IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 27 July 2023, accepted 11 August 2023, date of publication 21 August 2023, date of current version 25 August 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3306957

RESEARCH ARTICLE

A Broad Ensemble Learning System for Drifting Stream Classification

SEPEHR BAKHSHI[1], POUYA GHAHRAMANIAN[1], HAMED BONAB[2,3], AND FAZLI CAN[1], (Member, IEEE)

[1]Department of Computer Engineering, Bilkent University, 06800 Ankara, Turkey
[2]Manning College of Information and Computer Sciences, Amherst, MA 01003, USA
[3]Amazon.com, Inc., Seattle, WA 98109, USA

Corresponding author: Fazli Can (canf@cs.bilkent.edu.tr)

This work was supported in part by the Türkiye Bilimsel ve Teknolojik Araştırma Kurumu (TÜBİTAK) under Grant 120E103.

ABSTRACT In a data stream environment, classification models must effectively and efficiently handle concept drift. Ensemble methods are widely used for this purpose; however, the ones available in the literature either use a large data chunk to update the model or learn the data one by one. In the former, the model may miss the changes in the data distribution, while in the latter, the model may suffer from inefficiency and instability. To address these issues, we introduce a novel ensemble approach based on the Broad Learning System (BLS), where mini chunks are used at each update. BLS is an effective lightweight neural architecture recently developed for incremental learning. Although it is fast, it requires huge data chunks for effective updates and is unable to handle dynamic changes observed in data streams. Our proposed approach, named Broad Ensemble Learning System (BELS), uses a novel updating method that significantly improves best-in-class model accuracy. It employs an ensemble of output layers to address the limitations of BLS and handle drifts. Our model tracks the changes in the accuracy of the ensemble components and reacts to these changes. We present our mathematical derivation of BELS, perform comprehensive experiments with 35 datasets that demonstrate the adaptability of our model to various drift types, and provide its hyperparameter, ablation, and imbalanced dataset performance analysis. The experimental results show that the proposed approach outperforms 10 state-of-the-art baselines, and supplies an overall improvement of 18.59% in terms of average prequential accuracy.

INDEX TERMS Data stream mining, concept drift, ensemble learning, neural networks, big data.

I. INTRODUCTION

Various data stream sources generate an immense amount of data in the blink of an eye. Social media, IoT devices, and sensors are all examples of such sources. The ''3 V's of Big Data Management'' summarizes the hurdles in this field. These are the Volume of the data, Variety which refers to numerous data types, and Velocity, which is one of the major problems in handling data streams due to its fast data arrival rate [1]. As a result of these hurdles, building models that are capable of learning in a streaming environment is a challenging task. The developed methods require an approach specifically designed for this task, as it faces problems different from traditional machine learning.

The associate editor coordinating the review of this manuscript and approving it for publication was Jerry Chun-Wei Lin.

A major issue in data stream mining is *concept drift*, which refers to changes in the probability distribution of data over time [2], [3], [4], [5]. If the change affects the decision boundaries it is referred to as *real drift*, and if the distribution is altered without affecting the decision boundaries, it is called *virtual drift* [6]. Concept drift is mainly categorized into four types: *Abrupt*, *Gradual*, *Incremental*, and *Recurring* [3]. In abrupt drift, the concept is suddenly altered to a new one which usually results in a prompt accuracy decline; however, gradual drift refers to the replacement of the old concept with a new one in a gradual way. In incremental drift, an old concept changes to a new one incrementally over a period of time. The main difference between gradual and incremental drift is that in the former one, the class distribution is also prone to changes; however, in the latter, the values of the data are changed during a span of time [7]. Recurring drift occurs

when an old and previously observed concept replaces the current one.

In terms of their implementation, data stream classification models in the literature could be categorized into two main approaches: *chunk-based methods* and *online learning* [8]. In a chunk-based approach, a fixed-size chunk of data is typically collected to update the model as new data items arrive; however, in an online learning approach, one data point at a time is used to update the system. Each approach has its pros and cons. Compared to online methods, a chunk-based approach learns faster at the start of the training because the model is seeded with a large initial chunk of data. This makes the update process more effective at the initial steps; however, a concept drift may occur later in the learning process, and if the drift is located within a chunk, it may be missed. Such a concept drift may result in a significant reduction in model accuracy. In the case of online learning, the latter problem does not affect the performance of the model; however, the model may suffer from slower initial learning performance [9]. Another problem of an online model is its runtime. Compared to the chunk-based models, online learning methods are less efficient and require more computations. Online models also suffer from instability as they learn the data one at a time [9].

A. MOTIVATION

The disadvantages of using an online or chunk-based model motivate us to propose a method to alleviate the issues in these two approaches while keeping their positive features. Our main goal is to propose a resilient model that is able to function effectively and efficiently in a stream environment. To do so, we propose our solution based on Broad Learning System (BLS) [10], a lightweight neural network. Using lightweight neural networks for concept drift adaptation is a relatively unexplored research avenue [11] and in this work, we aim to explore its potential for stream processing.

BLS solves a much simpler least square equation for training the model in place of time-consuming loss calculations and backpropagation. Although an incremental version of BLS is introduced in the original paper, BLS is not suitable yet to be used in a stream environment for the following two reasons: (i) in incremental mode, BLS is effective only when the chunk size is large; and (ii) the model has no mechanism to handle concept drift. Since using large chunks may result in missing concept drifts inside a chunk, we use mini chunks ($2 \leq$ chunk size ≤ 50). Above all, to handle the problem of slow learning in the initial steps of an online model, or a model that is trained with mini chunks, we adapt the BLS to learn with small chunk sizes and have a comprehensive feature mapping and output layer. We track the changes in the accuracy of the ensemble components, and remove, replace, or add them to the model at each time step. With this frequent exchange of the ensemble components between a pool of removed ensemble components and the ensemble itself, we create an ensemble of diverse classifiers, as each of these ensemble components is trained with different data. This enables our model to react faster to concept drifts, and in the case of false removal of a component, it is quickly returned to the learning process. The same feature also helps our approach to be resilient. As mentioned by Vardi [12], "*resilience via distributivity and redundancy is one of the great principles of computer science*", and this principle is one of the main reasons that ensemble approaches function effectively in a stream environment, and compared to a single classifier, are more resilient to changes in the data distribution. This redundancy (using several classifiers in an ensemble) decreases the efficiency of the model. Given that learning with mini chunks and using an ensemble both have a negative impact on the efficiency of the model in terms of computational cost, we only use output layers of the Broad Learning System as our ensemble components and use a single feature mapping and enhancement layer. Meaning that the ensemble consists of a part of BLS and not the whole BLS model. This results in a significant reduction in computational burden. In-depth technical aspects of our ensemble model are demonstrated in the following sections.

B. CONTRIBUTIONS

Our main contributions are the following. We

- Design and mathematically derive an enhanced version of BLS suitable for stream environment, trained with small chunks of data;
- Propose an efficient and effective passive ensemble approach for concept drift handling based on tracking the changes in the accuracy of each ensemble component, and utilizing the output layers of the BLS as the ensemble components;
- Conduct experiments on 35 datasets with various concept drift types, and compare our results with 10 state-of-the-art baselines.
- Assess the performance of our model in imbalanced drifting data streams and show that it maintains a competitive behavior with the most recent method designed specially for imbalanced learning, named ROSE [13];
- Provide our implementation on GitHub[1] so that all of our results can be reproduced and our method can be easily used as a baseline in future studies and modified for new purposes.

In the upcoming sections, we first review the related works in the literature in Section II. Next, we explain our proposed approach in detail in Section III. Then the experimental design is presented in Section IV. We report our experimental results and compare our model with the baselines in Section V. We conclude our work and specify a future direction in Section VI.

II. RELATED WORK

In this section, we study the proposed approaches for data stream classification from three different perspectives.

[1] https://github.com/sepehrbakhshi/BELS

We begin by reviewing the methods for concept drift adaptation in the literature (1), then we present the ensemble approaches and categorize them into *active vs. passive* (2), and *chunk-based vs. online* algorithms (3) [14].

A. CONCEPT DRIFT ADAPTATION

For concept drift adaptation, two approaches are mainly studied in the literature. As mentioned by Gama et al. [4], these two *model management* techniques are *single classifiers* [8] and *ensemble methods* [15].

A single classifier is normally accompanied by a concept drift detector. The detection algorithm utilizes alerts to warn the classifier of a drift. To find a drift point, detection methods usually follow the error rate and trigger an alarm in the case of an unusual decline in overall accuracy. DDM [5], EDDM [16], ADWIN [17], and OCDD [18] use this strategy.

Another approach is to keep track of the statistical changes in the data distribution. PCA-CD [19], EDE [20], and CM [21] are designed based on this method. After an alarm is triggered, the classifier usually restarts learning from that point on. kNN and the Hoeffding Tree [22] are among the most popular classifiers used for this purpose. All the above-mentioned methods are supervised; however, unsupervised detection of concept drift is also studied by Gözüaç1k et al. in a recent work [23]. Another work on unsupervised concept drift detection, but this time on multi-label classification, is presented by Gulcan and Can [24].

Models that use a single classifier are efficient; however, retraining the model from scratch or replacing it with a new classifier, results in a delay in the learning process, since the model loses useful information learned so far; furthermore, in the case of a recurrent drift, the model should learn an already learned concept from scratch.

Ensemble methods are one of the most powerful tools for analyzing data that enable us to handle various issues in knowledge discovery from imbalanced learning to handling noisy data [25]. Many works are proposed based on ensemble learning for static environment [26], [27], [28]. Inspired by their impressive performance in static settings, ensemble methods are widely studied in data stream environments. In ensemble-based models, a combination of learners is used to make the final decision. For concept drift adaptation, ensemble methods use various techniques; however, there are a few similar strategies that most of them utilize to maintain their high effectiveness during the learning process. For instance, they may have an adaptive strategy that adds and removes the classifiers based on their performance. Some ensemble methods preserve the old classifiers that are removed from the ensemble [4], and utilize these classifiers in the later stages of the learning process based on the needs of the model. This approach helps the ensemble model to have an effective reaction to concept recurrence. Our proposed approach uses this strategy, but one of the drawbacks of preserving the old classifiers is the high storage and computational burden. Using a limit for the number of preserved classifiers is a simple yet effective strategy that we use for alleviating this issue. Another considerable advantage of declaring a limit for the number of classifier components of an ensemble is that controlling the ensemble size provides a consistent runtime during stream classification. The computational load of the model increases substantially as the number of classifier components increases, and this leads to an inefficient and sometimes broken system that is unable to process new data.

Apart from the brief description of ensemble methods presented above, we classify them from two perspectives. Initially, we consider the way they handle concept drift. In this sense, ensemble approaches fall into two subgroups: *Active and Passive*. Furthermore, when it comes to data processing criteria, two primary options are available: *Chunk-Based and Online*.

B. ACTIVE vs. PASSIVE ENSEMBLE METHODS

Active methods rely on a concept drift detection method to trigger an alarm. Then, the ensemble model reacts to this drift by adding new classifiers, updating them, or restarting their learning process. Adaptive Random Forest (ARF) [29], Leveraging bagging [30], Adaptive Classifiers Ensemble (ACE) [31], Heterogeneous Dynamic Weighted Majority (HDWM) [32], comprehensive active learning method for multi-class imbalanced streaming data with concept drift (CALMID) [33], Streaming Random Patches (SRP) [34], and Robust online self-adjusting ensemble for continual learning on imbalanced drifting data streams (ROSE) [13] are among the well-known algorithms in this category.

In passive models, no concept drift detector is used, and a weighting strategy is commonly employed to help the model adapt to the new changes [35]. The model may also add or remove classifiers from the ensemble. With this method, the ensemble relies on the most recent data to make a prediction [8]. Additive Expert Ensembles (AddExp) [36], Dynamic Weighted Majority (DWM) [37], Geometrically Optimum and Online-Weighted Ensemble (GOOWE) [38], Learn++. NSE [39], Resample-based Ensemble Framework for Drifting Imbalanced Stream (R-DI) [40], and Kappa Updated Ensemble (KUE) [41] fall into this category.

C. CHUNK-BASED vs. ONLINE ENSEMBLES

In a chunk-based ensemble, a chunk of data is collected before each update. Using large chunks of data causes some problems like missing the actual drift point. This leads to delayed response in case of a concept drift and decreases the accuracy of the model. In contrast to online models, chunk-based models are more efficient as they process a chunk of data simultaneously instead of processing the data one by one. Accuracy Weighted Ensemble (AWE) [42], Learn++. NSE [39], and Accuracy Updated Ensemble (AUE) [43] are in this category.

Unlike the chunk-based models, online ensembles process each data item separately, eliminating the need to gather a

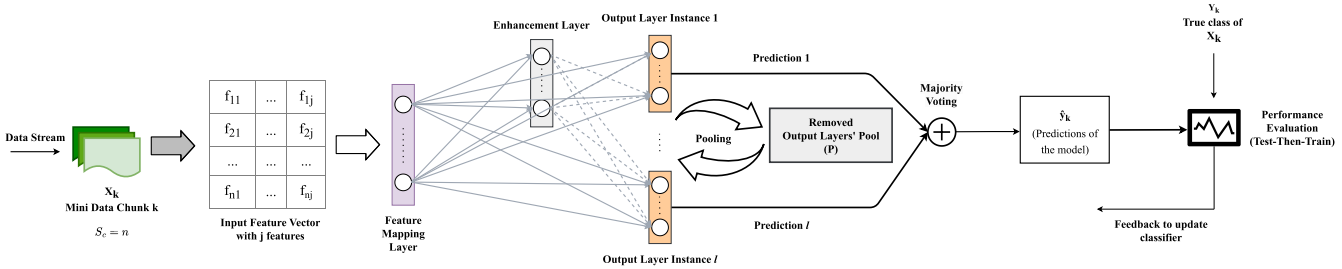


FIGURE 1. Overall schema of Broad Ensemble Learning System (BELS) for drifting stream classification.

chunk of data. This approach increases memory efficiency. On the other hand, processing a single data instance sequentially is time-consuming, and may result in the instability of the model. Meaning that small changes in the input may negatively affect the learning power of the model. Diversity of Dealing with Drift (DDD) [44], DWM [37], and AddEXP [36] are examples of online ensembles.

III. METHOD

This section is organized as follows. First, we introduce the problem definition. Next, we provide a brief overview of the BLS approach. Then, we present the technical details of our proposed approach, which we call BELS.

A. PROBLEM DEFINITION

In a data stream environment, data is continuously generated via a source over time. We refer to a data chunk at time step k as X_k, and define the problem of data stream classification as follows. First, the model generates the probability values for each class to predict the class of incoming data chunk X_k. Assuming that there are C predefined classes, s_k is the probabilities vector with the length of the number of labels. For each instance, we assume that the correct label becomes available at time step $k+1$, making the prequential evaluation [45] process possible. The model is then incrementally updated using the correct labels of X_k and the obtained features. Fundamentally, this process is the assumption used in the majority of the data stream classification studies and is known as the test-then-train learning paradigm in the literature [45]. As a result, every data instance is used both for training and testing. We propose BELS as being specifically designed for this problem setting. In the following subsections, we introduce our method step by step.

B. BLS: BROAD LEARNING SYSTEM

Broad Learning System (BLS) [10] achieves high performance in static environments, both in terms of runtime and accuracy. In its architecture, the number of layers is minimized and instead, each layer consists of several nodes. In the first stage, BLS takes the input data and creates a feature mapping layer using a linear function. Then an activation function is used in the enhancement layer, and the output of the feature mapping layer is fed to it. Next, the output of both the feature mapping layer and the enhancement layer is concatenated and fed to the output layer. To determine the output weights, BLS solves the least squares problem by finding the pseudoinverse.

In the feature mapping layer, the data is mapped using $\phi_i(XW_{e_i} + \beta_{e_i})$ function, where W_{e_i} and β_{e_i} are random weights and bias of the ith mapped feature respectively. W_{e_i} and β_{e_i} are initiated randomly. Each mapped feature is used to form a group of n mapped features by concatenation, where n is the maximum number of feature mapping nodes in each group [10].

This concatenation is denoted as $Z^n = [Z_1, \ldots, Z_n]$ [10]. Next, the output of this feature mapping layer is fed to an activation function and forms the enhancement layer. Furthermore, ith feature map is used to form the jth enhancement node H_j as follows: $\xi(Z^i W_{hj} + \beta_{hj})$ [10]. Like the feature mapping layer, the concatenation of m enhancement nodes is grouped as $H^m = [H_1, \ldots, H_m]$ [10] where m is the maximum number of enhancement nodes. Based on the BLS paper [10], the broad model is defined as follows:

$$\begin{aligned} Y &= [Z_1, \ldots, Z_n | \xi(Z^n W_{h1} + \beta_{h1}), \ldots, \xi(Z^n W_{hm} + \beta_{hm})]W \\ &= [Z_1, \ldots, Z_n | H_1, \ldots, H_m]W \\ &= [Z^n | H^m]W \end{aligned}$$

The ultimate goal of the system is to find W by solving a least square problem.

BLS proposes an incremental way of updating the system for large datasets. The incremental approach uses a large chunk of data to update the system at every step; however, there are three main reasons that make BLS an ineffective system while dealing with data streams: i) BLS uses the initial set of incoming data to calculate the feature mapping. The problem is that in each time step, the system uses the same data representation without any update. ii) By using a smaller chunk size, the proposed pseudoinverse updating in BLS is not effective, since it needs a very large chunk of data for an effective update. iii) There is no mechanism to handle concept drift in BLS and its proposed variants. Note that our proposed model is different from other variations in [46]. In [46], the proposed variations of BLS are not designed for a stream environment, and can not handle concept drift. Furthermore, our model utilizes an ensemble approach and

focuses on using small chunk sizes for tracking the changes in the accuracy of the ensemble components.

C. BELS: BROAD ENSEMBLE LEARNING SYSTEM

To tackle the problems mentioned in Section III.B we propose BELS [47]. In Sections III-C1 and III-C2 we propose a solution for considering the whole data items for generating the feature mapping, enhancement, and output layers, which prepares the model for effective updates with smaller chunk sizes ($S_c \leq 1000$). In Section III-C3 we introduce our ensemble approach for dealing with concept drift.

Figure 1 shows the overall structure of BELS. As we can see in this figure, when the first chunk of data is received, it is fed to the feature mapping layer. Next, the output of the feature mapping layer is fed to the enhancement layer. Subsequently, the output of these two layers is concatenated and fed to an ensemble of output layers, and the final decision is made based on the majority voting. We utilize a pool of removed output layers where the ones removed from the ensemble are kept in it. Frequent exchange of the removed output layer instances in the pool and the active ones in the ensemble is used as a strategy for handling concept drift. A detailed explanation of each step is presented in the following subsections.

1) UPDATING THE FEATURE MAPPING IN BELS

BLS employs a sparse autoencoder to overcome the randomness of the generated features. We use the same method to deal with this issue; however, despite BLS, we update this feature representation after each chunk of the data.

To obtain this feature representation, BLS uses an iterative method. Eq. (1) is defined in BLS paper for this purpose [10]. The output of these iterative steps is denoted as μ.

$$\begin{cases} w_{i+1} := \left(z^T z + \rho I\right)^{-1} \left(z^T X_k + \rho \left(o^i - u^i\right)\right) \\ o_{i+1} := S_{\lambda/\rho}\left(w_{i+1} + u^i\right) \\ u_{i+1} := u^i + (w_{i+1} - o_{i+1}) \end{cases} \tag{1}$$

In Eq. (1), X_k is the input data in time step k and z is the projection of the input data using $XW_{e_i} + \beta_{e_i}$ for the ith feature group. W_e and β_e are randomly generated with proper dimensions initialized at the beginning of the first time interval. Note that we apply the same W_e and β_e during the training for each update. u^i, o^i, and w^i are initialized as zero matrices at the beginning of the iteration. These matrices are only used in the updates of the iterative steps of Eq. (1). In the formula, $\rho > 0$, I is the identity matrix, and S is the soft thresholding operator [10]. In Eq. (2), (a) is the sum of w_{i+1} and u^i. In our experiments, we use $\kappa = 0.001$. S is calculated as follows [10]:

$$S_\kappa(a) = \begin{cases} a - \kappa, & a > \kappa \\ 0, & |a| \leq \kappa \\ a + \kappa, & a < -\kappa \end{cases} \tag{2}$$

While considering the incremental input in a stream environment, the projection of data is different for each chunk of data. Meaning that we can not utilize the first set of data to calculate the μ, and use the same μ for the rest of the incoming data. Additionally, if the data chunk is small, then this projection is not comprehensive enough. To solve this problem, we propose an updating system for Eq. (1), that in each step k, μ^i is the projection of the entire data from step one to step k. In each time step, the system uses the renewed μ, and this helps the model to have a comprehensive feature mapping layer that represents the entire data up to that time step. This technique improves the accuracy of the model drastically when dealing with both small and large chunks.

To implement this idea, we need to update $z^T X_k$ and $z^T z$ in each time step for every set of mapping features in k, separately.

Let us denote $z^T X_k$ as T_1. We know that the dimensions of T_1 are independent of the number of data instances in each time step, and depend on the number of feature mapping nodes, as well as the number of features in each data instance. Based on this fact, we can use the following theorem in our method:

Theorem 1: For two matrices A and A' with the same number of columns, if we multiply A^T with A, and A'^T with A', the result of both multiplications are square matrices with the equivalent size. We refer to them as A_t and A'_t, respectively. Let us concatenate A and A' vertically and denote the new matrix as A_c. The product of A_c^T and A_c is a square matrix equal to the sum of A_t and A'_t. The hypothesis can be formulated as follows:

$$A_c^T A_c = A^T A + A'^T A' \quad \textit{where:} \quad A_c = \left[A \mid A'\right] \tag{3}$$

Proof: Let A be an m by n matrix and let A' be an m' by n matrix. A_c is obtained by concatenating A and A' matrices vertically as follows:

$$A_c = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \\ a'_{11} & a'_{12} & \dots & a'_{1n} \\ a'_{21} & a'_{22} & \dots & a'_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a'_{m'1} & a'_{m'2} & \dots & a'_{m'n} \end{bmatrix}$$

The left-hand side of the Eq. (3) is the multiplication of A_c^T and A_c matrices. Let us denote the element at i*th* row and j*th* column of the resulting matrix as l_{ij}. Also, let the (i, j)*th* element of the resulting matrix on the right-hand side of the Eq. (3) be represented by r_{ij}. In the following, we show that these two elements are equal regardless of the i and j values, meaning that the resulting matrices in the left and right-hand

sides of the Eq. (3) are equal.

$$l_{ij} = \sum_{k=1}^{m+m'} A_c^{ki} \times A_c^{kj} = \sum_{k=1}^{m} A_c^{ki} \times A_c^{kj} + \sum_{k=m+1}^{m+m'} A_c^{ki} \times A_c^{kj}$$
$$= \sum_{k=1}^{m} a_{ki} \times a_{kj} + \sum_{k'=1}^{m'} a_{k'i} \times a_{k'j}$$
$$r_{ij} = (A^T A)^{ij} + (A'^T A')^{ij} = \sum_{k=1}^{m} a_{ki} \times a_{kj} + \sum_{k'=1}^{m'} a_{k'i} \times a_{k'j}$$

□

Based on Theorem 1, Eq. (4) is used to update T_1 value.

$$T_{1_k} = \sum_{i=1}^{k} T_{1_i} \tag{4}$$

Let us denote $z^T z$ as T_2. Assuming that the number of columns in z does not change during the training phase, based on Theorem 1, we define the Eq. (5) to update T_2 at each step k:

$$T_{2_k} = \sum_{i=1}^{k} T_{2_i} \tag{5}$$

We use T_{2_k} and T_{1_k} as the input for the modified version of Eq. (1), and utilize it as in Eq. (6).

$$\begin{cases} w_{i+1} := \left(T_{2_k} + \rho I\right)^{-1} \left(T_{1_k} X_k + \rho \left(o^i - u^i\right)\right) \\ o_{i+1} := S_{\lambda/\rho}\left(w_{i+1} + u^i\right) \\ u_{i+1} := u^i + (w_{i+1} - o_{i+1}) \end{cases} \tag{6}$$

The enhancement nodes are created using the following formula [10]:

$$H_j = [\tanh(Z_k{}^n W_{h_j} + \beta_{h_j})] \tag{7}$$

$Z_k{}^n$ is a set of feature mapping nodes at time step k, and W_{h_j} and β_{h_j} are generated randomly. It's noteworthy that the enhancement layer is used as it is proposed in [10].

While updating the model, the random weights are fixed and the enhancement nodes are updated at each step. The process of updating the feature mapping and enhancement layers is shown in Algorithm 1.

After concatenating the output of the feature mapping layer and the enhancement layer horizontally, the next step is calculating the pseudoinverse for the least square problem discussed in BLS [10].

2) UPDATING THE PSEUDOINVERSE IN BELS

Unlike BLS, where the pseudoinverse is calculated based on the instances of the last chunk of the data, we revise this calculation such that it represents the pseudoinverse of the whole data until that time step.

Suppose that A is the result of concatenating the output of the feature mapping layer and the output of the enhancement layer horizontally. To obtain the pseudoinverse, BLS uses Eq. (8) [10]:

$$W = (\lambda I + AA^T)^{-1} A^T Y \tag{8}$$

where Y is the labels of a chunk of training data and λ is a small positive value added to the diagonals of A. To calculate weights of the output layer, which is considered as the solution for the least square problem, we update AA^T at each time step k and refer to it as A_t:

$$A_t = A^T{}_k A_k \tag{9}$$

Then, we multiply $A^T{}_k$ by Y_k, which is the set of labels at time step k and define it as follows:

$$D_t = A_k^T Y_k \tag{10}$$

A_{tk} and D_{tk} values are obtained as:

$$A_{t_k} = \sum_{i=1}^{k} A_{t_i} \quad \text{and} \quad D_{t_k} = \sum_{i=1}^{k} D_{t_i} \tag{11}$$

Based on theorem 1, we know that at the end of step k, A_{tk} and D_{tk} are equal to A_t and D_t of the entire data until that time step, respectively. The process is shown in Algorithm 2. First we calculate A_{tk} and D_{tk} (Algorithm 2:1-9). Then, Eq. (12) is used to update the pseudoinverse (Algorithm 2:10).

$$W = \left(\lambda I + A_{t_k}\right)^{-1} D_{t_k} \tag{12}$$

Algorithm 1 Feature Mapping and Enhancement Layer Update

Input: data chunk X_k
Output: a set of feature mapping and enhancement nodes denoted as A_k

initiate random W_e, W_h, β_e and β_h at the beginning
X_k = data instances at step k
for i=0; i ≤ n **do**
  $z = (X_k W_{e_i} + \beta_{e_i})$
  $T_1 = X_k z^T$
  $T_2 = z^T z$
  if $k = 0$ **then**
    $T_{1_k} = T_1$
    $T_{2_k} = T_2$
  else
    $T_{1_k} = T_{1_{k-1}} + T_1$
    $T_{2_k} = T_{2_{k-1}} + T_2$
  end if
  calculate μ_i with Eq. (6)
  $Z_i = X\mu_i$
end for
set the feature mapping group $Z_k{}^n = [Z_1, \ldots, Z_n]$
for j ← 1; j ≤ m **do**
  calculate $H_j = [\tanh(Z_K{}^n W_{h_j} + \beta_{h_j})]$ with Eq. (7)
end for
set the enhancement node group $H_k{}^m = [H_1, \ldots, H_m]$
  $A_k = [Z_k{}^n | H_k{}^m]$

Algorithm 2 Output Layer Weight Calculation

Input: a set of feature mapping and enhancement nodes denoted as A_k

Output: W

$A_t = A_k^T A_k$
$D_t = A_k^T Y_k$
if $k = 0$ **then**
  $A_{t_k} = A_t$
  $D_{t_k} = D_t$
else
  $A_{t_k} = A_{t_{k-1}} + A_t$
  $D_{t_k} = D_{t_{k-1}} + D_t$
end if
$W = (\lambda I + A_{t_k})^{-1} D_{t_k}$

TABLE 1. Additional symbols and notation for BELS (Algorithm 3).

Symbol	Meaning
ξ	Ensemble of output layers $\xi = \{O_1, O_2, O_2, ..., O_l\}$
P	The pool containing the removed output layers
L	List of indexes of output layer instances with an accuracy lower than the threshold (for each chunk)
${s_i}^k$	Relevance scores for the ith classifier for kth data chunk in the ensemble. ${s_i}^k =< {s_{i1}}^k, {s_{i2}}^k, {s_{i3}}^k, ... >$
bP	Output layer instances from P which achieve an accuracy higher than a threshold in a chunk (bP stands for best of pool)
M_o	Maximum number of output layer instances
M_p	Maximum number of output layer instances in P
δ	Accuracy threshold for removing an ensemble component
S_c	Chunk size

For testing, we use a similar approach to BLS [10]; however, like the previous steps, we separate the calculations related to feature mapping and enhancement layers, and the output layer. Let us assume that $A_{test\,k}$ is the concatenation of test results of feature mapping and enhancement layers. We use $A_{test\,k}$ for producing the prediction based on the following formula in the final stage [10]:

$$\hat{y}_k = A_{test\,k} W \tag{13}$$

3) CONCEPT DRIFT ADAPTATION IN BELS

For concept drift handling, we use an ensemble approach. Our approach is passive as we do not use any concept drift detection mechanism. Simply, we keep updating the feature mapping layer and enhancement layer as new data chunk arrives; however, an ensemble of output layer instances is used to determine the final result. Each output layer instance is a collection of output layer nodes.

There are two main reasons for excluding the feature mapping and enhancement layers from our ensemble: (i) Using an ensemble of a complete BLS model, which includes feature mapping, enhancement, and output layers, is not efficient as it requires additional calculations,(ii) Initializing the feature mapping and enhancement layer for each incoming data chunk delays the learning process, because the initial feature mapping and enhancement layers of the data are not comprehensive. In our ensemble, the output layer instances with the best prediction accuracy in the last chunk are kept in the model, and those with an accuracy less than the threshold δ are replaced with a new one, or one of the output layer instances in the pool. The pool consists of removed output

Algorithm 3 BELS (Broad Ensemble Learning System)

Require: D: data stream, X_k: data chunk at step k, Y_k: labels of the data chunk at step k, δ: accuracy threshold

Ensure: $\hat{y}_k$: prediction of the ensemble as a score vector at step k

while D has more instance **do**
  if ξ is not full **then**
    $\xi \leftarrow$ add new output layer
  else if ξ is full **or** L length $>$ (ξ length /2) **then**
    for i $\leftarrow$ 0; i $\leq L$ length - 1 **do**
      remove ξ[L[i]]
      if L length $>$ (ξ length/2) **then**
        $P \leftarrow \xi[L[i]]$
      end if
    end for
    while ξ is not full and $i <$ bP length **do**
      $\xi \leftarrow$ bP[i]
      remove bP[i] from P
    end while
    if P length $> M_p$ **then**
      keep the last M_p instances of P
    end if
  end if
  calculate $A_{test\,k}$
  for i $\leftarrow$ 0; i $\leq \xi$ length **do**
    if O_i is initialized **then**
      ${s_i}^k$, ${\text{acc}_i}^k \leftarrow$ prediction & acc of O_i Eq.(13)
      if S_c!= 2 **then**
        δ = overall acc
      end if
      if ${\text{acc}_i}^k < \delta$ **then**
        $L \leftarrow$ i
      end if
    end if
  end for
  $\hat{y}_k \leftarrow$ use the set of ${s_i}^k$ for hard voting
  for j $\leftarrow$ 0; j $\leq P$ length **do**
    ${\text{acc}_j}^k \leftarrow$ test $P[j]$ using Eq. (13)
    if ${\text{acc}_j}^k > \eta$ **then** bP $\leftarrow P[j]$
    end if
  end for
  $A_k \leftarrow$ update F and E using Algorithm 1.
  for i $\leftarrow$ 0; i $\leq \xi$ length **do**
    update O_i using Algorithm 2.
  end for
end while

layer instances. Basically, our proposed approach for managing concept drift involves regularly swapping out the output layer instances that have been removed from the pool or newly generated ones, with the active ones in the ensemble. Using small chunk sizes results in more exchanges, and this helps our ensemble to have a set of diverse components. Besides, it also reacts swiftly to any changes in the distribution of data. Symbols and notations used in the ensemble learning process and the detailed steps of our approach are shown in Table 1 and Algorithm 3, respectively.

Our model is defined with three independent but connected parts. Feature mapping layer denoted by F, enhancement layer denoted by E and output layer denoted by O. BELS consists of a single F and E and an ensemble of l output layer instances $\xi = \{O_1, O_2, O_2, \ldots, O_l\}$. To update these parts at each time step, Algorithm 1 and Algorithm 2 are used. A set of new data instances $I = \{I_1, I_2, I_3, \ldots, I_{S_c}\}$ where $2 \leq S_c \leq 50$ is used for each update.

Let us assume that we have received the first chunk of data and our model is initialized. In the upcoming iterations, first, we check if the number of output layer instances has reached the predefined maximum number (Algorithm 3: 2-3). If ξ is full, then the model removes the output layer instances that have an accuracy less than a threshold. If the number of output layer instances in L is more than the instances in ξ, then the output layer instances in L are added to P (Algorithm 3:4-10). Next, the model adds the previously removed output layer instances from bP back to the model (Algorithm 3:11-14). bP Consists of output layer instances that were removed once and now are eligible to return to the learning process. Then we check the size of the pool, and if it passes the predefined threshold (M_p), we only keep the last M_p output layer instances in the pool.

In the testing phase, the accuracy of each output layer instance is calculated, and the index of the worst-performing ones is added to L (Algorithm 3:20-30).

Hard voting is used for calculating the final prediction. In hard voting, the score vector of an output layer instance s^k is first transformed into a one-hot vector, and then combined with the s^k of the other output layer instances to determine the final prediction (Algorithm 3:31). Next, output layer instances in P are tested. If any of them has an accuracy more than a threshold (denoted by η), then that output layer instance is added to bP (Algorithm 3:32-36). This process gives the output layer instances in P a chance to be added to ξ and used in the learning process as of the next time step. Finally, the model is updated (Algorithm 3:38-40) and the same processes are repeated.

4) TIME COMPLEXITY ANALYSIS

Since we assume that the calculations for building each feature mapping and enhancement node take $O(1)$ time, let us assume that building the feature mapping layer and enhancement layer takes $O(n)$ and $O(m)$ time respectively, where n is the number of feature mapping nodes, and m is the number of enhancement nodes. Based on this assumption, we conclude that Algorithm 1 and the first phase of testing (which includes generating the feature mapping and enhancement layers) take $O(n + m)$ time. Next, we have our ensemble method in Algorithm 3. Execution time for initializing the model and removing the output layer instances, or adding them back to the model (Algorithm 3:2-18) are negligible.

Let us denote the ensemble size as S_ξ. Then, the final prediction for each output layer instance is calculated in (Algorithm 3:20-30), and it takes $O(S_\xi)$ time.

Later, we test the output layer instances in the pool in (Algorithm 3:32-36). This process takes $O(P)$ time. For training, we first update the feature mapping and enhancement layer using Algorithm 1. This process is executed once for each chunk, and it takes $O(n+m)$ time. Finally, we update the output layer instances using Algorithm 2. In this algorithm, we calculate the pseudoinverse. Let us denote chunk size as S_c. Based on the dimensions of D_{tk} and A_{tk}, we conclude that Algorithm 2 takes $O(\max(S_c, (n+m))^2 \min(S_c, (n+m)))$ time.

For $\frac{N}{S_c}$ chunks of data, the whole process complexity is as follows:

$$O\Big(\frac{N}{S_c}\Big(2(m+n) + S_\xi + P + \big(S_\xi \ \max(S_c, (n+m))^2 \min(S_c, (n+m))\big)\Big)\Big) \tag{14}$$

Obviously, among different parts of the algorithm, the one with $O(\xi \ \max(S_c, (n+m))^2 \min(S_c, (n+m)))$ time complexity dominates the whole process. The final complexity of the algorithm is as follows:

$$O\Big(\frac{N}{S_c}\big(S_\xi \ \max(S_c, (n+m))^2 \min(S_c, (n+m))\big)\Big) \tag{15}$$

The analysis shows that our model's efficiency in terms of processing time is agnostic of the feature set size, and depends on the chunk size, number of ensemble components, and the number of nodes. This ability allows the user of the model to set the hyperparameters based on the needs of the project in a real-world environment. As we discuss later, the efficiency of the baseline methods heavily relies on the size of the feature set, and it significantly decreases as the feature set size increases.

IV. EXPERIMENTAL DESIGN

In this section, first, the datasets and baseline methods are introduced, then we elaborate on our experimental setup.

A. DATASETS

In our experiments, we use 30 datasets to evaluate the effectiveness of our model (in terms of accuracy) and efficiency (in terms of runtime) and compare it with the baselines. Five additional synthetic datasets are also used to assess the performance of the proposed model on an imbalanced learning scenario. Therefore, in our experiments, the total number of datasets is 35. Our datasets cover a wide spectrum

TABLE 2. Properties of 35 Datasets used in the experiments (Type: R - Real, S - Synthetic; DT: A - Abrupt, I - Incremental, G - Gradual, R- Recurring).

Dataset	Type	DT	# F	# C	Size
Airlines	R	U	7	2	539,383
CoverTypes**	R	U	54	7	581,012
Electricity[2] [48] **	R	U	8	2	45,312
Email[1] [49]	R	A&R	913	2	1,500
Phishing[2] [50]	R	U	46	2	11,055
Poker [2] [51]**	R	U	10	10	829,201
Spam[2] [50] **	R	U	499	2	6,213
Usenet[3] [52]	R	A&R	99	2	1,500
Weather[2] [39]**	R	U	8	2	18,152
Agrawal-Gradual-456	S	G	9	2	100,000
hyperplane-25	S	G	10	2	100,000
hyperplane-50	S	G	10	2	100,000
Interchanging RBF[2] [53]	S	A	2	15	200,000
LED-drift	S	A	7	10	100,000
MG2C2D[4] [54]	S	I&G	2	2	200,000
Mixed-Abrupt-Rec-1212	S	A&R	4	2	100,000
Moving Squares[2] [53]	S	I	2	4	200,000
RBF-Fast	S	G	10	4	100,000
RBF-Slow	S	G	10	4	100,000
Rotating Hyperplane[2] [53]	S	I	10	2	200,000
SEA-Abrupt-01210*	S	A&R	3	2	100,000
SEA-Abrupt-13123*	S	A&R	3	2	100,000
SEA-Incremental-01210*	S	I&R	3	2	100,000
SEA-Incremental-13123*	S	I&R	3	2	100,000
Sine-Gradual-234	S	G	4	2	100,000
Sine-Gradual-Rec-121	S	G&R	4	2	100,000
Stagger-Abrupt-Rec-1212	S	A&R	3	2	100,000
Stagger-Abrupt-Rec-1232	S	A&R	3	2	100,000
Waveform	S	-	21	3	100,000
Waveform-Noisy*	S	-	40	3	100,000
Imb-Mixed-Gradual-Rec-1212**	S	G&R	4	2	100,000
Imb-RBF-Fast**	S	G	10	4	100,000
Imb-RBF-Slow**	S	G	10	4	100,000
Imb-Staggar-Abrupt-Rec-1212**	S	A&R	3	2	100,000
Imb-Staggar-Abrupt-Rec-1231**	S	A&R	3	2	100,000

* Five noisy datasets are marked with (*).
** Five imbalanced datasets are marked with (**).

TABLE 3. List of parameters.

Parameter	Definition	Default Value
n	No. of feature mapping nodes	{25, 100}
m	No. of enhancement nodes	{1, 50, 100}
M_0	Max. No. of output layer instances	75
M_p	Max. No. of output layer instances in P	300
S_c	Chunk size	{2, 5, 10, 20, 50}
η	Accuracy threshold used in the pool	0.5
δ	Accuracy threshold used in the pool	Dynamic Update

of possibilities that can be observed in data streams. Their properties are presented in Table 2.

All four drift types are used in the experiments: Gradual (G), Incremental (I), Abrupt (A), and Recurring (R). In the same table, (U) stands for unknown drift type. For each dataset, its type (real or synthetic), the drift type, the number of features, and the number of class labels are denoted as Type, (DT), (# F), and (# C); respectively.

[2]https://github.com/ogozuacik/concept-drift-datasets-scikit-multiflow
[3]http://mlkd.csd.auth.gr/concept_drift.html
[4]https://sites.google.com/site/nonstationaryarchive/datasets

The first four synthetic datasets in Table 2 are from the available works in the literature [53], [54], and the LED, SEA, and Waveform datasets are generated using the scikit-multiflow library [57]. In the LED dataset, seven drifting features are used without noise. For the SEA dataset, each number in the dataset names represents one of the classification functions (there are four classification functions: {0, 1, 2, 3}). At each drift point, a new classification function is used to generate the stream. We introduce five drift points in each dataset. SEA-Abrupt and SEA-Incremental datasets have 10% and 20% noise, respectively. For adding noise to the datasets, the labels of the dataset change from 0 to 1 and vice versa, according to a probability distribution [57]. The Waveform-Noisy dataset adds 19 irrelevant attributes as noise to the dataset [57]. Hyperplane, RBF, Agrawal, Sine, and Stagger generators are also used to create synthetic datasets with various types and speeds of drift. These datasets are generated using the MOA library [58]. In the hyperplane dataset, two different magnitudes of change (0.25 and 0.50) are utilized. The RBF datasets have two different drift speeds indicated as 'Fast' and 'Slow' in their names. For the remaining datasets, the numbers indicate the sequence of classification functions used to synthesize the drift.

B. EXPERIMENTAL SETUP

The evaluation is based on the *interleaved-test-then-train* approach [45]. It is the most common technique for evaluating classification models in a data stream environment. In this approach, a data instance or a chunk of instances is first used for testing, then for training the model.

One of the most challenging tasks in data stream mining is optimizing hyperparameters for neural networks. Recently, a potential straightforward solution has been proposed, which involves using a pool of neural networks with varying hyperparameters [59]. During the learning process, all neural networks in the pool are trained, and after receiving each data instance, the model with the lowest loss (excluding the current data instance) is used for prediction; however, in our case, such an approach may result in a significant computational burden. To address this issue, we utilize a modified version of this method that only considers the first 1,000 data items to choose a set of hyperparameters for the rest of the learning process (for Usenet and Email datasets we use the first 100 data items due to their small size). For each dataset, we use this subset of data to perform a local grid search to determine the optimal number of feature mapping nodes, number of enhancement nodes, and chunk size. We then choose the combination of hyperparameters with the highest accuracy and continue using the same set of hyperparameters during the learning process. We limit the number of nodes and chunk sizes to a predefined set to reduce the time needed for the grid search. Therefore, three combinations of BELS are used as follows: $BELS_1$ (n: 25, m: 1), $BELS_2$ (n: 25, m: 50), $BELS_3$ (n: 100, m: 100), where, as defined earlier, n is the number of feature mapping nodes, and m is the number of

TABLE 4. Baseline methods with a short description.

Baseline Name	Short Description*
ARF [29]	Adaptive Random Forest is a tree-based classifier. Each base tree has a concept drift that signals a possible reset.
DWM [37]	Dynamic Weighted Majority uses an ensemble of weighted classifiers. Weights are assigned based on the performance of the classifiers.
GOOWE [38]	Geometrically Optimum and Online-Weighted Ensemble uses a geometrical approach for assigning weights for the classifiers.
HAT [55]	Hoeffding Adaptive Tree Classifier uses an ADWIN concept drift detector for detecting the drift and pruning the tree.
KUE [41]	Kappa Updated Ensemble combines online and chunk-based approaches using Kappa statistics to update the weights of classifiers.
LevBag [30]	Leveraging Bagging Classifier is an active ensemble approach built on top of the Oza Bagging method.
OzaADWIN [56]	Oza Bagging ADWIN Classifier utilizes a concept drift detector to enhance Online Bagging Algorithm's performance.
ROSE [13]	Robust Online Self-Adjusting Ensemble is a novel online method for handling concept drift and imbalance data in data stream settings.
SAM-kNN [53]	The Self Adjusting Memory (SAM) is a kNN-based ensemble method for data stream classification.
SRP [34]	Streaming Random Patches is an ensemble method that utilizes both bagging and random subspaces.

* Descriptions of each method are based on the information on scikit-mutliflow [57] web page or their original paper.

TABLE 5. Ablation study results: Comparison between BELS, its variants, and BLS in terms of average prequential accuracy and runtime (runtime is reported in centiseconds for processing of 1,000 data items). Accuracy improvement of BELS wrt. to BLS is provided in the last column. The best results for each dataset are in bold.

	BLS		BELS-FPs		BELS-Ens		BELS		% Acc. Imp.
Datasets (DT)	Accuracy	Runtime	Accuracy	Runtime	Accuracy	Runtime	Accuracy	Runtime	BELS wrt. BLS
Electricity (U)	49.94	**177.08**	60.54	180.79	84.68	285.95	**87.21**	597.06	74.62
Usenet (A & R)	54.76	**13.29**	59.71	14.49	**70.64**	16.58	70.29	25.80	28.36
MG2C2D (I & G)	61.37	**17.67**	91.06	22.81	92.06	83.61	**92.94**	301.65	51.44
Rotating Hyperplane (I)	81.54	**6.42**	81.76	7.96	85.94	10.08	**90.79**	54.66	11.34

enhancement nodes. For chunk size, we use a set of five different chunk sizes: {2, 5, 10, 20, 50}. These hyperparameters are tuned based on a grid search on several values, and the ones that contribute positively to both runtime and accuracy are chosen as default. The list of parameters is provided in Table 3.

The other three hyperparameters are set to default when the model is initiated: $M_o = 75$, $M_P = 300$, $\eta = 0.5$, and δ is updated dynamically in the model (See Algorithm 3:23-25).

To compare the performance of BELS with other methods, we choose 10 state-of-the-art models as baselines. The names of the baseline models and a short explanation of their approach are supplied in Table 4. All the baselines (except GOOWE, KUE, and ROSE) are available in the MOA library [58]. The source codes for GOOWE,[5] KUE,[6] and ROSE[7] are available on their GitHub. To have a fair comparison, we use the default configuration for BELS and baselines. For the baseline methods, the default values introduced in their papers are utilized in the experiments. All of the experiments are conducted in a chunk setting and the default chunk size of MOA is utilized except ROSE, which is inherently online and the chunk-based evaluation of the model is not available.

The experiments are conducted on a PC with Intel(R) Xeon(R) Gold 5118 CPU @ 2.30 GHz and 128 GB RAM on an Ubuntu 18.04.4 LTS operating system.

[5]https://github.com/abuyukcakir/goowe-python
[6]https://people.vcu.edu/ acano/KUE/
[7]https://github.com/canoalberto/ROSE

V. EXPERIMENTAL RESULTS AND EVALUATION

In this section, we first compare our method with its variants and also with the original BLS. Then we perform a thorough experimental evaluation with various baselines in terms of prequential accuracy and runtime. Next, we discuss the results of statistical significance analysis. Finally, the results of hyperparameter sensitivity analysis and the performance of our method in imbalanced data streams are discussed.

A. BELS VS. BLS - ABLATION STUDY

To study the effect of our approach on the learning process and the handling of concept drift, we conduct a step-by-step study to observe average prequential accuracy and runtime. We analyze the effects of the different features of BELS on accuracy by performing experiments on three different BELS variants and compare them with the original BLS model in terms of runtime effectiveness and efficiency.

- **BLS**: The original Broad Learning System method.
- **BELS-FPs**: BELS method with the enhancements mentioned in Section III-C1 and III-C2, which includes feature mapping layer and pseudoinverse update.
- **BELS-Ens**: BELS as an ensemble method. This part does not contain the pool of removed output layer instances.
- **BELS**: Complete BELS version with all of its features including BELS-Ens and the pool.

We report the results of four data streams of a good variety on four datasets. The chosen datasets contain all types of concept drift (incremental, gradual, abrupt, and recurring). To have a fair comparison, we use the same hyperparameters

TABLE 6. Average prequential accuracy results (in %). For each row, the highest value is marked with bold text. Avg. % Imp. by BELS wrt a baseline is the mean value of % improvements obtained for individual datasets. Average Rank is the mean rank of each method for individual datasets.

Datasets	BELS	ARF	DWM	GOOWE	HAT	KUE	LevBag	OzaADWIN	ROSE	SAM-kNN	SRP
Airlines	61.85	67.52	60.37	63.59	62.74	66.68	66.49	64.06	66.40	59.58	**67.96**
CoverTypes	87.29	89.38	77.91	82.77	77.32	87.32	86.12	81.01	60.97	75.85	**90.32**
Electricity	87.21	76.28	70.37	77.00	72.94	76.89	76.22	75.89	**90.39**	63.50	77.08
Email	**68.36**	62.44	50.71	57.81	52.34	52.70	48.37	52.38	62.77	53.29	60.38
Phishing	93.74	**93.99**	90.15	90.42	88.38	92.06	92.21	90.19	92.70	93.38	93.83
Poker	**91.71**	76.45	54.18	61.26	58.74	90.67	73.96	58.85	50.41	65.11	81.70
Spam	**95.30**	79.38	85.92	88.39	81.60	80.52	82.28	82.80	90.44	75.94	80.20
Usenet	**70.29**	62.55	46.60	62.96	54.64	66.30	56.33	55.74	66.92	55.85	61.34
Weather	77.36	**79.05**	68.38	71.83	72.73	74.95	77.19	73.89	77.53	74.90	78.88
Agrawal-Gradual-456	73.95	**89.03**	70.85	77.49	83.02	82.88	88.37	81.85	87.55	70.46	86.94
hyperplane-25	90.24	88.77	89.27	88.54	88.07	88.30	89.67	89.28	**91.23**	83.85	88.08
hyperplane-50	90.28	88.84	89.33	88.35	88.13	88.16	89.66	89.29	**91.14**	83.76	88.11
Interchanging RBF	98.20	6.22	71.83	60.30	45.91	69.30	71.16	69.30	**100.00**	64.60	71.74
LED-drift	**100.00**	99.58	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	29.47	95.46	99.39
MG2C2D	92.94	49.60	91.94	92.48	92.65	**92.95**	92.61	92.13	92.54	92.55	92.56
Mixed-Abrupt-Rec-1212	92.33	95.26	89.34	87.11	90.94	90.88	93.25	89.65	**96.97**	94.17	84.19
Moving Squares	**89.99**	25.00	32.17	31.78	33.09	30.27	34.01	33.90	64.60	33.92	33.87
RBF-Fast	30.26	**37.63**	27.67	33.57	34.50	31.44	36.87	34.28	34.37	30.55	37.55
RBF-Slow	**46.94**	32.75	32.86	29.61	32.19	31.76	30.33	30.81	40.94	28.95	32.71
Rotating Hyperplane	**90.85**	84.80	89.46	87.90	86.15	85.44	86.70	87.71	86.66	82.77	85.38
SEA-Abrupt-01210*	88.65	88.52	88.33	87.71	87.38	88.21	88.66	87.60	**89.48**	86.46	87.60
SEA-Abrupt-13123*	88.68	89.64	88.51	87.52	88.21	87.77	89.30	87.73	**89.97**	86.12	88.47
SEA-Incremental-01210*	78.53	78.52	79.26	78.35	77.98	78.28	78.90	87.36	**79.07**	74.04	77.84
SEA-Incremental-13123*	78.53	79.31	79.08	78.28	78.65	78.17	79.43	78.74	**79.81**	74.04	77.96
Sine-Gradual-234	78.69	94.78	83.78	90.73	93.45	94.49	95.93	92.30	**97.95**	91.38	96.29
Sine-Gradual-Rec-121	95.64	96.35	91.67	94.15	96.20	95.35	96.40	96.14	**99.25**	91.94	96.47
Stagger-Abrupt-Rec-1212	**99.75**	98.27	95.90	98.26	98.19	98.23	98.27	98.27	87.58	95.58	98.24
Stagger-Abrupt-Rec-1232	**99.53**	98.41	98.37	94.71	98.37	98.38	98.41	98.37	87.62	97.83	98.52
Waveform	**84.96**	84.77	80.45	82.87	81.85	83.67	84.81	83.78	84.35	81.61	84.77
Waveform-Noisy *	**85.77**	84.70	80.61	82.93	81.43	83.26	84.46	83.53	84.21	81.35	85.17
Avg. Accuracy	**83.59**	75.93	75.18	76.96	75.93	78.84	78.88	77.56	78.44	74.62	79.45
Avg. % Imp. by BELS	-	62.94	17.23	14.09	16.48	11.48	10.95	12.89	13.31	17.19	9.29
Avg. Rank	**3.20**	4.71	7.63	7.20	7.70	6.32	4.38	6.40	4.18	9.07	5.20

* Five noisy datasets are marked with (*).

for all three variants of BELS and the BLS. The results are shown in Table 5.

BLS is not designed for a data stream environment. Based on this fact, we know that BLS is faster than our algorithm but it has lower accuracy in a stream environment. As we see in Table 5, by utilizing each feature of the BELS, the accuracy improves, which yields the best performance in the complete version of BELS with an average accuracy improvement of 41.44% on four datasets used in this experiment. Average accuracy improvement is the mean of improvement over all datasets, and is calculated as (Accuracy Increase $\times$ 100/Original Accuracy).

B. EFFECTIVENESS AND EFFICIENCY ANALYSIS

The prequential evaluation results are shown in Table 6. The results demonstrate that our proposed method outperforms the baseline methods on 12 out of 30 datasets, and has a marginal difference with the best-performing baseline in the remaining datasets. Our observations on datasets of varying drift types with different numbers of class labels and features demonstrate the adaptability of our model under a variety of classification conditions. On average BELS improves the accuracy by 18.59% compared to the baselines used in the experiments across 30 datasets. Experiments on noisy datasets also show that our model is robust to noise.

We can see the runtime of the models in Table 7. Runtime is considered as the processing time for test and training of 1,000 data instances in the test-then-train method. The results show that our model has an average processing time of 625.63 centiseconds for processing 1,000 data items. It is worth mentioning that, being implemented in Python, the BELS model functions slower compared to Java-based models. Despite this inherent difference, BELS still achieves a competitive average runtime. As mentioned earlier in Section III-C4, unlike the baselines, our runtime results are independent of the feature set size. For instance, the runtime results for Email and Spam datasets with 913 and 499 features, respectively, highlight this issue (See Table 7.). Compared to other datasets, the baseline methods exhibit a longer runtime on these two particular datasets.

The plots in Figure 2 show that BELS has robust concept drift-resistant performance: it maintains better performance when concept drift occurs (indicated by the fluctuated accuracies as the data stream progresses). Furthermore, it provides a higher level of accuracy compared to the three top-ranked baselines (ARF, LevBag, and ROSE) in five out of the six plots.

In the Electricity dataset, we observe abrupt fluctuations in the prequential accuracy. This suggests that there might be an abrupt drift in this dataset. In the Poker dataset, there are mild abrupt changes in the accuracies; and in the case of BELS,

TABLE 7. Average runtime for processing 1,000 data instances (in centiseconds). For each row, the lowest value is marked with bold text. Average Rank is the mean rank of each method for individual datasets.

Datasets	BELS	ARF	DWM	GOOWE	HAT	KUE	LevBag	OzaADWIN	ROSE	SAM-kNN	SRP
Airlines	159.76	1,431.73	6.41	858.24	**1.28**	52.20	108.36	19.96	88.93	13.94	5,122.81
CoverTypes	716.79	2,630.10	20.29	5,133.06	**4.74**	429.92	165.85	42.12	230.57	374.94	7,811.76
Electricity	597.06	641.46	4.166	449.81	**1.21**	26.70	62.38	16.92	125.35	55.25	1,974.97
Email	120.20	28,165.33	301.18	2,043.33	**88.63**	462.67	1,650.03	5,678.30	22,271.33	1,891.06	43,988.11
Phishing	803.80	1,695.79	19.75	1,107.82	**3.31**	68.29	100.22	29.88	336.68	184.43	2,263.77
Poker	1,554.27	872.50	5.46	1,602.62	**1.09**	122.20	44.63	19.25	99.55	68.64	4,070.42
Spam	7,004.02	16,220.18	160.62	6,624.77	**45.47**	516.01	821.18	232.81	6,160.95	1,075.33	21,244.91
Usenet	25.80	2,855.33	28.87	210.00	**5.73**	89.33	188.37	211.05	581.07	115.66	2,374.04
Weather	65.17	719.09	6.85	319.30	**1.12**	13.17	31.78	17.60	140.77	37.47	880.86
Agrawal-Gradual-456	204.27	1,142.46	20.26	552.18	**1.47**	19.95	60.82	25.45	29.89	60.42	3,501.40
hyperplane-25	613.68	1,278.78	21.02	589.75	**2.22**	33.18	39.89	23.06	103.55	233.49	7,802.28
hyperplane-50	623.67	1,246.49	21.09	593.33	**1.79**	20.42	45.47	22.84	115.92	123.36	7,774.75
Interchanging RBF	321.03	99.19	1.56	8,330.16	**0.68**	12.86	24.81	18.19	85.90	75.50	331.87
LED-drift	269.88	171.45	0.44	413.81	**0.37**	67.98	26.94	16.98	69.48	200.25	512.41
MG2C2D	301.65	77.14	3.50	224.54	**0.75**	5.50	65.36	17.10	57.83	49.97	1,682.14
Mixed-Abrupt-Rec-1212	147.73	151.91	16.57	294.69	**0.80**	7.67	25.67	18.99	49.21	177.44	1,683.80
Moving Squares	509.15	77.80	1.08	413.81	**0.92**	4.54	44.87	17.05	50.82	35.98	244.41
RBF-Fast	114.69	711.68	26.19	804.75	**1.88**	33.81	41.22	26.01	95.98	25.80	903.01
RBF-Slow	166.04	330.11	21.95	1,042.27	**1.72**	54.86	32.32	22.72	148.92	36.78	2,503.18
Rotating Hyperplane	54.66	8,526.04	7.49	221.87	**1.78**	23.27	131.07	21.49	68.66	169.49	8,426.76
SEA-Abrupt-01210	119.91	2,107.96	1.93	129.80	**0.94**	10.64	29.99	17.51	54.07	63.85	1239.03
SEA-Abrupt-13123	126.51	2,645.36	2.47	133.92	**0.98**	13.64	34.77	17.29	59.84	51.51	1602.55
SEA-Incremental-01210	112.99	3,065.71	3.61	133.90	**0.97**	11.39	26.90	16.82	64.67	33.20	2,034.56
SEA-Incremental-13123	115.87	3,379.04	3.00	132.98	**0.98**	12.18	29.46	17.72	57.28	35.18	1426.31
Sine-Gradual-234	1,170.20	1,088.75	18.56	321.10	**1.14**	8.73	31.99	18.56	16.88	92.75	1,142.60
Sine-Gradual-Rec-121	1,187.96	689.53	20.55	317.43	**1.15**	8.61	32.40	20.59	17.91	75.59	1,032.37
Stagger-Abrupt-Rec-1212	546.75	68.70	15.47	271.08	**0.58**	4.01	15.64	18.08	15.29	145.75	242.78
Stagger-Abrupt-Rec-1232	535.56	60.88	14.33	266.27	**0.53**	3.80	16.63	18.41	34.18	103.95	230.22
Waveform	235.46	5,271.77	30.68	865.36	**6.09**	46.34	171.63	44.85	134.80	164.13	5,271.77
Waveform-Noisy	244.34	5,526.82	62.27	3,209.17	**4.96**	80.37	158.03	47.35	136.69	144.08	9,419.07
Avg. Runtime	625.63	3,098.30	28.92	1,253.70	**6.18**	75.47	141.96	224.50	1,050.10	197.17	4,957.96
Avg. Rank	8.13	9.45	2.47	9.07	**1.00**	3.70	5.47	3.90	6.23	6.07	10.52

we see soft transitions in the plot that demonstrate the ability of BELS to handle concept drifts in such cases. The remaining four datasets (Usenet, RBF-Slow, Rotating Hyperplane, Stagger-Abrupt-Rec-1212) include all four types of drifts. In these datasets, BELS demonstrates its effectiveness in handling drift. In three of these datasets, Rotating Hyperplane, RBF-Slow, and Stagger-Abrupt-Rec-1212, BELS is the top-performing model during the whole process. In Usenet, which has multiple drifts within a short span of 1,500 data items, BELS demonstrates its overall effectiveness and recovers swiftly.

To gain a better perspective of the average rankings of the methods used in our experimental evaluation, we demonstrate them in Figure 3. Each bar shows the average ranking of the models in terms of accuracy and runtime.

C. STATISTICAL SIGNIFICANCE ANALYSIS

In this part, we present a statistical test on both accuracy and runtime. The analysis is conducted for 11 models and 30 datasets. By using the *Friedman Test* we first reject the null hypothesis that there is no statistically significant difference between the mean values of the populations. Then we use *post-hoc Bonferroni-Dunn* test to check if there is a significant difference between the results of our proposed model and the baselines [60].

For this test, we first rank the models based on their performance. Then based on the post-hoc Bonferroni-Dunn test, we calculate the Critical Difference as $CD = 2.40$. In our experiment $\alpha = 0.05$. Figure 4 shows the critical distance diagram for average prequential accuracy and runtime.[8]

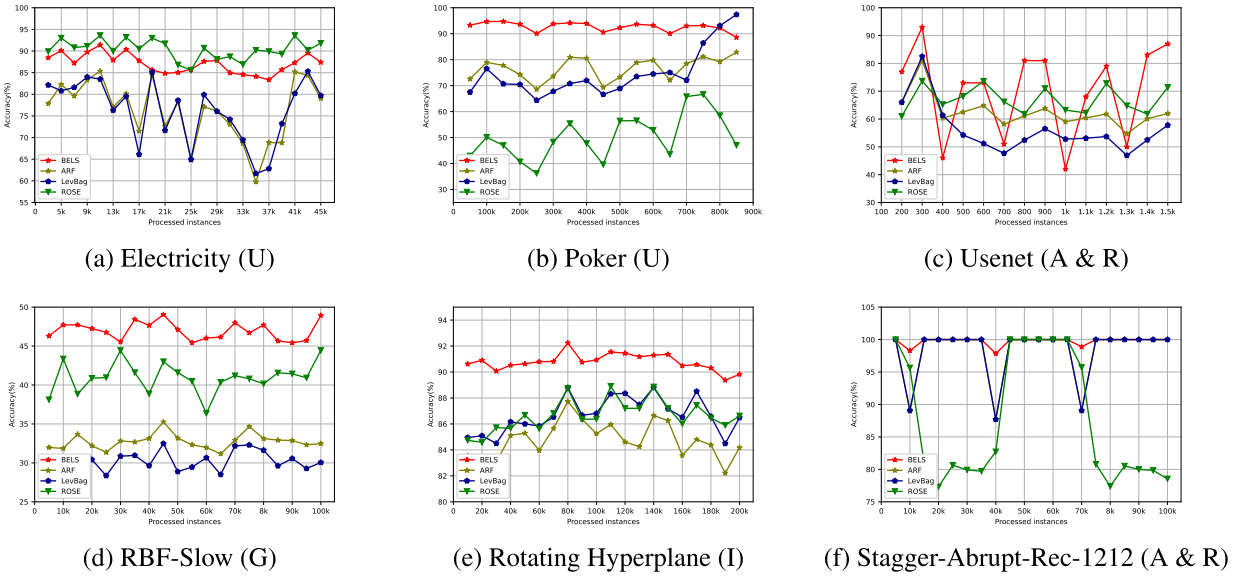
The CD diagram reveals that BELS ranks higher compared to the 10 state-of-the-art models in terms of average prequential accuracy. BELS statistically significantly outperforms six of the models (KUE, OzaADWIN, GOOWE, DWM, HAT, and SAM-kNN) and is in the same group as ROSE, LevBag, ARF, and SRP.

Regarding runtime, BELS does not exhibit a significant difference from SAM-kNN, ROSE, GOOWE, ARF, and SRP, which are all defined for data streams. However, HAT, DWM, KUE, OzaADWIN, and LevBag statistically significantly outperform the proposed model in terms of runtime.

D. HYPERPARAMETER SENSITIVITY ANALYSIS

In this part, we study the effects of the main hyperparameters on the overall performance of BELS. Four datasets are used for this purpose. With this intent, S_c (chunk size), M_o (maximum number of output layer instances), and M_p (maximum number of output layer instances in P) are studied because of their importance in the learning process and handling concept drift. Table 8 shows the results for hyperparameter sensitivity analysis. Initially, we set a default setting for each dataset based on the first 1,000 data instances (check Section IV-B for details). We only modify the target hyperparameter, and the remaining ones stay the same in all experiments. Based on the results for chunk size, we observe that the best accuracy

[8]https://orangedatamining.com



(a) Electricity (U) (b) Poker (U) (c) Usenet (A & R)

(d) RBF-Slow (G) (e) Rotating Hyperplane (I) (f) Stagger-Abrupt-Rec-1212 (A & R)

FIGURE 2. **Prequential temporal accuracy results of BELS and three top-ranked baselines (ARF, LevBag, and ROSE) for real (first row), and synthetic datasets (second row). The initial capital of drift types is given within parentheses (U: Unknown, A: Abrupt, G: Gradual, I: Incremental, R: Recurring).**

TABLE 8. **Hyperparameter analysis results: Observations are reported in terms of average prequential accuracy and runtime (in centiseconds for processing 1,000 data items). For the Usenet dataset, we do not report the results for chunk sizes 250 and 500, since the dataset has only 1,500 data items. The best results for each hyperparameter for each dataset are in bold.**

		S_c: chunk size									M_o: maximum number of output layer instances					M_p: maximum number of output layer instances in P				
Datasets (DT)		1	2	5	10	20	50	100	250	500	5	25	50	75	150	50	100	200	300	400
Electricity (U)	Acc.	**88.18**	87.21	83.22	80.16	75.97	71.90	69.30	72.13	71.22	85.27	86.67	86.92	**87.21**	87.20	86.85	87.20	**87.21**	**87.21**	**87.21**
	Runtime	1,195.20	597.06	326.18	166.44	95.71	46.72	28.44	26.35	**11.78**	**199.95**	322.89	460.49	597.06	482.74	**448.57**	584.87	585.32	597.06	522.78
Usenet (A & R)	Acc.	68.79	**76.50**	72.07	71.14	70.50	70.29	63.54	-	-	65.35	**70.29**	**70.29**	**70.29**	**70.29**	**70.29**	**70.29**	**70.29**	**70.29**	**70.29**
	Ruuntime	1,304.66	536.66	293.33	112.00	48.66	21.33	**14.53**	-	-	**19.86**	21.52	20.36	**20.36**	**20.36**	**20.36**	**20.36**	**20.36**	**20.36**	**20.36**
MG2C2D (I & G)	Acc.	81.19	91.42	92.45	92.71	92.84	**92.94**	92.67	92.50	92.76	92.35	92.75	92.82	**92.94**	**92.94**	92.87	**92.99**	92.94	92.94	92.94
	Runtime	9,470.50	5,967.25	2,850.06	1,348.71	637.62	301.65	175.88	89.37	**55.86**	**50.24**	176.72	354.56	301.65	553.14	325.52	481.91	384.51	**301.65**	587.40
Rotating Hyperplane (I)	Acc.	81.99	86.19	89.82	90.81	**91.00**	90.85	90.34	90.34	88.49	88.36	90.88	**90.94**	90.85	90.36	**91.05**	90.85	90.85	90.85	90.85
	Runtime	1,116.64	567.79	366.41	241.53	122.31	54.66	36.44	34.82	**24.91**	**12.06**	23.83	40.12	54.66	104.36	**39.73**	54.47	54.58	54.66	54.20

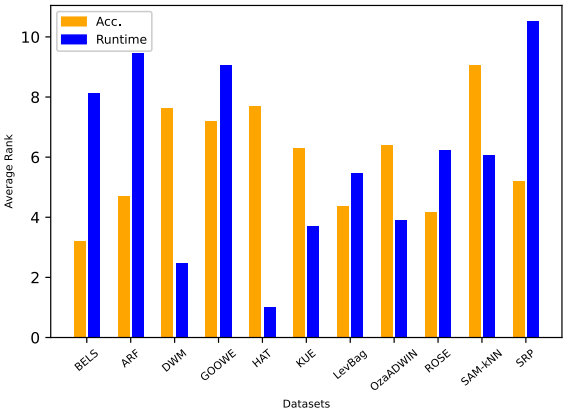


FIGURE 3. **Average rank comparison (lower is better) for prequential accuracy and runtime based on Tables 6 and 7.**

results for each dataset are for chunk sizes less than 100. BELS is designed for learning with small chunk sizes at every step, since it reacts to the changes in accuracy, and frequently replaces the ensemble components with the new ones or the ones in the pool. Based on the observations in this section, we suggest using chunk sizes smaller than 100. As mentioned earlier in Section IV-B, we use the first 1,000 instances to automatically determine the chunk size which is chosen from a set of {2, 5, 10, 20, 50}. The reason for removing a chunk size equal to one from this set is its inefficiency. As we can see in Table 8, the runtime of the model is substantially dependent on the chunk size.

Based on the results in Section V-A, we observe that having an ensemble of output layer instances improves the results, meaning that the ensemble size should be greater than one. In our experiments, the default number of output layer instances (M_o) is set to 75. The reason is that adding more instances leads to longer runtime, and fewer output layer instances may result in poor performance in terms of accuracy.

Next, we analyze the effect of pool size (M_p). After increasing M_p to 200 and more, the accuracy remains the same. The reason is that based on our ensemble method, after adding an output layer instance back to the learning process, we remove

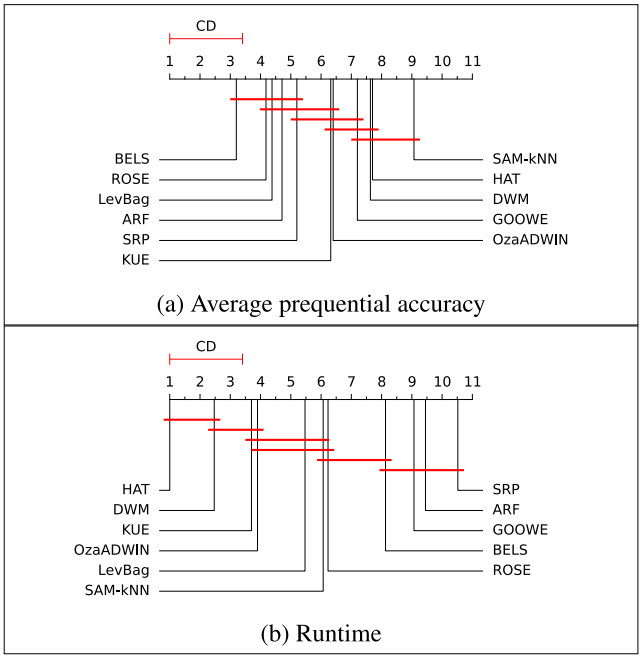


FIGURE 4. Critical distance diagrams for the (a) Average prequential accuracy and (b) runtime using the data on Tables 6 and 7. (CD = 2.40).

it from P (see Algorithm 3:13). For this reason, usually M_p does not exceed the defined limit for P. We choose an upper bound of 300 for M_p in all of the experiments.

Our suggestions for default hyperparameters are presented in Section IV-B (Table 3), and they are repeated here for ease of reference: The chunk size is selected from a set of {2, 5, 10, 20, 50}, M_o= 75, and M_P= 300.

E. HANDLING IMBALANCED DRIFTING DATA STREAMS

To assess the performance of our method in imbalanced streams, we compare it with the top five approaches in the accuracy ranking in Table 6: ROSE, LevBag, ARF, SRP, and KUE. We report the Kappa metric results for comparison [61], [62], [63]. The Kappa measure is applicable for both multi-class and binary classification problems and has proven to be a dependable measure for evaluating the performance of data stream classification models, particularly when dealing with imbalanced datasets [61].

In this experiment, we use five real (R) and five synthetic (S) datasets. The real datasets are chosen from the set of imbalanced datasets included in a recent survey [61]. The imbalance ratio in synthetic datasets is set to 0.9:0.1 in binary and 0.6:0.2:0.1:0.1 in multi-class datasets with four labels. The synthetic datasets contain concept drift.

The average Kappa scores are reported in Table 9. The Kappa score is between (−1, 1). Kappa scores closer to one suggest higher levels of agreement between predicted and true labels, signifying better handling of imbalanced datasets. The results show that our model outperforms the baselines in terms of average Kappa. In terms of average rank, our model achieves the same rank as ROSE on the examined datasets.

TABLE 9. Imbalanced data streams analysis results: Kappa scores (in %) for the BELS and top five baselines in the accuracy table (Table 6).

Datasets (Typpe)	BELS	ARF	KUE	LevBag	ROSE	SRP
CoverTypes (R)	73.41	79.02	74.75	73.17	41.27	**80.53**
Electricity (R)	73.12	50.50	50.73	50.75	**79.85**	51.71
Poker (R)	**85.17**	51.62	79.58	43.94	0.03	60.13
Spam (R)	**85.09**	54.21	58.13	55.66	63.60	56.50
Weather (R)	42.29	46.41	37.18	42.88	**48.92**	45.40
Imb-RBF-Fast (S)	0.06	4.43	3.33	4.26	**10.93**	4.03
Imb-RBF-Slow (S)	5.72	-0.01	0.15	0.06	**17.43**	-0.24
Imb-Mixed-Gradual-Rec-1212 (S)	66.63	81.02	88.94	93.51	**95.92**	9.04
Imb-Staggar-Abrupt-Rec-1212 (S)	91.89	91.44	92.49	**93.12**	80.38	91.12
Imb-Staggar-Abrupt-Rec-1231 (S)	**96.88**	96.60	95.58	96.60	91.44	74.67
Avg. Kappa	**62.03**	55.52	58.09	55.40	52.98	47.29
Avg. Rank	**3.00**	3.75	3.60	3.55	**3.00**	4.10

Applying the Friedman test to the models in this section [60], indicates that the null hypothesis could not be rejected, and there is no significant difference between the classifiers.

VI. CONCLUSION AND FUTURE WORK

In this work, we present BELS, a novel ensemble model for data stream classification in non-stationary environments. We describe real-world and unique challenges that data stream causes and focus on handling concept drift using a novel approach. BELS tracks changes in the accuracy of the ensemble components and frequently reacts to these changes by exchanging classifier components between the pool of removed ensemble components and the active ensemble members. The results show that on average, BELS improves the accuracy, compared to the state-of-the-art models designed specifically for data streams, by 18.59%, and in terms of processing time, it is a suitable choice for evolving environments.

Despite the competitive results in the imbalanced learning scenario, our future plans involve enhancing our model to handle this issue more effectively. Another problem in data stream environments is the lack of available class labels in real-world scenarios. As a part of our future work, we also plan to improve BELS for semi-supervised data stream classification.

REFERENCES

[1] D. Laney, "3D data management: Controlling data volume, velocity, and variety," *META Group Res. Note*, vol. 6, no. 70, p. 1, 2001.
[2] G. Widmer and M. Kubat, "Learning in the presence of concept drift and hidden contexts," *Mach. Learn.*, vol. 23, no. 1, pp. 69–101, Apr. 1996.
[3] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, "Learning under concept drift: A review," *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 12, pp. 2346–2363, Dec. 2019.
[4] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Comput. Surv. (CSUR)*, vol. 46, no. 4, pp. 1–37, Apr. 2014.
[5] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with drift detection," in *Proc. Brazilian Symp. Artif. Intell.* Berlin, Germany: Springer, 2004, pp. 286–295.
[6] S. Ramírez-Gallego, B. Krawczyk, S. García, M. Woźniak, and F. Herrera, "A survey on data preprocessing for data stream mining: Current status and future directions," *Neurocomputing*, vol. 239, pp. 39–57, May 2017.
[7] A. S. Iwashita and J. P. Papa, "An overview on concept drift learning," *IEEE Access*, vol. 7, pp. 1532–1547, 2019.
[8] G. Ditzler, M. Roveri, C. Alippi, and R. Polikar, "Learning in nonstationary environments: A survey," *IEEE Comput. Intell. Mag.*, vol. 10, no. 4, pp. 12–25, Nov. 2015.

[9] Z. Li, W. Huang, Y. Xiong, S. Ren, and T. Zhu, "Incremental learning imbalanced data streams with concept drift: The dynamic updated ensemble algorithm," *Knowl.-Based Syst.*, vol. 195, May 2020, Art. no. 105694.

[10] C. L. P. Chen and Z. Liu, "Broad learning system: An effective and efficient incremental learning system without the need for deep architecture," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 1, pp. 10–24, Jan. 2018.

[11] L. Yuan, H. Li, B. Xia, C. Gao, M. Liu, W. Yuan, and X. You, "Recent advances in concept drift adaptation methods for deep learning," in *Proc. 31st Int. Joint Conf. Artif. Intell. (IJCAI)*, L. D. Raedt, Ed., Jul. 2022, pp. 5654–5661, doi: 10.24963/ijcai.2022/788.

[12] M. Y. Vardi, "Efficiency vs. resilience: What COVID-19 teaches computing," *Commun. ACM*, vol. 63, no. 5, p. 9, Apr. 2020.

[13] A. Cano and B. Krawczyk, "ROSE: Robust online self-adjusting ensemble for continual learning on imbalanced drifting data streams," *Mach. Learn.*, vol. 111, no. 7, pp. 2561–2599, Jul. 2022.

[14] G. Ditzler and R. Polikar, "Incremental learning of concept drift from streaming imbalanced data," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 10, pp. 2283–2301, Oct. 2013.

[15] H. Bonab and F. Can, "Less is more: A comprehensive framework for the number of components of ensemble classifiers," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 9, pp. 2735–2745, Sep. 2019.

[16] M. Baena-Garcıa, J. D. Campo-Ávila, R. Fidalgo, A. Bifet, R. Gavalda, and R. Morales-Bueno, "Early drift detection method," in *Proc. 4th Int. Workshop Knowl. Discovery Data Streams*, vol. 6, 2006, pp. 77–86.

[17] A. Bifet and R. Gavaldà, "Learning from time-changing data with adaptive windowing," in *Proc. SIAM Int. Conf. Data Mining*, Apr. 2007, pp. 443–448.

[18] Ö. Gözüaçık and F. Can, "Concept learning using one-class classifiers for implicit drift detection in evolving data streams," *Artif. Intell. Rev.*, vol. 54, no. 5, pp. 3725–3747, Jun. 2021.

[19] A. A. Qahtan, B. Alharbi, S. Wang, and X. Zhang, "A PCA-based change detection framework for multidimensional data streams: Change detection in multidimensional data streams," in *Proc. 21st ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2015, pp. 935–944.

[20] F. Gu, G. Zhang, J. Lu, and C.-T. Lin, "Concept drift detection based on equal density estimation," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2016, pp. 24–30.

[21] N. Lu, J. Lu, G. Zhang, and R. L. de Mantaras, "A concept drift-tolerant case-base editing technique," *Artif. Intell.*, vol. 230, pp. 108–133, Jan. 2016.

[22] G. Hulten, L. Spencer, and P. Domingos, "Mining time-changing data streams," in *Proc. 7th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2001, pp. 97–106.

[23] Ö. Gözüaçık, A. Büyükçakır, H. Bonab, and F. Can, "Unsupervised concept drift detection with a discriminative classifier," in *Proc. 28th ACM Int. Conf. Inf. Knowl. Manage.*, Nov. 2019, pp. 2365–2368.

[24] E. B. Gulcan and F. Can, "Unsupervised concept drift detection for multi-label data streams," *Artif. Intell. Rev.*, vol. 56, no. 3, pp. 2401–2434, Mar. 2023.

[25] X. Dong, Z. Yu, W. Cao, Y. Shi, and Q. Ma, "A survey on ensemble learning," *Frontiers Comput. Sci.*, vol. 14, no. 2, pp. 241–258, 2020.

[26] S. Akbar, A. Ahmad, M. Hayat, A. U. Rehman, S. Khan, and F. Ali, "iAtbP-Hyb-EnC: Prediction of antitubercular peptides via heterogeneous feature representation and genetic algorithm based ensemble learning model," *Comput. Biol. Med.*, vol. 137, Oct. 2021, Art. no. 104778.

[27] A. Ahmad, S. Akbar, M. Tahir, M. Hayat, and F. Ali, "iAFPs-EnC-GA: Identifying antifungal peptides using sequential and evolutionary descriptors based multi-information fusion and ensemble learning approach," *Chemometric Intell. Lab. Syst.*, vol. 222, Mar. 2022, Art. no. 104516.

[28] S. Akbar, M. Hayat, M. Iqbal, and M. A. Jan, "iACP-GAEnsC: Evolutionary genetic algorithm based ensemble classification of anticancer peptides by utilizing hybrid feature space," *Artif. Intell. Med.*, vol. 79, pp. 62–70, Jun. 2017.

[29] H. M. Gomes, A. Bifet, J. Read, J. P. Barddal, F. Enembreck, B. Pfharinger, G. Holmes, and T. Abdessalem, "Adaptive random forests for evolving data stream classification," *Mach. Learn.*, vol. 106, nos. 9–10, pp. 1469–1495, Oct. 2017.

[30] A. Bifet, G. Holmes, and B. Pfahringer, "Leveraging bagging for evolving data streams," in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discovery Databases*. Berlin, Germany: Springer, 2010, pp. 135–150.

[31] K. Nishida, K. Yamauchi, and T. Omori, "Ace: Adaptive classifiers-ensemble system for concept-drifting environments," in *Proc. Int. Workshop Multiple Classifier Syst.* Berlin, Germany: Springer, 2005, pp. 176–185.

[32] M. M. Idrees, L. L. Minku, F. Stahl, and A. Badii, "A heterogeneous online learning ensemble for non-stationary environments," *Knowl.-Based Syst.*, vol. 188, Jan. 2020, Art. no. 104983.

[33] W. Liu, H. Zhang, Z. Ding, Q. Liu, and C. Zhu, "A comprehensive active learning method for multiclass imbalanced data streams with concept drift," *Knowl.-Based Syst.*, vol. 215, Mar. 2021, Art. no. 106778.

[34] H. M. Gomes, J. Read, and A. Bifet, "Streaming random patches for evolving data stream classification," in *Proc. IEEE Int. Conf. Data Mining (ICDM)*, Nov. 2019, pp. 240–249.

[35] B. Krawczyk, L. L. Minku, J. Gama, J. Stefanowski, and M. Woźniak, "Ensemble learning for data stream analysis: A survey," *Inf. Fusion*, vol. 37, pp. 132–156, Sep. 2017.

[36] J. Z. Kolter and M. A. Maloof, "Using additive expert ensembles to cope with concept drift," in *Proc. 22nd Int. Conf. Mach. Learn.*, 2005, pp. 449–456.

[37] J. Z. Kolter and M. A. Maloof, "Dynamic weighted majority: An ensemble method for drifting concepts," *J. Mach. Learn. Res.*, vol. 8, pp. 2755–2790, Dec. 2007.

[38] H. R. Bonab and F. Can, "GOOWE: Geometrically optimum and online-weighted ensemble classifier for evolving data streams," *ACM Trans. Knowl. Discovery from Data (TKDD)*, vol. 12, no. 2, pp. 1–33, Apr. 2018.

[39] R. Elwell and R. Polikar, "Incremental learning of concept drift in non-stationary environments," *IEEE Trans. Neural Netw.*, vol. 22, no. 10, pp. 1517–1531, Oct. 2011.

[40] H. Zhang, W. Liu, S. Wang, J. Shan, and Q. Liu, "Resample-based ensemble framework for drifting imbalanced data streams," *IEEE Access*, vol. 7, pp. 65103–65115, 2019.

[41] A. Cano and B. Krawczyk, "Kappa updated ensemble for drifting data stream mining," *Mach. Learn.*, vol. 109, no. 1, pp. 175–218, Jan. 2020.

[42] H. Wang, W. Fan, P. S. Yu, and J. Han, "Mining concept-drifting data streams using ensemble classifiers," in *Proc. 9th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2003, pp. 226–235.

[43] D. Brzeziński and J. Stefanowski, "Accuracy updated ensemble for data streams with concept drift," in *Proc. Int. Conf. Hybrid Artif. Intell. Syst.* Berlin, Germany: Springer, 2011, pp. 155–163.

[44] L. L. Minku and X. Yao, "DDD: A new ensemble approach for dealing with concept drift," *IEEE Trans. Knowl. Data Eng.*, vol. 24, no. 4, pp. 619–633, Apr. 2012.

[45] J. Gama, R. Sebastiao, and P. P. Rodrigues, "Issues in evaluation of stream learning algorithms," in *Proc. 15th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2009, pp. 329–338.

[46] C. L. P. Chen, Z. Liu, and S. Feng, "Universal approximation capability of broad learning system and its structural variations," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 4, pp. 1191–1204, Apr. 2019.

[47] S. Bakhshi, "BELS: A broad ensemble learning system for data stream classification," M.S. thesis, Dept. Comput. Eng., Bilkent Universitesi, Ankara, Turkey, 2021. Accessed: Jul. 27, 2023. [Online]. Available: http://hdl.handle.net/11693/76862

[48] M. Harries, *Splice-2 Comparative Evaluation: Electricity Pricing* (PANDORA Electronic Collection). Univ. of New South Wales, School of Computer Science and Engineering, 1999. [Online]. Available: https://books.google.com.tr/books?id=1Zr1vQAACAAJ

[49] I. Katakis, G. Tsoumakas, and I. Vlahavas, "Tracking recurring contexts using ensemble classifiers: An application to email filtering," *Knowl. Inf. Syst.*, vol. 22, no. 3, pp. 371–391, Mar. 2010.

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

[51] D. Dua and C. Graff. (2017). *UCI Machine Learning Repository*. [Online]. Available: http://archive.ics.uci.edu/ml

[52] I. Katakis, G. Tsoumakas, and I. P. Vlahavas, "An ensemble of classifiers for coping with recurring contexts in data streams," in *Proc. 18th Eur. Conf. Artif. Intell. (ECAI)*, 2008, pp. 763–764.

[53] V. Losing, B. Hammer, and H. Wersing, "KNN classifier with self adjusting memory for heterogeneous concept drift," in *Proc. IEEE 16th Int. Conf. Data Mining (ICDM)*, Dec. 2016, pp. 291–300.

[54] K. B. Dyer, R. Capo, and R. Polikar, "COMPOSE: A semisupervised learning framework for initially labeled nonstationary streaming data," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 1, pp. 12–26, Jan. 2014.

[55] A. Bifet and R. Gavalda, "Adaptive learning from evolving data streams," in *Proc. Int. Symp. Intell. Data Anal.* Berlin, Germany: Springer, 2009, pp. 249–260.
[56] N. C. Oza and S. J. Russell, "Online bagging and boosting," in *Proc. 8th Int. Workshop Artif. Intell. Statist.*, 2001, pp. 229–236.
[57] J. Montiel, J. Read, A. Bifet, and T. Abdessalem, "Scikit-multiflow: A multi-output streaming framework," *J. Mach. Learn. Res.*, vol. 19, no. 72, pp. 1–5, 2018. [Online]. Available: http://jmlr.org/papers/v19/18-251.html
[58] A. Bifet, G. Holmes, B. Pfahringer, P. Kranen, H. Kremer, T. Jansen, and T. Seidl, "MOA: Massive online analysis, a framework for stream classification and clustering," in *Proc. 1st Workshop Appl. Pattern Anal.*, 2010, pp. 44–50.
[59] N. Gunasekara, H. M. Gomes, B. Pfahringer, and A. Bifet, "Online hyperparameter optimization for streaming neural networks," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2022, pp. 1–9.
[60] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, pp. 1–30, Jan. 2006.
[61] G. Aguiar, B. Krawczyk, and A. Cano, "A survey on learning from imbalanced data streams: Taxonomy, challenges, empirical study, and reproducible experimental framework," 2022, *arXiv:2204.03719*.
[62] D. Brzezinski, J. Stefanowski, R. Susmaga, and I. Szczech, "On the dynamics of classification measures for imbalanced and streaming data," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 8, pp. 2868–2878, Aug. 2020.
[63] D. Brzezinski, J. Stefanowski, R. Susmaga, and I. Szczęch, "Visual-based analysis of classification measures and their properties for class imbalanced problems," *Inf. Sci.*, vol. 462, pp. 242–261, Sep. 2018.

SEPEHR BAKHSHI received the B.S. degree in computer engineering from the University of Bonab, Bonab, Iran. He is currently pursuing the M.S. degree with Bilkent University, under the supervision of Prof. Fazl Can. In 2019, he joined the Bilkent Information Retrieval Group (BilIR). His current research interests include data stream mining, natural language processing, and information retrieval.

POUYA GHAHRAMANIAN received the B.S. degree in computer software engineering from the Iran University of Science and Technology (IUST), in 2018. He is currently pursuing the M.S. degree with Bilkent University, under the supervision of Prof. Fazl Can. His current research interests include machine learning, natural language processing, and data stream mining. He is a member of the Bilkent Information Retrieval Group (BilIR).

HAMED BONAB received the B.S. degree in computer engineering from the Iran University of Science and Technology, Tehran, Iran, the M.S. degree in computer engineering from Bilkent University, Ankara, Turkey, and the Ph.D. degree in computer science from the College of Information and Computer Sciences, University of Massachusetts Amherst, Amherst, MA, USA. During the Ph.D. degree, he was with the Center for Intelligent Information Retrieval (CIIR) along with Prof. James Allan. He is currently an Applied Scientist II with Amazon.com Inc., where he is focusing on various product search problems. His current research interests include information retrieval, natural language processing, machine learning, and stream processing.

FAZLI CAN (Member, IEEE) received the B.S. and M.S. degrees in electrical-electronics and computer engineering and the Ph.D. degree in computer engineering from Middle East Technical University, Ankara, Turkey, in 1976, 1979, and 1985, respectively. He conducted the Ph.D. research under the supervision of Prof. E. Ozkarahan; at Arizona State University, Tempe, AZ, USA, and Intel, Chandler, AZ, USA; as a part of the RAP Database Machine Project. He is currently a Faculty Member with Bilkent University, Ankara. Before joining Bilkent, he was a tenured Full Professor with Miami University, Oxford, OH, USA. He co-edited ACM SIGIR Forum from 1995 to 2002 and is the Co-Founder of the Bilkent Information Retrieval Group (BilIR), Bilkent University. His interest in dynamic information processing dates back to his incremental clustering paper in *ACM Transactions on Information Systems*, in 1993, and some other earlier work with Prof. E. Ozkarahan on dynamic cluster maintenance. His current research interests include information retrieval and data mining.