BELS: A BROAD ENSEMBLE LEARNING SYSTEM FOR DATA STREAM CLASSIFICATION

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF
MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

By
Sepehr Bakhshi
December 2021

BELS: A Broad Ensemble Learning System for Data Stream Classification

By Sepehr Bakhshi

December 2021

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Fazlı Can(Advisor)

Uğur Doğrusöz

İsmail Sengör Altıngövde

Approved for the Graduate School of Engineering and Science:

Ezhan Karaşan
Director of the Graduate School

ABSTRACT

BELS: A BROAD ENSEMBLE LEARNING SYSTEM FOR DATA STREAM CLASSIFICATION

Sepehr Bakhshi
M.S. in Computer Engineering
Advisor: Fazlı Can
December 2021

Data stream classification has become a major research topic due to the increase in temporal data. One of the biggest hurdles of data stream classification is the development of algorithms that deal with evolving data, also known as concept drifts. As data changes over time, static prediction models lose their validity. Adapting to concept drifts provides more robust and better performing models. The Broad Learning System (BLS) is an effective broad neural architecture recently developed for incremental learning. BLS cannot provide instant response since it requires huge data chunks and is unable to handle concept drifts. We propose a Broad Ensemble Learning System (BELS) for stream classification with concept drift. BELS uses a novel updating method that greatly improves best-in-class model accuracy. It employs a dynamic output ensemble layer to address the limitations of BLS. We present its mathematical derivation, provide comprehensive experiments with 11 datasets that demonstrate the adaptability of our model, including a comparison of our model with BLS, and provide parameter and robustness analysis on several drifting streams, showing that it statistically significantly outperforms seven state-of-the-art baselines. We show that our proposed method improves on average 44% compared to BLS, and 29% compared to other competitive baselines.

Keywords: Data stream mining, Concept drift, Ensemble learning, Neural networks, Big data.

ÖZET

BELS: VERİ AKIŞI SINIFLANDIRMASI İÇİN GENİŞ BİR TOPLULUK ÖĞRENİM SİSTEMİ

Sepehr Bakhshi
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Danışmanı: Fazlı Can
Aralık 2021

Veri akışı sınıflandırması, zamansal verilerdeki artış nedeniyle önemli bir araştırma konusu haline gelmiştir. Kavram kaymaları, veri akışı sınıflandırmasındaki en önemli sorunlardan biridir ve veri akışının istatistiki özelliklerinin zamanla değişmesi olarak tanımlanabilir. Verilerin zamanla değişmesi, statik tahmin modellerinin işlevsizleşmesine neden olur. Daha sağlam ve başarılı modeller geliştirmek için kavram kaymasına uyum sağlayabilen modeller gereklidir. Geniş Öğrenme Sistemi (BLS), artımlı öğrenme için yakın zamanda geliştirilen etkili bir geniş nöral mimaridir. BLS, büyük veri yığınları gerektirmesi nedeniyle kavram kaymalarını anlık tanımlamada ve uyum sağlamada başarısızdır. Bu çalışmada, kavram kaymalarına uyum sağlayabilen bir akış sınıflandırması için Geniş Topluluk Öğrenme Sistemi'ini (BELS) öneriyoruz. BELS, BLS'nin limitasyonlarını dinamik bir çıktı topluluğu katmanı kullanarak çözer ve "sınıfının-en-iyisi" model doğruluğunu artıran yeni bir güncelleme yöntemi kullanır. BELS'in matematiksel çıkarımının yanında, BLS ile karşılaştırılması da dahil 11 veri kümesi üzerinden gerçekleştirilen kapsamlı deney sonuçlarını sunuyoruz. Bu deneyler ile birlikte BELS'in teknolojiyi temsil eden yedi temel çizgiden önemli ölçüde üstün olduğunu gösteriyoruz. Önerilen yöntemimiz, BLS'ye kıyasla ortalama %44 ve diğer rekabetçi modellere kıyasla ortalama %29 oranında daha başarılıdır.

Anahtar sözcükler: Veri akışı madenciliği, Kavram kayması, Topluluk öğrenimi, Sinir ağları, Büyük veri.

Acknowledgement

First of all, I would like to thank my advisor, Prof. Fazlı Can for trusting in me and giving me an opportunity to do research as part of his research group BILIR. His support, guidance, and inspiring comments kept me motivated, specially during the hard times of pandemic. I would also like to thank the Scientific and Technological Research Council of Turkey(TUBİTAK) 1001 program for supporting our research as part of the 120E103 project. I also want to thank the rest of my thesis committee, Prof. Uğur Doğrusöz and Prof. İ. Sengor Altıngovde, for their feedback and valuable comments.

Besides, I would like to thank Bilkent University Computer Engineering Department for their financial support, creating a friendly environment, and providing facilities for students that help them with their research.

During my studies, I learned a lot from a friend and former member of our research group, Hamed Bonab. I will never forget his unwavering support and guidance. I would also like to thank Alper Can for his valuable comments and contributions on my work.

Making a decision about studying abroad was tricky. It was a turning point in my life and a great experience. All this would not have been possible without the help of my beloved family and girlfriend. Their never-ending support kept me motivated and encouraged me during my studies. I want to thank them for being there for me all the time.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, first, data stream and concept drift notions are discussed. The two major approaches for handling concept drift in a stream environment is explained, and finally, contributions of this work are discussed.

1.1 Data Streams and Concept Drift

Various data stream sources generate an immense amount of data in the blink of an eye. Social media, IoT devices, sensors are all examples of such a source. 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 [37]. Building models that are capable of learning in streaming environments, is a challenging task. The developed method requires an approach specifically designed for this task, as it faces problems different from traditional machine learning. In a data stream environment, data items arrive at a fast rate. As a result, a fast but at the same time, an effective model is needed. Concept drift requires a dynamic change of the learning model. In this study, we consider a multi-class classification environment, where for each incoming data

item one of the possible class labels is assigned. A multi-label environment is also another possibility [11].

Concept drift refers to changes in the probability distribution of data over time [52]. If the change affects decision boundaries it is referred to as *real drift*, and if the distribution is altered without affecting decision boundaries, it is called *virtual drift* [48]. Concept drift is mainly categorized into three types: *Abrupt*, *Gradual* and *Incremental* [41]. 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, this change is completed over a period of time. There is a potential of concept recurrence in all three categories, when an old and previously observed concept may replace the present one.

1.2 Chunk Based vs. Online Methods

In terms of their implementation, models in the literature could be categorized into two main approaches: *chunk-based methods* and *online learning* [16]. In a chunk-based model, usually, a fixed-size chunk of data is collected to update the model as new data items arrive; however, in an online learning model, one data point at a time is used for learning. Each approach has its own pros and cons. Compared to online methods, the model learns faster at the beginning of training in a chunk-based approach since the model is seeded with a big initial chunk of data that makes the update process more effective; however, a concept drift may occur later in the learning process, and if the drift is located within a chunk, it is possible that it will be missed, resulting 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[38]. Another problem with such a model is its runtime. Compared to the chunk-based models, online learning methods are less efficient and require more computations[38].

The issues with using an online or a chunk-based model motivate us to propose a method to alleviate these issues. Our idea is to use a model that is effective and efficient while training with mini chunks as small as 2. Our approach is based on Broad Learning System (BLS) [13]. BLS uses a broad architecture. In this method, instead of the time-consuming process of loss calculation and backpropagation used in deep learning models, a least square problem is solved for training the model, which is much faster [13]. 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 the incremental mode of BLS, the model needs very large chunks of data in each step to reach the desired outcome, which may not be possible when dealing with stream data, (ii) and the model is not able to handle concept drift.

1.3 Work Done and Contributions

In our approach, we first enhance BLS by introducing a new updating system for feature mapping and output layers. This improvement makes the model suitable for training with small chunk sizes.

Next, to handle concept drift, we use a dynamic output layer where the worst output layer instances are continuously replaced with new ones, or where the best ones of the removed output layer instances are brought back to the system. In our method, each output layer instance is added to the pool after it has a major accuracy decline. The output layer instances and the ones in the pool are repeatedly exchanged based on their performance on the most recent data items.

A related point to consider is that our model does not build an ensemble of the BLS model. The model uses a single feature mapping and enhancement layer. Output layer instances are used as ensemble components. In-depth technical aspects of our ensemble model are provided in the following chapters.

Our main contributions are the following. We

- propose and mathematically derive an enhanced version of BLS suitable for using in a stream environment, trained with small chunks of data;
- introduce a dynamic output layer approach to BLS to deal with the concept drift problem;
- conduct experiments on seven state-of-the-art baseline and 11 datasets with various concept drift types. We also perform experiments to demonstrate the model's robustness in presence of various degrees of noise and drift.

Chapter 2

Related Work

Concept drift adaptation: For concept drift adaptation, two approaches are mostly studied in the literature. As mentioned by Gama *et al.* [23] these two *model management* techniques are single classifier [16] and ensemble methods [6].

A single classifier is 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 follow the error rate and trigger an alarm in the case of a sharp decrease in overall accuracy. DDM [21], EDDM [1], ADWIN [2] and OCDD [27] use this strategy. OCDD observes outliers and it is unsupervised. The number of outliers above a threshold indicates a concept drift. Another approach is to keep track of statistical changes in the data distribution. PCA-CD [47], EDE [28] and CM [42] are designed based on this method. After an alarm is triggered, the classifier usually restarts learning from that point on. KNN and Hoeffding Tree [30] are among the most popular classifiers for this purpose. In recent studies unsupervised detection of concept drift is also studied by Gözüaçık *et al.* [26]. In these kinds of methods, it is assumed that the true labels of data items are unavailable.

Single classifier-based models are efficient; however, restarting the model or replacing it with a new one negatively affects the effectiveness of a system. Since

restarting the model loses some useful information learned so far; furthermore, in the case of a recurrent drift, the model cannot use its previous knowledge and should learn each concept from scratch.

In ensemble models, however, a combination of learners is used to make the final decision. Our work is most commonly related to ensemble methods. Some of these models have a strategy to deal with recurring drifts. They preserve previously used classifiers and employ them to replace ineffective active classifiers, helping them in having a much faster adapting strategy in such cases. As this problem is fairly common in real-world applications, ensemble techniques are gaining in popularity [10]. Learn++.NSE [19] is one the most famous models which utilize this strategy.

One of the drawbacks of preserving the old classifiers is its storage burden. Using a limit for the number of preserved classifiers is a simple strategy 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. Since as the number of classifier components grows, the computational complexity of the model increases dramatically, which leads to an inefficient and sometimes broken system that is unable to process new data.

Active vs. Passive Ensemble models: In ensemble methods, the outputs of different learners are combined using a voting strategy. Solutions to deal with varying concepts fall into two categories: *Active* and *Passive* [16].Active models rely on a detection method to trigger an alarm. Then, the ensemble method reacts to this drift by adding new classifiers, updating them, or restarting their learning process. Adaptive Random Forest (ARF) [25], Leverage bagging [4], Adaptive Classifiers-Ensemble (ACE) [45], Heterogeneous Dynamic Weighted Majority (HDWM) [31] and comprehensive active learning method for multi-class imbalanced streaming data with concept drift (CALMID) [39] are among the well-known algorithms in this category.

In passive models, however, no detection algorithm is used, and usually, a weighting strategy helps the model to adapt to the new changes [36]. The model may also use a dynamic approach where learners are added and removed dynamically, and rely on the most recent data to make a prediction [16]. Addictive Expert Ensembles (AddExp) [35], Dynamic Weighted Majority (DWM) [34], Geometrically Optimum and Online-Weighted Ensemble (GOOWE) [7], Learn++.NSE [19], Resample-based Ensemble Framework for Drifting Imbalanced Stream (REDI) [54] and Dynamic Adaption to Concept Changes (DACC) [12] fall into this category.

Chunk-based vs. Online algorithms: In terms of learning procedure and manner of instance arrival, ensemble methods are categorized into chunk-based and online learning algorithms [24]. In chunk-based models, 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 a late response and decreases the effectiveness of the model. In contrast to online models, chunk-based models are more efficient. Accuracy Weighted Ensemble (AWE) [50], Learn++.NSE [19] and Accuracy Updated Ensemble (AUE) [9] are in this category.

Online methods process each data item separately, and they do not need to gather a chunk of data, which leads to less memory usage. Processing a single data instance at a time is time-consuming. As we mentioned earlier, one of the most important factors in mining data streams is the efficiency of the model, and based on this fact, inefficiency is the most important drawback of online models. Diversity of Dealing with Drift (DDD) [43], DWM [34], AddEXP [35] are examples for online ensembles.

Chapter 3

Method

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, assuming a chunk of data arrives at that time. We define the problem of data stream classification as follows: First, the model predicts the class of incoming data X_k by generating the probability values for each class.

Assuming that there is C, predefined classes, s_k is the probabilities vector with the length of the number of labels that is pre-defined. We assume that, for every instance, the correct label becomes available at time-step $k + 1$, making possible the evaluation procedure. Next, using the correct classes of X_k, and the obtained features, the model is updated incrementally. Such a method can be useful in some environments such as the stock market. Fundamentally, the explained procedure is the assumption used in the majority of the data stream classification studies. This procedure is known as *prequential learning* or test-then-train learning paradigm in the literature [22]. As a result, every data instance is used both for training and testing. We propose BELS as being specially designed for this problem setting. In the following chapter, we introduce our method step by step.

3.1 BLS: Broad Learning System

Broad Learning System (BLS) utilizes a broad neural architecture [13] and achieves high performance, both in runtime and accuracy in traditional environments. In the first stage, BLS takes the input data and creates a feature mapping layer to overcome the data's randomness. Then the feature mapping layer is concatenated with a set of randomly generated enhancement nodes. Next, the output of both the feature mapping layer and the enhancement layer is directly fed to the output layer. To determine the output weights, BLS solves the least-squares problem by finding the pseudoinverse.

Assume that we take the input data as X and denote the output matrix as Y. In the case of a chunk of data with size S_c, Y would be a $S_c * C$ matrix where C stands for the number of class labels. In the feature mapping layer, the data is mapped using $\phi_i(XW_{ei} + \beta_{ei})$ where W_{ei} and β_{ei} are random weights and bias of the ith mapped feature respectively. W_{ei} and β_{ei} are initiated randomly. Each mapped feature is used to form a group of n mapped features by concatenation. n is the maximum number of feature mapping nodes.

We denote this concatenation as $Z^n = [Z_1, ..., Z_n]$. Next, the output of this feature mapping layer is enhanced using random weights in the enhancement layer, which is constructed by transforming the mapped features. We use ith feature map to form the jth enhancement node H_j as follows: $\xi(Z^i W_{hj} + \beta_{hj})$. Like the feature mapping layer, the concatenation of m enhancement nodes are grouped as $H^m = [H_1, ..., H_m]$. m is the maximum number of enhancement nodes. Based on the original BLS paper [13], the broad model is defined as follows:

$$Y = [Z_1, ..., Z_n | \xi(Z^n W_{h1} + \beta_{h1}), ..., \xi(Z^n W_{hm} + \beta_{hm})] W^m$$

$$= [Z_1, ..., Z_n | H_1, ..., H_m] W^m$$

$$= [Z^n | H^m] W^m$$

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

W^m is the connecting weights of the system.

BLS proposes an incremental way of updating the system for large datasets. The incremental approach uses a large chunk of data for updating the system at every step; however, there are a couple of reasons that make BLS an ineffective system while dealing with data streams. First, BLS uses the initial set of incoming data for generating the sparse features using a sparse autoencoder. The problem is that in each time step, the system uses the same data representation, without any update. Besides, by using a smaller chunk size, the proposed pseudoinverse updating in BLS is not effective since the focus of learning is now on the remaining data in the last chunk. Meaning that the representation of the former data instances is not taken into account. Moreover, there is no mechanism to handle this issue in BLS and its proposed variants. It is worth mentioning that our proposed model is different from other variations in [14]. In [14], the proposed approach is not designed for a streaming environment, and cannot handle concept drift. Furthermore, our model focuses on using small chunk sizes wherein these variations, this problem is not taken into account.

In the following section, we extend BLS on feature mapping and the output layer by introducing a new updating system. Then we elaborate an ensemble method for handling various concept drifts in non-stationary environments.

3.2 BELS: Broad Ensemble Learning System

To tackle the problems mentioned in section 3.1 we propose BELS. In sections 3.3 and 3.4 we propose a solution for updating the model with small chunks. Our solution enhances the accuracy of the model dramatically. In section 3.5 we introduce our ensemble approach for dealing with concept drift. Our experimental results show that the proposed model is able to deal with various drift types.

For generating enhancement nodes, we use the same function in BLS.

3.2.1 Updating the Sparse 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. 3.1 is defined in BLS paper for this purpose [13]. The output of these iterative steps is denoted as μ.

$$\begin{cases} w_{k+1} := \left(z^T z + \rho I\right)^{-1} \left(z^T X + \rho \left(o^k - u^k\right)\right) \\ o_{k+1} := S_{\lambda/\rho}\left(w_{k+1} + u^k\right) \\ u_{k+1} := u^k + \left(w_{k+1} - o_{k+1}\right) \end{cases} \tag{3.1}$$

In Eq. 3.1, X is the input data in a time interval 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^k, o^k, and w^k are initialized as zero matrices at the beginning of the iteration. These matrices are only used for updating purposes in iterative steps of Eq. 3.1. In the formula, $\rho > 0$, I is the identity matrix, and S is the soft thresholding operator. The input of S is the sum of $w_{k+1}$and u^k and a small value κ. In our experiments we use $\kappa = 0.001$. S is calculated as follows :

$$S_\kappa(a) = \begin{cases} a - \kappa, & a > \kappa \\ 0, & |a| <= \kappa \\ a + \kappa, & a < -\kappa \end{cases} \tag{3.2}$$

While considering the incremental input in a stream environment, the projection of data varies over time. Meaning that we cannot 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. 3.1, that in each step k, μ^k is the projection of the entire data from step 0 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 effectiveness of the model drastically when dealing with small and large chunks.

Algorithm 1 Feature mapping and enhancement layers

Input: *Data chunk* X
Output: *A set of Feature mapping and enhancement node* A_k

Initiate random W_e, W_h, β_e and β_h at the beginning
X = instances at step k
for i=0 ; i $\leq$ n **do**
    $z = (XW_{e_i} + \beta_{e_i})$
    $T_1 = Xz^T$
    $T_2 = z^T z$
    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. 3.5
    $Z_i = X\mu_i$
end for
Set the feature mapping group ${Z_k}^n = [Z_1, ..., Z_n]$
for j $\leftarrow$ 1 ; j $\leq$ number of enhancement nodes m **do**
    calculate $H_j = [\tanh({Z_K}^n W_{h_j} + \beta_{h_j})]$ with Eq. 3.6
end for
Set the enhancement node group ${H_k}^m = [H_1, ..., H_m]$
    $A_k = [{Z_k}^n | {H_k}^m]$

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

Let us denote $z^T X$ as T_1. We know that 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 Theorem 1, Eq. 3.3 is used to update T_1 value. To the best of our knowledge,

our study is the first work that introduces and proves this theorem.

$$T_{1_k} = \sum_{i=1}^{k} T_{1_i} \tag{3.3}$$

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. 3.4 to update T_2 at each step k:

$$T_{2_k} = \sum_{i=1}^{k} T_{2_i} \tag{3.4}$$

We use T_{2_k} and T_{1_k} as an input for the modified version of Eq. 3.1, and utilize it as in Eq. 3.5.

$$\begin{cases} w_{k+1} := \left(T_{2_k} + \rho I\right)^{-1} \left(T_{1_k} X + \rho \left(o^k - u^k\right)\right) \\ o_{k+1} := S_{\lambda/\rho}\left(w_{k+1} + u^k\right) \\ u_{k+1} := u^k + \left(w_{k+1} - o_{k+1}\right) \end{cases} \tag{3.5}$$

The enhancement nodes are created using the following formula:

$$H_j = [\tanh({Z_K}^n W_{h_j} + \beta_{h_j})] \tag{3.6}$$

${Z_K}^n$ is a set of feature mapping nodes at each time step k, and like feature mapping layer, W_{h_j} and β_{h_j} are generated randomly. It is worth mentioning that the enhancement layer is used as it is proposed in [13].

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

After concatenating the output of the feature mapping layer and the output of

the enhancement layer horizontally, the next step is calculating the pseudoinverse for the least square problem discussed in BLS [13].

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 sum of A_t and A'_t. The hypothesis can be formulated as follows:*

$$A_c^T A_c = A^T A + A'^T A' \quad \textit{where:} \quad A_c = \begin{bmatrix} A & | & A' \end{bmatrix} \tag{3.7}$$

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:

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

Left-hand side of the Eq. 3.7 is the multiplication of A_c^T and A_c matrices. Let us denote the element at ith row and jth 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. 3.7 be represented by r_{ij}. In the following, we show that these two elements are equal regardless of i and j values, meaning that the resulting matrices in the left and right-hand sides of the Eq. 3.7 are equal.

$$l_{ij} = \sum_{k=1}^{m+m'} A_c^{ki} \times A_c^{kj} = \sum_{k=1}^{m} A_c^{ki} \times A_c^{kj} + \sum_{k=m+1}^{m+m'} A_c^{ki} \times A_c^{kj}$$
$$= \sum_{k=1}^{m} a_{ki} \times a_{kj} + \sum_{k'=1}^{m'} a_{k'i} \times a_{k'j}$$

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

□

3.2.2 Updating the Pseudoinverse in BELS

Unlike BLS, where the pseudoinverse is calculated based on the instances of the last chunk of 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 output of the enhancement layer horizontally. To obtain the pseudoinverse, BLS uses Eq. 3.8 [13]:

$$W = (\lambda I + AA^T)^{-1} A^T Y \tag{3.8}$$

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:

$$A_t = {A^T}_k A_k \tag{3.9}$$

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

$$D_t = A_k^T Y_k \tag{3.10}$$

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

$$A_{t_k} = \sum_{i=1}^{k} A_{t_i} \quad \text{and} \quad D_{t_k} = \sum_{i=1}^{k} D_{t_i} \tag{3.11}$$

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 point respectively. The procedure is shown in Algorithm 2. First we calculate A_{tk} and D_{tk} (Alg.2:1-9). Next, Eq. 3.12 is used to update the pseudoinverse (Alg.2:10).

$$W = (\lambda I + A_{t_k})^{-1} D_{t_k} \tag{3.12}$$

For testing, we use a similar approach to BLS; however, just like the previous steps, we separate the calculations related to feature mapping and enhancement layers, and the output layer. Algorithm 3 shows the procedures of the testing phase. First, features are mapped and enhanced (Alg.3:1-8). Then, they are concatenated and the final prediction is calculated (Alg.3:9-10). In this phase, A_{testk} is generated and then it is used for producing the prediction based on the following formula in the final stage:

$$\hat{y} = A_{testk} W \tag{3.13}$$

Algorithm 2 Output layer weight calculation

Input: *A set of Feature mapping and enhancement node at step k denoted as* A_k

Output: W

1: $A_t = A_k^T A_k$
2: $D_t = A_k^T Y_k$
3: **if** $k = 0$ **then**
4: $\quad A_{t_k} = A_t$
5: $\quad D_{t_k} = D_t$
6: **else**
7: $\quad A_{t_k} = A_{t_{k-1}} + A_t$
8: $\quad D_{t_k} = D_{t_{k-1}} + D_t$
9: **end if**
10: $W = (\lambda I + A_{t_k})^{-1} D_{t_k}$

Algorithm 3 BELS Test

Input: *Data chunk* X_{test}

Output: *Set of Feature mapping & enhancement node at step k denoted as* A_{ktest}

for i= 0 ; i $\leq$ n **do**
    $Z_{testi} = X_{test}\mu_i$
end for
Set the feature mapping group
    ${Z_{testk}}^n = [Z_{test1}, ..., Z_{testn}]$
for j $\leftarrow$ 1 ; j $\leq$ number of enhancement nodes m **do**
    calculate H_{testj} with Eq. 3.6
end for
Set the enhancement node group
    ${H_{testk}}^m = [H_{test1}, ..., H_{testm}]$
$A_{testk} = [{Z_{testk}}^n | {H_{testk}}^m]$
$\hat{y} = A_{testk}W$

3.2.3 Concept Drift Adaptation in BELS

For concept drift handling, we use an ensemble approach. Our approach is considered 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 arrives; however, an ensemble of output layer instances is used to determine the final result. First, using an ensemble of a whole BLS is not efficient as it needs more calculations. Furthermore, initializing the feature mapping and enhancement layer for each arriving data chunk, delays the learning process, because the initial feature mapping and enhancement layer of the data is not comprehensive. Only the output layer instances with the best prediction in the last chunk are kept in the model, and those with an accuracy less than threshold δ are replaced with a new layer, or one of the output layers in the pool, which consists of removed output layers. Symbols and notations used in the ensemble learning part are in Table 3.1.

Table 3.1: Additional Symbols and Notation for BELS (Algorithm 4)

Symbol	Meaning
ξ	Ensemble of output layers $\xi = \{O_1, O_2, O_2, ..., O_m\}$
P	The pool, it contains the removed output layers
L	List of indexes of output layer instances with an accuracy lower than 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
M_o	Maximum number of output layer instances
M_p	Maximum number of output layer instances in P

Algorithm 4 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

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**
    $\xi \leftarrow$ *new output layer added*
  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 $bP[i]$
    end while
    if P length $> M_p$ **then**

keep the last M_p instances of P
end if
end if
Calculate A_{testk} using Algorithm 3.
for i $\leftarrow 0$; i $\leq \xi$ length **do**
if O_i is initialized **then**
${s_i}^k$, ${acc_i}^k \leftarrow$ Prediction & acc of O_i (Eq.3.13)
if chunk size ==2 **then**
$\delta = 0.5$
else
$\delta =$ overall accuracy
end if
if ${acc_i}^k < \delta$ **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**
${acc_j}^k \leftarrow$ Test $P[j]$ using (Eq. 3.13)
if ${acc_j}^k > \eta$ **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

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 $\xi = \{O_1, O_2, O_2, ..., O_l\}$. For updating these parts at each

time step, Algorithm 1 and Algorithm 2 are used. A set of new data instances $I = \{I_1, I_2, I_3, ..., I_{S_c}\}$ where $1 < S_c \leq 50$ is used for each update. .

To initialize the procedure, we first add an output layer instance to ξ. (Alg.4:4-5). Then we initialize the model (Alg.4:21-24). In the next iterations, first, we check if the number of output layer instances has reached the predefined maximum number. If ξ is full, then the model removes the worst-performing output layers instances. 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 (Alg.4:6-12). In the testing phase, the accuracy of each output layer instance is calculated and the indexes of the worst-performing ones are added to L (Alg.4:27-39).

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 (Alg.4:41). Next, output layer instances in P are tested. If their accuracy is more than a threshold (denoted by η), they are added to bP (Alg.4:42-46). This procedure gives the output layer instances in P a chance to be added to ξ, and used in the learning process once again. Finally, the model is updated (Alg.4:47-50) and the same procedures are repeated.

Figure 3.1 shows the overall structure of BELS. As we see in the figure, the model uses a mini chunk of data for updating the model. The same chunk is also used for testing.

3.3 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 Algorithm 3 take

$O(n+m)$ time. Our main ensemble method is based on Algorithm 4. Execution time for initializing the model and removing the output layer instances, or adding them back to the model (Alg.4:2-20) is negligible. Next, for testing, we use Algorithm 2. For each chunk, it is executed once in $O(n+m)$ time. Let us denote the ensemble size as S_ξ. Then, the final prediction for each output layer instance is calculated in (Alg.4:28-40), and it takes $O(S_\xi)$ time. Later, we test the output layer instances in the pool in (Alg.4:42-46). This procedure takes $O(P)$ time. Next, 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:

$$O\Big(\frac{N}{S_c}\Big(2(m+n)+S_\xi+P+\big(S_\xi\, max(S_c,(n+m))^2 \\ min(S_c,(n+m))\big)\Big)\Big) \tag{3.14}$$

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:

$$O\Big(\frac{N}{S_c}\big(S_\xi\, max(S_c,(n+m))^2 min(S_c,(n+m))\big)\Big) \tag{3.15}$$

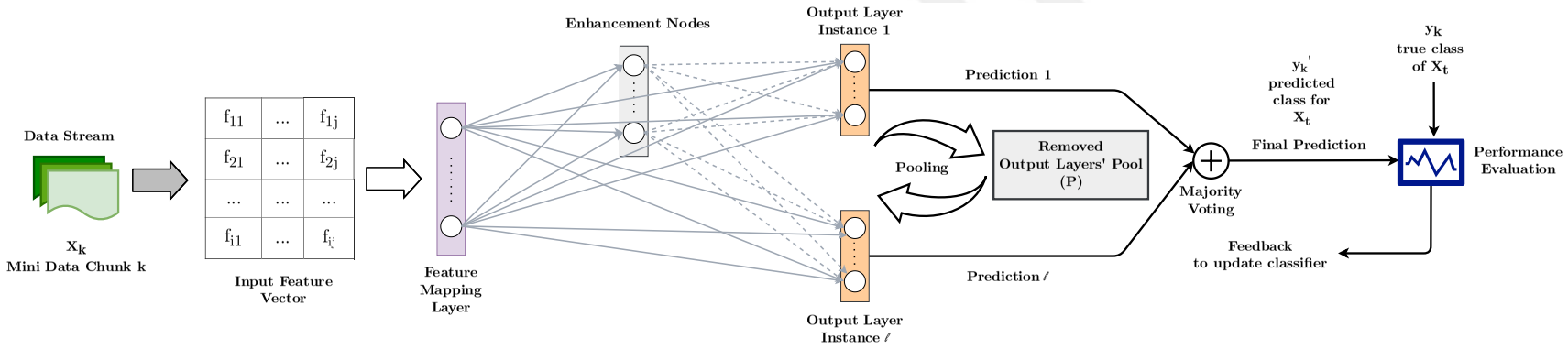

Figure 3.1: Overall schema of Broad Ensemble Learning System (BELS) for drifting stream classification.

Chapter 4

Experimental Design

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

4.1 Datasets

In our experiments, we use 11 datasets to evaluate the efficiency and effectiveness of our model, and compare it with baselines. Seven Real (Re) and four Synthetic (Syn) datasets are used. The detailed explanation of the datasets is presented in Table 4.1. Four different drift types are used in the experiments: Gradual (G), Incremental (I), Abrupt (A), and Recurring (R). In Table 4.1, (U) stands for unknown drift type. In the same table, the number of features, the number of class labels, and $Drift\ Type$ are denoted as (# F), (# C), and DT respectively. As we see in Table 4.1, the set of datasets used in our experiments contains all different concept drift types.

Table 4.1: Summary of Datasets

Dataset	# F	# C	Size	Type	DT
Electricity[1] [29]	8	2	45,312	Re	U
Email[1] [32]	913	2	1,500	Re	A&R
Phishing[2] [49]	46	2	11,055	Re	U
Poker [2] [17]	10	10	829,201	Re	U
Spam[2] [49]	499	2	6,213	Re	U
Usenet[2] [33]	99	2	1,500	Re	A&R
Weather[2] [19]	8	2	18,152	Re	U
Inetrchanging RBF[2] [40]	2	15	200,000	Syn	A
MG2C2D[3] [18]	2	2	200,000	Syn	I&G
Moving Squares[2] [40]	2	4	200,000	Syn	I
Rotating Hyperplane[2] [40]	10	2	200,000	Syn	I

4.2 Experimental Setup

Evaluation of each method is based on the *prequential accuracy* [22]. This method is the most common technique for evaluating the models in a data stream environment. In this method, a data instance or a chunk of instances are first used for testing, then the same instances are used in training. Another name for this method is *interleaved-test-then-train* [22].

In this part, we explain the default settings used in our experiments. We suggest using these configurations for experiments with new datasets and stream environments. In our experiments, the maximum number of output layer instances are M_o= 75, M_P= 300, η= 0.5, and δ is updated dynamically in the model (See Alg.4:31-35). The parameters are tuned based on the result of the experiments, and the best ones are used as default. The details for setting the parameters are later provided in section 5.5.

To determine the number of feature mapping groups and the number of feature mapping and enhancement nodes, we use the first 300 data instances. For this

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

purpose, two combinations of $BELS$ is used as follows: $BELS_1$ ($N1$: 5, $N2$: 5, $N3$: 1), $BELS_2$ ($N1$: 10, $N2$: 10, $N3 : 100$). N_1, N_2, N_3 stand for the number of feature mapping groups, the number of feature mapping nodes, and the number of enhancement nodes respectively.

A combination of one of the two BELS models and chunk sizes $\{2, 5, 20, 50\}$ are used. In total, we start with 8 different architectures and keep learning with the one with the highest accuracy after 300 instances. The number of instances for the model can be determined based on the size of the dataset.

To compare the performance of BELS with other methods, we choose 7 state-of-the-art models as baselines: **DWM** [34], **AddExp** [35], **Learn++.NSE** [19], **OzaADWIN** [46], **HAT** [3], **LevBag** [4] and **GOOWE** [7]. All the baselines (Except GOOWE) are implemented using scikit-multiflow [44]. The source code for GOOWE is available on the website[4]. For a fair comparison, we utilize the default settings of all of the baselines and our proposed approach.

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

[4]https://github.com/abuyukcakir/goowe-python

Chapter 5

Experimental Evaluation

In this chapter, we first compare our method with its variants and also the original BLS. Then we perform a thorough experimental evaluation with various baselines in terms of prequential accuracy and execution time. Next, parameter sensitivity analysis are provided. Finally, an experiment on LED dataset with different noise and drift intensities is provided to show the robustness of our method in presence of noise and concept drift.

5.1 BELS vs. BLS

In this section, we compare our method with the original BLS algorithm [13]. 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 show the improvement on each measure. We analyze the effects of the different features of BELS on accuracy by performing the experiments on three different BELS variants, and compare it with the original BLS algorithm in terms of efficiency and effectiveness.

- **BLS**: the original Broad Learning System method.

- **BELS-FPs**: BELS method with enhancements mentioned in Section 4.1 and 4.2, which includes a feature mapping layer and a pseudoinvese 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.

In this section, for brevity, we report the results of four data streams of a good variety and confirm that outcomes supporting our conclusion are observed with the other datasets. The chosen datasets contain all types of concept drift (Incremental, Gradual, Abrupt, Recurrence). Results are shown in Table 5.1. Methods are evaluated with the default configuration. Parameters are the same in all four models.

BLS is not designed for a data stream environment. Based on this fact, we know that BLS would be faster than our algorithm and at the same time, it will have extremely low accuracy. As we see in Table 5.1, by utilizing each technique, the accuracy improves, which yields the best performance in the complete version of BELS with an average accuracy improvement of 44% in Table 5.1.

In our ablation study, we compare BELS and BLS with the same number of chunks and the results show that BELS outperforms the original BLS model in terms of accuracy when the data chunks are small. However, studying the effect of larger chunk sizes (larger than 50) on BELS is another experiment that could give a clearer view of the differences between BELS and BLS, and their performance capabilities in different situations. The original BLS algorithm is not able to handle small chunks of data, and better performance is achieved by using larger chunks of data. However, based on the fact that BELS employs a drift handling mechanism in its structure, it would outperform BLS even with larger chunk sizes.

Table 5.1: Comparison between BELS, its variants and BLS in terms of prequential accuracy and runtime (in seconds). The best results for each dataset is in bold

	BLS		BELS-FPs		BELS-Ens		BELS		
Datasets (DT)	Accuracy	Runtime	Accuracy	Runtime	Accuracy	Runtime	Accuracy	Runtime	% Acc. Imp. of BELS wrt BLS
Electricity (U)	49.94	**36.36**	63.44	183.61	83.34	220.27	**87.17**	392.43	74.54
Usenet (A & R)	54.76	**1.16**	59.56	4.82	72.75	10.14	**75.68**	12.68	38.20
MG2C2D (I & G)	61.37	**14.67**	91.47	32.76	92.10	109.33	**92.92**	357.03	51.40
Rotating Hyperplane (I)	81.54	**6.33**	81.75	28.16	85.92	41.01	**90.64**	175.83	11.16

5.2 Effectiveness and Efficiency

Prequential evaluation results can be seen in Table 5.2. The results show that our proposed method outperforms other baseline methods. Our evaluation on datasets of varying drift types with different numbers of class labels and features demonstrates the adaptability of our model under different classification conditions. The best performance for each dataset is in bold.

Based on the results, it seems that if the dataset contains abrupt drift with short time intervals and recurring drift (Usenet and Emails), baseline models are not able to handle it effectively. Compared to other baseline models, BELS can handle different drift types effectively.

We can see the runtime of the models in Table 5.3. Runtime is considered as the total time for calculations in the *interleaved-test-then-train* method. The results show that our model has an average rank equal to 3.72. Further statistical analysis of runtime and accuracy is provided in the next section. Figure 5.1 shows how the average rank of models compares in terms of both runtime and accuracy. On average BELS improves 29% compared to other competitive baselines used in the experiments across 11 datasets we examined.

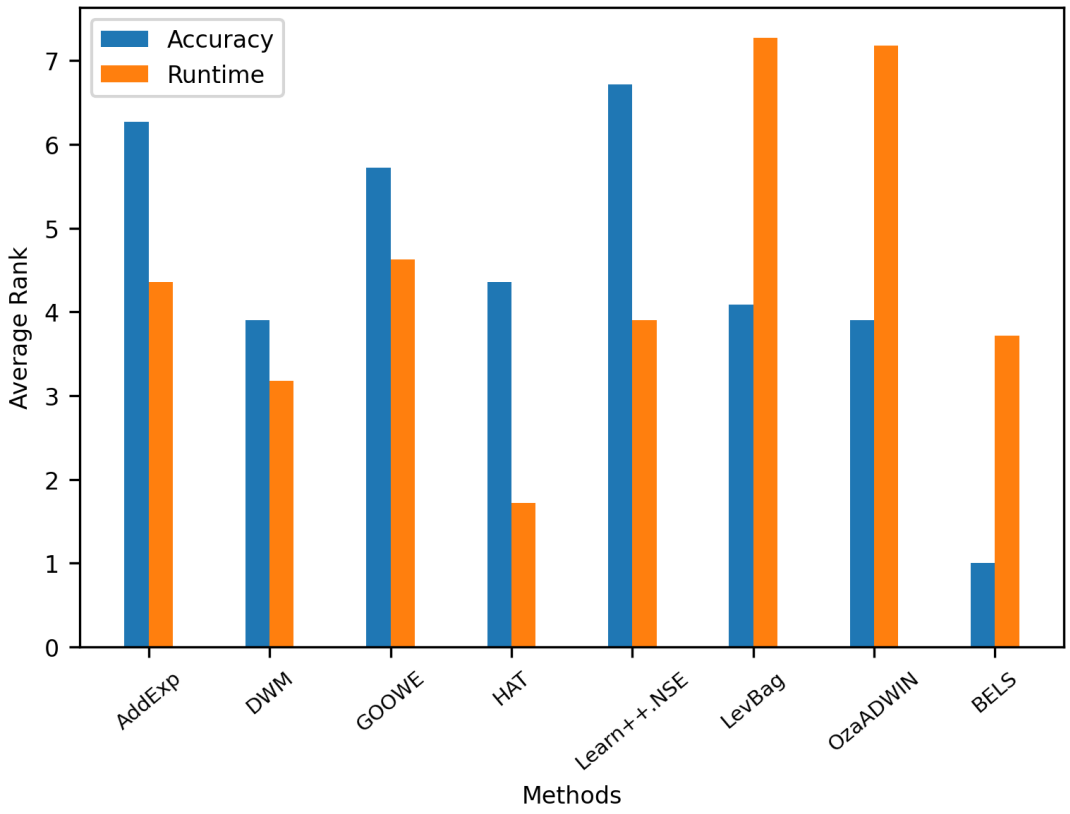

Figure 5.1: Average rank comparison for accuracy and runtime.

Table 5.2: 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

Datasets	BELS	AddExp	DWM	GOOWE	HAT	Learn++.NSE	LevBag	OzaADWIN
Electricity	**87.17**	73.30	79.87	76.25	83.32	75.19	83.51	75.99
Email	**76.42**	56.20	57.33	55.00	55.46	45.66	58.00	55.33
Phishing	**93.03**	91.40	91.83	90.17	89.28	89.74	90.44	92.87
Poker	**83.03**	59.00	72.10	61.41	66.64	74.52	81.43	82.21
Spam	**95.43**	89.31	88.34	82.84	86.06	82.52	94.46	90.85
Usenet	**75.68**	64.80	64.26	62.90	65.73	54.26	54.20	58.13
Weather	**78.32**	69.22	71.35	70.09	73.63	69.41	73.34	78.02
Interchanging RBF	**98.22**	17.42	92.72	69.30	61.95	92.57	96.40	94.14
MG2C2D	**92.92**	55.56	91.97	90.90	92.73	90.00	90.51	92.58
Moving Squares	**89.45**	32.73	34.13	70.62	74.76	41.41	60.74	43.04
Rotating Hyperplane	**90.64**	81.55	89.62	87.77	85.76	71.39	73.44	82.17
Avg. Accuracy	**87.30**	62.77	75.77	74.29	75.94	71.52	77.86	76.85
Avg. % Imp. by BELS	-	77.24	24.06	19.34	17.08	28.97	14.88	19.31
Avg. Rank	**1.00**	6.27	4.18	5.54	4.36	6.63	4.09	3.90

Table 5.3: Average runtime for the whole dataset (in seconds). For each row the lowest value is marked with bold text

Datasets	BELS	AddExp	DWM	GOOWE	HAT	Learn++.NSE	LevBag	OzaADWIN
Electricity	392.43	65.13	62.05	128.05	**30.13**	110.43	421.95	332.87
Email	12.46	100.41	83.36	30.65	34.28	**0.98**	839.07	230.31
Phishing	47.80	84.43	79.54	122.44	**27.06**	28.80	375.49	288.41
Poker	8,101.16	5,219.57	2,620.84	6,923.92	**1,213.25**	6,252.57	7,938.82	21,872.37
Spam	90.49	475.60	444.33	424.14	173.62	**11.45**	1797.14	579.76
Usenet	12.68	11.44	7.85	3.15	4.53	**0.82**	77.53	19.36
Weather	25.23	51.38	26.06	48.22	**13.02**	46.04	115.61	99.61
Interchanging RBF	417.12	394.01	158.35	954.48	**152.92**	982.77	713.39	2410.79
MG2C2D	357.03	178.34	158.77	305.07	662.28	762.70	**101.07**	679.99
Moving Squares	151.40	225.39	167.37	422.57	**113.65**	554.21	629.65	1,281.14
Rotating Hyperplane	**175.83**	308.40	297.96	795.46	1,518.72	662.34	190.56	1,438.28
Avg. Runtime	889.42	646.74	373.32	923.49	**358.50**	855.74	1,200.03	2,657.54
Avg. Rank	3.72	4.36	3.18	4.63	**1.72**	3.90	7.27	7.18

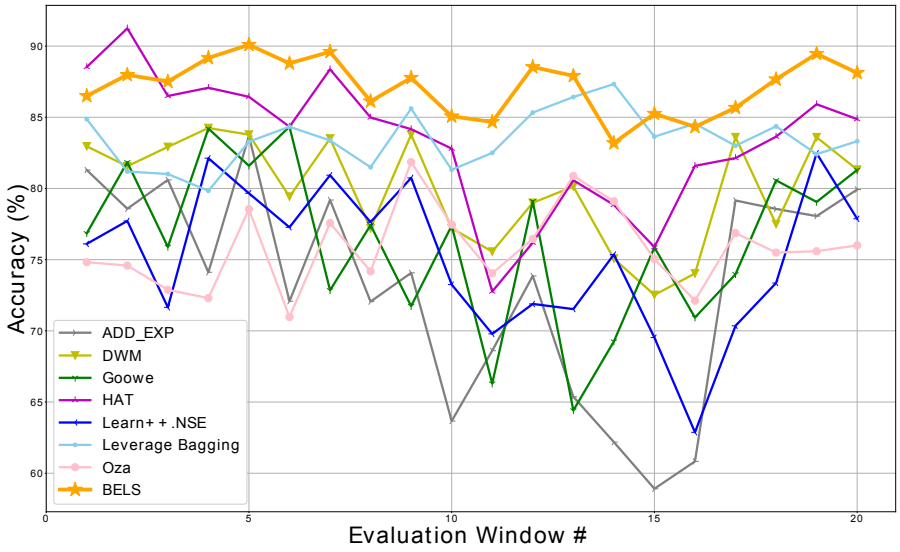

Figure 5.2: Electricity dataset accuracy plot.

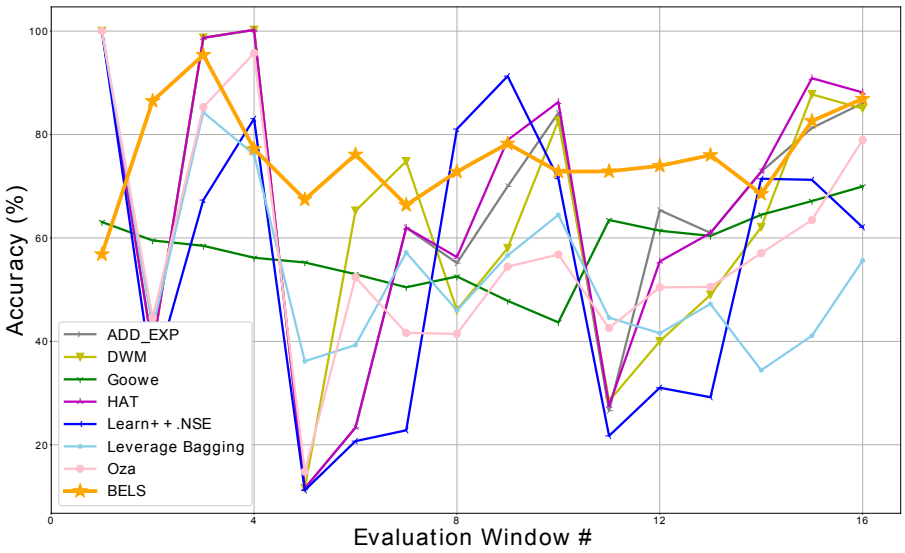

Figure 5.3: Usenet dataset accuracy plot.

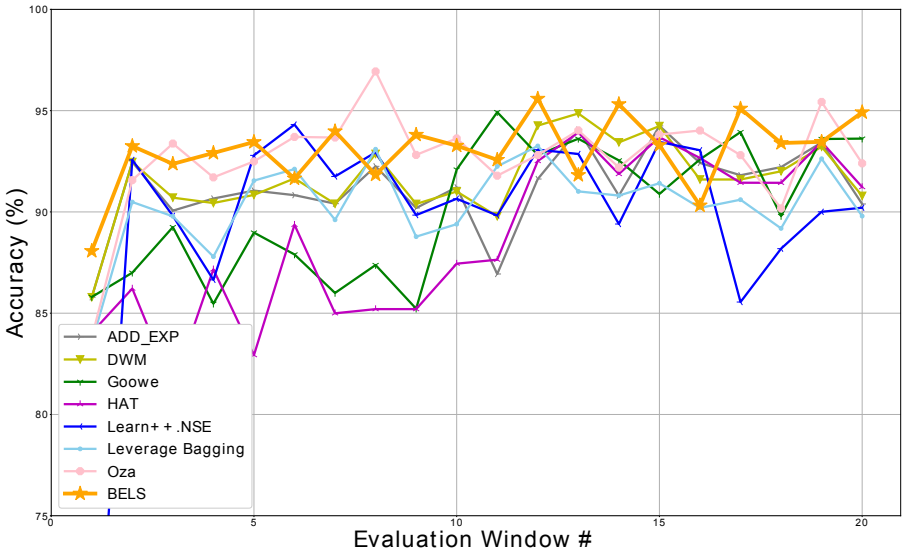

Figure 5.4: Phishing dataset accuracy plot.

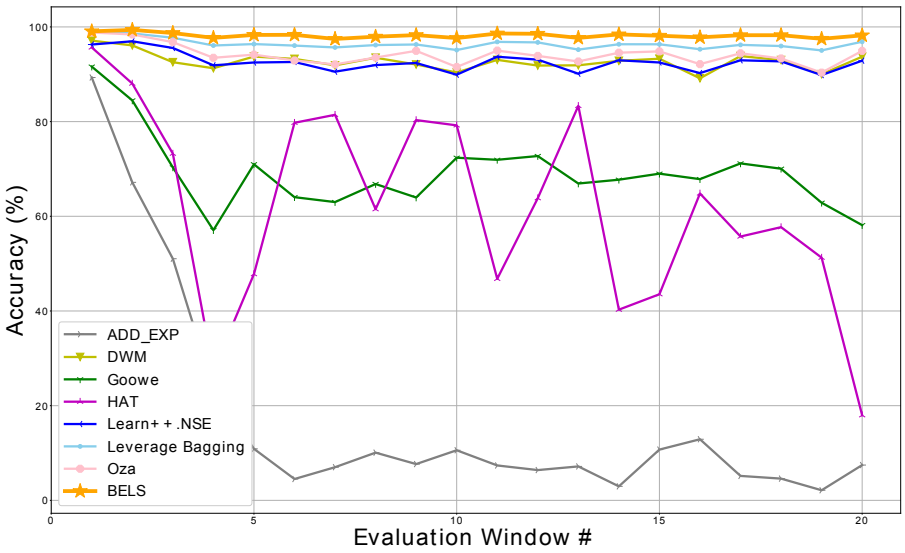

Figure 5.5: InterchangingRBF dataset accuracy plot.

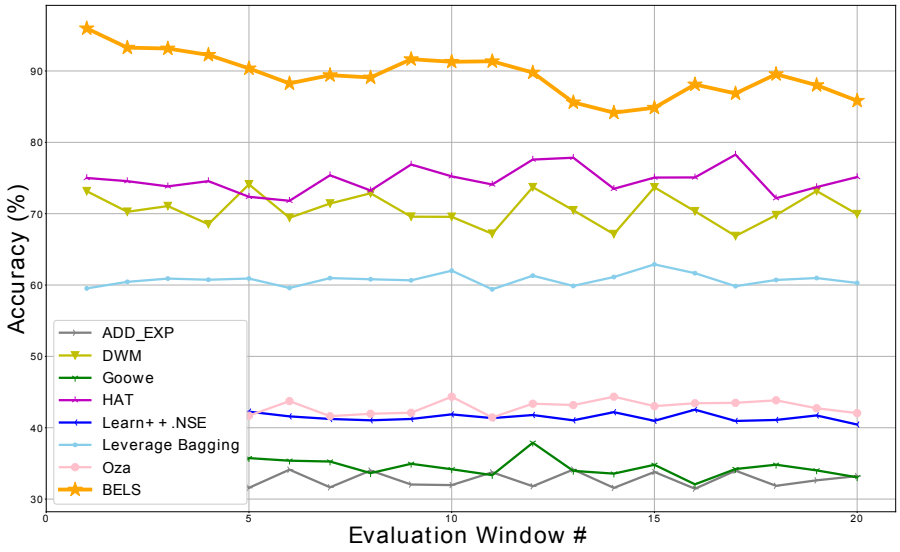

Figure 5.6: MovingSquares dataset accuracy plot.

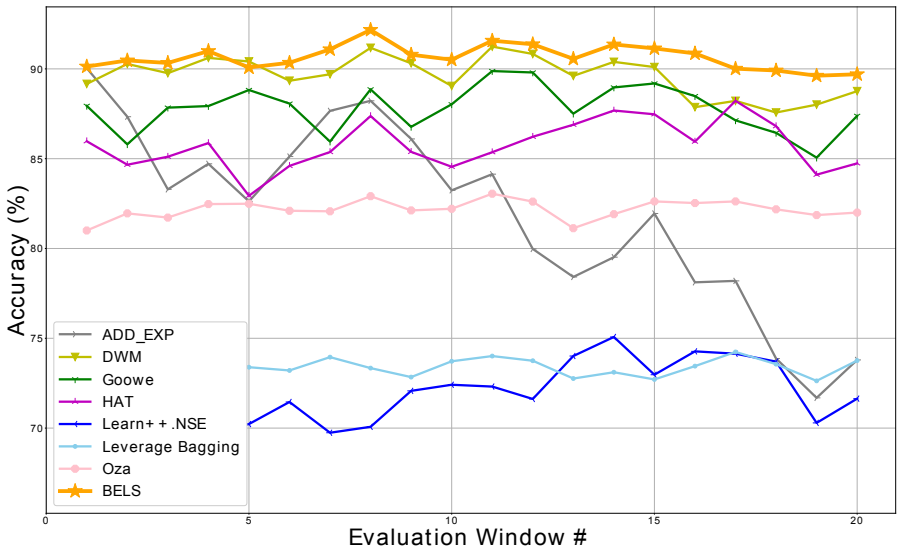

Figure 5.7: Rotating Hyperplane dataset accuracy plot.

The plots in figures (5.2 - 5.7) show that BELS has robust concept drift-resistant performance under different drift types, and it maintains much better performance when concept drift occurs (indicated by the fluctuated accuracies as the data stream progresses). Furthermore, it maintains a higher level of accuracy in the baselines; for example, in 4 of 6 cases, it is always the top-performing model during the entire process. These plots are typical and we observe similar results for other datasets.

5.3 Evaluation of Statistical Significance

In this section, we aim at using a statistical test to show the significance of our model in terms of accuracy. We also provide statistical analysis for runtime.[1]

As the differences between the accuracies of each model for each dataset is small, we used average ranks for statistical significance test. Besides, in a data stream environment in the real-world, we are dealing with a huge amount of data. Based on this fact, although the differences between each model in terms of accuracy may be small in our experiments, however, in the real-world scenario, the difference would be significant. To show this significance, we used the average ranks of the models for comparison. Another reason for choosing average ranks as the basis of our significance test is that this kind of comparison is commonly used in the data stream mining literature [15, 11, 25].

The analysis is conducted for 8 models and 11 datasets. In the experiment $\alpha = 0.05$. 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 see if there is a significant difference between the results of our proposed model and other baselines.

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 $CD=$

[1] https://orangedatamining.com

2.80. Figure 5.8 shows the critical distance diagram. It shows that our method is statistically significantly better, compared to other baselines used in this work.

Another statistical analysis is conducted based on the runtime results in Table 5.3. We use the same tests and the same procedure. First, based on the *Friedman Test* where α= 0.05 we reject the hypothesis that all of the measurements come from the same distribution. Based on the post-hoc Bonferroni-Dunn test, we calculate the *Critical Difference* as CD= 2.80. Based on the results we conclude that there are no significant differences within the following groups: HAT, DWM, BELS, Learn++.NSE and AddExp; DWM, BELS, Learn++.NSE, AddExp and GOOWE; GOOWE, OzaADWIN and LevBag . The critical distance diagram is shown in Figure 5.9.

To decrease the computational burden of the ensemble model, we use a single feature mapping and enhancement layer, and an ensemble of output layers. This technique also positively affects the accuracy of the model by providing a comprehensive feature mapping layer.

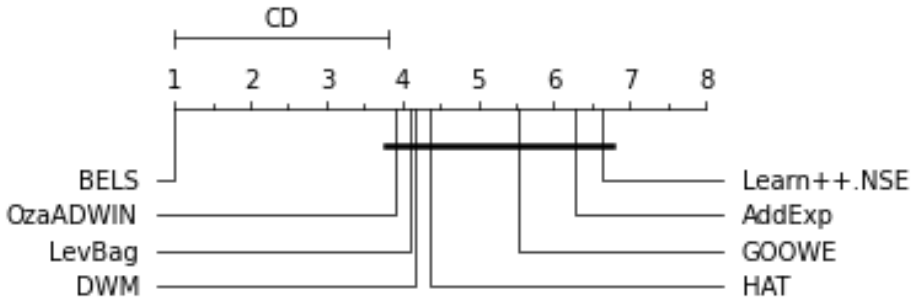

Figure 5.8: Critical distance diagram for the prequential accuracy using the data provided on Table 5.2. (CD=2.80)

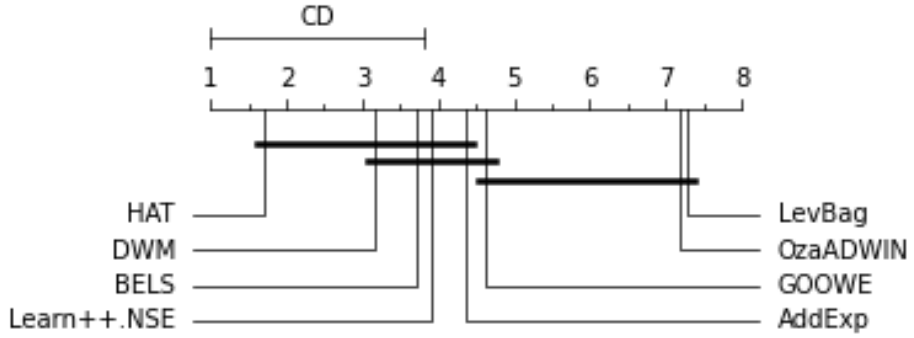

Figure 5.9: Critical distance diagram for the runtime using the data provided on Table 5.3. (CD=2.80)

5.4 Parameter Sensitivity Analysis

In this section, we study the effect of the main parameters on the overall performance of BELS. We use the same four datasets as in section 5.1. Chunk size, M_p (maximum number of output layer instances in P), and M_o (maximum number of output layer instances) are studied because of their importance in the learning process and handling the drifts. Table 5.4 shows the results for parameter sensitivity analysis.

Based on the results for chunk size, we observe that after increasing the chunk size to 100, the accuracy decreases drastically in presence of concept drift. BELS is designed in a way to use a small chunk size at every step. Based on the observations in this section, we suggest using chunk sizes equal to or smaller than 50. As mentioned earlier in section 4.2, we use the first 300 instances to automatically determine the chunk size which is chosen from a set of $\{2, 5, 20, 50\}$.

Based on the results in section 5.1, we observed that having an ensemble of output layer instances improves the results, so the ensemble size should be greater than 1. 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 higher runtime, and

fewer output layer instances may result in poor performance.

As we see in the results, after increasing M_p to 200 and more, the effectiveness remains the same. The reason is that based on our ensemble method (see Algorithm 4), after adding an output layer instance back to the learning process, we remove it from P. For this reason, usually, M_p does not exceed the defined limit for P; however, as we illustrated in section 5.1, adding the pool has a positive impact on the performance of the model. We set the M_p limit to 300 in all the experiments. Our suggestions for default parameters and experiments with other datasets are provided in Section 4.2, and they are repeated here for ease of reference: $M_o = 75$,$M_P = 300$, and chunk size is selected from a set of $\{2, 5, 20, 50\}$.

5.5 Sensitivity to Drift Intensity and Noise Percentage

As mentioned by Wang and Machida [51], error labels have a huge impact on the performance of the model in presence of concept drift. To show the effectiveness of our model in presence of different drift and noise levels, we conduct experiments using the LED generator [8] implemented in the scikit-multiflow [44] library. The LED dataset is one of the most famous datasets used in the literature [53, 5, 20]. The generator produces 24 binary features with 10 class labels. We generate 9 datasets with different noise percentages (10, 30, 70 percent), and a number of drifting features (1, 5, 10). We compare **BELS** with 3 top models including **DWM**, **Oza**, **LevBag**.

Figures (5.10 - 5.18) shows the prequential temporal accuracies for each dataset. BELS is robust to variations in noise and drift intensity, according to our experiments. OzaADWIN has a better performance in comparison to the other two baselines. As we see in Figure (5.10 - 5.18), DWM and LevBag methods have poor performance in presence of noise.

Table 5.4: Parameter sensitivity analysis in prequential accuracy. Best results for each parameter is in bold

	Chunk size					M_o: maximum number of output layer instance					M_p: maximim number of output layer instances in P				
Datasets (DT)	2	5	20	50	100	5	25	50	75	150	50	100	200	300	400
Electricity (U)	**87.37**	87.45	77.78	74.41	71.37	85.18	86.06	86.85	**87.17**	87.10	86.78	87.07	**87.17**	**87.17**	**87.17**
Usenet (A & R)	75.76	**76.70**	75.19	70.19	63.58	**76.42**	71.95	75.02	75.68	76.14	73.88	**76.35**	75.68	75.68	75.68
MG2C2D (I & G)	90.64	92.47	92.85	**92.92**	92.80	92.41	92.88	92.90	92.92	**93.01**	92.89	**93.03**	92.92	92.92	92.92
Rotating Hyperplane (I)	85.23	88.86	**90.94**	90.64	90.36	88.41	90.82	**90.94**	90.64	90.31	**91.02**	90.64	90.64	90.64	90.64

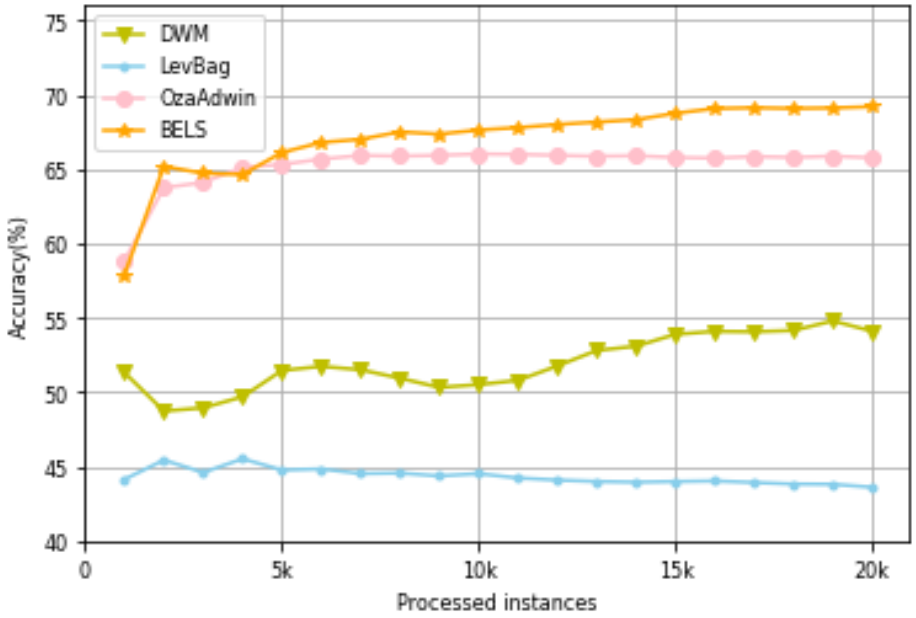

Figure 5.10: LED dataset: 10% noise, 1 drifting feature.

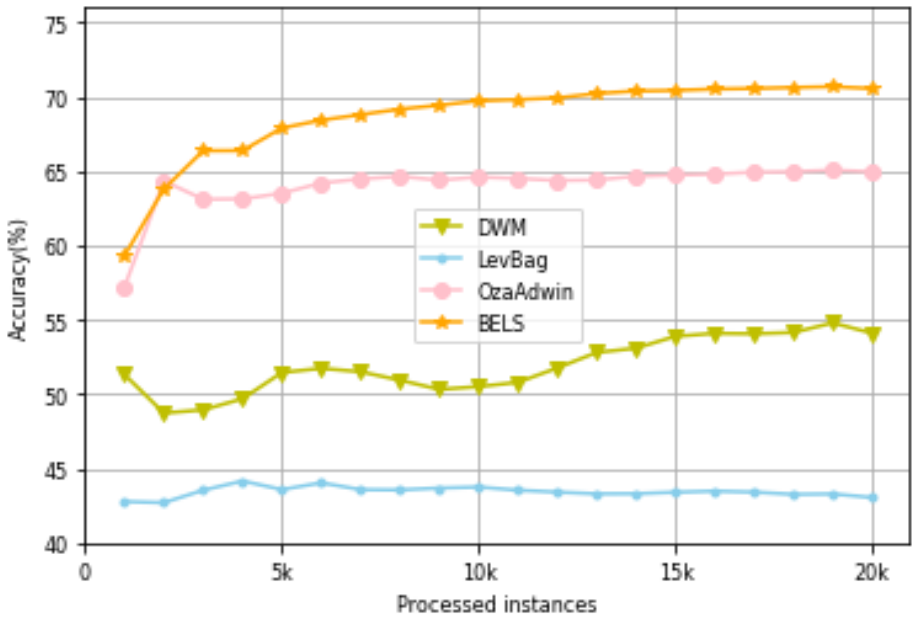

Figure 5.11: LED dataset: 10% noise, 5 drifting features.

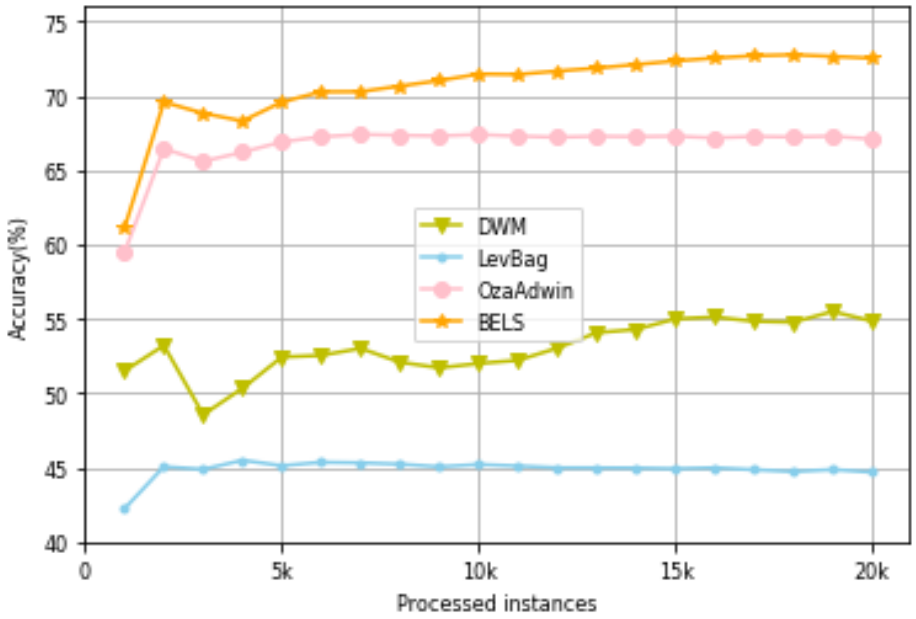

Figure 5.12: LED dataset: 10% noise, 10 drifting features.

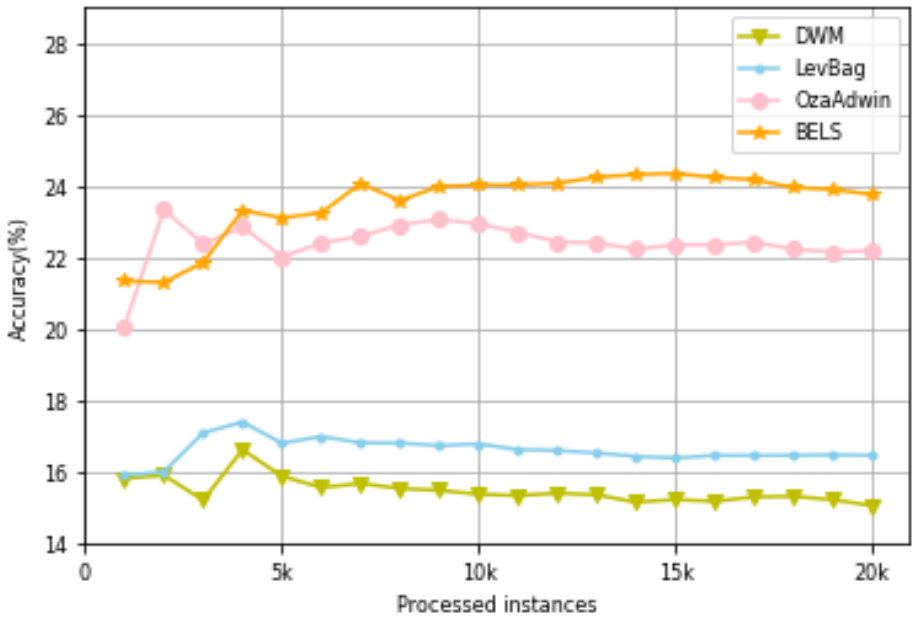

Figure 5.13: LED dataset: 30% noise, 1 drifting feature.

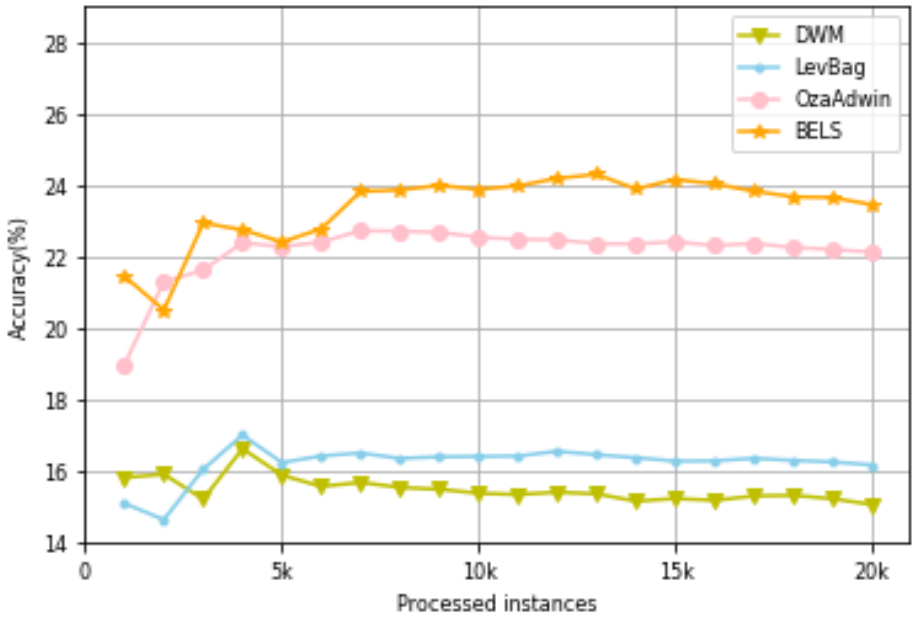

Figure 5.14: LED dataset: 30% noise, 5 drifting features.

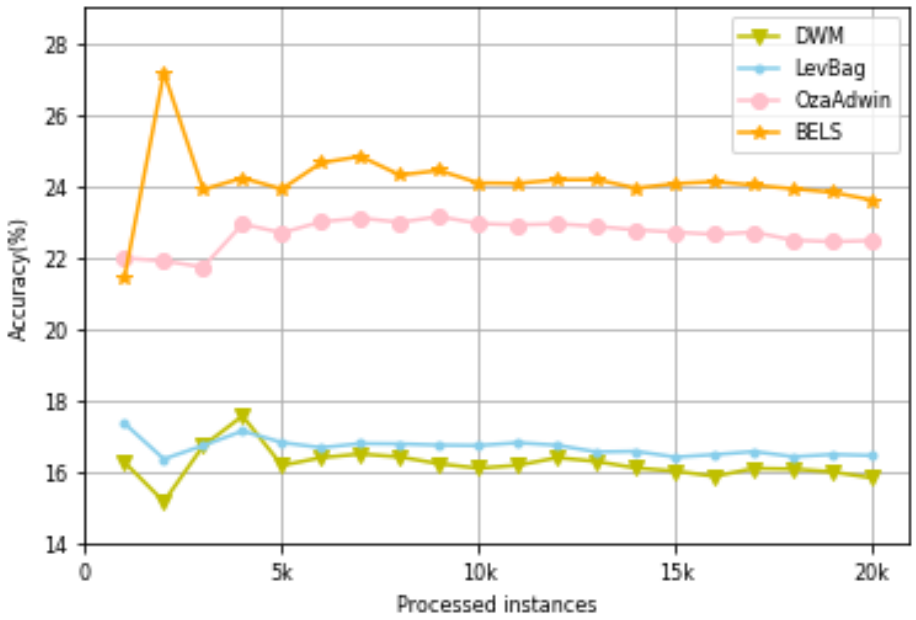

Figure 5.15: LED dataset: 30% noise, 10 drifting features.

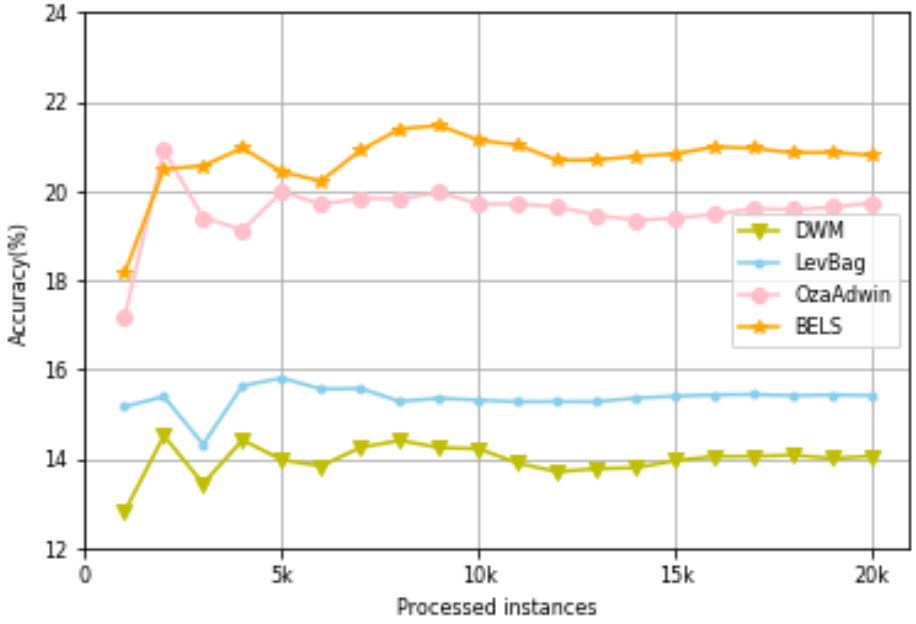

Figure 5.16: LED dataset: 70% noise, 1 drifting feature.

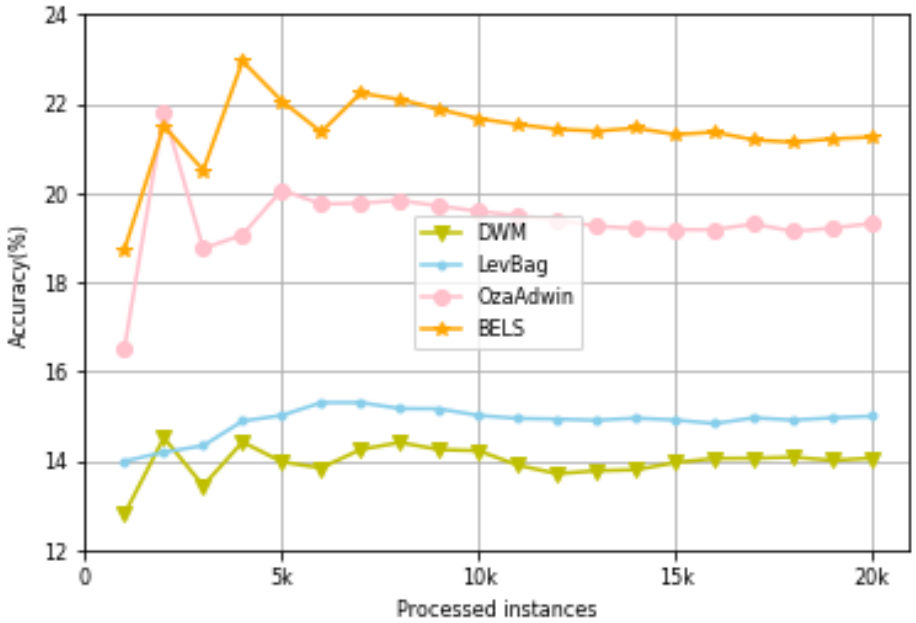

Figure 5.17: LED dataset: 70% noise, 5 drifting features.

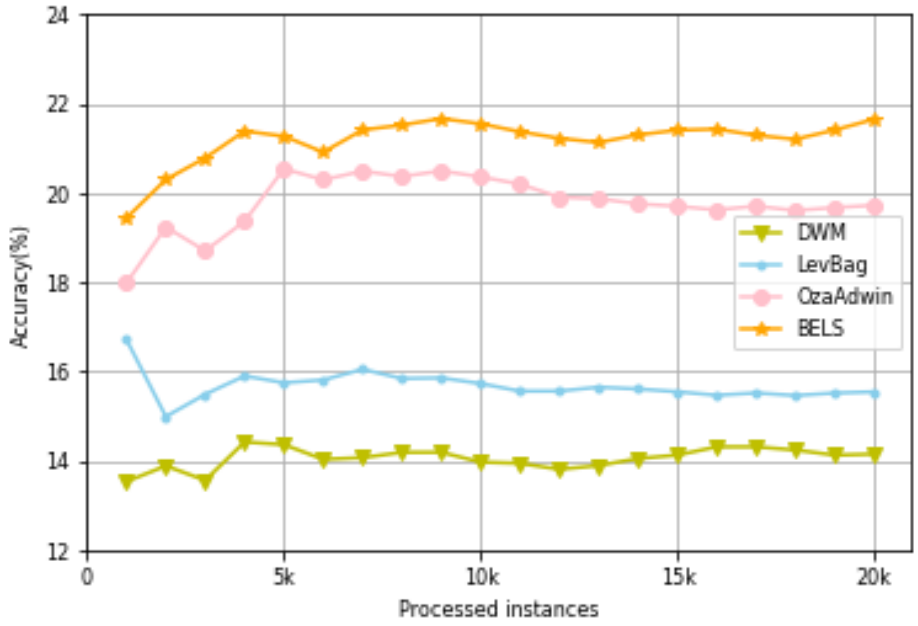

Figure 5.18: LED dataset: 70% noise, 10 drifting features.

Chapter 6

Conclusion and Future Work

In this work, we present a novel ensemble model for data stream classification in non-stationary environments, called BELS. We describe real-world and unique challenges that data stream causes and focus on handling the problems like concept drift adaptation in data streams.

The statistical test results show that our model is statistically significantly better than the state-of-the-art models designed specifically for data streams and we illustrate that BELS is a suitable choice for evolving environments. Moreover, we show that, in terms of efficiency, BELS is suitable for data stream classification.

Our proposed method is able to handle numerical data as input. Results on numeric datasets, and text datasets with a limited number of words and one-hot encoding as input, show that BELS is able to handle different types of data. However, further analysis of the performance of our model on text datasets with embedding vectors as input, and image data needs to be done. There are some problems in the data stream mining that we plan to consider as an extension to our proposed method.

- **Lack of labeled data**: Lack of available class labels in a stream may cause serious problems for supervised methods and reduce their efficacy.

- **Concept evolution**: Emerging new classes (as known as "concept evolution") is another important issue that is inherent to the stream environment.

Our aim is to propose a model based on BELS which is able to handle label scarcity and concept evolution as future work.

Bibliography

[1] Manuel Baena-Garcıa, José del Campo-Ávila, Raúl Fidalgo, Albert Bifet, R Gavalda, and R Morales-Bueno. Early drift detection method. In *Fourth International Workshop on Knowledge Discovery from Data Streams*, volume 6, pages 77–86, 2006.

[2] Albert Bifet and Ricard Gavalda. Learning from time-changing data with adaptive windowing. In *Proceedings of the 2007 SIAM International Conference on Data Mining*, pages 443–448. SIAM, 2007.

[3] Albert Bifet and Ricard Gavalda. Adaptive learning from evolving data streams. In *International Symposium on Intelligent Data Analysis*, pages 249–260. Springer, 2009.

[4] Albert Bifet, Geoff Holmes, and Bernhard Pfahringer. Leveraging bagging for evolving data streams. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 135–150. Springer, 2010.

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

[6] Hamed Bonab and Fazli Can. Less is more: A comprehensive framework for the number of components of ensemble classifiers. *IEEE Transactions on Neural Networks and Learning Systems*, 30(9):2735–2745, 2019.

[7] Hamed R Bonab and Fazli Can. GOOWE: Geometrically optimum and online-weighted ensemble classifier for evolving data streams. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 12(2):1–33, 2018.

[8] Leo Breiman, Jerome H Friedman, Richard A Olshen, and Charles J Stone. *Classification and regression trees*. Routledge, 2017.

[9] Dariusz Brzeziński and Jerzy Stefanowski. Accuracy updated ensemble for data streams with concept drift. In *International Conference on Hybrid Artificial Intelligence Systems*, pages 155–163. Springer, 2011.

[10] Dariusz Brzezinski and Jerzy Stefanowski. Reacting to different types of concept drift: The accuracy updated ensemble algorithm. *IEEE Transactions on Neural Networks and Learning Systems*, 25(1):81–94, 2013.

[11] Alican Büyükçakir, Hamed Bonab, and Fazli Can. A novel online stacked ensemble for multi-label stream classification. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 1063–1072, 2018.

[12] Alberto Cano and Bartosz Krawczyk. Kappa updated ensemble for drifting data stream mining. *Machine Learning*, 109(1):175–218, 2020.

[13] CL Philip Chen and Zhulin Liu. Broad learning system: An effective and efficient incremental learning system without the need for deep architecture. *IEEE Transactions on Neural Networks and Learning Systems*, 29(1):10–24, 2017.

[14] CL Philip Chen, Zhulin Liu, and Shuang Feng. Universal approximation capability of broad learning system and its structural variations. *IEEE Transactions on Neural Networks and Learning Systems*, 30(4):1191–1204, 2018.

[15] Salah Ud Din, Junming Shao, Jay Kumar, Waqar Ali, Jiaming Liu, and Yu Ye. Online reliable semi-supervised learning on evolving data streams. *Information Sciences*, 525:153–171, 2020.

[16] Gregory Ditzler, Manuel Roveri, Cesare Alippi, and Robi Polikar. Learning in nonstationary environments: A survey. *IEEE Computational Intelligence Magazine*, 10(4):12–25, 2015.

[17] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[18] Karl B Dyer, Robert Capo, and Robi Polikar. Compose: A semisupervised learning framework for initially labeled nonstationary streaming data. *IEEE Transactions on Neural Networks and Learning Systems*, 25(1):12–26, 2013.

[19] Ryan Elwell and Robi Polikar. Incremental learning of concept drift in nonstationary environments. *IEEE Transactions on Neural Networks*, 22(10):1517–1531, 2011.

[20] Isvani Frias-Blanco, José del Campo-Ávila, Gonzalo Ramos-Jimenez, Rafael Morales-Bueno, Agustin Ortiz-Diaz, and Yaile Caballero-Mota. Online and non-parametric drift detection methods based on hoeffding's bounds. *IEEE Transactions on Knowledge and Data Engineering*, 27(3):810–823, 2014.

[21] Joao Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. Learning with drift detection. In *Brazilian Symposium on Artificial Intelligence*, pages 286–295. Springer, 2004.

[22] Joao Gama, Raquel Sebastiao, and Pedro Pereira Rodrigues. Issues in evaluation of stream learning algorithms. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 329–338, 2009.

[23] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM Computing Surveys (CSUR)*, 46(4):1–37, 2014.

[24] Jing Gao, Wei Fan, Jiawei Han, and Philip S Yu. A general framework for mining concept-drifting data streams with skewed distributions. In *Proceedings of the 2007 SIAM International Conference on Data Mining*, pages 3–14. SIAM, 2007.

[25] Heitor M Gomes, Albert Bifet, Jesse Read, Jean Paul Barddal, Fabrício Enembreck, Bernhard Pfharinger, Geoff Holmes, and Talel Abdessalem. Adaptive random forests for evolving data stream classification. *Machine Learning*, 106(9):1469–1495, 2017.

[26] Ömer Gözüaçık, Alican Büyükçakır, Hamed Bonab, and Fazli Can. Unsupervised concept drift detection with a discriminative classifier. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 2365–2368, 2019.

[27] Ömer Gözüaçık and Fazli Can. Concept learning using one-class classifiers for implicit drift detection in evolving data streams. *Artificial Intelligence Review*, 54(5):3725–3747, 2021.

[28] Feng Gu, Guangquan Zhang, Jie Lu, and Chin-Teng Lin. Concept drift detection based on equal density estimation. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 24–30. IEEE, 2016.

[29] Michael Harries and New South Wales. Splice-2 comparative evaluation: Electricity pricing. 1999.

[30] Geoff Hulten, Laurie Spencer, and Pedro Domingos. Mining time-changing data streams. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 97–106, 2001.

[31] Mobin M Idrees, Leandro L Minku, Frederic Stahl, and Atta Badii. A heterogeneous online learning ensemble for non-stationary environments. *Knowledge-Based Systems*, 188:104983, 2020.

[32] Ioannis Katakis, Grigorios Tsoumakas, and Ioannis Vlahavas. Tracking recurring contexts using ensemble classifiers: an application to email filtering. *Knowledge and Information Systems*, 22(3):371–391, 2010.

[33] Ioannis Katakis, Grigorios Tsoumakas, and Ioannis P Vlahavas. An ensemble of classifiers for coping with recurring contexts in data streams. In *ECAI*, pages 763–764, 2008.

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

[35] Jeremy Z Kolter and Marcus A Maloof. Using additive expert ensembles to cope with concept drift. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 449–456, 2005.

[36] Bartosz Krawczyk, Leandro L Minku, João Gama, Jerzy Stefanowski, and Michał Woźniak. Ensemble learning for data stream analysis: A survey. *Information Fusion*, 37:132–156, 2017.

[37] Doug Laney et al. 3d data management: Controlling data volume, velocity and variety. *META Group Research Note*, 6(70):1, 2001.

[38] Zeng Li, Wenchao Huang, Yan Xiong, Siqi Ren, and Tuanfei Zhu. Incremental learning imbalanced data streams with concept drift: The dynamic updated ensemble algorithm. *Knowledge-Based Systems*, 195:105694, 2020.

[39] Weike Liu, Hang Zhang, Zhaoyun Ding, Qingbao Liu, and Cheng Zhu. A comprehensive active learning method for multiclass imbalanced data streams with concept drift. *Knowledge-Based Systems*, 215:106778, 2021.

[40] Viktor Losing, Barbara Hammer, and Heiko Wersing. Knn classifier with self adjusting memory for heterogeneous concept drift. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 291–300. IEEE, 2016.

[41] Jie Lu, Anjin Liu, Fan Dong, Feng Gu, Joao Gama, and Guangquan Zhang. Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering*, 31(12):2346–2363, 2018.

[42] Ning Lu, Jie Lu, Guangquan Zhang, and Ramon Lopez De Mantaras. A concept drift-tolerant case-base editing technique. *Artificial Intelligence*, 230:108–133, 2016.

[43] Leandro L Minku and Xin Yao. DDD: A new ensemble approach for dealing with concept drift. *IEEE Transactions on Knowledge and Data Engineering*, 24(4):619–633, 2011.

[44] Jacob Montiel, Jesse Read, Albert Bifet, and Talel Abdessalem. Scikit-multiflow: A multi-output streaming framework. *Journal of Machine Learning Research*, 19(72):1–5, 2018.

[45] Kyosuke Nishida, Koichiro Yamauchi, and Takashi Omori. Ace: Adaptive classifiers-ensemble system for concept-drifting environments. In *International Workshop on Multiple Classifier Systems*, pages 176–185. Springer, 2005.

[46] Nikunj C Oza and Stuart J Russell. Online bagging and boosting. In *International Workshop on Artificial Intelligence and Statistics*, pages 229–236. PMLR, 2001.

[47] Abdulhakim A Qahtan, Basma Alharbi, Suojin Wang, and Xiangliang Zhang. A pca-based change detection framework for multidimensional data streams: Change detection in multidimensional data streams. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 935–944, 2015.

[48] Sergio Ramírez-Gallego, Bartosz Krawczyk, Salvador García, Michał Woźniak, and Francisco Herrera. A survey on data preprocessing for data stream mining: Current status and future directions. *Neurocomputing*, 239:39–57, 2017.

[49] Tegjyot Singh Sethi and Mehmed Kantardzic. On the reliable detection of concept drift from streaming unlabeled data. *Expert Systems with Applications*, 82:77–99, 2017.

[50] Haixun Wang, Wei Fan, Philip S Yu, and Jiawei Han. Mining concept-drifting data streams using ensemble classifiers. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 226–235, 2003.

[51] Sixiang Wang and Fumio Machida. A robustness evaluation of concept drift detectors against unreliable data streams.

[52] Gerhard Widmer and Miroslav Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23(1):69–101, 1996.

[53] Shuliang Xu and Junhong Wang. Dynamic extreme learning machine for data stream classification. *Neurocomputing*, 238:433–449, 2017.

[54] Hang Zhang, Weike Liu, Shuo Wang, Jicheng Shan, and Qingbao Liu. Resample-based ensemble framework for drifting imbalanced data streams. *IEEE Access*, 7:65103–65115, 2019.