GOOWE: GEOMETRICALLY OPTIMUM AND ONLINE-WEIGHTED ENSEMBLE CLASSIFIER FOR EVOLVING DATA STREAMS

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

Hamed Rezanejad Asl-Bonab

July 2016

GOOWE: Geometrically Optimum and Online-Weighted Ensemble Classifier for Evolving Data Streams
By Hamed Rezanejad Asl-Bonab
July 2016

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)

Özgür Ulusoy

Ismail Sengor Altingovde

Approved for the Graduate School of Engineering and Science:

Levent Onural
Director of the Graduate School

ABSTRACT

GOOWE: GEOMETRICALLY OPTIMUM AND ONLINE-WEIGHTED ENSEMBLE CLASSIFIER FOR EVOLVING DATA STREAMS

Hamed Rezanejad Asl-Bonab
M.S. in Computer Engineering
Advisor: Fazlı Can
July 2016

Designing adaptive classifiers for an evolving data stream is a challenging task due to its size and dynamically changing nature. Combining individual classifiers in an online setting, the ensemble approach, is one of the well-known solutions. It is possible that a subset of classifiers in the ensemble outperforms others in a time-varying fashion. However, optimum weight assignment for component classifiers is a problem which is not yet fully addressed in online evolving environments. We propose a novel data stream ensemble classifier, called Geometrically Optimum and Online-Weighted Ensemble (GOOWE), which assigns optimum weights to the component classifiers using a sliding window containing the most recent data instances. We map vote scores of individual classifiers and true class labels into a spatial environment. Based on the Euclidean distance between vote scores and ideal-points, and using the linear least squares (LSQ) solution, we present a novel dynamic and online weighting approach. While LSQ is used for batch mode ensemble classifiers, it is the first time that we adapt and use it for online environments by providing a spatial modeling of online ensembles. In order to show the robustness of the proposed algorithm, we use real-world datasets and synthetic data generators using the MOA libraries. We compare our results with 8 state-of-the-art ensemble classifiers in a comprehensive experimental environment. Our experiments show that GOOWE provides improved reactions to different types of concept drift compared to our baselines. The statistical tests indicate a significant improvement in accuracy, with conservative time and memory requirements.

Keywords: Ensemble classifier, concept drift, evolving data stream, dynamic weighting, geometry of voting, least squares, spatial modeling for online ensembles.

ÖZET

GOOWE: DEĞİŞEN VERİ AKIŞLARI İÇİN GEOMETRİK AÇIDAN OPTİMUM AĞIRLIKLI ÇEVRİM İÇİ ÇOKLU SINIFLANDIRICI

Hamed Rezanejad Asl-Bonab
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Danışmanı: Fazlı Can
Temmuz 2016

Değişmekte olan veri akışlarının büyüklüğü ve dinamik yapısı bu ortamlar için hedeflenen uyabilen sınıflandırıcıların tasarımını zorlaştırmaktadır. Veri akışının tasnifinde çevrim içi bireysel sınıflandırıcıların bir topluluk içinde çoklu bir yaklaşımla kullanılması bilinen yöntemlerden biridir. Çoklu sınıflandırıcının bileşenlerinden bir kısmının, zamanla değişen bir biçimde, diğerlerinden daha iyi olması olasıdır. Bileşen sınıflandırıcılara optimum ağırlık atanması tam olarak incelenmiş bir problem değildir. Bu çalışmada, en son veri örneklerini içeren kayan bir pencere kullanarak bileşen sınıflandırıcılara geometrik açıdan optimum ağırlık atayan çevrim içi bir çoklu sınıflandırıcı (GOOWE) yaklaşımı önerilmektedir. Bu amaçla, bileşen sınıflandırıcıların verdikleri oylar ve gerçek sınıf etiketleri uzaydaki noktalarla eşleştirilmektedir. Önerdiğimiz yeni yöntem, en küçük kareler (EKK) çözüm yaklaşımında, bileşenlerin oy puanları ve ideal noktalar arasındaki Öklid mesafesini kullanarak, bileşenlere optimum ağırlık atamaktadır. EKK yaklaşımı yığınlar için tasarlanmış olan çoklu sınıflandırıcılar için daha önceden kullanılmıştır. Çalışmada, bu yaklaşım ilk kez çevrim içi çoklu sınıflandırıcılar için uzaysal bir model yaklaşımıyla kullanılmaktadır. Algoritmanın sağlamlığını göstermek için MOA kütüphanelerinin yanı sıra gerçek ve sentetik veri derlemlerini de kullanan, literatürde önde gelen 8 çoklu sınıflandırıcının sonuçlarını içeren, kapsamlı bir karşılaştırma sunulmaktadır. Deneyler, farklı kavram değişimi gözlenen bilgi akışı ortamlarında, GOOWE ile elde edilen başarının istatistiksel anlamda daha iyi olduğunu göstermektedir.

Anahtar sözcükler: Çoklu sınıflandırıcı, kavram değişimi, değişen veri akışı, dinamik ağırlıklandırma, oylama geometrisi, en küçük kareler, çevrim içi çoklu sınıflandırıcılar için uzaysal modelleme.

Acknowledgement

It is with immense gratitude that I acknowledge the support and help of my supervisor, Dr. Fazlı Can. He continually and convincingly conveyed a spirit of adventure and excitement in regard to research. Without his guidance and persistent help this dissertation would not have been possible. His heartwarming advice, support and friendship has been invaluable on both academic and personal level, for which I cannot find words to express my extreme gratitude.

I would like to thank other jury members for being a part of my thesis examining committee in such a vital transitional stage of my academic life: Dr. Özgür Ulusoy and Dr. Ismail Sengor Altingovde.

I cannot express enough thanks to Dr. Manouchehr Takrimi from Bilkent University, Dr. Jon M. Patton from Miami University of OH, and Alper Can for their valuable comments and contributions on this thesis.

I would also like to acknowledge the financial, academic and technical support of the Computer Engineering Department at Bilkent University. I would like to express my special appreciations and thanks to the faculty and staff of the department, especially Dr. Hakan Ferhatosmanoglu, Dr. Öznur Taştan, and Dr. Selim Aksoy for their kindness, friendship and support, and the respected department chair Dr. Altay Güvenir and administrative assistant Ebru Ateş for their kind helps.

Last but not least, I am indebted to my Mother, Father, Saeed, and Vahid for being such a huge support and encouragement through my experiences. I would not be here without you. To all my friends, thank you for your understanding and encouragement in my life. Your friendship makes my life a wonderful experience. I cannot list all the names here, but you are always on my mind.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Automation of several processes in our daily life has dramatically increased the number of data stream generators. Mining the data generated in real-world applications; like traffic management data, click streams in web exploration, detailed call logs, stock market and business transactions, social and computer network logs, and many other such examples; introduced several challenges to the domain. These challenges are mostly due to the size and time-evolving nature of these data streams. The cost and effort of storing and retrieving this type of data made the on-the-fly real-time analysis of the incoming data extremely crucial [2].

In such dynamically evolving and non-stationary environments, data distribution can change over time, which is referred to as concept drift [1]. However, some of these changes are not real concept drifts, and they do not need to be reacted to by adaptive classifiers. Real concept drift is referred to as change in the conditional distribution of the output, given the input features, while the distribution of the input may stay unchanged [2, 1]. An example of evolving environments is filtering spam emails, in which the definition of the spam class label may change with time. Since users specify these class labels, and their preferences may change with time, the conditional distribution of labels for incoming emails can change [3].

Designing a classifier for time-evolving data streams has some considerations to be addressed, compared to traditional classifiers. Since data arrives continuously, any proposed algorithm needs to process it under strict time constraints. Handling large volumes of data in main memory is impractical, so the proposed algorithm must use limited memory. Patterns of change in target concepts are categorized into sudden/abrupt, incremental, gradual, and reoccurring drifts [4, 1, 5]. Effective classifiers should be able to handle these concept drifts.

More recently, many drift-aware adaptive learning algorithms are developed. Among these algorithms, ensemble methods are naturally more consistent with needs of the problem and they proved to outperform single algorithms statistically and computationally [4, 6, 7, 3, 8]. It is possible that a subset of classifiers in the ensemble outperforms others in a time-varying fashion. However, optimum weight assignment for component classifiers is a problem which is not yet fully addressed in online evolving environments [9]. We propose a novel data stream ensemble classifier which assigns optimum weights to the component classifiers using a sliding window containing the most recent data instances. Since ensemble methods use individual classifiers inside their models, this does not decrease the importance of designing more adaptive individual classifiers for evolving data streams. Improving the performance of individual classifiers in terms of accuracy and resource usage can increase the performance of an ensemble as well.

In this thesis, we concentrate on designing a geometrical framework for dynamic weighting of component classifiers for ensemble methods. We model our ensemble in a spatial environment and use the Euclidean distance as our measure of closeness. We try to find an optimum weighting function based on LSQ, leading to a system of linear equations which describes the ensemble more precisely. Based on this system of linear equations, we design our algorithm called Geometrically Optimum and Online-Weighted Ensemble (GOOWE)—pronounced gooey (/'gü-ē/). It is inspired from the geometry of voting which is a well-known domain in the political and social sciences, and economics. The geometrical analysis of individual votes for their aggregation has proved to outperform the existing solutions in these fields. In aggregation, various rules may have conflicting votes, i.e., "the paradox of voting." Finding classes of profiles, uncovering paradoxes, and

determining the likelihood of disagreements are among the problems addressed by the geometry of voting [10].

In a time-evolving data stream domain, for evaluating the performance of an algorithm it is necessary to use tens of millions of examples [4]. However, gathering this much of real-world data, especially with substantial concept drifts, is not feasible. Another problem is that we cannot find out when a concept drift happens. Because of these problems, like earlier studies, we use a combination of real-world and synthetic data streams in our experiments.

We experimentally evaluate our algorithm using several real-world and synthetic datasets representing gradual, incremental, sudden/abrupt, and reoccurring concept drifts. We use the most popular real-world datasets and for generating synthetic data streams, we use the MOA libraries [4]. For the sake of comparison, we use 8 state-of-the-art ensemble methods as baselines in our experiments. We follow the tradition and use classification accuracy, processing time, and memory costs as our comparison measurements. For classification accuracy measurement, we use the Interleaved Test-Then-Train approach [4].

The summary of main contributions of this study are the following. We

- Provide a spatial modeling for online ensembles and use the linear least squares (LSQ) solution [11] for optimizing the weights of components of an ensemble classifier for evolving environments. While LSQ is used for batch mode component weighting [12, 13], for the first time in the literature, we adapt and use it for online environments, as a stacking algorithm,
- Introduce an ensemble algorithm, called GOOWE. We use data chunks for training and sliding instance window containing the latest available data for testing; such an approach provides more robust behavior as shown by our experiments,
- Conduct an extensive experimental evaluation on 16 synthetic and 4 real-world data streams for comparing GOOWE with 8 state-of-the-art ensemble classifiers, and

Table 1.1: Symbol Notation

Notation	**Definition**
S	Data stream
$I = \{I_1, I_2, ..., I_n\}$	Instance window, $I_i; (1 \leq i \leq n)$
$DC = \{I_1, I_2, ..., I_h\}$	Data chunk, $I_j; (1 \leq j \leq h)$
$I_t = x_t \in S$	Incoming data instance in time t
y_t / y'_t	Vector of true/predicted class label
$C = \{C_1, C_2, ..., C_p\}$	Set of p class labels, $C_k; (1 \leq k \leq p)$
$\xi = \{CS_1, CS_2, ..., CS_m\}$	Ensemble of m individual classifiers, $CS_j; (1 \leq j \leq m)$
$s_{ij} =< S_{ij}^1, S_{ij}^2, ..., S_{ij}^p >$	Score vector for I_i and CS_j, $S_{ij}^k; (1 \leq k \leq p)$
$o_i =< O_i^1, O_i^2, \cdots, O_i^p >$	Ideal-point for I_i, $O_i^k; (1 \leq k \leq p)$
$w =< W_1, W_2, \cdots, W_m >$	Weight vector for ξ, $W_j; (1 \leq j \leq m)$

- Carry out comprehensive statistical tests to show that GOOWE provides a statistically significant improvement in terms of accuracy while using conservative resources.

The rest of this thesis includes a brief chronological survey of the related works in Chapter 2; GOOWE in Chapter 3; our experimental evaluation in Chapter 4; and statistical tests in Chapter 5. Chapter 6 offers a conclusion and directions for future research. Table 1.1 presents the notation of symbols that we use in the succeeding chapters.

Chapter 2

Background and Related Work

In this chapter, we explain our assumptions and specifications for time-evolving data streams. We distinguish different types of concept drifts based on the literature. We discuss different approaches of adapting concept drifts in evolving environments focused on ensemble methods, since they are naturally more capable of handling concept drift and they proved to outperform individual classifiers [1, 4].

2.1 Basic Concepts and Notations

The traditional supervised classification problem aims to map a vector of attributes, x, into a vector of class labels, y', i.e. $x \mapsto y'$. The domain of attribute values in x, can be either numerical or nominal. However, for the domain of class labels in y', we assume binary values for each label indicating selection or not-selection of that specific class label. We compare mapped class label vectors, y', with true class label vectors, y. Instances from our data stream, $I_t = x_t \in S$, appear sequentially in temporal order, and we need to process the data in an online fashion. We map x_t into y'_t and when the true class labels, y_t, are available we can evaluate our predictions. Due to the size of stream data, we only able to

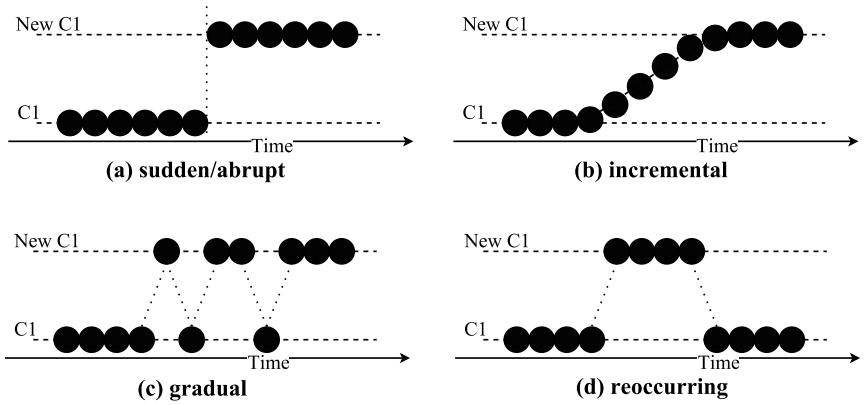


Figure 2.1: Four patterns of real concept drift over time (revised from [1]).

store a finite number of instances to process in a window, and we need to discard old instances. Based on the availability of true class labels (data constraints) and our resources (solution/resource constraints) we can determine the length of the window. Classifiers are supposed to use limited memory and limited processing time per instance [4, 1, 3].

In dynamically evolving environments, the conditional distribution of the output (i.e. true class labels) given the input vector, may change with time, i.e. $P(y_{t+1}|x_{t+1}) \neq P(y_t|x_t)$, while the distribution of the input vector itself, $P(x_t)$, may remain the same [1]. This is referred to as real concept drift and raised several challenges for detecting and reacting to these changes.

Zhang et al. [14] categorized real concept drifts into two scenarios; *Loose Concept Drift (LCD)* where only change in $P(y_t|x_t)$ cause the concept drift, and *Rigorous Concept Drift (RCD)* where change in both $P(y_t|x_t)$ and $P(x_t)$ cause the concept drift. The general assumption in the concept drift setting is that the change happens unexpectedly and is unpredictable. We do not consider the situation for some real-world problems where the change is predictable. We do not address concept-evolution or arrival of a novel class label, and time-constrained classification [15, 16, 17, 18, 19]. The reader is referred to [1] for various settings of the problem. We assume the most general setting of evolving data stream classification problem.

There are several forms of change patterns over time for real concept drift, as shown in Fig. 2.1. If we consider a non-changing conditional distribution of the output given the input as one concept, a drift may happen *suddenly/abruptly* by replacing one concept with another (e.g. C1 with New C1 in Fig. 2.1-(a)) at a moment in time t. Drift may happen *incrementally* between the first and last concepts (e.g. C1 and New C1 in Fig. 2.1-(b), respectively), where there are many intermediate concepts which connects the dots in a smooth way. *Gradual drift* happens when there are no intermediate concepts and both of the first and last concepts are occurring for a period of time, Fig. 2.1-(c). Drifts may introduce new concepts that were not seen before, or previously seen concepts may *reoccur* after some time, Fig. 2.1-(d). Once-off random anomalies or blips are called *outlier/noise* and there should not be any reaction as we do not consider them as a concept drift. Since most of the real-world problems are complex mixtures of these concept drifts, we expect any classifier to react and adapt reasonably to different types of concept drifts and remain robust to outliers and predict with acceptable resource requirements [1].

2.2 Ensemble Classifiers for Evolving Online Environments

A recently published survey by Gama et al. [1], presents a new taxonomy of adaptive classifiers using four existing modules of various learning methods in time-evolving environments. They are *memory management*, *change detection*, *learning property*, and *loss estimation*. In this study, we concentrate on model management strategies, as a learning property, to present state-of-the-art ensemble methods in chronological order. In addition, since we do provide a novel stacking algorithm for online ensemble classifiers, we cover vote combination techniques of these ensembles. The remaining modules, other than learning property, are out of the scope of this study.

Based on the model management categories of Kuncheva [3], there are five

possible strategies for adaptive online classifiers:

1. *Horse Racing:* The dynamic combination ensemble strategy that aims to have the most proper combination rule of existing individual components in an ensemble;

2. *Updated Data Feeding:* Feeding individual classifiers with the most recent available data;

3. *Scheduled Feeding of Ensemble Members:* Scheduling the update of individual classifiers either by retraining in a batch mode or incrementally in an online mode with newly available data;

4. *Add/Drop Classifiers:* Adding fresh classifiers to the ensemble or pruning the deteriorating classifiers; and

5. *Feature Regulation:* Regulating importance of features along with the life of an ensemble.

Practically any combination of these strategies can be used together and they do not need to be necessarily mutually exclusive.

Elwell et al. [20] explains *active* versus *passive* approaches. Active approaches benefit from a drift detection mechanism, reacting only when drift is detected. On the other hand, passive approaches continuously update the model with each incoming data. Since training identical hypotheses with the same data produces identical classifiers, we need some mechanisms to increase their diversity. This is accomplished mostly by Kuncheva's third and fourth strategies. In addition, there are some works to measure and maintain the diversity of component classifiers [21, 22].

The WINNOW [23], Weighted Majority (WM) [24], and Hedge(β) [25], algorithms are the initial adaptive ensemble methods for large-scale changing environments. They mainly use the horse racing strategy for developing better combination rules in an off-line setting. They begin by creating a set of classifiers

Table 2.1: Summary of Related Ensemble Classifiers for Evolving Online Environments

	Spec.		Kuncheva's Strategies				
Ensemble	Study	Type	St. 1	St. 2	St. 3	St. 4	St. 5
WINNOW	[23]	Passive	✓	×	✓	×	×
WM	[24]	Passive	✓	×	✓	×	×
Hedge(β)	[25]	Passive	✓	×	✓	×	×
SEA	[26]	Passive	×	×	✓	✓	×
OzaBag/OzaBoost	[27, 28]	Passive	×	✓	✓	×	×
DWM	[29, 30]	Passive	✓	×	✓	✓	×
AWE	[8]	Passive	✓	×	✓	✓	×
ACE	[31]	Active	✓	✓	✓	×	×
LevBag	[32]	Active	✓	✓	✓	✓	×
Learn++.NSE	[20]	Passive	✓	×	✓	✓	×
AUE2	[6]	Passive	✓	×	✓	✓	×
OAUE	[33]	Passive	✓	✓	✓	✓	×
GOOWE	Current work	Passive	✓	×	✓	✓	×

with an initial weight (usually 1). They adapt the ensemble's behavior using a reward-punishment system to keep track of the most trustworthy expert in each time slot. In particular, WINNOW uses $\alpha > 1$ (usually $\alpha = 2$) for its promotion ($w_i \leftarrow w_i \times \alpha$) and demotion ($w_i \leftarrow w_i \div \alpha$) steps. WM excludes the promotion step and if an expert incorrectly classifies the instance, the algorithm decreases its weight by a multiplicative constant, $\beta \in [0, 1]$. Hedge(β) algorithm operates in the same way but instead of taking the weighted majority vote, chooses one classifier's decision as the ensemble decision. They provide a general framework for weighting component classifiers. However, they do not suggest any mechanism for dynamically adding or removing new components.

Streaming Ensemble Algorithm (SEA) [26], provides a block-based and fixed-size ensemble of classifiers, each trained on the incoming chunk of instances—addressing Kuncheva's fourth model management strategy. If ensemble has space, SEA adds the new classifier to the ensemble, otherwise, it puts the new classifier into the place of a weaker classifier. SEA uses majority vote for predictions in an off-line setting. Due to batch-mode component classifiers which stop learning after being formed and replacing the worst performing classifier in an unweighted

ensemble, the learner was unable to track properly concept drifts in the stream data.

Oza [27, 28], uses Kuncheva's second and third model management strategies together with the traditional bagging and boosting algorithms in online settings for designing OzaBagging and OzaBoosting. For stream data environments, as the number of training examples and component classifiers tend to go to infinity, Oza uses the Poisson distribution with $\lambda = 1$ for approximating the binomial distribution of sampling. A similar idea is used for the OzaBoosting algorithm. It employs incremental values of λ, starting from 1, for training and sampling of classifiers.

Dynamic Weighted Majority (DWM) [29, 30], introduced an ensemble of incremental learning algorithms, each with an associated weight in an online setting. Models are generated by the same learning algorithm on different batches of data. DWM uses the WM approach for assigning weights and makes predictions using a weighted-majority vote of the components where weights are dynamically changing. Pruning components with weights less than a threshold helps to avoid creating an excessive number of components. An extension to DWM, additive expert ensemble (AddExp) [7], provides a general theoretical expert analysis to prove mistakes and loss bounds for a discrete and a continuous ensemble.

Accuracy Weighted Ensemble (AWE) [8], alternatively suggests a general framework for mining changing data streams using weighted ensemble classifiers by re-evaluating ensemble components with incoming data chunks. Inspired by the framework of SEA, a new static learning algorithm is trained and the previous components of ensemble are evaluated on each incoming data chunk. However, these evaluations are done with a special version of Mean Square Error (MSE) allowing the algorithm to select the k best classifiers to create a new ensemble ($MSE_i = \frac{1}{|D|}\sum_{x \in D}(1 - M_c^i(x))^2$; where D is the latest data chunk and $M_c^i(x)$ is the probability score that x belongs to its true class label c, generated by a specific classifier system indexed i). Briefly, it assigns weights to component classifiers based on their expected classification accuracy—according to Bayes error optimization [34]. Moreover, the structure of ensemble is pruned

if errors of individual classifiers are worse than the MSE of a random classifier ($MSE_r = \sum_c P(c) \times (1 - P(c))^2$; where $P(c)$ is the probability of observing class label c). All in all, the weight of classifier i is determined by a linear function ($w_i = MSE_r - MSE_i$).

Since larger data chunks can provide a better distribution of data, they are more capable of building more accurate classifiers but may contain more than one change. Smaller chunks can separate drifting places better, but usually lead to poorer classifiers. In particular, ensembles built on large data chunks may react too slowly to sudden drifts occurring inside the chunk [4, 6]. To overcome this problem, Adaptive Classifier Ensemble (ACE) [31], proposed an algorithm which uses a hybrid of one online classifier and a collection of batch classifiers (a mixture of active and passive approaches) along with a drift detection mechanism. ACE does not benefit from pruning strategies and possibly using of a drift detector leads to poor reactions for gradual drifts.

Bifet [32], introduced Leverage Bagging (LevBag) as an extended version of OzaBagging, using the first four strategies of Kuncheva. It aims to increase the resampling rate using a larger value of λ in the Poisson distribution. Additionally, it adapts output detection codes [35] for handling multi-class problems using only binary classifiers and the ADWIN [36] change detector for dealing better with concept drifts in stream data.

Learn++.NSE [20], is a batch learning ensemble that uses the weighted majority voting. It updates the weights dynamically with respect to the time-adjusted errors of the classifiers on current and past environments. Similar to the AWE model management approach, evaluation of classifiers is considered by giving more credit to the ones capable of identifying previously unknown instances. On the other hand, classifiers that misclassify previously known instances are penalized. Moreover, Learn++.NSE does not discard any component from the ensemble when its knowledge is not relevant to the current chunk of data. Although temporary forgetting model management is particularly useful in cyclical environments, it causes some resource overuse. Ditzler and Polikar extended Learn++.NSE for class imbalanced data stream [37].

Brzezinski et al. [6], proposed Accuracy Updated Ensemble (AUE2), for combining the chunk-based algorithms with incremental learning components. Its model management strategy is based on AWE, while suggesting a non-linear weighting function using the same MSE functions ($w_{ij} = \frac{1}{(MSE_r + MSE_{ij} + \epsilon)}$). The online version of AUE2 [33], called Online Accuracy Updated Ensemble (OAUE), uses a sliding window for the last n instances of the data stream.

A summary of these online ensemble classifiers is provided in Table 2.1. The method we introduced in this work, GOOWE that we present in the next chapter, is also included in the table for comparison. As we can see, GOOWE's model management strategies are the same as AWE and AUE2.

Chapter 3

GOOWE: Geometrically Optimum and Online-Weighted Ensemble

Unlike traditional batch learning, the assumption of independent and identical distribution (i.i.d) of whole stream data is not true for evolving online environments [38]. Possibilities of changes are; "feature changes" or evolving of $p(x)$ with time stamp t, "conditional change" or the changes of class label y assignment to feature vector x, and "dual changes" which includes both [39]. Four recognized patterns of conditional change are given in Fig. 2.1. The same patterns of change are possible for feature changes. As mentioned in section 2.1, Zhang et al. [14] categorized these change into LCD and RCD scenarios. An effective classification algorithm should be able to handle these continuous changes.

3.1 Concepts and Motivation

The data stream is sliced into chunks, each representing single distribution. Almost all state-of-the-art stream classifiers divide the data into fix chunk sizes, as

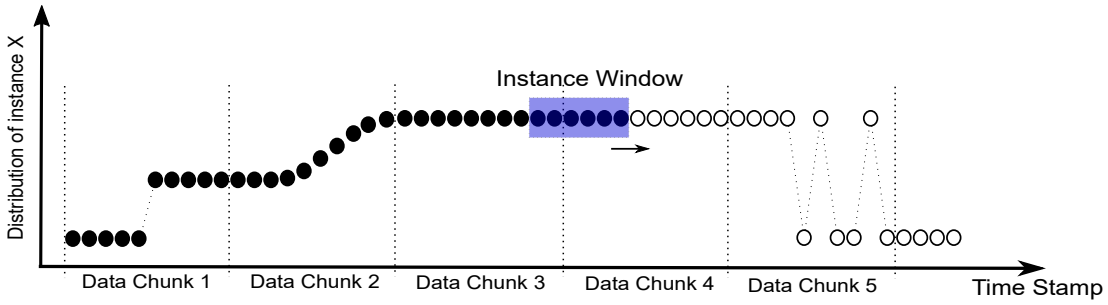


Figure 3.1: Data Chunk (DC) vs. Instance Window (I)—stream data is sliced into equal chunks with size of h and sliding instance window takes the latest n instances with available labels; filled circles are instances with available labels and unfilled circles are yet-to-come instances.

h [40]. There is a recent study for dynamic determination of chunk size according to concept drift speed [40]. This problem is beyond the scope of our study.

Depending on when the labeled training data becomes available Gao et al. [39] categorized stream classifiers into two groups: The first group updates the training distribution as soon as labeled instance becomes available and the second group receives labeled data in chunks and updates the model. Since updating classifiers is a costly operation, the second group of classifiers can be more time efficient. However, these methods perform well when the up-to-date data chunk has identical or similar distributions to the yet-to-come data chunk, which is called stationary assumption in data stream. This assumption ignores the instability nature of evolving data streams when concept drift occurs frequently.

For making our ensemble more efficient we update component classifiers when a new chunk of labeled data is received. Although we do not address the concept drift adaption directly, our extensive experiments show that using a proper component weighting system based on the very recent instances would adapt existing component classifiers for recent concept changes. Consequently, having an optimum weighting function would be extremely beneficial for handling concept drift. For this purpose, we exploit a sliding instance window with latest n labeled instances. The size of instance window can be different with the chunk size, $h \neq n$. Instance window size can be determined by performance and accuracy requirements of the problem. Fig. 3.1 shows this combination usage of data chunk and

instance window.

Inspired from the geometry of voting [10] and using the least squares problem (LSQ) [11], we designed a geometrically optimum and online-weighted ensemble method for evolving environments, called GOOWE. While LSQ is used for component weighting of ensemble classifiers in batch mode [12, 13], it is the first time that we provide a spatial modeling for online environments as a stacking algorithm.

The motivation of this study is to design an ensemble that assigns optimum weights to component classifiers, in an on-line setting with different types of concept drifts. For combining votes, as an stacking algorithm, we model scores of the ensemble' individual classifiers in a spatial environment as vectors and try to establish a clear relationship between a geometric feature of vectors and their effectiveness. Its novelty is based on dynamically changing component optimum weight assignment approach for online ensembles in evolving data streams.

3.2 Design

GOOWE's model management approach is similar to AWE and AUE2 with a passive approach for handling concept drift. Basically, a new incremental learning algorithm is trained on each incoming data chunk and the previous components of the ensemble are re-evaluated on the same data chunk. However, these evaluations are done with a special function of mean square error (MSE) allowing the algorithm to weight component classifiers dynamically, relative to each other, and in an on-line setting.

In training scenario, we use data chunks according to Fig. 3.1 as they become available. When a new data chunk is received, we train a new component classifier using these instances and we add to the ensemble. If there is no space for the new classifier, we substitute it with the worst performing component. For testing the ensemble and classifying a new instance, we use our LSQ-based stacking

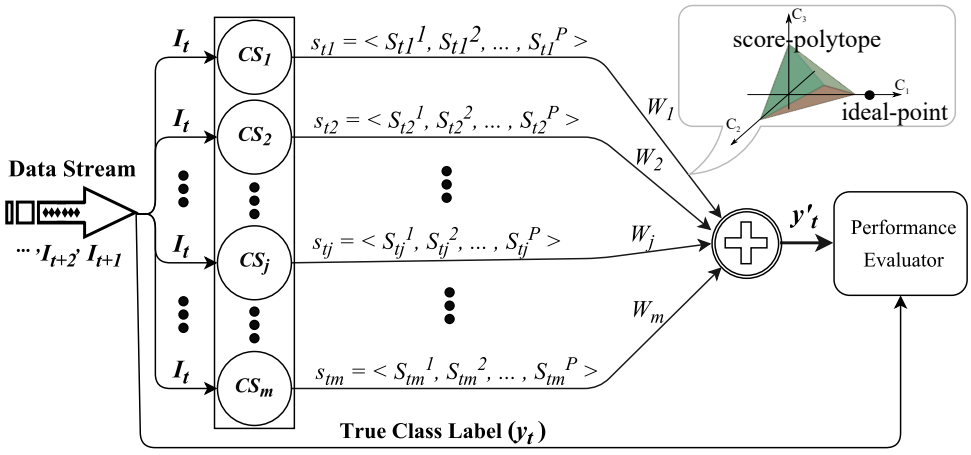


Figure 3.2: General schema of GOOWE; each $I_t \in S$ delivered to $CS_j (1 \leq j \leq m)$, produces relevance score vector, s_{tj}. GOOWE maps these score vectors, as score-polytope, and the true class label, as ideal-point, into a p-dimensional space. It assigns weights, W_j, using the linear least squares (LSQ) solution. Predicted class label, y'_t, is obtained using the weighted majority voting.

algorithm based on the sliding instance window for getting the most updated weights for adapting existing components. Briefly, GOOWE uses a combination of data chunk and instance window, as shown in Fig 3.1. A *data chunk (DC)* has h instances of equally divided data stream, and *instance window (I)* has the latest n instances of data stream, with available true class labels. In our implementation, we build the instance window with the length of $max(n, h)$, and simply add a counter with the maximum value of h into the instance window for providing data chunk. If the length of the instance window is less than the length of data chunk (i.e. $n < h$), we set the length of instance window to h and use the latest n instances of it.

In our geometrical framework, we use the Euclidean norm as the system's loss function for optimization purpose. There are clear statistical, mathematical, and computational advantages to use the Euclidean norm [11]. We calculate weights based on the latest n instances in our window, and for making prediction we use weighted majority voting approach.

As shown in Fig. 3.2, we have an ensemble of component classifiers $\xi =$

$\{CS_1, CS_2, \cdots, CS_m\}$. Each component classifier, $CS_j (1 \leq j \leq m)$, processes instance I_t of evolving data stream, S, and produces relevance scores, $s_j =< S_j^1, S_j^2, \cdots, S_j^p >$, for each of class labels, $C = \{C_1, C_2, \cdots, C_p\}$. Since each classifier produces relevance scores in different ranges, we use Eq. 3.1 for normalizing the scores into the range of [0, 1].

$$S_j^k \leftarrow \frac{S_j^k}{\sum_{a=1}^{p} S_j^a} \quad (1 \leq k \leq p) \tag{3.1}$$

Assuming each class label as one dimension, enables us to map each component's score $(s_j; 1 \leq j \leq m)$ into a point in a p-dimensional Euclidean space. Mapping all score points of I_t in the same way, builds a polytope in a p-dimensional Euclidean space, which we call the *score-polytope* of I_t. We define *score-vector* by using the origin point as the starting point and score point as the terminal point in our spatial environment. Using the vector of the true class label for I_t as y_t, we can assume an *ideal-point* in the p-dimensional space as $o =< O^1, O^2, \cdots, O^p >$. For example, if the number of class labels is 4, and the true class label of I_t is C_2, then the ideal-point would be $o =< 0, 1, 0, 0 >$.

3.3 Optimum Weight Assignment

For making prediction, we use n latest instances $I = \{I_1, I_2, \cdots, I_n\}$, as an instance window, where I_n is the latest instance and all true class labels are available. For each instance $I_i (1 \leq i \leq n)$, each component classifier $CS_j (1 \leq j \leq m)$ has a score-vector as $s_{ij} =< S_{ij}^1, S_{ij}^2, \cdots, S_{ij}^p >$. For the true class label of I_i we have $o_i =< O_i^1, O_i^2, \cdots, O_i^p >$ as the ideal-point. We aim to find optimum weight vector $w =< W_1, W_2, \cdots, W_m >$ to minimize the distance between score-polytope and ideal-point. Using the squared Euclidean norm as our measure of closeness for the linear least squares problem (LSQ) results

$$\min_w ||o - Sw||_2^2 \tag{3.2}$$

The corresponding residual vector is $r = o - Sw$, where for each instance I_i, $S \in \mathbb{R}^{m \times p}$ is the matrix with relevance scores s_{ij} in each row, w is the vector of weights to be determined, and o is the vector of ideal-point [11]. Since we have n instances in our window, we use the following function for our optimization solution.

$$f(W_1, W_2, \cdots, W_m) = \sum_{i=1}^{n} \sum_{k=1}^{p} (\sum_{j=1}^{m} (W_j S_{ij}^k) - O_i^k)^2 \tag{3.3}$$

Taking a partial derivation over $W_q (1 \leq q \leq m)$ and finding optimum points will give us our weight vector. The gradient equations become

$$\frac{\partial f}{\partial W_q} = \sum_{i=1}^{n} \sum_{k=1}^{p} 2(\sum_{j=1}^{m} (W_j S_{ij}^k) - O_i^k) S_{iq}^k, \quad (1 \leq q \leq m) \tag{3.4}$$

Setting the gradient to zero, $\nabla f = 0$

$$\sum_{j=1}^{m} W_j (\sum_{i=1}^{n} \sum_{k=1}^{p} S_{iq}^k S_{ij}^k) = \sum_{i=1}^{n} \sum_{k=1}^{p} O_i^k S_{iq}^k, \quad (1 \leq q \leq m) \tag{3.5}$$

and assuming below summations as a_{qj} and d_q

$$a_{qj} = \sum_{i=1}^{n} \sum_{k=1}^{p} S_{iq}^k S_{ij}^k, \quad (1 \leq q, j \leq m) \tag{3.6}$$

$$d_q = \sum_{i=1}^{n} \sum_{k=1}^{p} O_i^k S_{iq}^k, \quad (1 \leq q \leq m) \tag{3.7}$$

lead to m linear equations with m variables (weights). The proper weights in the following matrix equation are our intended optimum weight vector. We present weight assignment equation in matrix representation to make the later example easier to follow.

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mm} \end{bmatrix} \times \begin{bmatrix} W_1 \\ W_2 \\ \vdots \\ W_m \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_m \end{bmatrix} \tag{3.8}$$

Briefly, $Aw = d$, where A is the coefficients matrix and d is the remainders vector. According to Eq. 3.6, A is a symmetric square matrix. In the sense of

least squares solution [11], since it is probable that A is rank-deficient, we may not have a unique solution and we denote the minimizer by w^*. According to theorem 9 of [11], the *normal equations* for w^* can be written as

$$A^T A w = A^T d \tag{3.9}$$

In this equation, $A^T A$, is also a symmetric square matrix. In addition, if A has full rank, $A^T A$ is positive definite and our problem has a unique solution. In the rank-deficient case, it is non-negative definite and we have a set of possible weight vectors. The QR factorization suggests less expensive solutions for both full rank and rank-deficient cases [11]. In such cases, the weights are near optimum which.

Since we predict scores for each incoming instance separately, we define A_i and $d_i (1 \leq i \leq n)$ according to equations 6 and 7. Matrix A and vector d can be calculated simply by adding all A_i and d_i for all instances of a given window, respectively.

$$a_{qj}^i = \sum_{k=1}^{p} S_{iq}^k S_{ij}^k, \quad (1 \leq i \leq n) \tag{3.10}$$

$$d_q^i = \sum_{k=1}^{p} O_i^k S_{iq}^k, \quad (1 \leq i \leq n) \tag{3.11}$$

Using the weighted majority vote approach gives the aggregated score vector. Since we calculate scores in a spatial environment, it is possible that these score values become negative. Using the following normalization in advance to Eq. 3.1 gives the proper aggregated score vector.

$$S_k \leftarrow \frac{S_k - min(S_k)}{max(S_k) - min(S_k)}, \quad (1 \leq k \leq p) \tag{3.12}$$

3.4 Example of Assigning Optimal Weights for Component Classifiers

Suppose that we have 2 classifiers and 2 class labels, as shown in Fig. 3.3. Our instance window has 2 instances as I_1 and I_2. We want to find optimum weight vector for aggregating scores for a newly arrived instance as I_t.

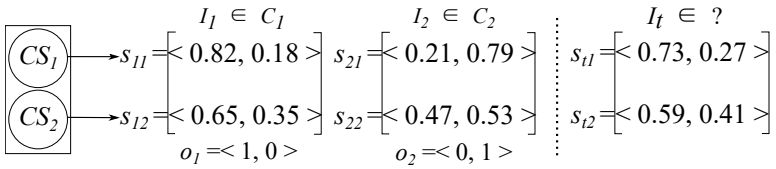
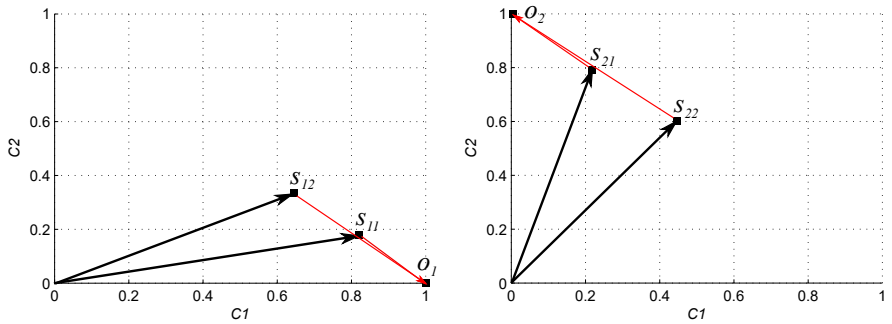


Figure 3.3: An example of GOOWE component classifiers weighting.

We have a 2-dimensional Euclidean space, as shown in Fig. 3.4. Score vectors and their intended projections are illustrated with black and red lines, respectively.



(a) I_1 (b) I_2

Figure 3.4: Score vectors for window instances of example.

Putting the values into Equation 3.6 and 3.7, gives the following matrix equation.

$$\begin{bmatrix} 1.37 & 1.11 \\ 1.11 & 1.05 \end{bmatrix} \times \begin{bmatrix} W_1 \\ W_2 \end{bmatrix} = \begin{bmatrix} 1.61 \\ 1.18 \end{bmatrix}$$

Solving this equation gives the intended weight vector, $w =< 1.88, -0.87 >$. Multiplying these weights with the score vectors of the components, results in the aggregated score vector, $s =< 0.86, 0.14 >$. We have much stronger vote compared to each individual classifier.

3.5 Pseudocode of GOOWE Algorithm

It is given in Algorithm 1. In training scenario (lines 9-23), having the proper number of instances from each class label, as our training data, is crucial for more accurate individual classifiers. On the other hand, for testing scenario (lines 3-8), static weighting component classifiers can result in relatively poor aggregated predictions, especially with existence of frequent concept drifts in data stream. Using a combination of data chunk and instance window enables us to think about training and testing of our algorithm separately. These two values can be adjusted according to the drift rate of data stream.

When the number of instances in data chunk, DC, reaches its maximum value (line 9), GOOWE trains a new incremental classifier (line 10). If the ensemble has its maximum number of classifiers, m, then GOOWE calculates weights of classifiers using Eq. 3.8 and instances in data chunk (lines 12-16). The more the obtained weight value is close to zero, the more we want to cancel its effectiveness in our aggregated score vector. As a result, we take the absolute value of weight values and omit the classifier with the least weight (line 17). We first incrementally update all the existing classifiers with DC (lines 19-21), and then add a fresh classifier into the ensemble (line 22). Most of the incrementally updated classifiers, needs to be pruned after some updates. Since we have memory constraints in our problem, we prune these classifiers when the consumed memory exceeds the memory limit (lines 24-26). For example, in our experiments we use the Hoeffding tree [41] and prune the least active leaves of the tree to satisfy the user specified memory constraint.

For each incoming instance, for making prediction, GOOWE calculates weights

Algorithm 1: GOOWE (Geometrically Optimum and Online-Weighted Ensemble)

Require: S: data stream, I: window of n latest instances, DC: latest data chunk with length of h, m: maximum number of classifiers, CS: single classifier system, p: number of class labels, L: memory limit.
Ensure: ξ: set of weighted classifiers, s_T: aggregated score vector.

$\xi \leftarrow \emptyset$;
for all instances $I_t \in S$ **do**
  for all instances $I_i \in I$ **do**
    $A \leftarrow A + A_i$; {using Eq. 3.10}
    $d \leftarrow d + d_i$; {using Eq. 3.11}
  end for
  $w \leftarrow solve(Aw = d)$; {see Eq. 3.8}
  $s_T \leftarrow \sum_{j=1}^{m}(W_j s_j)$; {weighted majority vote}
  if *DC has h instances* **then**
    $CS' \leftarrow$ new single classifier built on DC;
    if ξ *has m classifiers* **then**
      for all instances $I_i \in DC$ **do**
        $A \leftarrow A + A_i$; {using Eq. 3.10}
        $d \leftarrow d + d_i$; {using Eq. 3.11}
      end for
      $w \leftarrow solve(Aw = d)$; {see Eq. 3.8}
      $\xi \leftarrow \xi \backslash$ {classifier with $min(|W_j|)$; $1 \leq j \leq m$};
    end if
    for all $CS_j \in \xi$ **do**
      train CS_j with DC;
    end for
    $\xi \leftarrow \xi \cup \{CS'\}$;
  end if
  if $memory\_usage(\xi) \geq L$ **then**
    prune all component classifiers;
  end if
end for

of classifiers using Eq. 3.8 and instances in instance window (lines 3-7). It multiplies resulted weights with score vectors and using the weighted majority voting approach, calculates the aggregated score vector. Adjusting the length of instance window and data chunk depends on the data stream and types of concept drift. There is no general solution to this problem. However, setting relatively small numbers to the instance window and relatively large numbers to the data chunk, according to available resources, can result in better accuracy. Experimental evaluation, presented in the next two sections, illustrate that GOOWE can react statistically significantly better compared to its state-of-the-art rivals.

Chapter 4

Experimental Evaluation

The main concern of evolving data stream classifiers is having more accurate predictions with less processing time and memory consumptions. In the following sections, we present the experimental results for different simulation scenarios conducted to evaluate our proposed ensemble method. We describe the datasets, discuss the experimental setup, and finally analyze the simulation results. For the sake of comparison, we include 8 state-of-the-art adaptive ensemble methods proposed for evolving data streams. We did not include single classifiers in our experiments because the comparison between online ensemble methods and single classifiers is well studied [6]. Instead, we chose to evaluate the latest ensemble classifiers on evolving data streams specifically to see the difference in performance more clearly. In this evaluation, we use the Massive Online Analysis (MOA)[1] framework [42]. MOA is an open-source software package to run data streaming experiments and, to the best of our knowledge, is the most popular framework for data stream mining. We use JAva MAtrix (JAMA)[2] package, a basic linear algebra library, for matrix operations and least squares solutions in our implementation of GOOWE.

[1]MOA webpage: http://moa.cms.waikato.ac.nz/
[2]JAMA webpage: http://math.nist.gov/javanumerics/jama/

4.1 Datasets as Data Streams

Selecting proper time-evolving data stream is one of the vital steps for comparing different algorithms. There are two types of data stream sets; synthetic and real-world datasets. We have the whole dataset before the experiment and use the terms dataset and data stream equivalently. Similar to the other domains of prediction algorithms, real-world datasets are the best. However, the problem with them is that we do not know when drift occurs or if there is any drift at all. Some studies use real-world datasets with artificial concept drifts, called real-world data with forced/synthetic concept drift [1]. These datasets cannot be considered as real examples of drifts. Synthetic data has several benefits like easy to reproduce, low cost of storage and transmission, but most importantly, it provides an advantage of knowing where exactly drift has happened [4, 1].

A proposed algorithm should be capable of handling large data streams—potentially an infinite number of instances [4]. As a result, for comparisons of several algorithms, we need to have large datasets in the order of tens of millions of instances. Similar to common approaches [4, 6, 33, 26], in order to cover all patterns of changes over time; sudden/abrupt, incremental, gradual, and reoccurring as concept drifts including blips or noise; we use synthetic data stream generators, implemented in the MOA framework. Using these generators, we prepared 16 synthetic datasets. In addition, we have 4 widely used real-world data streams.

Following are a brief description of each dataset including their generation and preparation. Table 4.1 summarizes the specifications of each dataset. We report the average of accuracy, processing time, and maximum memory consumption for each dataset in Table 4.2, 4.3, and 4.4, respectively.

4.1.1 Synthetic Datasets

According to the concept drifting scenarios of Zhang et al. [14], we have 8 Rigorous Concept Drifting (RCD) and 8 Loose Concept Drifting (LCD) synthetic datasets. Bifet et al. [4] specified Random RBF generator as the RCD data stream and the rest of synthetic data stream generators as the LCD data stream.

Random RBF. It assigns a fixed number of random positioned centroids, with a random standard deviation value, class label and weight. For generating new instances, we randomly select a center, considering weights, so that centroids with higher weights are more likely to be chosen. A random direction is chosen for displacement, using a Gaussian distribution, and drift is defined by moving the centroids, with constant speed. Attributes are all numerical values. Using this generator we prepared 8 different datasets, each containing 1 million instances with 20 attributes and 0 percent noise. Here are 3 important alternate factors we changed among these 8 datasets. We reflect these, respectively, in the naming of RBF datasets in Table 4.1.

- *Concept Drift Type (Gradual: G and Abrupt: A).* The way generator moves centroids make the data stream gradually changing. We add some outliers during generations of gradual changing datasets in order to have blips. We generate abruptly changing data streams using the sigmoid join operator ($c = a \oplus_{t_0}^{W} b$; t_0: point of change, W: length of change) [4].

- *Number of Classes (Four: 4 and Ten: 10).* The ability for generating arbitrary number of classes is useful for evaluating an algorithm. We generate our datasets with either four or ten class labels.

- *Drift Frequency (Slow: S and Fast: F).* For gradually changing datasets, we generate instances with 0.01 (fast) and 0.0001 (slow) concept changing speed. For abruptly changing datasets, we switch to a new random stream 10 (slow) or 100 (fast) times evenly distributed over 1 million instances.

SEA Concepts. It involves 3 numerical attributes varying between 0 and 10

Table 4.1: Summary of Dataset Characteristics

Dataset	#Instance	#Att	#CL	%N	Drift Spec.
RBF-G-4-S	1×10^6	20	4	0	Gr., Bp., DS=0.0001
RBF-G-4-F	1×10^6	20	4	0	Gr., Bp., DS=0.01
RBF-G-10-S	1×10^6	20	10	0	Gr., Bp., DS=0.0001
RBF-G-10-F	1×10^6	20	10	0	Gr., Bp., DS=0.01
RBF-A-4-S	1×10^6	20	4	0	Abrupt, #D=10
RBF-A-4-F	1×10^6	20	4	0	Abrupt, #D=100
RBF-A-10-S	1×10^6	20	10	0	Abrupt, #D=10
RBF-A-10-F	1×10^6	20	10	0	Abrupt, #D=100
SEA-S	1×10^6	3	2	10	Abrupt, #D=3
SEA-F	2×10^6	3	2	10	Abrupt, #D=9
HYP-S	1×10^6	10	2	5	Incrm., DS=0.001
HYP-F	1×10^6	10	2	5	Incrm., DS=0.1
TREE-S	1×10^6	10	4	0	Reoc., #D=4
TREE-F	1×10^5	10	6	0	Reoc., #D=15
LED-M	1×10^6	24	10	10	Mixed, #D=3
LED-ND	1×10^7	24	10	20	No drift
CoverType	581,012	54	7	-	Unknown
PokerHand	1×10^7	10	10	-	Unknown
CovPokElec	1,455,525	72	10	-	Unknown
Airlines	539,383	7	2	-	Unknown

#CL: No. of Class Labels, %N: Percentage of Noise, DS: Drift Speed, #D: No. of Drifts, Gr.: Gradual, Bp.: Blips.

[26]. In our experiment, we use this generator in 2 different settings, both with 10 percent noise. First, 1 million instances, with drifts occurring every 250,000 examples (slow: SEA-S), and second, 2 million instances with drifts occurring every 200,000 examples (fast: SEA-F) are generated.

Rotating Hyperplane. It assigns points in a multi-dimensional hyperplane and classifies them positively and negatively. Concept drift is defined by changing the orientation and position of the hyperplane [43]. We set the hyperplane generator to create 2 datasets, each with 1 million instances described by 10 numerical features. We add 5 percent class noise to both of them. The modification weight of slowly changing dataset (HYP-S) is set to $w_i = 0.001$, and for the rapidly changing one (HYP-F) to $w_i = 0.1$.

Random Tree. It produces nominal and numerical attributes using a randomly constructed tree. Drift is defined by abruptly changing the tree after a given number of examples [42]. For both slow and fast tree datasets, we set the generator to have 5 nominal and 5 numerical attributes. Slowly changing dataset (TREE-S) consists of 1 million instances with 4 reoccurring drifts evenly distributed. Rapidly changing dataset (TREE-F) contains 100,000 instances with 15 sudden drifts; it is the fastest changing dataset of our experiments.

LED. It tries to predict the digit displayed on a seven-segment LED display. Each instance has 24 binary attributes and each has a possibility of being inverted, which is defined as noise. We have 2 LED datasets. The first dataset, LED-M, has 1 million instances with 2 gradually drifting concepts abruptly switching after 0.5 million instances and 10 percent of noise. The second one, LED-ND, has 10 millions of instances without any drift and 20 percent of noise, makes it noisiest and largest dataset among others [6].

4.1.2 Real-World Datasets

The noise values, number of drifts, and drift speeds are unknown for these datasets. Access URL links are given in the footnote.

CoverType.[3] It contains the forest cover type from the US Forest Service (USFS), comprised of 581,012 instances and 54 attributes.

PokerHand.[4] It consists of 1 million instances and 10 attributes. Each record is a hand of 5 playing cards—2 attributes as suit and rank.

CovPokElec.[5] It combines the normalized CoverType, normalized PokerHand, and Electricity datasets using the sigmoid join operator. The Electricity dataset comes from the Australian New South Wales Electricity Market. CovPokElec is obtained by merging all attributes, and assuming that each dataset corresponds to a different concept [4].

Airlines.[6] It consists of 539,383 examples described by 7 attributes. The task is to predict whether a given flight will be delayed or not, given the information of the scheduled departure.

4.2 Experimental Design

In this study, we evaluate our method by comparing it with 8 well-known ensemble classifiers for non-stationary environments using the online block-based, bagging, and boosting methods. We select Accuracy Weighted Ensemble (AWE) [8], improved Accuracy Updated Ensemble (AUE2) [6], Dynamic Weighted Ensemble (DWM) [30] and Learn++.NSE (NSE) [20] ensemble methods from block-based approaches. In addition to these, we include Online Accuracy Updated Ensemble (OAUE) [33], Online Bagging (OzaBag) [28], Online Boosting (OzaBoost) [28] and Leverage Bagging (LevBag) [32] ensemble methods as popular online ensembles. All the algorithms were programmed in Java as part of the MOA framework that we extended to implement GOOWE. We used the MOA extensions library

[3]Access link: http://archive.ics.uci.edu/ml/datasets/Covertype
[4]Access link: http://archive.ics.uci.edu/ml/datasets/Poker+Hand
[5]Access link: http://www.openml.org/d/149
[6]Access link: http://moa.cms.waikato.ac.nz/datasets/

Table 4.2: Average Classification Accuracy in Percentage (%) —for each synthetic dataset a one-way ANOVA using Scheffe multiple comparisons statistical test is conducted and the top tier algorithms are underlined; for real-world datasets the most accurate algorithms are underlined

Dataset	DWM	NSE	AWE	AUE2	GOOWE	OAUE	OzaBag	LevBag	OzaBoost
RBF-G-4-S	75.157	72.355	75.329	91.174	92.014	91.817	87.084	85.779	88.353
RBF-G-4-F	74.102	72.041	73.837	94.250	94.590	93.322	87.213	85.947	87.995
RBF-G-10-S	79.549	77.365	81.326	83.102	92.298	83.059	80.901	80.671	76.951
RBF-G-10-F	79.669	78.455	80.875	83.055	92.189	82.726	80.748	80.256	76.459
RBF-A-4-S	76.628	73.308	78.046	96.543	96.901	96.267	95.618	95.676	97.367
RBF-A-4-F	75.452	72.519	77.591	96.779	97.019	95.867	95.461	95.988	96.258
RBF-A-10-S	81.297	79.446	84.832	91.943	96.477	85.771	95.017	94.901	95.136
RBF-A-10-F	82.338	80.471	85.657	92.592	96.730	88.473	95.504	95.480	95.923
SEA-S	86.030	86.847	87.897	89.718	89.031	89.749	89.628	89.633	89.360
SEA-F	86.084	86.849	87.923	89.812	89.637	89.831	89.739	89.742	89.551
HYP-S	86.819	87.175	90.483	88.486	88.891	89.044	83.467	89.222	86.306
HYP-F	90.734	88.714	90.994	92.564	92.567	92.748	82.032	92.148	89.495
TREE-S	32.921	24.638	33.926	35.222	35.932	36.286	37.135	37.124	24.639
TREE-F	28.870	09.512	30.154	31.858	33.827	32.024	32.217	32.217	9.518
LED-M	65.880	70.354	74.002	73.975	73.989	73.973	73.984	74.008	73.778
LED-ND	43.336	46.849	51.210	51.196	51.191	51.208	51.194	51.212	51.034
CoverType	86.266	79.437	80.600	84.951	89.779	88.205	84.264	84.080	90.570
PokerHand	47.591	49.550	49.470	50.217	54.604	50.031	52.995	52.995	53.681
CovPokElec	85.377	65.249	66.330	73.668	88.718	86.390	82.265	77.647	87.154
Airlines	61.206	60.797	60.650	61.395	61.834	62.516	61.404	62.164	61.015

for DWM and NSE. In addition, our implementation of GOOWE and some detailed information about experimental evaluation, such as standard deviations, and dataset generations are available on our webpage[7]. The experiments were performed on a machine equipped with an Intel Xeon E3-1200 v3@ 3.40 GHz processor and 32 GB of ECC RAM.

Due to having equivalent configurations of different methods we fixed some common settings. First, we set the maximum number of classifiers to 10. Increasing or decreasing this value, based on the observations in [6], has a linear effect on the accuracy, memory consumption and processing time. With 10 component classifiers for ensemble, we can see how our proposed weighting works compared to existing ones more clearly. In addition, we use the Hoeffding trees [41] as the base classifier components for all methods. We used the Hoeffding trees enhanced with adaptive Naive Bayes leaf predictions, with a grace period $n_{min} = 100$, split confidence $\delta = 0.01$, and tie-threshold $\tau = 0.05$ similar to experiments in [6, 33, 41].

In our experiments, according to the chunk size analysis of [8] and similar to the experimental evaluations of [6], the chunk size for block-based ensembles (namely DWM, NSE, AWE, AUE2, GOOWE and OAUE) is set to 500 instances. OAUE and GOOWE use a sliding window of recent data instances. To ensure a fair comparison, similar to block-based ensembles, we set instance window length as 500. Although this length can be smaller for most of the ensembles, to perform an equivalent comparison, we choose this value based on the suggested minimum chunk length of AWE [8]. The data chunk size and instance window size analysis is possible as a future work.

By considering the main requirements of data stream environments [4, 6, 26] in our experimental setup, we chose interleaved Test-Then-Train procedure for measuring prediction accuracy values. For synthetic datasets, our initial experiments showed that for the exact same settings of generators, accuracy values showed some variations. In order to have confident conclusions, for each synthetic dataset, we generate 10 time-seeded random data streams. For example,

[7]GOOWE webpage: http://www.cs.bilkent.edu.tr/∼canf/goowe/

when we say that RBF-G-4-F dataset has 1 million of instances, we examine 10 such datasets (i.e. total of 10 millions of instances).

4.3 Comparative Evaluation

We measured the class label prediction accuracy (in percentage), maximum memory usage (in MegaByte), and total processing time of every one thousand instances (in CentiSecond) for each of the ensemble algorithms—average values for synthetic datasets and exact values for real-world datasets reported in Table 4.2, 4.3, and 4.4, respectively. For each synthetic dataset, a one-way analysis of variance (ANOVA) using Scheffe multiple comparisons [44] are conducted and the best performing algorithms underlined. It is not possible to conduct the Scheffe statistical test for real-world datasets, since they only have a single value. For each of them, we underline the most accurate and least resource consumer algorithm. We do not report the standard deviation of the average values for synthetic datasets, due to their less importance and limited space.

We draw scatter diagrams of the algorithms on the arrival of new chunks of data stream as in [4, 20, 6]. Due to limited space, we provide only one plot of accuracy and memory behaviors for each category of RCD, LCD, and real-world datasets. For better understanding the behavior of ensembles in these situations, we present accuracy and memory plots for each gradual changing RCD and abrupt changing RCD datasets, separately. We provide these plots in Fig. 4.1, 4.2, 4.3, and 4.4—note that plots are in different scales.

4.3.1 RCD Data Streams with Gradual/Abrupt Drift Patterns

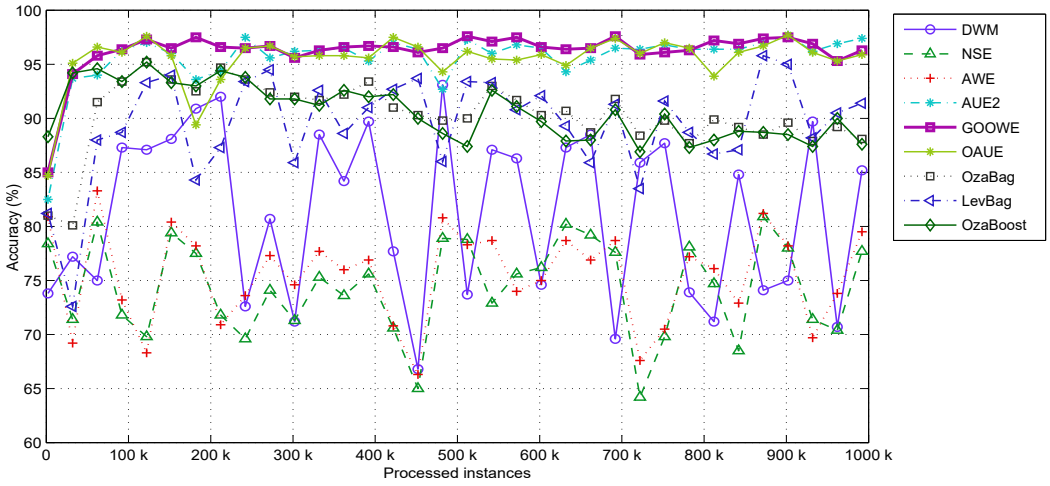
Table 4.2 for Random RBF data streams (the first 8 rows) shows the superiority of GOOWE over other algorithms, in terms of accuracy. Its superiority is more significant for the gradually changing data streams with respect to the

Table 4.3: Average Processing Time in CentiSecond (CS), for processing every one thousand instances —for each synthetic dataset a one-way ANOVA using Scheffe multiple comparisons statistical test is conducted and the top tier algorithms are underlined; for real-world datasets the least time-consumer algorithms are underlined

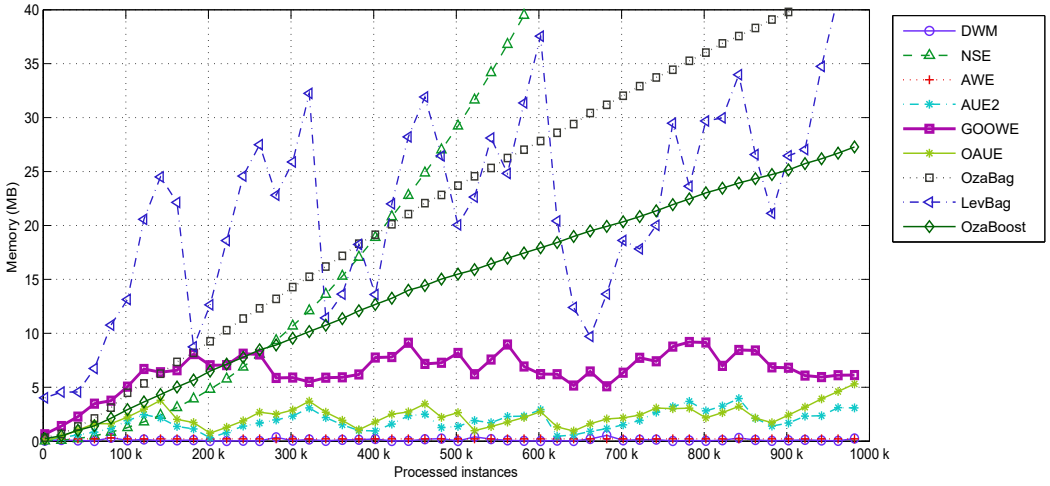
Dataset	DWM	NSE	AWE	AUE2	GOOWE	OAUE	OzaBag	LevBag	OzaBoost
RBF-G-4-S	6.718	439.071	18.458	20.563	14.887	17.482	8.299	17.825	7.231
RBF-G-4-F	6.628	444.261	21.930	21.039	14.400	17.986	11.234	18.801	14.403
RBF-G-10-S	31.854	1085.484	41.223	45.602	31.569	37.233	17.146	38.402	14.871
RBF-G-10-F	30.501	1124.648	49.549	47.105	31.294	37.599	23.036	38.789	28.333
RBF-A-4-S	6.598	443.676	21.900	17.218	13.784	15.367	9.901	16.425	12.124
RBF-A-4-F	6.621	445.520	21.913	17.037	13.806	15.376	10.007	16.381	11.921
RBF-A-10-S	30.027	1125.125	49.025	44.556	31.202	36.867	21.545	36.214	27.433
RBF-A-10-F	29.693	1119.683	48.678	43.537	30.718	36.231	20.482	35.397	26.59
SEA-S	0.615	43.714	2.752	2.772	2.428	2.760	1.509	2.840	1.808
SEA-F	0.616	94.267	2.740	3.065	2.452	2.958	1.684	3.102	1.963
HYP-S	1.504	115.595	7.390	6.617	6.321	6.547	3.955	5.103	4.241
HYP-F	1.443	79.532	7.534	5.680	5.707	5.877	3.580	4.290	3.873
TREE-S	6.337	85.346	11.468	9.479	9.333	9.528	7.217	8.777	7.009
TREE-F	8.193	14.276	13.639	11.026	10.518	11.036	7.641	9.478	7.295
LED-M	35.260	1288.899	57.355	54.824	45.966	40.135	13.874	25.615	14.764
LED-ND	38.603	1280.011	58.002	54.971	45.615	42.924	16.068	23.699	17.102
CoverType	11.526	335.569	28.269	24.409	21.858	21.952	11.721	18.868	14.359
PokerHand	8.578	68.910	5.184	7.123	6.842	8.816	5.634	8.012	6.148
CovPokElec	15.483	546.412	25.803	24.005	24.076	24.832	13.786	18.814	17.804
Airlines	0.982	11.304	2.692	2.710	3.338	3.146	2.086	2.673	2.676

Table 4.4: Maximum Memory Usage in MegaByte (MB) —for each synthetic dataset a one-way ANOVA using Scheffe multiple comparisons statistical test is conducted and the top tier algorithms are underlined; for real-world datasets the least memory-consumer algorithms are underlined

Dataset	DWM	NSE	AWE	AUE2	GOOWE	OAUE	OzaBag	LevBag	OzaBoost
RBF-G-4-S	0.261	116.193	0.320	0.483	1.632	0.467	13.379	15.221	13.232
RBF-G-4-F	0.294	116.193	0.319	2.885	9.750	2.711	35.258	39.568	24.517
RBF-G-10-S	0.333	116.201	0.394	3.470	11.066	3.358	18.063	20.293	8.085
RBF-G-10-F	0.691	116.200	0.394	5.451	11.837	5.408	16.877	17.382	13.343
RBF-A-4-S	0.931	116.192	0.319	4.535	15.657	4.687	27.618	42.854	30.531
RBF-A-4-F	0.553	116.192	0.319	5.004	12.107	5.026	26.646	40.845	28.919
RBF-A-10-S	0.569	116.200	0.395	4.453	11.320	4.194	19.184	35.068	16.292
RBF-A-10-F	0.842	116.200	0.394	4.682	13.784	5.334	17.168	34.660	15.368
SEA-S	0.181	116.109	0.229	18.974	3.975	21.146	7.046	9.009	6.861
SEA-F	0.204	464.351	0.230	36.584	6.332	42.023	13.826	15.858	13.629
HYP-S	0.384	116.145	0.614	2.376	4.849	2.636	22.918	25.218	21.19
HYP-F	0.505	116.147	0.643	1.063	2.999	1.863	20.912	22.882	21.097
TREE-S	0.172	116.137	0.210	1.426	5.931	7.343	6.288	9.265	6.343
TREE-F	0.181	1.251	0.224	0.681	3.118	0.762	0.538	0.564	0.606
LED-M	0.788	116.222	0.453	0.453	1.052	0.415	13.707	18.715	13.927
LED-ND	0.581	116.221	0.453	0.453	2.871	0.454	12.891	20.912	12.863
CoverType	1.252	39.376	0.437	0.736	1.128	1.079	55.631	71.475	62.185
PokerHand	0.186	116.146	0.224	0.254	0.924	0.229	3.460	4.390	3.524
CovPokElec	4.534	246.144	0.576	11.543	54.435	36.529	132.399	162.592	153.337
Airlines	0.131	33.825	0.157	0.300	2.507	0.535	7.592	10.322	7.773

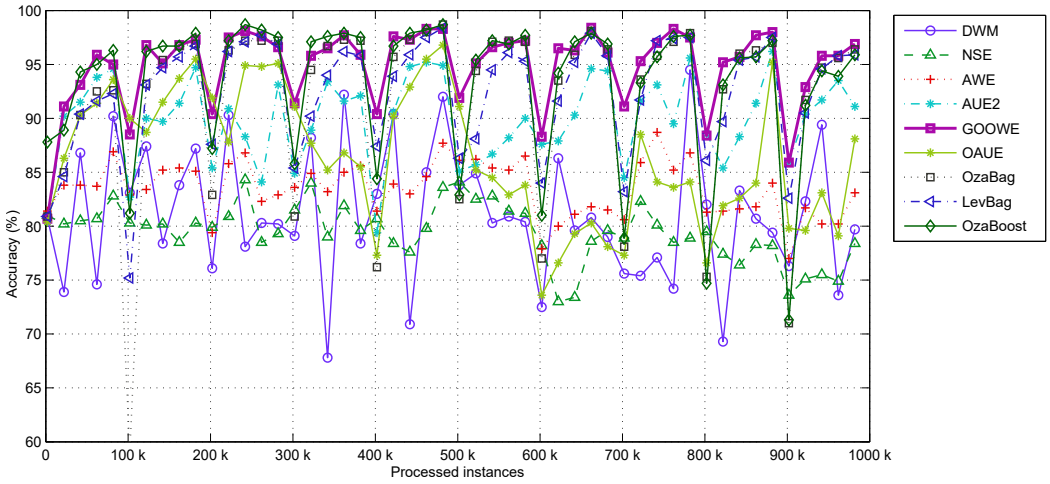


(a) accuracy

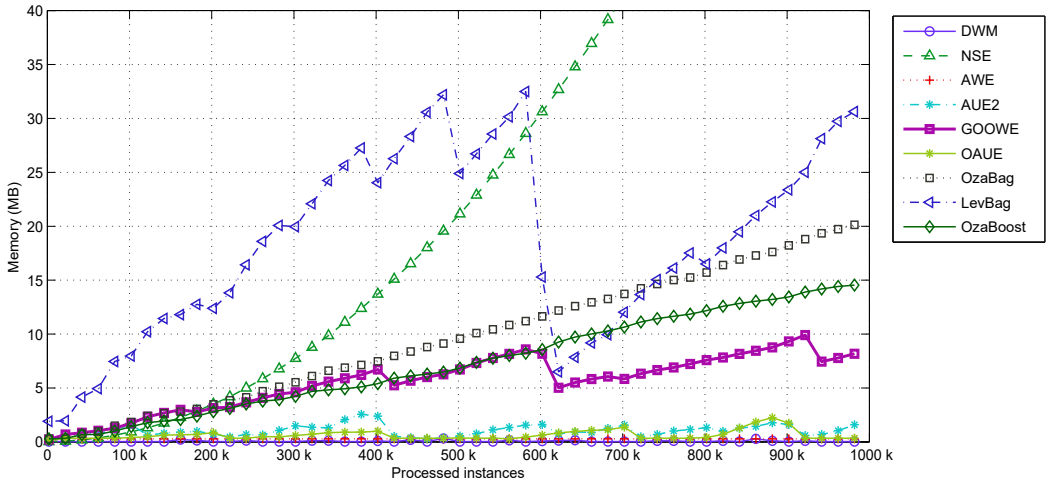


(b) memory

Figure 4.1: RCD example with gradually changing data stream: Classification accuracy and memory consumption for RBF-G-4-F dataset.



(a) accuracy



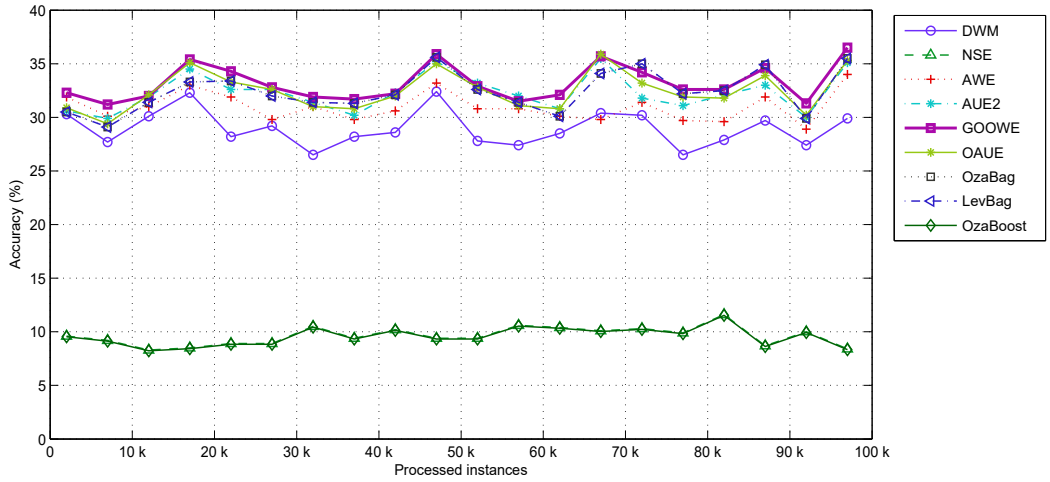
(b) memory

Figure 4.2: RCD example with abruptly changing data stream: Classification accuracy and memory consumption for RBF-A-10-S dataset.

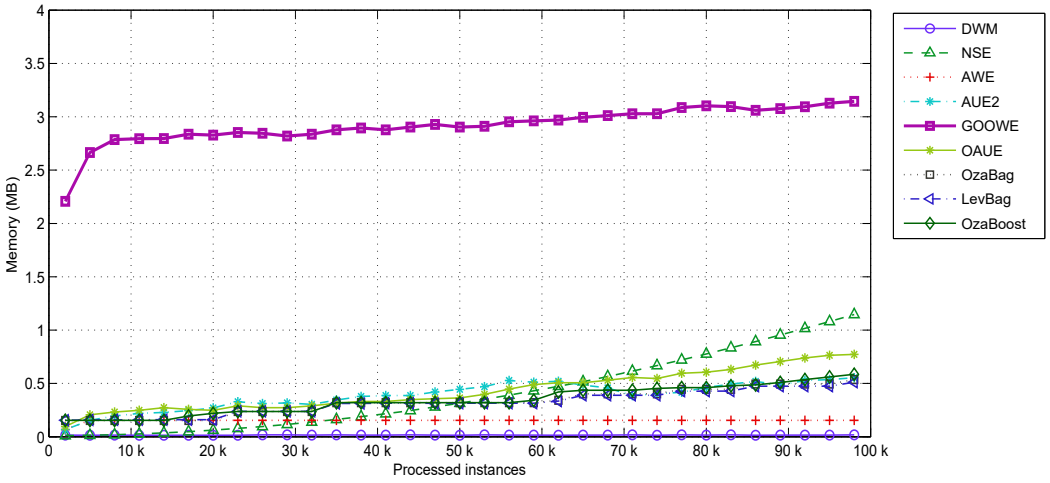
abruptly changing data streams. Comparing the number of class labels suggests that GOOWE performs better for RCD datasets with 10 class-labels rather than 4. The preliminary experiments show that this relationship changes with the number of component classifiers of the ensemble. For example, having 4 component classifiers can benefit more from data stream with 4 class labels.

As shown in Table 4.2, in most of the cases, GOOWE has higher average accuracy for the fast changing datasets, compared to the slow changing ones. We can intuitively understand the reason in accuracy plots of Fig. 4.1-(a) and 4.2-(a). They present behaviors of different ensemble methods with the arrival of new data chunks of gradually/abruptly changing RBF data streams. The place of abrupt drifts is clear in the classification accuracy plots, consistent with what we know from generation step of these synthetic datasets. In most abruptly changing points, it is obvious that GOOWE has significantly faster adaptive reactions than those of the others. While OzaBoost, LevBag and OzaBag perform similar to GOOWE in stationary phases of the data stream, they react slowly in changing phases. As a result, when more number of changes exist in stream data, GOOWE provides better performance. DWM, NSE and AWE are among the poorly performing algorithms.

Table 4.3 and 4.4 for the RBF datasets (the first 8 rows) show a conservative resource consumption of GOOWE, in terms of time and memory. We present memory usage behavior for the algorithms in RBF-G-4-F and RBF-A-10-S datasets in Fig. 4.1-(b) and 4.2-(b). They show that most ensemble methods drop one of the most memory-hungry component classifiers with drift occurrence. Among these algorithms, memory consumption of GOOWE is less than those of NSE, LevBag, OzaBag, and OzaBoost. Although it uses more memory than DWM, AWE, AUE2, and OAUE, it does not grow exponentially. As Brzezinski explained in [6], no pruning was used to limit the number of components for NSE and it requires much more time and memory than the other algorithms. As a result, memory usage of NSE does not react to concept drifts and it grows exponentially with the arrival of new instances.



(a) accuracy



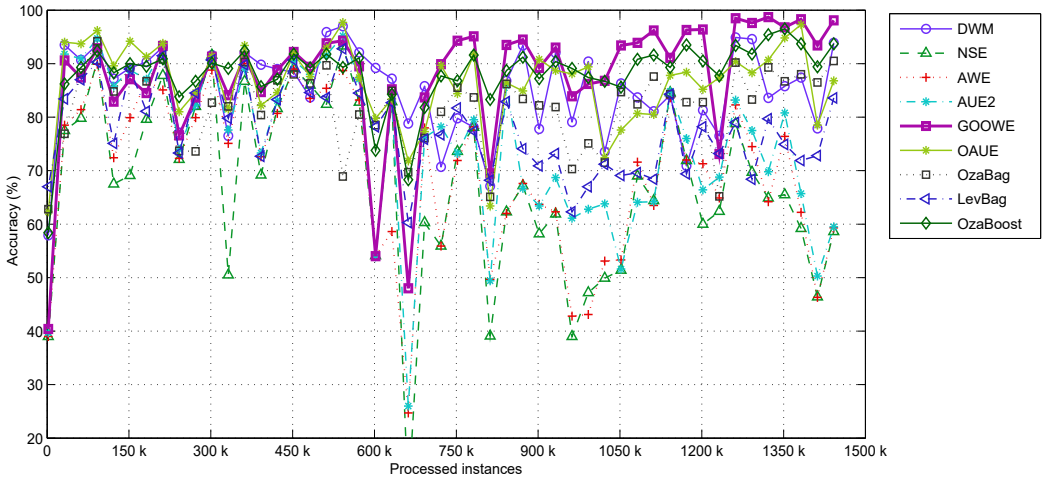
(b) memory

Figure 4.3: LCD example with reoccurring data stream: Classification accuracy and memory consumption for TREE-F dataset.

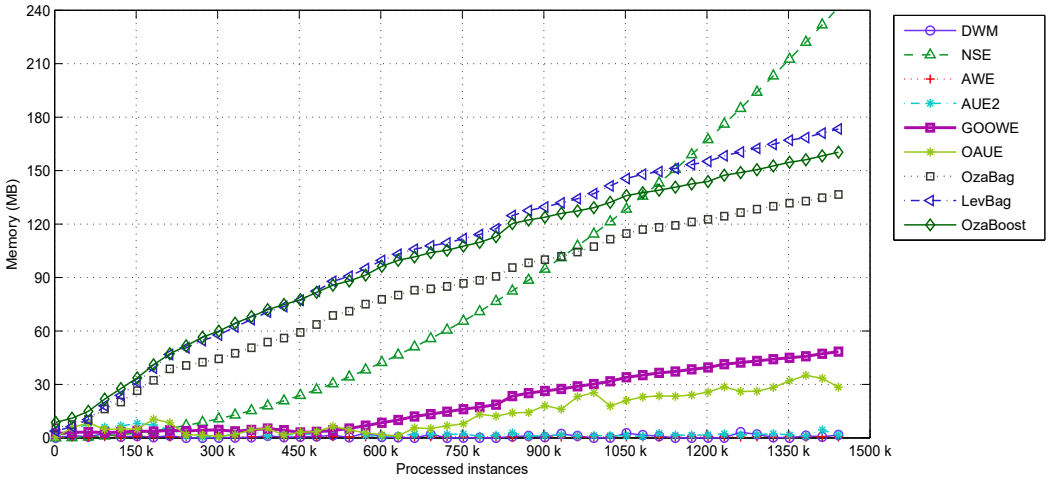
4.3.2 LCD Data Streams with Miscellaneous Drift Patterns

Table 4.2 for the LCD generators (the second 8 rows) shows that GOOWE is among the top tier algorithms in the Rotating Hyperplane, TREE-F, and LED datasets, in terms of accuracy. Similar to the gradually changing RBF datasets, Rotating Hyperplane dataset has incremental drifts. TREE-F is the smallest and fastest changing dataset with reoccurring drift patterns. These characteristics lead to superiority of GOOWE. Generally, for the LCD datasets, we can say that GOOWE acts better for the fast changing datasets compared to the slow ones. For the LED datasets there is no significant difference among various algorithms and most of them reacting similarly, since there are no clear concept drifts. For the SEA datasets, although GOOWE is not among the top tier algorithms, the differences among accuracy values are small. Moreover, Table 4.3 and 4.4 for LCD data streams (second 8 rows) show a comparable resource consumption of GOOWE, in terms of time and memory, similar to RCD data streams.

We can see the behavior of the algorithms for TREE-F dataset, in terms of accuracy (Fig. 4.3-(a)) and memory usage (Fig. 4.3-(b)). Similar to other accuracy plots, GOOWE reacts robustly to concept drifts. The memory usage plot suggests that GOOWE is the worst memory consumer algorithm. However, its memory growth rate is slow and its maximum memory usage is under 4 MB. In other words, it uses a limited amount of memory. In contrast to other plots, in Fig. 4.3-(b), NSE shows a small consumption of memory. In general, for NSE on small datasets, when only a few components are created, memory usage is reasonable.



(a) accuracy



(b) memory

Figure 4.4: Real-world example data stream: Classification accuracy and memory consumption for CovPokElec dataset.

4.3.3 Real-World Data Streams with Unknown Drift Patterns

Table 4.2 for real-world datasets (the last 4 rows) shows superiority of GOOWE over other algorithms in PokerHand and CovPokElec datasets, in terms of accuracy. For CoverType and Airlines datasets, although GOOWE is not the best performing algorithm, still the difference with the best performing algorithms are less than 1 percent. In addition, Table 4.3 and 4.4 for real-world datasets (the last 4 rows) show a feasible resource consumption of GOOWE, in terms of time and memory.

Fig. 4.4 shows classification accuracy and memory usage behaviors for the CovPokElec real-world dataset with arrival of new data chunks. The accuracy plot (Fig. 4.4-(a)) shows that GOOWE, OzaBoost, OAUE and DWM are among the best performing ensemble methods. There are diverse types of concept drift in the CovPokElec dataset. For example, comparing the behavior of the algorithms around 350k and 400k demonstrates that, although all the algorithms prove the existence of concept drift, for the first evolving point OzaBoost reacts best in contrast to the second point that DWM shows the best reaction. By looking at 750k or 1050k points, we can say that, in some situations, different algorithms are not synchronous; while some of them (DWM, OzaBag, LevBag) show decrease in performance, the others (NSE, AWE, AUE2, GOOWE, OAUE, OzaBoost) show an increase. Considering the first 500k of instances, belonging to the normalized CoverType dataset, DWM and OAUE outperform the others and react faster to unknown drift types. For the second 500k of instances, belonging to the normalized PokerHand dataset, we see more robust behavior from GOOWE and OAUE. However, for the last 500k of instances, belonging to the Electricity dataset, except one evolving point around 1250k, the best performing algorithm is GOOWE.

The memory plot (Fig. 4.4-(b)) suggests that while AWE, DWM and AUE2 are the least memory consumers, OAUE and GOOWE algorithms are far better than NSE, OzaBag, LevBag and OzaBoost which grow exponentially.

Chapter 5

Statistical Analysis and Further Discussion

In order to assure the significant difference of average values for classification accuracies, processing time and memory usage, we carried out statistical tests. First, a one-way analysis of variance (ANOVA) test using Scheffe multiple comparisons [44] were conducted on the results of different algorithms for each dataset. The null-hypothesis for each dataset when considered individually is: There is no significant difference between the algorithms.

We conducted the Scheffe test at the significance level of $\alpha = 0.05$. The most accurate, least processing time consumer, and least memory consumer algorithms are underlined for each row of Table 4.2, 4.3, and 4.4. We underlined the top tier group of the Scheffe's comparison results for each synthetic dataset [44]. As we mentioned earlier, it is not possible to conduct the Scheffe statistical test for real-world datasets, since they only have a single value. For each of them, we underline the most accurate and least resource consumer algorithm. As it is shown in Table 4.2, for 15 out of 20 datasets, GOOWE is consistently among the most accurate algorithms. OAUE is placed in the second rank of the most accurate algorithms with 11 out of 20 datasets. For the case of time and memory usage, Table 4.3 and 4.4, we can see that GOOWE is among the conservative consumers of resources.

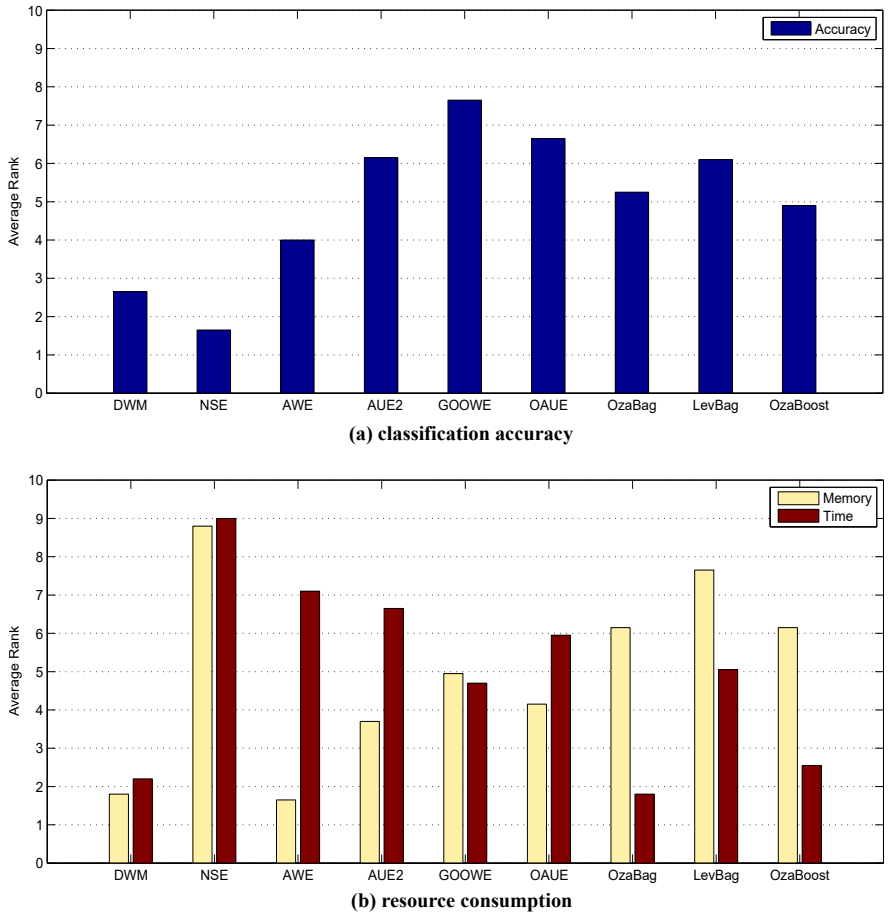


Figure 5.1: The Friedman statistical test average rank plots; for classification accuracy plot (a) higher average rank means better prediction, and for resource consumption plot (b) lower average ranks mean better performance.

Despite this fact, when we compare resource usage with the worst ones, we can see that the costs are much less and affordable. In addition, comparing the resource usage of OAUE and LevBag with GOOWE shows that our algorithm is in the same range of memory and time consumption.

To extend the analysis, we carried out the non-parametric Friedman statistical test for comparing multiple classifiers over multiple datasets [45, 46]. The null-hypothesis for this test states that all the algorithms are equivalent on all datasets when considered together. Since we have 9 algorithms and 20 datasets in our experiment, F_F is distributed according to the F distribution with $9 - 1 = 8$ and $(9 - 1) \times (20 - 1) = 152$ degrees of freedom. We run the statistical tests at the

significance level of $\alpha = 0.05$, and the Critical Distances (CD) for $F(8, 152)$, and average ranks of algorithms are given in Table 5.1. If the Friedman test results in a P-value less than α, the null-hypothesis is rejected and we can conclude that at least 2 of the algorithms are significantly different from each other. The tests for accuracy, memory, and time results in P-values less than 0.00001, and the null-hypotheses are rejected for all cases. We plot these average ranks in Fig. 5.1. Note that for classification accuracy, Fig. 5.1-(a), higher average rank means better prediction; and for resource consumption, Fig. 5.1-(b), lower ranks mean better performance.

Table 5.1 shows that, according to the Friedman test, GOOWE outperforms DWM, NSE, AWE, AUE2, OzaBag, LevBag, and OzaBoost; except OAUE. The CD value is 1.238 and their rank difference is less than this value ($7.650 - 6.650 = 1.000$). Since the difference of average ranks between GOOWE and OAUE is close to CD, we performed the Wilcoxon signed rank test to further analyze this pair of algorithms [45]. It ranks the absolute values of the differences between paired samples, and calculates a statistic on the number of negative and positive differences. For our case, the positive differences are 13, and the negative differences are 7. The two-tailed probability value, $P = 0.014$, is less than $\alpha = 0.05$; Therefore, it can be accepted that GOOWE is significantly better than OAUE, in terms of accuracy.

Similar to the accuracy test, we performed time and memory statistical comparisons using the Friedman test, summarized in Table 5.1. For the memory test, we reject the null-hypothesis. The average ranks show that GOOWE uses more memory compared to DWM, AWE, AUE2, and OAUE. On the other hand, it uses less memory compared to NSE, OzaBag, LevBag, and OzaBoost. The CD value shows that, GOOWE is in the middle of the baselines as a singleton with a significant difference from upper and lower range algorithms. For the processing time test, we again reject the null-hypothesis; and the average ranks show that GOOWE is faster than NSE, AWE, AUE2, and OAUE. It is significantly slower than DWM, OzaBag,and OzaBoost, somehow equivalent to LevBag.

In summary, we can say that there is a trade-off between prediction accuracy

Table 5.1: Summary of the Friedman Statistical Test for Accuracy, Memory and Time; The underlined values are GOOWE and its rivals that are in the same range of rank with no significant difference

	Test Results		Average Algorithm Ranks								
Test	F_F	CD	DWM	NSE	AWE	AUE2	GOOWE	OAUE	OzaBag	LevBag	OzaBoost
Accuracy	19.153	1.238	2.650	1.650	4.000	6.150	<u>7.650</u>	<u>6.650</u>	5.250	6.100	4.900
Memory	76.198	0.784	1.800	8.800	1.650	3.700	<u>4.950</u>	4.150	6.150	7.650	6.150
Time	77.692	0.778	2.200	9.000	7.100	6.650	<u>4.700</u>	5.950	1.800	<u>5.050</u>	2.550

and resource consumption. In this trade-off, GOOWE can predict statistically significantly better compared to the most accurate algorithms. Furthermore, it uses affordable resources compared to the most memory and time efficient ensembles.

Chapter 6

Conclusions and Future Work

In this study, we provide a geometrically optimum and online-weighted ensemble classifier, called GOOWE, for non-stationary environments. The main contribution of the proposed algorithm is providing a spatial modeling for using the linear least squares (LSQ) solution for dynamically optimizing the weights of components of an ensemble classifier for evolving environments. Our algorithm, different from the use of LSQ in batch mode ensembles, has dynamically changing optimum weight assignment to component classifiers and continuous training and testing. We use data chunks for training and sliding instance window containing the latest available data for testing; such an approach provides more robust behavior as shown in our experiments. We use the Euclidean norm as the measure of closeness for LSQ. The LSQ proved to react well in noisy situations [11], as it did in our algorithm for data streams with concept drifts.

We experimentally evaluate GOOWE on 20 datasets as data streams with tens of millions of instances, where 16 of them are synthetic and 4 of them are real-world datasets. For the synthetic streams, we use two categories of data stream generators: Rigorous Concept Drift (RCD) and Loose Concept Drift (LCD), each with 8 datasets. We include all possible patterns of change (sudden/abrupt, incremental, gradual, and reoccurring as concept drifts including blips and noise)

into our datasets. We compare GOOWE with 8 state-of-the-art ensemble classifiers for evolving data streams as our baselines. The statistical tests prove the superiority of GOOWE and its robustness in reacting to different types of concept drift, in terms of accuracy. Furthermore, we show that it requires a conservative resource consumption, in terms of memory and processing time.

In future work, the impact of the length of data chunk size on the performance in the presence of different concept drifts and its dynamic determination are possible studies. Effects of instance window size on performance can also be analyzed. In addition, GOOWE can be used in various sub-problem domains, such as semi-supervised and multi-label classification. It can be applied to unbalanced datasets and streams with concept-evolution.

Bibliography

[1] J. Gama, I. Zliobaite, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Comput. Surv.*, vol. 46, no. 4, pp. 44:1–44:37, 2014.

[2] J. Gama, *Knowledge discovery from data streams*. CRC Press, 2010.

[3] L. I. Kuncheva, "Classifier ensembles for changing environments," in *Multiple Classifier Systems, 5th International Workshop, MCS 2004, Cagliari, Italy, June 9-11, 2004, Proceedings*, pp. 1–15, 2004.

[4] A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavaldà, "New ensemble methods for evolving data streams," in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, June 28 - July 1, 2009*, pp. 139–148, 2009.

[5] L. I. Kuncheva, "Classifier ensembles for detecting concept change in streaming data: Overview and perspectives," in *2nd Workshop SUEMA*, vol. 2008, pp. 5–10, 2008.

[6] D. Brzezinski and J. Stefanowski, "Reacting to different types of concept drift: The accuracy updated ensemble algorithm," *IEEE Trans. Neural Networks Lear. Sys.*, vol. 25, no. 1, pp. 81–94, 2014.

[7] J. Z. Kolter and M. A. Maloof, "Using additive expert ensembles to cope with concept drift," in *Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005), Bonn, Germany, August 7-11, 2005*, pp. 449–456, 2005.

[8] H. Wang, W. Fan, P. S. Yu, and J. Han, "Mining concept-drifting data streams using ensemble classifiers," in *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 24 - 27, 2003*, pp. 226–235, 2003.

[9] X. Zhu, P. Zhang, X. Lin, and Y. Shi, "Active learning from stream data using optimal weight classifier ensemble," *IEEE Trans. Systems, Man, and Cybernetics, Part B*, vol. 40, no. 6, pp. 1607–1621, 2010.

[10] D. G. Saari, "Complexity and the geometry of voting," *Mathematical and Computer Modelling*, vol. 48, no. 9-10, pp. 1335–1356, 2008.

[11] P. C. Hansen, V. Pereyra, and G. Scherer, *Least squares data fitting with applications*. Baltimore, Md.: Johns Hopkins University Press, 2013.

[12] L.-W. Chan, "Weighted least square ensemble networks," in *International Joint Conference on Neural Networks, 1999. IJCNN'99*, vol. 2, pp. 1393–1396, IEEE, 1999.

[13] J. H. Friedman, "Stochastic gradient boosting," *Computational Statistics & Data Analysis*, vol. 38, no. 4, pp. 367–378, 2002.

[14] P. Zhang, X. Zhu, and Y. Shi, "Categorizing and mining concept drifting data streams," in *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, August 24-27, 2008*, pp. 812–820, 2008.

[15] D. M. Farid, L. Zhang, A. Hossain, C. M. Rahman, R. Strachan, G. Sexton, and K. Dahal, "An adaptive ensemble classifier for mining concept drifting data streams," *Expert Systems with Applications*, vol. 40, no. 15, pp. 5895–5906, 2013.

[16] D.-H. Han, X. Zhang, and G.-R. Wang, "Classifying uncertain and evolving data streams with distributed extreme learning machine," *Journal of Computer Science and Technology*, vol. 30, no. 4, pp. 874–887, 2015.

[17] M. M. Masud, J. Gao, L. Khan, J. Han, and B. M. Thuraisingham, "Classification and novel class detection in concept-drifting data streams under time

constraints," *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 6, pp. 859–874, 2011.

[18] Y. Sun, K. Tang, L. L. Minku, S. Wang, and X. Yao, "Online ensemble learning of data streams with gradually evolved classes," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 6, pp. 1532–1545, 2016.

[19] S. Wang, L. L. Minku, and X. Yao, "Resampling-based ensemble methods for online class imbalance learning," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 5, pp. 1356–1368, 2015.

[20] R. Elwell and R. Polikar, "Incremental learning of concept drift in nonstationary environments," *IEEE Trans. Neural Networks Lear. Sys.*, vol. 22, no. 10, pp. 1517–1531, 2011.

[21] L. L. Minku, A. P. White, and X. Yao, "The impact of diversity on online ensemble eearning in the presence of concept drift," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 5, pp. 730–742, 2010.

[22] 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, 2012.

[23] N. Littlestone, "Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm," *Machine Learning*, vol. 2, no. 4, pp. 285–318, 1987.

[24] N. Littlestone and M. K. Warmuth, "The weighted majority algorithm," *Inf. Comput.*, vol. 108, no. 2, pp. 212–261, 1994.

[25] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of online learning and an application to boosting," *J. Comput. Syst. Sci.*, vol. 55, no. 1, pp. 119–139, 1997.

[26] W. N. Street and Y. Kim, "A streaming ensemble algorithm (SEA) for large-scale classification," in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, San Francisco, CA, USA, August 26-29, 2001*, pp. 377–382, 2001.

[27] N. C. Oza, *Online Ensemble Learning*. PhD thesis, Computer Science Division, Univ. California, Berkeley, CA, USA, 09 2001.

[28] N. C. Oza and S. Russell, "Experimental comparisons of online and batch versions of bagging and boosting," in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 359–364, ACM, 2001.

[29] J. Z. Kolter and M. A. Maloof, "Dynamic weighted majority: A new ensemble method for tracking concept drift," in *Data Mining, 2003. ICDM 2003. Third IEEE International Conference on*, pp. 123–130, IEEE, 2003.

[30] J. Z. Kolter and M. A. Maloof, "Dynamic weighted majority: An ensemble method for drifting concepts," *The Journal of Machine Learning Research*, vol. 8, pp. 2755–2790, 2007.

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

[32] A. Bifet, G. Holmes, and B. Pfahringer, "Leveraging bagging for evolving data streams," in *Machine Learning and Knowledge Discovery in Databases, European Conference, ECML PKDD 2010, Barcelona, Spain, September 20-24, 2010, Proceedings, Part I*, pp. 135–150, 2010.

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

[34] K. Tumer and J. Ghosh, "Analysis of decision boundaries in linearly combined neural classifiers," *Pattern Recognition*, vol. 29, no. 2, pp. 341–348, 1996.

[35] T. G. Dietterich and G. Bakiri, "Solving multiclass learning problems via error-correcting output codes," *J. Artif. Intell. Res. (JAIR)*, vol. 2, pp. 263–286, 1995.

[36] A. Bifet and R. Gavaldà, "Learning from time-changing data with adaptive windowing," in *Proceedings of the Seventh SIAM International Conference on Data Mining, April 26-28, 2007, Minneapolis, Minnesota, USA*, pp. 443–448, 2007.

[37] 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, 2013.

[38] J. Gama, R. Sebastião, and P. P. Rodrigues, "On evaluating stream learning algorithms," *Machine Learning*, vol. 90, no. 3, pp. 317–346, 2013.

[39] J. Gao, W. Fan, and J. Han, "On appropriate assumptions to mine data streams: Analysis and practice," in *Seventh IEEE International Conference on Data Mining (ICDM 2007)*, pp. 143–152, IEEE, 2007.

[40] A. Mustafa, A. Haque, L. Khan, M. Baron, and B. Thuraisingham, "Evolving stream classification using change detection," in *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2014 International Conference on*, pp. 154–162, IEEE, 2014.

[41] P. M. Domingos and G. Hulten, "Mining high-speed data streams," in *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, Boston, MA, USA, August 20-23, 2000*, pp. 71–80, 2000.

[42] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, "MOA: massive online analysis," *Journal of Machine Learning Research*, vol. 11, pp. 1601–1604, 2010.

[43] G. Hulten, L. Spencer, and P. M. Domingos, "Mining time-changing data streams," in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, San Francisco, CA, USA, August 26-29, 2001*, pp. 97–106, 2001.

[44] H. Scheffe, *The Analysis of Variance*. New York: John Wiley, 1959.

[45] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *The Journal of Machine Learning Research*, vol. 7, pp. 1–30, 2006.

[46] W. Conover, *Practical Nonparametric Statistics*. New York: John Wiley & Sons, 1999.