BIRMINGHAM CITY UNIVERSITY

DOCTORAL THESIS

Hybrid Metaheuristic Methods for Ensemble Classification in Non-stationary Data Streams

---

Author:
Hossein GHOMESHI

Supervisors:
Prof. Mohamed GABER
Dr. Yevgeniya KOVALCHUK

A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy

in the

School of Computing and Digital Technology
Birmingham City University

June 8, 2020

Declaration of Authorship

BIRMINGHAM CITY UNIVERSITY

Abstract

Faculty of Computing, Engineering and the Built Environment
School of Computing and Digital Technology

Doctor of Philosophy

Hybrid Metaheuristic Methods for Ensemble Classification in Non-stationary Data Streams

by Hossein GHOMESHI

The extensive growth of digital technologies has led to new challenges in terms of processing and distilling insights from data that generated continuously in real-time. To address this challenge, several data stream mining techniques, where each instance of data is typically processed once on its arrival (i.e. online), have been proposed. However, such techniques often perform poorly over non-stationary data streams, where the distribution of data evolves over time in unforeseen ways. To ensure the predictive ability of a computational model working with evolving data, appropriate data-stream mining techniques capable of adapting to different types of concept drifts are required. So far, ensemble-based learning methods are among the most popular techniques employed for performing data stream classification tasks in the presence of concept drifts. In ensemble learning, multiple learners forming an ensemble are trained to obtain a better predictive performance compared to that of a single learner.

This thesis aims to propose and investigate novel hybrid metaheuristic methods for performing classification tasks in non-stationary environments. In particular, the thesis offers the following three main contributions. First, it presents the Evolutionary Adaptation to Concept Drifts (EACD) method that uses two evolutionary algorithms, namely, Replicator Dynamics (RD) and Genetic algorithm (GA). According to this method, an ensemble of different classification *types* is created based on various feature sets (called *subspaces*) randomly drawn from the target data stream. These subspaces are allowed to grow or shrink based on their performance using RD, while their combinations are optimised using GA. As the second contribution, this thesis proposes the REplicator Dynamics & GENEtic (RED-GENE) algorithm. RED-GENE builds upon the EACD method and employs the same approach to creating different classification *types* and GA optimisation technique. At the same time, RED-GENE improves the EACD method by proposing three different modified versions of RD to accelerate the concept drift adaptation process. The third contribution of the thesis is the REplicator Dynamics & Particle Swarm Optimisation (RED-PSO) algorithm that is based on a three-layer architecture to produce classification types of different sizes. The selected feature combinations in all classification types are optimised using a non-canonical version of the Particle Swarm Optimisation (PSO) technique for each layer individually.

An extensive set of experiments using both synthetic and real-world data streams proves the effectiveness of the three proposed methods along with their statistical significance to the state-of-the-art algorithms. The proposed methods in this dissertation are consequently compared with each other that proves each of the proposed methods has its strengths towards concept drift adaptation in non-stationary data stream classification. This has led us to formulate a list of suggestions on when to use each of the proposed methods with regards to different applications and environments.

Acknowledgements

The research presented in this thesis was funded by Birmingham City University as part of the STEAM scholars program and supported by many individuals.

First and foremost, I would like to express my sincere gratitude to my supervisory team, including my first supervisor, Prof. Mohamed Medhat Gaber, and my second supervisor, Dr. Yevgeniya Kovalchuk, for their continuous support throughout my PhD study.

I am indebted to Mohamed for his insights, motivation, ideas, immense knowledge and believing in me. His guidance has always helped me during my research and writing of this thesis. I could not have imagined having a better supervisor and mentor for my research.

I would like to gratefully thank Yevgeniya for her advice, inputs, ideas and patience throughout my research. Her feedback, attention to detail and knowledge has allowed me to shape this thesis and express the concepts behind my research.

I would also like to thank the staff in the Doctoral Research College (DRC) and the faculty of Computing, Engineering and the Built Environment (CEBE) of Birmingham City University for their administrative support during my research.

I want to acknowledge my colleagues and fellow PhD students at the Data Analytics and Artificial Intelligence (DAAI) research group and other school friends, who helped to make my PhD study an enjoyable and memorable journey.

My special thanks go to my Mom and Dad, Batool and Mehdi, as well as everyone else in my family, including Mohammad, Zahra, Ahmad, Parisa, Sarina, Samyar and Sophia, for their continuous support and courage during my study. No words can truly express how grateful I am for having them in my life.

Last but not least, I owe a great thank you to my best friends, who were of great emotional support during my journey. Thanks for being by my side at all times.

Contents

List of Figures

List of Tables

List of Abbreviations

ADOB	**A**daptable **D**iversity-based **O**nline **B**oosting
ARF	**A**daptive **R**andom **F**orest
CV	**C**ross **V**alidation
DDM	**D**rift **D**etection **M**ethod
DWM	**D**ynamic **W**eighted **M**ajority
EACD	**E**volutionary **A**daptation (to) **C**oncept **D**rifts
EDDM	**E**arly **D**rift **D**etection **M**ethod
GA	**G**enetic **A**lgorithm
GP	**G**enetic **P**rogramming
Lev-Bag	**Lev**veraging **Bag**ging
MOA	**M**assive **O**online **A**nalysis
NA	**N**ash **E**quilibrium
OAUE	**O**nline **A**ccuracy **U**pdated **E**nsemble
OGA	**O**nline **G**enetic **A**lgorithm
PSO	**P**article **S**warm **O**ptimisation
RD	**R**eplicator **D**ynamics
RED-GENE	**RE**plicator **D**ynamics (&) **GENE**tic algorithm
RED-PSO	**RE**plicator **D**ynamics (&) **P**article **S**warm **O**ptimisation

Dedicated to my loving parents,

Mehdi and Batool,

whose unconditional love, affection, support and prayers

are my strength in everything I do.

Chapter 1

Introduction

1.1 Preamble

Data mining is the science of extracting useful knowledge from massive data repositories [1]. This interdisciplinary field of computer science, which involves machine learning and statistical techniques, has been widely applied to problems in both industry and science [2]. Machine learning is a field of computer science aiming at teaching computers how to learn from data and automatically improve with experience [3].

A system built using a machine learning technique is called a machine learning model. In general, machine learning techniques can be categorised into supervised and unsupervised methods. In supervised learning, the aim is to build a machine learning model using labelled instances to predict the labels of future data. This is in contract with unsupervised learning, where the aim is to find or distinguish patterns in data while the labels of instances are not given. Other categories of machine learning techniques that have different degrees and patterns of supervision exist (e.g. semi-supervised learning and active learning where the number of labelled instances is limited) [4]. In this thesis, the terms machine learning and data mining are used interchangeably providing their significant overlap.

Classification is a supervised data mining technique used to label a data record/instance with one of the predefined classes using a machine learning model trained over a set of labelled data. This area of research has been extensively studied considering many different applications [5] and is the main focus of this thesis.

To perform a typical classification task, a static set of labelled data is employed to train and test a classification model. Training is performed to build a model (classifier) using a machine learning technique, while testing is performed to assess the model's performance. Evaluation techniques such as holdout or cross validation (CV) are typically used for the training and testing purposes in classical machine learning supervised tasks [6]. The holdout method divides the considered dataset into two mutually exclusive subsets of data called a training set and a testing (holdout) set. The most common way of splitting is to designate 70% of data to the training set and the remaining 30% to the testing set [6]. In contrast, the CV method repeatedly splits the dataset into a training set and a validation set – so that each data record (instance) can contribute to both training and validation sets.

Nevertheless, in many modern-world applications such as sensor network analysis [7], traffic monitoring [8], weather forecasting [9], social media analysis, spam filtering systems [10], stock market prediction, fraud and intrusion detection [11], healthcare systems [12] and web searches [7], data come mainly in the form of data streams, with a high-volume of data being produced continuously. Mining data in such time-critical applications requires online techniques to find recent patterns and perform accurate predictions.

Online mining of data streams is more challenging than classical data mining of static datasets in the following aspects:

- First, due to the massive amount of continuous data points in data streams, along with the time and memory constraints, each instance (record) of data is processed only once on the arrival in most cases. Hence, iterative processing and storing the data in-memory are not possible [13].

- Second, classical data mining evaluation techniques such as holdout and CV are not suitable for data streams due to the infinite size and high-volume of the data, where training over data streams cannot be independent from testing the continuously updated model [14].

- Third, the underlying distribution of data often changes in many real-world data streams. This can be caused by different factors such as changes in the temporal and/or spatial context and possibly other environmental factors (e.g. natural disasters) [15].

These challenges necessitate proposing new data mining techniques or modifying the existing classical techniques to enable the extraction of knowledge structures from data streams. Indeed, a plethora of data stream mining techniques has been proposed during the last two decades [16].

The main objective in many data stream mining applications is to predict the class of the incoming data records over the target data stream. The classification procedure in this context should be performed based on a trained model (classifier(s)) built upon previous labelled data records (once available) using their features and the recent data distribution. This is related to the problem of adaptation to concept drifts that will be discussed in the next section.

As opposed to single-classifier models, multiple-classifier systems offer diverse mechanisms for coping with concept drifts and have been demonstrated to have a superior performance in stream classification tasks in non-stationary environments [14][13]. Such multiple-classifier systems are typically based on ensemble learning, which is defined in the following section.

Metaheuristic algorithms is an emerging area in computing aiming at solving problems using the principles of heuristic (partial search algorithms) that may provide a sufficiently good solution to an optimisation problem [17]. Such algorithms offer a natural solution to the changing environment problems. In particular, it is believed that metaheuristic algorithms allow the creation of new solutions with promising characteristics such as fault-tolerance, self-replication or cloning, reproduction, evolution, adaptation, learning and growth [18].

Providing that non-stationary data streams are typically generated by evolving environments, metaheuristic methods are hypothesised to be a suitable solution for non-stationary data stream classification tasks. Hence, this thesis studies the impact of using metaheuristic approaches when solving non-stationary data stream classification tasks.

1.2 Motivation

A considerable effort of recent research has focused on data stream classification tasks in non-stationary environments [15]. The main challenge in this research area concerns the adaptation to *concept drifts*, that is, when the data distribution changes over time in unforeseen ways. Concept drifts occur in different forms and can be divided into four general types: abrupt (sudden), gradual, incremental and recurrent (reoccurring). The different types of concepts are depicted in Figure 1.1.

In abrupt (sudden) concept drifts, the data distribution at the time *t* suddenly changes to a new distribution at the time *t+1*. Incremental concept drifts occur when the data distribution changes and stays in the new distribution after going through some new, unstable, median data distributions. In gradual concept drifts, the proportion of the new probability distribution of incoming data increases, while the proportion of the data belonging to the former probability distribution decreases over time. Recurring concept drifts happen when the same old probability distribution of data reappears after some time of a different distribution.

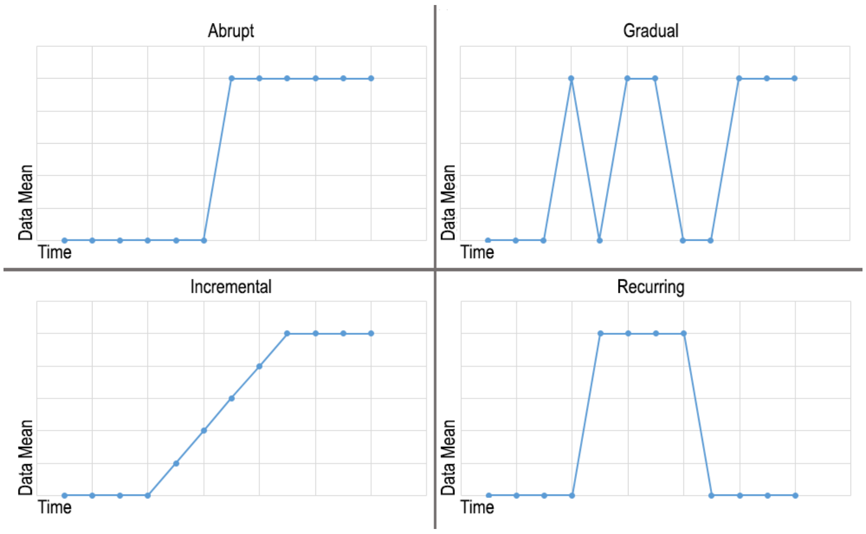


FIGURE 1.1: Different types of concept drifts. Adapted from [15].

To cope with the concept drift problem in a data stream, it is important to build a classification system that adapts to different concept drifts as quickly as possible. Ensemble learning has proved its superiority for stream classification in non-stationary environments over other classification techniques [14][13]. Ensemble learning is a machine learning approach, in which predictions of individual classifiers are combined using a combination rule to predict incoming instances more accurately compared to a single learner. In particular, a voting mechanism is used to combine different classifiers' outputs to establish a single class label as the output of the ensemble (Figure 1.2). The combination procedure is typically performed using majority voting or weighted majority voting.

The advantage of using ensemble learning techniques in non-stationary data stream classification lies in their ability to update swiftly according to the most recent data instances. This is usually achieved by training the existing classifiers in the ensemble and changing their weights according to their performance: adding new, well-performing classifiers, removing outdated, poor-performing classifiers and updating the existing classifiers. In single-classifier systems, the only strategy to address the concept drift problem is to update the classifier; however, this task can be challenging when dealing with different types of concept drifts. The extensive range of applications makes the task of non-stationary data stream classification even more challenging since various applications seek diverse purposes and have different conditions.

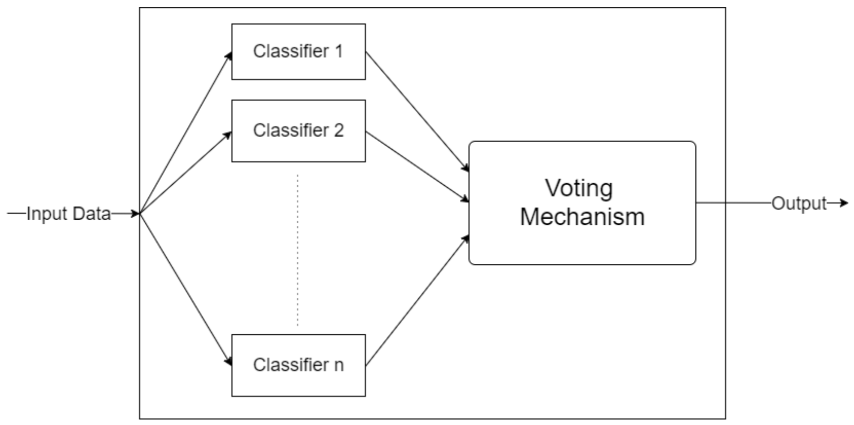


FIGURE 1.2: A typical ensemble learning system.

1.3 Problem Statement

The following main features should be taken into consideration when designing a versatile yet robust ensemble approach to non-stationary data stream classification:

- **Reliability**: the main target of any classification approach is to achieve a high degree of reliability in the results. The reliability of a classification technique can be tested by calculating the method's performance using different performance metrics such as accuracy, precision, recall and F1 score along with inter-rater reliability score (i.e. Cohen's Kappa statistic [19]) which is the score of consistency of agreement among different experts (classifiers) in a pool of experts. A versatile approach should have reliable results over different applications having different characteristics such as a various number of features/classes and various imbalance ratios.
- **Efficiency**: in many applications, there are constraints on the system in terms of the time and memory usage. When the time calculating an

output or the amount of available memory is limited, the learning time and space of an approach should be minimised.

- **Adaptation**: when a concept drift happens in a data stream, the accuracy of the ensemble drops. This is due to the change in the data distribution and/or the target concept. It is important to minimise the rate of misclassification and time of recovery upon different types of concept drifts.

Consequently, an ideal approach to non-stationary data stream classification should satisfy the following many-fold objective: *having the highest degree of reliability while minimising the computational complexity and quickly adapting to possible concept drifts*. However, to the best of our knowledge, there is no comprehensive approach able to satisfy these conditions at the same time in different environments. The majority of the state-of-the-art ensemble methods for data stream classification are focused on either a specific type of concept drifts or a specific type of applications.

1.4 Aims and Objectives

This thesis proposes different hybrid metaheuristic ensemble learning methods with the following aims:

- satisfy the recognised gap in the state-of-the-art research that is the lack of a comprehensive approach able to satisfy all the conditions of an ideal approach in non-stationary data stream classification tasks;
- perform well regardless of the concept drift type or application;
- demonstrate robustness using different data streams having a various number of features/classes and various imbalance ratios;
- offer a natural solution to improving the accuracy of predictions.

To achieve these aims, the following objectives have been identified:

- study the state-of-the-art algorithms for non-stationary data stream classification tasks and explore different mechanisms for coping with the current challenges in such tasks;
- employ evolutionary algorithms to develop a novel ensemble learning technique for concept drift adaptation in data-streams;

- propose efficient ways of deploying evolutionary algorithms in ensemble methods designed to solve non-stationary data stream classification tasks;
- optimise the proposed metaheuristic solutions for non-stationary data stream classification by balancing out the exploration and exploitation of metaheuristics.

1.5 Contributions

Achieving the objectives of the thesis has led to the following contributions.

An Analytical Study on Ensemble Dynamics in Non-stationary Data Stream Classification

To address the first objective, the state-of-the-art ensemble methods for non-stationary data stream classification are introduced and analysed comprehensively. Furthermore, we study the ensembles' dynamic behaviour of the existing ensemble methods (e.g. adding, removing and updating classifiers). We propose a novel taxonomy to categorise the current methods based on their dynamic behaviour. Finally, a new, compact, yet informative formalisation of state-of-the-art methods is proposed [20].

Evolutionary Adaptation to Concept Drifts

To address the second objective, the Evolutionary Adaptation to Concept Drifts (EACD) [21] algorithm is proposed. EACD is a novel ensemble learning method for data stream classification in non-stationary environments. This method uses random selections of features and two evolutionary algorithms, namely, Replicator Dynamics (RD) and Genetic Algorithm (GA).

GA is a metaheuristic algorithm inspired by the process of natural selection, which is a subset of a bigger class of algorithms called evolutionary algorithms. Such algorithms are commonly used to generate high-quality solutions to optimisation and search problems relying on bio-inspired operators such as *mutation*, *crossover* and *selection*.

RD is a simple model of evolution and prestige-biased learning in game theory [22][23]. It provides a solution for selecting useful *types* from a population of diverse *types*. In this model, the act of selection happens at discrete times and the effectiveness of each *type* in the next selection operation is specified by the replicator equation as a function of the *type's* payoff and its current proportion in the population [24].

In the proposed EACD method, an ensemble of different classification *types* comprising randomly drawn features (subspaces) of the target data stream is trained. These randomly drawn subspaces are then optimised using GA to cope with different concept drifts over time. The training of the proposed ensemble is performed on sequential data blocks of the stream. The proposed ensemble technique allows a dynamic set of classification *types* to take action over time. In addition, the number of decision trees in a classification *type* (subspace) depends on the performance of this *type* on the most recent data. Hence, well-performing *types* increase in size, while poor-performing *types* decrease in size. Since the original RD and GA methods are designed to work only on static sets of data, in the proposed method, modified versions of RD and GA are proposed to enable their successful performance over streaming data.

In summary, EACD allows the ensemble to handle different types of concept drifts by employing two different evolutionary techniques. RD is used to continuously determine well-performing and poor-performing *types* and expand or shrink them accordingly. GA is used to compose new, improved *types* out of the existing ones by iterating over the most recent data.

The comprehensive experiments conducted in this research show the significant contribution of the EACD in non-stationary data stream classification by comparing the proposed method to state-of-the-art methods in this area and thus, led us to achieve the second objective outlined in this dissertation (Section 1.4).

Replicator Dynamics & Genetic Algorithm

To address the third objective of this thesis, the REplicator Dynamics & GENEtic algorithm (RED-GENE) method [25] built upon EACD [21] is proposed. Both EACD and RED-GENE employ the same approach to creating different classification *types* and GA optimisation technique. However, EACD employs only the most basic modified version of RD. RED-GENE improves the state-of-the-art methods and offers the following further contributions: (1) introducing three different modified versions of RD to accelerate the concept drift adaptation process; (2) improving the classification accuracy for the majority of the considered experimental cases; and (3) reducing the running time of the algorithm by generating a lower number of *types* while improving the total accuracy.

A set of experiments employing four artificial and five real-world data streams are conducted to compare the performance of the proposed method, RED-GENE, to EACD method and other state-of-the-art algorithms. Comparing to the EACD method, RED-GENE improves the performance of EACD in most cases, however, there are some cases that EACD outperforms the RED-GENE method as seen by average accuracy. Furthermore, comparing to five state-of-the-art algorithms, RED-GENE achieves the highest average accuracy in five out of nine datasets. Finally, the results show the significance of deploying the proposed modifications of RD in improving the efficiency of the metaheuristic algorithms in non-stationary data stream classification tasks.

Replicator Dynamics & Particle Swarm Optimisation

To address the final objective of this thesis, the REplicator Dynamics & Particle Swarm Optimisation (RED-PSO) algorithm is proposed. RED-PSO [26] is a novel method for non-stationary data stream classification that uses a modified metaheuristic algorithm, namely, Particle Swarm Optimisation (PSO), to comply with the aforementioned characteristics of an ideal approach to coping with evolving data streams in classification challenges.

PSO is a metaheuristic algorithm [27] inspired by the social behaviour of the movement of organisms in a bird flock or fish school. The main target of the PSO algorithm is finding the global minimum of a function. While PSO does not guarantee an optimal solution, it has been shown to have promising results in various applications [28].

The proposed RED-PSO technique comprises a three-layer architecture. Each layer is initially assigned some predefined classification *types* randomly created from a pool of features of the target data stream. RD is used first to seamlessly cope with smooth (i.e. gradual or incremental) concept drifts; it allows the classification *types* with a good performance to grow and those with a poor performance to shrink in size. The combination of features in all types is then optimised using a modified version of PSO for each layer individually. This helps the method to cope with sudden (i.e. recurring or abrupt) concept drifts. PSO allows the types in each layer to go towards local (within the same type) and global (across all types) optimums with a specified velocity.

The RED-PSO method is compared to seven state-of-the-art algorithms in an extensive set of experiments. The results show the significant performance of RED-PSO method in different environments and led us to achieve the final objective identified in Section 1.4.

In summary, each of the proposed methods in this dissertation is a standalone contribution to non-stationary data stream classification tasks that has its strengths. In particular, when a concept drift is manifested in having some features become irrelevant, EACD can outperform any other methods, because of its active addition and deletion of classifiers periodically. Furthermore, when the presented concept drifts in the data stream are mostly abrupt or having a severe magnitude, RED-GENE can adapt to such concept drifts more quickly than any other methods because of its dynamic RD modifications and also its concept drift detection mechanism. Eventually, when the target data stream contains more recurrent concept drifts or noisy data, RED-PSO can outperform any other methods. The reason for this is due to its capability of investigating both exploration and exploitation aspects of the search space and also its implicit mechanism towards concept drifts that makes this method resistant to false alarms.

We thoroughly analyse the conditions under which each method outperforms the others and provide recommendations on which method should be used in which applications and also investigate their performance in different scenarios (simulated by the datasets) in Section 8.2.

1.6 Publications

The following book chapter and journal papers have been published while working towards this thesis:

- H. Ghomeshi, M.M. Gaber, and Y. Kovalchuk, Ensemble dynamics in non-stationary data stream classification, in Learning from Data Streams in Evolving Environments, Springer, 2018, pp. 123—153.
- H. Ghomeshi, M. M. Gaber, and Y. Kovalchuk, EACD: Evolutionary adaptation to concept drifts in data streams, Data Mining and Knowledge Discovery, vol. 33(3), pp. 663–694, Springer, 2019, ISSN: 1573-756X.
- H. Ghomeshi, M.M. Gaber, and Y. Kovalchuk, A non-canonical hybrid metaheuristic approach to adaptive data stream classification, Future Generation Computer Systems, vol. 102, pp. 127–139, 2020.

- H. Ghomeshi, M.M. Gaber, and Y. Kovalchuk, An Evolutionary Game Theoretic Approach to Adaptive Data Stream Classification, vol. 7, pp. 173944– 173954, IEEE Access (2019).

1.7 Thesis Overview

The rest of this thesis is organised as follows.

Chapter 2 provides an overview of related work in non-stationary data stream classification tasks. Different ensemble learning techniques are reviewed and categorised into implicit and explicit methods based on their concept drift adaptation mechanism. This is followed by an overview of the current metaheuristic methods for concept drift adaptation. Then, we review the dynamic behaviours of existing ensemble learning methods such as adding new classifiers, removing old classifiers and updating current classifiers. We propose a novel taxonomy for defining ensemble's dynamics when performing non-stationary data stream classification tasks. Finally, a formalisation method for classification algorithms for streaming analytics is presented to simplify the process of understanding different approaches in this area.

Chapter 3 reviews the necessary background and theoretical explanation of the algorithms used/modified in this thesis. RD, GA, PSO and Hoeffding Trees (VFDT) are comprehensively reviewed and discussed in detail. This is followed by a discussion on the concept drift detection mechanism used as the main concept drift detector in the methods proposed in this thesis. Furthermore, different evaluation techniques employed in our experiments are introduced and justified.

Chapter 4 describes the experimental methodology containing the basic information for conducting the experiments in this thesis. Particularly, this chapter includes the experimental settings such as different methods used for comparison, experimental environment and specification along with different datasets and their characteristics (e.g. imbalance ratio, number of classes, number of features and number of records) in the experiments conducted in this thesis.

Chapter 5 presents the EACD method as our contribution towards the first objective. This method employs modified versions of RD and GA to enable their usage in data stream mining problems. Furthermore, it implements a seamless concept drift adaptation mechanism to reduce the recovery time upon different concept drifts and maximise the overall performance of the ensemble. The chapter also outlines the set of experiments employing four artificial and five real-world data streams we conducted to compare the performance of EACD with that of the state-of-the-art algorithms using the immediate and delayed prequential evaluation methods.

Chapter 6 presents RED-GENE as the second contribution of this thesis. This method is built upon EACD and employs three novel RD modifications to improve the performance of the ensemble learning technique over different types of concept drifts. The performance of RED-GENE is compared to that of the EACD and other state-of-the-art methods using the same set of experiments as described in Chapter 5.

Chapter 7 presents a novel ensemble technique called RED-PSO, which is the last contribution of this thesis. RED-PSO is based on a three-layer architecture to produce classification *types* of different sizes. RD is used in this method to seamlessly adapt to different concept drifts. In addition, the selected feature combinations in all classification *types* are optimised using a non-canonical version of PSO for each layer individually. PSO allows the types in each layer to go towards local (within the same type) and global (across all types) optimums with a specified velocity. A similar set of experiments as described in Chapter 5 is then employed to examine the performance of RED-PSO.

Chapter 8 summarises the work presented in this thesis and draws conclusions by comparing the proposed methods in this thesis to each other. It contains a brief description of the novel methods proposed in this thesis and points out a few important directions for future research.

Chapter 2

An Analytical Study on Ensemble Dynamics in Non-stationary Data Stream Classification

To achieve the first objective of this thesis as identified in the previous chapter, a comprehensive study of the current state-of-the-art algorithms for non-stationary data stream classification problems is conducted in this chapter. Furthermore, a novel taxonomy characterising ensemble's dynamics in non-stationary data stream classification is proposed. The work presented in this chapter has been published as a book chapter titled "Ensemble dynamics in non-stationary data stream classification" in "Learning from Data Streams in Evolving Environments" [20].

2.1 Introduction

With the rapid growth of digital technology, more data become available in the form of data streams (e.g. continuous sensor readings in the Internet of Things (IoT) or credit card transactions). Hence, data stream classification has started to play an important role in the areas of knowledge discovery and big data analytics over the past few years. In the context of data streams, the goal of classification is to predict the class label of incoming instances from continuous data records that, generally, can be read only once providing a limited time and memory. This can be achieved by extracting useful knowledge from the past data of a stream using machine learning techniques. However, knowledge discovery from data streams is more complex than that in the domains where all data are available at once.

General characteristics of data streams as considered by [29] include their unlimited size, the online arrival of data elements, order of the data elements that is not governable and restriction to process these elements only one time (it is possible to process an element more than once but at a high cost of storing elements).

From the data distribution point of view, there are two types of data streams: *stationary* (stable) data streams, where the probability distribution of instances is fixed, and *non-stationary* (evolving) data streams, where the probability distribution of incoming data evolves or target concepts (labelling mechanism) change over time. The latter phenomena is called *concept drift*. The presence of concept drifts in data streams makes classification tasks more complex and difficult to handle. This chapter focuses on non-stationary data stream classification.

2.2 Ensemble Methods for Non-stationary Data Stream classification

The majority of the existing data stream mining approaches to non-stationary environments use ensemble learning techniques for classification tasks [30] [15] [13]. Ensemble learning methods offer more flexibility (by allowing addition, removal and retraining of classifiers) [14] compared to single classifier techniques that use only one classifier for the classification task. This is an advantage especially in the non-stationary environments in which the classification approach needs to have a swift adaptation to the new concept once a concept drift happens.

Furthermore, many of the state-of-the-art ensemble learning methods for non-stationary data stream classification tasks are adapted versions of bagging [31] and boosting [32] and can be categorised into **explicit** and **implicit** methods based on whether or not they use a concept drift detector to cope with concept drifts. A concept drift detector is an algorithm able to detect concept drifts in a data stream on the basis of the information about new incoming examples and the model's performance. Explicit methods use a concept drift detection mechanism and have an explicit (immediate) reaction to a drift when it is detected, whereas implicit methods do not have an immediate reaction to concept drifts, and as such, adapt to drifts implicitly by updating the state of the ensemble according to the most recent instances. We review some of the ensemble learning explicit and implicit and techniques for non-stationary data stream classification next.

2.2.1 Explicit Methods

Adaptive Boosting (Aboost) [30] is one of the approaches that uses a concept drift detection method. It builds one classifier per every block of data received from a stream and classifies these instances. Then, it evaluates the ensemble's output and updates the weights of all classifiers based on whether or not an instance is classified correctly by the ensemble, as well as the classifier itself. Whenever a concept drift is detected, the weight of each classifier in the ensemble is reset to one. Finally, once the size of the ensemble is exceeded, the oldest classifier is removed from it.

Adwin Bagging (AdwinBag) [33] is an approach that uses Oza's online bagging algorithm [34] for its learning mechanism and adds a concept drift detector called ADaptive WINdowing (ADWIN) [35] to specify when a new classifier is required. AdwinBag is enhanced in the Leveraging Bagging (LevBag) algorithm [36] by the same authors. LevBag aims to add randomisation to the input and output of the classifiers and increase the extent of re-sampling in the bagging technique. The re-sampling rate in LevBag is changed from $Poisson(1)$ to $Poisson(\lambda)$, where λ is a user defined parameter.

Yet another explicit approach is Recurring Concept Drift (RCD) [37]. It uses a buffer to store the context of each data type in the stream. This method employs a two-phase concept drift detection mechanism. First, a new classifier is created and trained alongside a new buffer when the drift detection mechanism signals a warning. If it then signals a drift, which means the concept drift is approved, the system checks whether or not the new concept is similar to another concept that has been previously stored in the buffer. If there has been a recurring concept drift, RCD uses the classifier created with that concept drift to classify the incoming data and then starts training the classifier. If no similar concept drift can be found in the buffer, RCD stores the newly trained buffer and classifier in the system and uses them to classify the incoming instances. If the system does not get the drift signal to approve the drift, it assumes it to be a false alarm; the system ignores the stored data and continues to classify using the current classifier. Note, only one classifier is activated at a time in this approach, while the rest are deactivated, unless the same data concept happens again.

Adaptive Random Forest (ARF) [38] is an explicit ensemble learning technique, which is an adaptation of the classical Random Forest algorithm [39] that grows decision trees by training them on re-sampled versions of the original data and randomly selecting a small number of features that can be inspected at each node for split. ARF is based on a warning and drift detection scheme per tree, such that after a warning has been detected for one tree, another one (background tree) starts growing in parallel and replaces the original tree only if the warning escalates to a drift.

Adaptable Diversity-based Online Boosting (ADOB) [40] is a modified version of the online boosting [34], which aims to speed up the recovery of classifiers after concept drifts. It uses ADaptive WINdowing (ADWIN) [35] change detector as its concept drift detector. This algorithm changes the Poisson distribution parameter from a fixed value of 1 to an adjustable value of λ according to the accuracy of its base classifiers, so that the samples can be distributed efficiently among its base classifiers.

2.2.2 Implicit Methods

OzaBag [34] is an implicit method modifying the standard bagging technique for it to work with data streams (i.e. in online environments). In this method, every classifier in the pool is trained with k copies of the data received recently. OzaBoost [34] is an online version of the standard boosting algorithm. In OzaBoost method, each incoming instance is used to train all experts sequentially: the highest possible weight is assigned to the first decision tree, while the weights calculated for the next decision trees are based on the evaluation of the older ones.

OSBoost [41] is an implicit algorithm that uses online boosting and combines weak learners by producing a connection between online boosting and batch boosting algorithms. It has been theoretically proven to achieve a small error rate, as long as the numbers of weak learners and examples are sufficiently large.

Dynamic Weighted Majority (DWM) [42] is an implicit approach for dealing with online data by classifying them immediately. If a classifier misclassifies an instance after a predefined period (p instances), the weight of this classifier is reduced by a constant value regardless of the ensemble's output and all weights are normalised. Then, the classifiers with weights lower than a predefined threshold (θ) are removed from the ensemble. Finally, when the whole ensemble misclassifies an instance, a new classifier is built and added to the ensemble. All classifiers are trained incrementally with incoming samples.

The Accuracy Updated Ensemble (AUE) [43] incrementally trains all old classifiers and weights them based on their error within a constant time and memory. In this algorithm, the incremental nature of Hoeffding trees [44] is combined with a normal block-based weighting mechanism. This approach does not remove any old classifiers; therefore, a threshold for memory is assigned, so that whenever it is met, a pruning method is used to reduce the size of classifiers. An online version of this approach (OAUE) was introduced by the same authors [45].

Anticipative Dynamic Adaptation to Concept Changes (ADACC) [46] is an implicit method that attempts to optimise stability of the ensemble by recognising incoming concept changes. This is achieved by establishing an enhanced forgetting strategy for the ensemble. ADACC takes snapshots of the ensemble when a concept is recognised as *stable* and uses them when there is instability in the system to cope with concept drifts.

Social Adaptive Ensemble (SAE) [47] is a method that has the same learning strategy as that of the DWM algorithm. It maintains an ensemble arranged as a network (undirected graph) of classifiers. Two classifiers are connected to each other when they produce similar predictions. These connections are weighted according to a similarity coefficient equation. The ensemble is updated after a predefined number of instances. The same authors extended their method to *SAE2* [48].

2.2.3 Research Issues and a Proposed Approach

In summary, the main issue with explicit methods is their sensitivity to false alarms (noise). Therefore, accuracy of the system using such methods can be degraded severely by a wrongly detected concept drift. Furthermore, employing a good drift detection mechanism that can recognise different types of concept drifts (gradual, recurring, abrupt and incremental) [15] is a difficult task. Thus, in this thesis, we propose an adapted version of RD offering a smooth yet effective way of improving the performance of the ensemble by increasing or reducing the number of trees in classification *types*. A classification *type* is a set of randomly drawn features (subspaces) of the target data stream used to create a diverse set of learners (classifiers). Furthermore, the main issue with implicit algorithms is their slowness in coping with concept drifts since these algorithms do not react immediately to drifts. This is the reason for using a concept drift detection algorithm in our proposed methods, along with GA to immediately react to concept drifts and optimise the combination of features in classification *types*. Overall, by combining RD with concept drift detection methods and GA or different layers of Particle

Swarm Optimisation (PSO), it is feasible to have the advantages of both explicit and implicit methods alongside in the ensemble.

2.3 Metaheuristic Methods for Concept Drift Adaptation

To achieve the objectives identified in this thesis, different metaheuristic methods are applied to the problem of classification in non-stationary data streams. In this section, we study the existing metaheuristic methods for concept drifts adaptation.

Metaheuristic algorithms cannot be applied in their original state to the problems in streaming applications since the entire set of instances is not accessible to the stream processing system. However, such algorithms can be adapted to streaming data in different ways. In particular, the following algorithms have been proposed in the literature for non-stationary data stream classification.

The StreamGP algorithm [49] builds an ensemble of classifiers using Genetic Programming (GP) along with the boosting algorithm to generate decision trees, each trained on different parts of the data stream. This algorithm is an explicit algorithm that uses a concept drift detection mechanism. Whenever a concept drift is detected, a new classifier is created using CGPC [50], which is a Cellular GP method generating classifiers as decision trees. According to this algorithm, each population is initialised as a set of individual data blocks (nodes) drawn randomly. The newly created classifier is then added to the ensemble, and all classifiers are boosted by updating their weights.

StreamGP is different from the methods proposed in this thesis. In particular, the aim of the optimisation technique used in StreamGP is to find the best set of data blocks to create a new classifier, whereas the aim of the metaheuristic algorithms used in our methods is to optimise the combination of features for the learning purposes.

Online Genetic Algorithm (OGA) [51] is a rule-based learning algorithm that builds and updates a set of candidate rules for a data stream based on the evolution of the data stream itself. In this algorithm, the rules are initially set randomly, and after fully receiving a new data block, an iteration of GA is performed to search for new (better) candidate rules for all classes in the received data block. This process is repeated until the end of the stream.

The differences between OGA and our proposed algorithms are as follows. Primarily, OGA is a rule-based learning algorithm, whereas *EACD*, *RED-GENE* and *RED-PSO* are ensemble learning algorithms. The aim of GA in OGA is to create new rules or update the current rules, whereas the aim of RD and GA in *EACD* and *RED-GENE*, and PSO in *RED-PSO* is to optimise the classification *types* inside the ensemble. Furthermore, the iterations in OGA are performed over different data blocks (an iteration per each data block) and GA never stops its iterations (the maximum number of generations is unlimited), whereas in *EACD* and *RED-GENE*, the iterations are performed over the same fixed data in the buffer for each round of GA, and the number of generations is limited. The main issue with OGA is the long time it takes to adapt to new concept drifts since GA takes only one data block at each iteration, potentially requiring a large number of iterations to completely cope with a concept drift. Further details regarding the methods proposed in this thesis, namely, EACD, RED-GENE and RED-PSO, are provided in Chapters 5, 6 and 7, respectively.

2.4 Ensemble Dynamics in Non-stationary Data Stream Classification

In non-stationary environments, where different types of concept drifts may happen, it is expected that an ensemble adapts to a new concept drift swiftly. Since the adaptation in such environments is typically achieved by adding a new classifier to the ensemble, removing old classifiers and changing the weights of the existing classifiers, understanding the dynamic behaviour of the ensemble towards different types of concept drifts can help us to choose the best approach for a specific application domain and develop new ensemble learning techniques for the required purpose. This section discusses the operations that form the ensemble dynamics of an approach. These operations are adding, removing and updating classifiers in the ensemble. The following sections describe each of these operations in detail and outline the comparison between different algorithms based on the criteria related to dynamics. Over 20 different ensemble methods for non-stationary environments are studied and compared for this purpose.

2.4.1 Adding Classifiers

Adding new classifiers trained with recent instances in a stream is one of the most important operations that should be applied to the ensemble when data is evolving. The aim of this operation is adapting to drifting data, as well as improving classification accuracy of the ensemble based on the fact that, in most cases, incoming data are more likely to be similar to upcoming instances. One decision that should be taken when making a strategy for the ensemble is to decide when to add new classifiers to the ensemble, or in other words, what time frame should be taken for the addition operation. Some algorithms use a *fixed* time of addition, while others use a *dynamic* time for it.

Fixed Time of Addition

The algorithms using a fixed time to train and add new classifiers usually use a similar strategy; they perform the addition operation after receiving a new block of data or after receiving a predefined p instances. A considerable number of the existing algorithms use this strategy to add new classifiers. The main challenge when building or using such algorithms is to pick a decent size of *blocks*, or p, to have the best possible output. Picking a large size might decelerate the adaption, while using a small size might make the ensemble sensitive to noise.

Dynamic Time of Addition

The algorithms using a dynamic time to train and add new classifiers are more diverse than the ones using a fixed time. Some of them use a method based on the concept drift detection to determine when to train and add new classifiers. These types of algorithms start to train a new classifier when the concept drift detector signals and identifies a concept drift. Such a mechanism is called the *detection-based dynamic* approach to the addition operation. Some algorithms start to build a new classifier once the ensemble misclassifies an example. This strategy is called in this thesis *misclassification-based dynamic*. Other mechanisms include adding a new classifier based on an acceptance factor [52]. This approach tries to add a new classifier when the threshold of the acceptance factor has been passed and a new classifier is required. Another approach trains and adds a new classifier once an old classifier is removed and some free space is available [53].

All studied algorithms and their addition mechanisms are listed in Table 2.1.

2.4.2 Removing Classifiers

Removing classifiers is a strategy to forget the knowledge previously gained from a data stream that is outdated in the current situation to adjust the ensemble to an updated state. In many cases, removing classifiers from the ensemble is performed when a predefined ensemble size is reached. However, in some algorithms, classifiers are removed when their accuracy drops below a predefined threshold. In yet other algorithms, the size of ensemble is set unlimited, hence no classifiers are eliminated from the ensemble unless a pruning method is utilised. In summary, the removing strategies can be categorised into the following four types (Table 2.1):

Full

Remove classifier(s) when a predefined ensemble size is reached and there is a new classifier required to be added to the ensemble. The algorithms employing this strategy eliminate classifiers based of the classifiers' age in the ensemble or their performance over recent data. These algorithms typically use a 'fixed' strategy for adding new classifiers.

Performance-based

Remove a classifier when its performance for the last predefined k examples drops below a specified threshold. When a classifier becomes 'inaccurate' in identifying a new concept, it is considered as an obsolete classifier and removed from the ensemble.

Drift-detection-based

Remove classifier(s) once the concept drift detection method identifies a concept drift. When the ensemble is full, one or multiple classifiers are eliminated from the ensemble for every new concept drift detected by the drift detection method. This is performed to forget the past concept of data and/or emphasise the new classifier(s) added to the ensemble. All of such algorithms use a dynamic mechanism for adding new classifiers.

No removal

A considerable number of algorithms do not remove any old classifiers from the ensemble and only change the weights of all classifiers to avoid inaccurate classification. This strategy is driven by the believe that when a classifier becomes weak in an environment, it can become an accurate classifier once again when a drift happens, especially when the drift is recurring. The algorithms using this strategy typically have a pruning method in place to avoid memory overload (providing that no classifier is removed from the ensemble).

2.4.3 Updating Classifiers

Updating classifiers in the ensemble can be performed using two main approaches: either updating the weight/ranking of each classifier or training old classifiers with incoming data. Providing that the latter approach requires a lot of memory to train all classifiers with incoming data, the majority of the current algorithms use the 'updating weights' mechanisms to improve accuracy (Table 2.1).

Updating the weight of each classifier is an efficient way to improve the accuracy of the ensemble, especially when a concept drift happens and there are diverse classifiers in the ensemble. This is usually achieved by evaluating the effectiveness of each classifier providing the current state of the environment and changing their weights/rank so that classifiers with a higher accuracy towards the current condition have a bigger impact on the ensemble's output than weaker classifiers. Note that the algorithms using a simple majority voting mechanism for selecting the output of the ensemble are unable to employ this procedure since there is no weight or rank set for each classifier. Similar to the addition stage, the mechanisms for updating the weights of classifiers can be categorised into those using *fixed* times and those using *dynamic* times. The methods using dynamic times for updating classifiers are typically employed when a drift is detected, except for the AddExp algorithm [54], where updating is performed when a classifier misclassifies an example.

2.4.4 Ensemble Dynamics Taxonomy

To summarise the above-mentioned operations, we propose a taxonomy for defining ensemble's dynamics in non-stationary data stream classification

(Figure 2.1). According to the proposed taxonomy, the dynamic behaviour of ensemble techniques is categorised into three main operations of addition, removal and updating. The addition mechanisms are partitioned into fixed and dynamic methods, with the dynamic methods being further divided into detection-based, performance-based and others (such as using an acceptance factor). The removal techniques are partitioned into four categories: full, performance-based, detection-based and no-removal. Finally, the updating approaches are divided into two categories: updating the classifiers' weights (or ranks) and training old classifiers. The first updating category is partitioned into those with fixed times, dynamic times and no-update, while the second category includes algorithms that train old classifiers (yes) and those that do not (no).

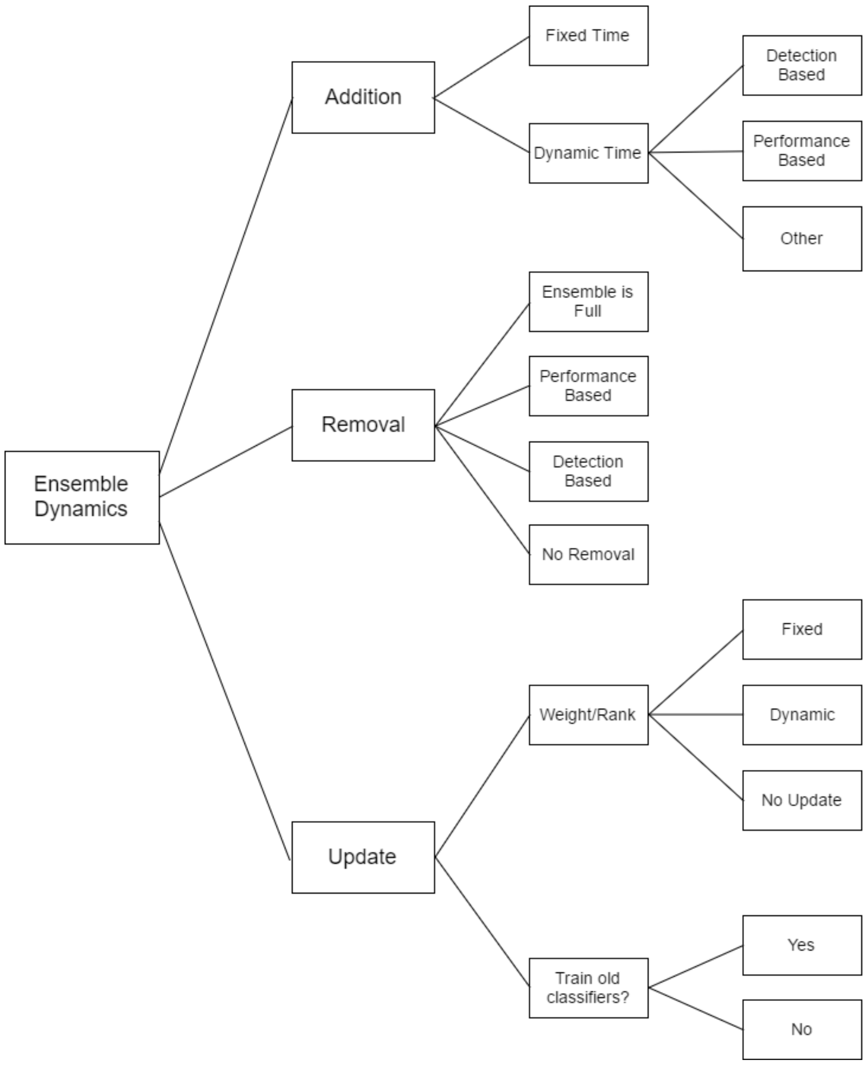


FIGURE 2.1: Proposed taxonomy for ensemble's dynamics in non-stationary data stream classification.

TABLE 2.1: Overview of the dynamic behaviour of studied algorithms

Algorithm	Adding	Removing	Updating	Training	Reference
SEA	Fixed	Full	No update	No	[55]
AWE	Fixed	Performance	Fixed	No	[56]
CDC	Other	Performance	Fixed	No	[53]
Aboost	Fixed	Full	Dynamic	No	[30]
CBEA	Fixed	Full	No update	No	[57]
AddExp	Misclassify	No removal	Dynamic	No	[54]
ACE	Detection	No removal	Dynamic	No	[58]
DWM	Misclassify	Performance	Fixed	Yes	[42]
TRE	Other	No removal	Fixed	No	[52]
Adwin Bag	Detection	Detection	No update	No	[33]
BWE	Detection	Detection	Fixed	No	[59]
Learn++	Fixed	No removal	Fixed	No	[60]
Heft-Stream	Fixed	Full	Fixed	No	[61]
WAE	Fixed	Full	Fixed	No	[62]
RCD	Detection	No removal	Dynamic	No	[37]
DACC	Fixed	Full	Fixed	Yes	[46]
ADACC	Fixed	Full	Fixed	Yes	[46]
AUE	Fixed	Full	Fixed	Yes	[43]
OAUE	Fixed	Full	Fixed	Yes	[45]
Fast-AE	Fixed	Full	Fixed	No	[63]

Legends. Fixed*: Fixed time of adding/updating classifiers;* Detection*: Detection-based (dynamic) times;* Misclassify*: Misclassification-based (dynamic) times of adding classifiers;* Full*: Removing old classifier(s) when the ensemble is full;* Performance*: removing when the performance of a classifier drops below a predefined threshold.*

2.5 A Generic Formal Description of Non-stationary Data Stream Classification Methods

Formalising algorithms is a suitable way to comprehend and modulate the existing approaches to develop novel methods. In this section, a formalised version of six different algorithms introduced earlier in this section is presented with the intention to simplify the process of examining and building new approaches. These six algorithms are chosen based on their dynamic behaviour covering a variety of mechanisms towards concept drift adaptation. At the same time, the proposed formalisation technique can be applied to any other algorithms.

The following functions are used in the chosen algorithms. Note that the sequence of the functions is important in the proposed formalisation, and the specific implementation of each function might be different for every algorithm.

- *Classify()*: The ensemble classifies data according to its combinational rule (e.g. weighted majority vote or majority vote).
- *Eval()*: Evaluating the whole ensemble or classifiers using an evaluation method.
- *Update()*: Updating the weights (or ranks) of all or one classifier using evaluation and updating mechanisms.
- *Build()*: Building a new classifier using recently received data.
- *Add()*: Adding the newly built classifier to the ensemble.
- *Remove()*: Removing one or some classifiers based on the ensemble's specific removal mechanism.
- *Train()*: Training all or some old classifiers using new data or a data block.
- *DriftDetection()*: Detecting drifts using a concept drift detection method.

Aboost [30] takes blocks of data, classifies the instances and evaluates the ensemble's performance (see Algorithm 1). If a concept drift is detected, Aboost updates all classifiers and assigns the default weight of one to them. Otherwise, it assigns a weight to each instance in the block according to whether or not the considered instance is classified correctly (lines 8-9 in Algorithm

1). High weights are assigned to misclassified instances, while the default weight of one is assigned to all instances classified correctly. Finally, the oldest classifier in the ensemble is removed and a new classifier is built and added to the ensemble (based on the weighted instances in the block).

Algorithm 1: ABOOST Adaptive Boosting Algorithm

```
Input: Continuous data blocks, $DB = \{db_1, db_2, .., db_n\}$
Output: C: A set of classifiers $c = \{c_1, c_2, .., c_m\}$ and their corresponding
        weights $w = \{w_1, w_2, .., w_m\}$
1  $i := 1$
2  while data stream is not empty do
3      Classify($db_i$)
4      Eval(Ensemble)
5      if DriftDetection()=1 (drift is detected) then
6          Update(c)
7      else
8          Eval($db_i$)
9          Update($db_i$)
10     Build($c_{i+1}$)
11     Add($c_{i+1}$)
12     Remove() //remove the oldest classifier
13     $i = i + 1$
```

In DWM [42], data arrive in real time and after a predefined period p (see Algorithm 2). If a classifier misclassifies an instance, the weight of this classifier is reduced by a constant value regardless of the ensemble's output (lines 8-9 in Algorithm 2). After the predefined period p, all weights are normalised and the classifiers with weights lower than a threshold (θ) are removed from the ensemble. Finally, when the ensemble misclassifies an instance, a new classifier is built and added to the ensemble. All classifiers are trained incrementally with incoming samples.

In TRE [52], a new classifier is added only when the ensemble error reaches a predefined permitted error rate (τ). Each classifier's weight is updated once the classifier's performance drops below an acceptance factor (θ). This approach does not remove old classifiers unless a pruning method is used. The formalised version of this algorithm is shown in Algorithm 3

AdwinBag [33] uses a concept drift detection method to specify when a new classifier is needed, in which case, the worst performing classifier is removed to make a room for a new classifier. The formalised version of this algorithm is shown in Algorithm 4.

Algorithm 2: DWM Dynamic Weighted Majority algorithm

```
   Input: A data stream, $DS = \{d_1, d_2, .., d_n\}$
   $l_i$: Real label of the $i^{th}$ example
1  $\Theta$: Threshold for removing classifiers
2  $p$: specified period for adding, removing and updating classifiers.
   Output: A set of classifiers $c = \{c_1, c_2, .., c_m\}$ and their corresponding
           weights $w = \{w_1, w_2, .., w_m\}$
3
4  $i := 1$
5  while data stream is not empty do
6      for $j = 1$ to $j = m$ do
7          Classify($d_i$)
8          if Output($b_j$) $\neq l_i$ and $i$ mod $p = 0$ then
9              Update()
10         if $i$ mod $p = 0$ then
11             while $w_j < \theta$ do
12                 Remove($c_j$)
13         Train($c_j$)
14     if Classify($d_i$) $\neq l_i$ then
15         Build()
16         Add()
17     $i := i + 1$
```

Algorithm 3: TRE Tracking Recurrent Ensemble

```
   Input: Continuous data blocks, $DB = \{db_1, db_2, .., db_n\}$
   $\tau$: Permitted error $\theta$: Acceptance factor
   Output: A set of classifiers $c = \{c_1, c_2, .., c_m\}$ and their corresponding
           weights $w = \{w_1, w_2, .., w_m\}$
1  $i := 1$
2  while data stream is not empty do
3      Classify($db_i$)
4      for $j = 1$ to $j = m$ do
5          Eval($c_j$)
6          if Eval($c_j$) $< \theta$ then
7              Update($c_j$)
8          Eval(Ensemble)
9          if Ensemble error $> \tau$ then
10             Build()
11             Add()
12     $i := i + 1$
```

Algorithm 4: ADWINBAG Adwin Bagging algorithm

```
Input: A data stream, $DS = \{d_1, d_2, .., d_n\}$
M: Ensemble size
Output: A set of classifiers $c = \{c_1, c_2, .., c_m\}$
1  $i := 1$
2  while data stream is not empty do
3      Classify($d_i$)
4      if DriftDetection()=1 then
5          Build()
6          Add()
7      for $j = 1$ to $j = m$ do
8          Eval($c_j$)
9      if Ensemble size = M then
10         Remove() //remove worst performing classifier
11     $i := i + 1$
```

RCD [37] uses a buffer to store the context related to each data distribution in the stream (see Algorithm 5). When the concept drift detector signals a warning, a new classifier is created and trained alongside with a new buffer. If the concept drift detector signals a drift, which means the concept drift is certain, RCD checks whether or not the new concept drift is similar to previous concepts in the buffer (in case it is a recurring concept drift). If the new concept is similar to an old concept based on a statistical test, RCD uses the classifier created with that concept to classify incoming data and starts training this classifier. If the data distribution (concept) is new, RCD stores the new buffer and classifier in the system and uses the new classifier to classify incoming data. Otherwise (i.e. if the signal was a false alarm), the system ignores the stored data and continues to classify using the latest classifier. In this approach, only one classifier is active at a time performing the classification task.

OAUE [45] is designed to incrementally train all old classifiers and weight them based on their error rate in a constant time and memory. Since this algorithm requires a lot of memory due to training all classifiers with incoming data, a threshold for memory is assigned, so that whenever the threshold is met, a pruning method is used to decrease the size of the classifiers. The formalised version of OAUE is presented in Algorithm 6.

Algorithm 5: RCD Recurring Concept Drift method

```
Input: A data stream, $DS = \{d_1,d_2,..,d_n\}$
Output: A set of classifiers $c = \{c_1,c_2,..,c_m\}$, Buffer list $b = \{b_1,b_2,..,b_m\}$
$c_a$= Active classifier, $b_a$= Active buffer
$c_n$= New classifier, $b_n$= New buffer
$i := 1$
while data stream is not empty do
    Classify($d_i$)
    DriftDetection()
    switch Drift Detection do
        case DriftDetection()= Warning and $c_n = null$ do
            Build($c_n$)
            Build($b_n$)
        case DriftDetection()= Warning and $c_n \neq null$ do
            Train($c_n$)
        case DriftDetection()= Drift do
            if Statistic-Test()= 1 then
                $c_a \leftarrow c_m$
                $b_a \leftarrow b_m$
            else
                $c_a \leftarrow c_n$
                $b_a \leftarrow b_n$
        otherwise do
            $c_n = b_n = null$
    $i := i + 1$
```

Algorithm 6: OAUE Online Accuracy Updated Ensemble algorithm

```
Input: A continuous blocks of data, $DB$ ={$db_1$,$db_2$,..,$db_n$}
M: Ensemble size, $\theta$: Memory threshold
Output: A set of classifiers $c$ ={$c_1$,$c_2$,..,$c_m$} and their corresponding
        weights $w$ ={$w_1$,$w_2$,..,$w_m$}
1  $i := 1$
2  while data stream is not empty do
3      Classify($db_i$)
4      Eval(c)
5      Build($c_i$)
6      if $i < M$ then
7          Add($c_i$)
8      else
9          Remove() //remove the least accurate classifier
10         Add($c_i$)
11     for $j = 1$ to $j = m$ do
12         Update($c_j$)
13         Train($c_j$)
14     if Memory usage $> \theta$ then
15         Prune(c) //decrease the size of classifiers
16     $i := i + 1$
```

2.6 Discussion

Analysing existing solutions is the first step towards proposing novel methods. Studying the dynamic behaviour of different ensemble learning techniques for non-stationary data stream classification and categorising them based on their dynamic behaviour have helped us to identify several research gaps. In particular, none of the reviewed methods offers a comprehensive solution to stream mining that is (i) able to cope with different types of concept drifts; (ii) resistant to noise and false alarms; and (iii) fast in adapting (reacting) to all types of concept drifts. Aiming at addressing these shortcomings, this thesis proposes three novel approaches to analysing evolving data streams based on several metaheuristic algorithms, namely, RD, GA and PSO. These algorithms can offer natural solutions to the changing environment problems with promising characteristics such as fault-tolerance, self-replication or cloning, reproduction, evolution, adaptation, learning and growth.

To the best of our knowledge, metaheuristic algorithms have been applied to this research area with a limited capacity. In particular, only two methods, namely, StreamGP [49] and OGA [51], are identified to propose solutions to the concept drift adaptation problem based on GP and GA. However, these methods are different from the ones proposed in this thesis in the following aspects. As the original metaheuristic algorithms cannot cope with the online nature of data streams, novel modifications of those algorithms are proposed in this thesis with the goal to optimise the combination of features used for building different classifiers inside the ensemble. In StreamGP, GP is used to select a data block (out of the available data blocks) to train a new classifier based on it, while in OGA, which is a rule-based learning algorithm, GA is used to create new rules and update the current ones. These applications of GP and GA differ from the main goal of ensemble learning, that is to create a pool of classifiers for the classification purpose. In particular, the main drawback of StreamGP is that no new classifier is created by the system unless a concept drift is detected. This might negatively affect the performance upon incremental and gradual concept drifts that are hard to detect. The main drawback of OGA is the long time it takes to adapt to new concept drifts since GA takes only one data block in each iteration, potentially requiring a large number of iterations to completely cope with a concept drift.

In summary, this thesis proposes for the first time some modifications of classical metaheuristic methods for performing online feature selection and solution adaptation when solving data stream classification tasks.

2.7 Summary

This chapter offered an overview of different ensemble-based methods proposed in the literature for solving non-stationary data stream classification tasks and studied their mechanisms for adapting to dynamic changes of data streams. Furthermore, a novel taxonomy was proposed based on the dynamic behaviour of these methods to help identifying different types of reactions to concept drifts. To simplify the process of understanding the dynamics of the current approaches and encourage the development of novel algorithms, a formalisation method for studying classification algorithms for streaming analytics was presented. The characteristics of some of the existing algorithms were investigated using this formalisation. Finally, this chapter discussed the contributions of the thesis in the light of the reviewed literature, demonstrating the originality of the proposed methods. The next chapter provides a detailed overview of the original classification technique and different metaheuristic algorithms employed in the methods proposed in this thesis.

Chapter 3

Background

In the previous chapter, a detailed study of the state-of-the-art work in the area of non-stationary data stream mining has been presented, along with a novel taxonomy characterising dynamic behaviours of the existing ensemble learning methods and a novel formalisation technique for describing non-stationary data stream classification methods. Understanding different classification methods and their dynamic behaviour has resulted in understanding how different mechanisms address the challenges in non-stationary data stream mining, which eventually has led to developing the novel methods presented in this thesis to address the shortcomings of the current algorithms. This chapter provides the background necessary to understand the proposed solutions, along with the theoretical explanation of the algorithms employed in these solutions. In particular, the algorithms introduced in this chapter are either used or modified in the methods presented in Chapters 5, 6 and 7.

3.1 Data Classification

The main focus of this thesis is the classification task in non-stationary data stream environments. This problem, in its general representation, can be defined as a set of N instances in the form of (x, y), where x denotes a set of numeric or qualitative features (attributes) d of an instance and y denotes the corresponding class label of this instance. The goal of the classification task is to train a model using the provided N instances to predict the class label y of future examples x.

There are many classification algorithms offering different ways of building and training a classifier, including Decision Tree [64], Naive Bayes [65], Artificial Neural Network (ANN) [66], Support Vector Machines (SVM) [67] and k-Nearest Neighbours (k-NN) [68].

The Decision Tree algorithm is one of the most effective and popular classification methods [44] extensively used for creating ensembles. Providing

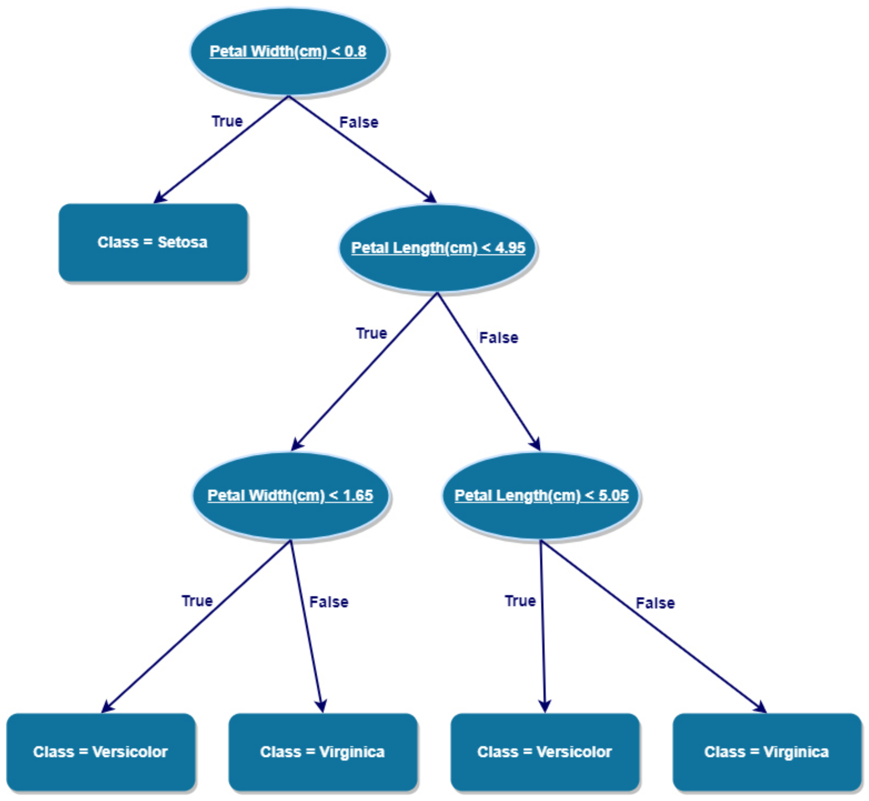


FIGURE 3.1: Illustration of a sample decision tree built over the Iris flower dataset.

that the majority of stream mining algorithms are ensemble-based, we used this algorithm as a base of the methods proposed in this thesis. Learners of this type induce models in the form of decision trees, where each external node (leaf) contains a class prediction and each internal node (non-leaf) is labelled with an input feature. The predicted label of an instance, $y = DT(x)$, is obtained by passing the instance through the tree starting from the root until it reaches a leaf. The reached leaf is then assumed to be the predicted value (y) of the instance. Figure 3.1 illustrates a sample decision tree built using the Iris dataset [69]. The Iris dataset contains three classes with 50 instances each, where each class refers to a type of the iris plant. Four features were measured from each sample, namely, the length and width of sepals and petals in centimetres[1].

[1]https://archive.ics.uci.edu/ml/datasets/iris

3.1.1 Classic Decision Tree Learners

Classic decision tree learners such as the Iterative Dichotomiser 3 (ID3) [70], C4.5 [71] and CART [72] assume that all training instances can be stored simultaneously in the main memory and are thus severely limited in the number of examples they can learn from. As an example, the ID3 algorithm considers an entire set S to set the root node. It repeats through every unused feature of the set S and measures the entropy $H(S)$ or information gain $IG(S)$ of that feature. The algorithm then chooses the feature with the largest information gain (or smallest entropy) value. Next, the set S is partitioned according to the selected feature to create subsets of the original data. As an example, a node in this algorithm can be split into child nodes based on the subsets of the population whose ages are less than 10, between 10 and 20, and greater than 20.) The algorithm proceeds to iterate on each subset, considering only the features that have never been chosen before.

$H(S)$ is a measure of the amount of uncertainty in the (data) set S calculated as in Equation 3.1.

$$H(S) = \sum_{x \in X} -P(x) \log_2 P(x), \tag{3.1}$$

where S denotes the current dataset for which entropy is being calculated, X denotes the set of classes in S and $P(x)$ denotes the proportion of the number of elements in class x to the number of elements in the set S.

When $H(S) = 0$, the set S is ideally classified (i.e. all elements in S are of the same class). In ID3, entropy is calculated for each remaining feature. The feature with the largest information gain is used to split the set S on this iteration.

Information Gain $IG(A)$ is the measure of the difference in entropy before to after the set S is split on a feature A. It measures how much uncertainty in S has been demoted after splitting the set S on feature A. $IG(A)$ can be calculated as in Equation 3.2.

$$IG(S, A) = H(S) - \sum_{t \in T} p(t)H(t) = H(S) - H(S|A), \tag{3.2}$$

where $H(S)$ denotes the entropy of the set S, T denotes the subsets created from splitting the set S by attribute A, $p(t)$ denotes the proportion of the number of elements in the subset t to the number of elements in the set S and $H(t)$ denotes the entropy of the subset t.

In ID3, information gain is calculated instead of entropy for each remaining attribute. The attribute with the largest information gain value is used to split the set S in the current iteration.

Iterations on a subset of data can terminate in one of the following cases:

- Every instance in the subset refers to the same class, in this case, the node is turned into a leaf node and identified as the class of the instances.
- There are no more features to be picked while the instances still do not refer to the same class. The node, in this case, is made a leaf node and labelled with the most common class of the instances in the subset.
- There are no instances in the subset. This is when no instance in the parent set was found to match a specific value of the selected feature. In this case, a leaf node is created and labelled with the most common class of the instances in the set of the parent node.

The output of ID3 algorithm is a decision tree with each non-terminal node (internal node) representing the selected feature on which the data was split and terminal nodes (leaf nodes) representing the class label of the final subset of this branch.

In the context of data stream classification, the learner can only process each instance once on its arrival to the system and the processed data are not accessible in the future due to the high-volume of data and possible infinite size of the entire dataset. Hence, ID3 is not suitable for data stream mining.

3.1.2 Hoeffding Trees

In contrast to ID3, the Hoeffding Tree algorithm [44] is an effective online learning mechanisms proposed to build potentially complex decision trees based on data streams. In particular, the first few instances of a given stream of data are used to choose the root attribute of the decision tree, while the succeeding instances are used to incrementally grow the tree employing the Hoeffding bound [73] to decide on optimal splitting features.

Consider a real-valued random variable r whose range is R (e.g. for information gain, the range is $\log c$, where c is the number of classes). Suppose we have made n independent observations of this variable and computed their mean $\bar{r}$. According to the definition of the Hoeffding bound, the true mean of the variable is at least $\bar{r} - \epsilon$ with the probability $1 - \delta$, where ϵ is calculated as in Equation 3.3.

$$\epsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}}. \tag{3.3}$$

The Hoeffding bound has a property of being independent of the probability distribution generating the observation. The price of this generality is that the bound is more conservative than distribution-dependent ones. Let $G(X_i)$ be the heuristic measure used to choose the best splitting attributes (e.g. the measure could be information gain as in ID3). The goal of the Hoeffding Tree algorithm is to ensure with a high probability that the attribute chosen using n instances (where n is as small as possible) is the same as that would be chosen using infinite examples.

3.2 Replicator Dynamics

RD is a simple model of evolutionary game-based learning in game theory [22][23]. Game theory is defined as the study of mathematical models of conflict and cooperation between intelligent rational decision makers. It provides general mathematical techniques for analysing the situation, in which two or more individuals (players) make decisions impacting the welfare of others [74]. It was appropriately fostered by evolutionary biologists[2]. The resulting population-based evolutionary game theory has been applied to many non-biological fields such as economics and learning theory, presenting an important enrichment of classical game theory, which is centred on the concept of a rational individual. The following sections discuss the mathematical fundamentals of the evolutionary game concepts, namely, the **Nash Equilibrium (NE)** and **Replicator Equation**.

3.2.1 Nash Equilibrium

The simplest kind of games in the game theory has only two players, I and II, each one has a limited set of options or *pure strategies*, Strat(I) and Strat(II), respectively. We denote the *payoff*, or expected value of the players I and II as a_{ij} and $b_i j$, when I uses a strategy $i \in Strat(I)$ and II uses a strategy $j \in Strat(II)$. Hence, the payoffs can be represented as $n \times m$ matrices A and B, where n and m denote the cardinalities of the sets of pure strategies.

[2]William D. Hamilton and John Maynard Smith

The mixed strategy of the player I considering $i \in Strat(I)$ with the probability x_i can be defined as a vector $x = (x_1, x_2, ..., x_n)^T$, which is an element of the unit simplex S_n spanned by the vectors e_i of the standard unit base; these vectors can be identified using the elements of $Strat(I)$. Similarly, the unit simplex S_m spanned by the vectors f_j corresponds to the set of mixed strategies for the player II. If the player$I$ uses $x \in S_n$ and the player II uses $y \in S_m$, then the former has the expected payoff $x^T Ay$ and the latter has the expected payoff $x^T By$. The strategy $x \in S_n$ is said to be the best reply to $y \in S_m$ if the condition in Equation 3.4 is satisfied for all $z \in S_n$.

$$z^T Ay \leq x^T Ay \tag{3.4}$$

The set of all best replies to y is denoted as $BR(y)$. A pair $(x, y) \in S_n \times S_m$ is a $Nashequilibrium(NE)$ if $x \in BR(y)$ and (with the obvious abuse of the notation) $y \in BR(x)$. A simple fixed-point argument shows that such a NE always exist. The pair is considered a strict NE if x is the unique best reply to y and vice versa. In this sense, such an outcome satisfies the consistency condition. To transfer this to a population setting, it is convenient at first to reduce attention to the case, where the two players I and II are interchangeable individuals within the population, i.e. consider only the case, where the two players do not appear in different roles but have the same strategy set and payoff matrix. In particular, we can first consider symmetric games defined as $Strat(I) = Strat(II)$ and $A = B^T$. For symmetric games, players cannot be distinguished and only symmetric pairs (x, x) of strategies are of interest. Therefore, we can say that a strategy $x \in S_n$ is a NE if the condition in Equation 3.5 is satisfied for all $z \in S_n$, i.e. if x is the best reply to itself.

$$z^T Ax \leq x^T Ax \tag{3.5}$$

The equilibrium is said to be *strict* if it equally holds only for $z = x$.

3.2.2 Replicator Equation

Now a population consisting of n types can be considered. Let x_i be the frequency of type i. Then, the state of the population can be defined as $x \in S_n$. Assume that x_i are differentiable functions of time t (which requires to assume that the population is infinitely large or that x_i are expected values for an ensemble of populations). We can then postulate the following law of motion for $x(t)$. If individuals meet randomly and engage in a symmetric game with the payoff matrix A, then $(Ax)_i$ is the expected payoff for an individual of type i and $x^T Ax$ is the average payoff in the population state x.

In this model, the act of selection happens at discrete times, and the population of each *type* in the next selection can be determined according to the replicator equation as a function of the *type's* payoff and its current proportion in the population [24]. In other words, a *type's* expected payoff can be determined using the payoff matrix, and hence, the population of each *type* can be determined according to its expected payoff. The *types* scoring above the average payoff increase in population, while the *types* scoring below the average payoff decrease in population. The *Replicator Equation* can be written as in Equation 3.6.

$$\dot{x}_i = x_i[(Ax)_i - x^T Ax], \tag{3.6}$$

where $(Ax)_i$ denotes the expected payoff for an individual and $x^T Ax$ denotes the average payoff in the population state x.

3.3 Genetic Algorithm

GA is a metaheuristic algorithm inspired by the process of natural selection, which is a subset of a bigger class of algorithms called evolutionary algorithms. These algorithms are commonly used to generate high-quality solutions to optimisation and search problems relying on bio-inspired operators such as *mutation*, *crossover* and *selection*. We applied GA in the methods proposed in this thesis due to it being superior to other optimisation methods when there is a relatively large number of local optima [75]. Using feature subspaces to optimise accuracy in a classification problem can typically form local optima. Since the proposed methods use subspaces of features, we hypothesise that GA can be a suitable solution to this problem (refer to Chapters 5 and 6 for further details).

A typical GA works as follows (Figure 3.2). An initial **population** is created from a group of individuals randomly. The individuals (**chromosomes**) in the population are then evaluated. An individual is characterised by a set of parameters (variables) known as **Genes**. The evaluation function is provided by the programmer and gives the individuals a score based on how well they perform at the given task. Some individuals are then selected based on their fitness; the higher the fitness, the higher the chance of being selected. These individuals then *reproduce* by exchanging their genes with another individual to create one or more offspring. This process is called crossover.

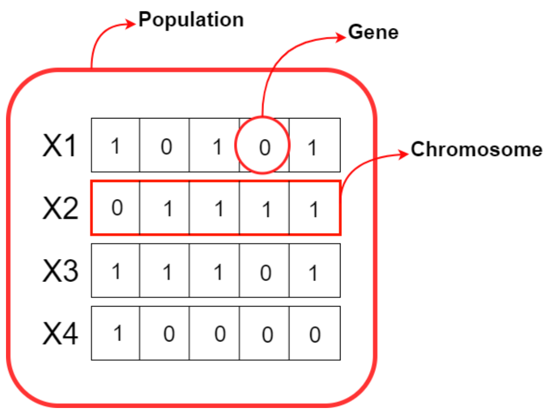


FIGURE 3.2: Illustration of Population, Chromosomes and Genes used in Genetic Algorithms (GA).

Mutation is then applied by adding random genes to the offspring to maintain the diversity within the population. This process continues until a suitable solution is found or a predefined maximum number of generations is reached [76].

The following operations are processed in a typical GA.

Fitness Function

The fitness function determines how fit an individual is (the ability of an individual to compete with other individuals). It is used to assign a fitness score to each individual. The probability that an individual is selected for reproduction is based on its fitness score.

Selection

The purpose of the selection phase is to select the fittest individuals and let them pass their genes to the next generation. The fittest individuals are selected based on their fitness scores. Individuals with high fitness have more chances to be selected for reproduction.

Crossover

Crossover is the most significant phase of GA. For each pair of parents to be mated, a crossover point is chosen at random within the genes. Processing this operation leads to creating offspring by exchanging the genes of parents among themselves until the crossover point is reached (Figure 3.3). The new offspring are then added to the population.

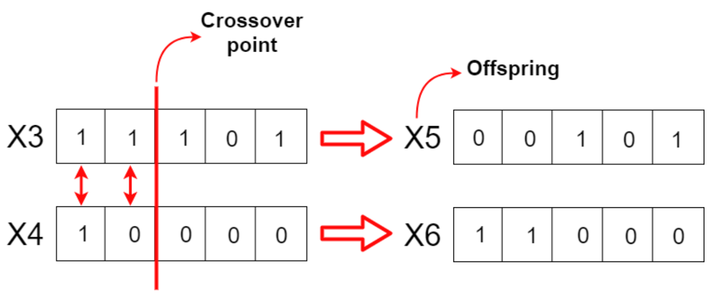


FIGURE 3.3: A sample crossover operation of GA leading to new offspring.

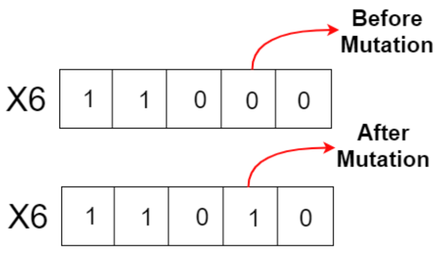


FIGURE 3.4: Before and after the mutation operation of GA.

Mutation

Some genes of the new offspring may be subjected to a mutation with a low random probability. This implies that some of the bits in the bit string of an offspring can be flipped (Figure 3.4). Mutation is applied to maintain the diversity within the population and prevent premature convergence. Figure 3.5 demonstrates how such a typical GA process works.

3.4 Particle Swarm Optimisation

PSO is a metaheuristic algorithm [27] inspired by the social behaviour of the movement of organisms in a bird flock or fish school. The main target of the PSO algorithm is finding the global minimum of a function. While PSO does not guarantee an optimal solution, it has been demonstrated to provide promising results in various applications [28].

A typical PSO algorithm is initialised by creating an initial random population (swarm) of candidate solutions (particles). The particles then move around the search space with a dynamic velocity (according to a specific formulae) to find the best possible solution. The movement of the particles is directed by two factors called 'local best' (Lbest) and 'global best' (Gbest) positions being updated from one iteration to another. The Lbest position is a particle's best position throughout the history, while the Gbest position is the

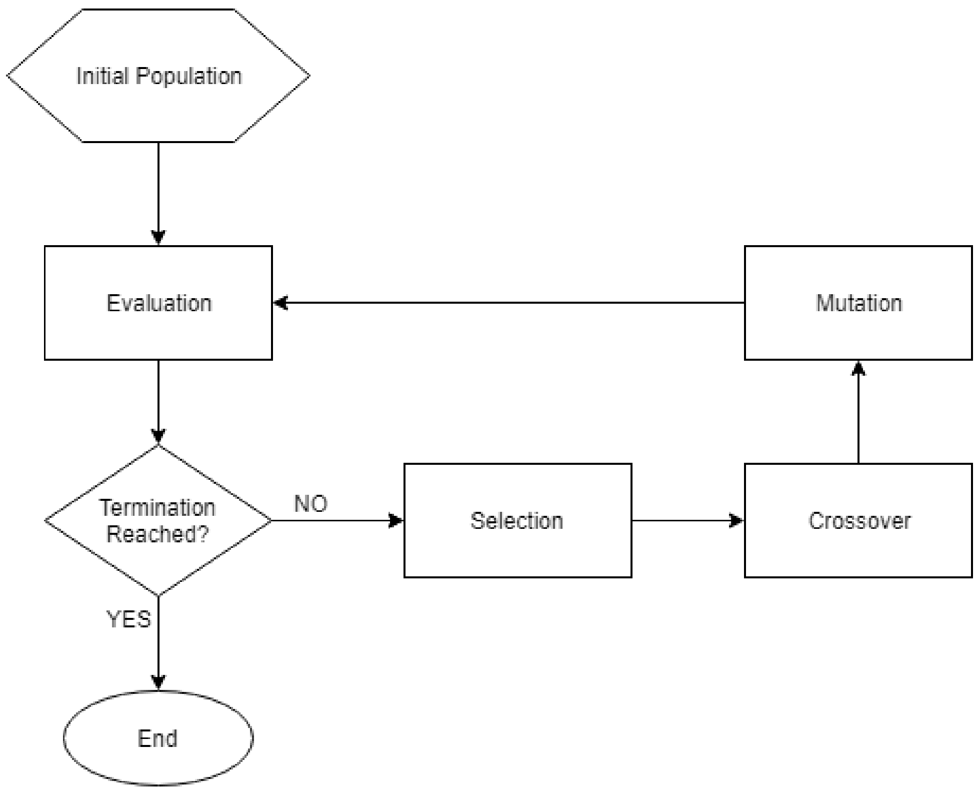


FIGURE 3.5: Illustration of a typical GA.

best position achieved by all particles in the swarm. This process is repeated at each iteration so that a satisfactory solution can be discovered. The velocity (V) and position (P) of the particles are updated according to Equations 3.7 and 3.8 respectively.

$$V_i(t+1) = \omega V_i(t) + \beta_1(Lbest(i,t) - P_i(t)) + \beta_2(Gbest(t) - P_i(t)), \tag{3.7}$$

$$P_i(t+1) = P_i(t) + V_i(t+1), \tag{3.8}$$

where ω denotes the inertia weight used to balance the global and local exploitation, while β_1 and β_2 are positive constant parameters called acceleration coefficients (inertia weights).

Figure 3.6 demonstrates how a particle (X) moves during a period from time i to $i+1$ based on the Gbest position, Lbest position and inertia weight.

A canonical PSO algorithm is designed to iterate over a static data, where there is only one possible optimal solution. In contract, in data stream classification tasks, data come in an online manner and the optimal solution is subject to change over time. Therefore, a non-canonical version of PSO is proposed in Chapter 7 to adapt PSO to streaming environments.

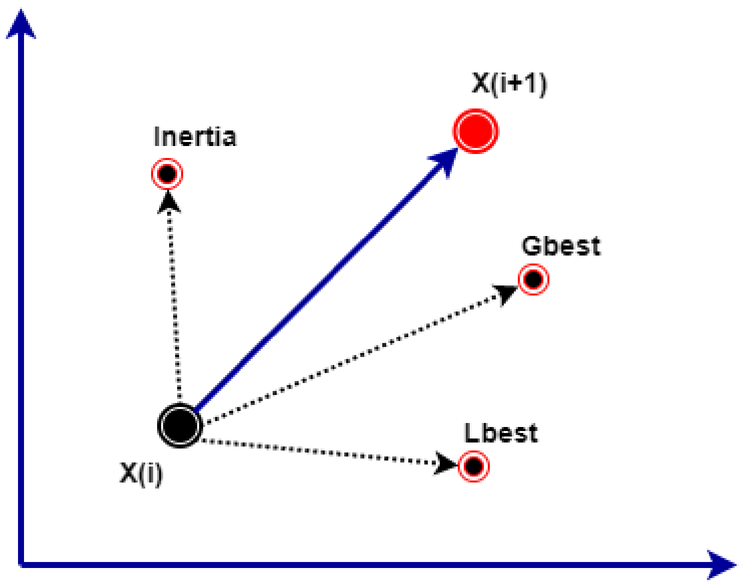


FIGURE 3.6: Illustration of a typical movement of a particle (X) according to the Particle Swarm Optimisation algorithm during a period from time i to $i+1$. Gbest and Lbest denote the best global and local positions, respectively.

3.5 Concept Drift Detection

An emerging problem in data stream mining is the detection of concept drifts. Concept drift detection methods aim at detecting changes in the concept being learnt so far. The majority of these methods are based on statistical analysis of recent data points received from a data stream. Below, we discuss two effective concept drift detection algorithms, namely, Drift Detection Method (DDM) and an extension of this method called Early Drift Detection Method (EDDM). We use EDDM as a base concept drift detector to start GA iterations in two of the three methods proposed in this thesis, namely, EACD and RED-GENE.

3.5.1 DDM: Drift Detection Method

Some approaches pay attention to the number of errors thrown by a learning model through the prediction stage. The DDM method [77] uses a binomial distribution providing a generic form of the probability for the random variable representing the number of errors in a sample of n examples. For each point i in the sequence being sampled, the error rate is the probability of misclassifying (p_i) with the standard deviation $s_i = \sqrt{p_i(1-p_i)/i}$. In DDM algorithm, it is assumed that the error rate of the learning algorithm (p_i) decreases while the number of examples increases if the distribution of examples is stationary. A significant increase in the error of the algorithm suggests that the class distribution is changing, therefore, the original decision model is supposed to be inappropriate. As a result, the values of p_i and s_i are stored when $p_i + s_i$ reaches its minimum value during the process (obtaining p_{min} and s_{min}), and the following conditions are triggered:

- $p_i + s_i \geq p_{min} + 2 \times s_{min}$ for the warning level; beyond this level, examples are stored in anticipation of a possible change of the context;

- $p_i + s_i \geq p_{min} + 3 \times s_{min}$ for the drift level; beyond this level, the concept drift is supposed to be true, the model induced by the learning method is reset and a new model is learnt using the examples stored since the warning level was triggered; the values for p_{min} and s_{min} are reset.

This approach has a good performance in detecting abrupt and gradual changes when the gradual change is not very slow. However, its performance drops when the change is slowly gradual, in which case, examples are stored for a long time, the drift level can take too much time to be triggered and the memory required to store examples can be exceeded.

3.5.2 EDDM: Early Drift Detection Method

The Early Drift Detection Method (EDDM) [78] was proposed to improve the performance of the DDM method in the presence of gradual concept drifts while keeping a good performance in the presence of abrupt concept drifts. The basic idea of EDDM is to consider the distance between the errors of two consecutive classification results instead of only the error values. In particular, the distance between two consecutive errors is expected to increase as the method improves its predictions while learning. The values of the average distance between two errors $p_i^{'}$ and its standard deviation $s_i^{'}$ can be calculated and stored when $p_i^{'} + 2 \times s_i^{'}$ reaches its maximum value (obtaining $p_{max}^{'}$ and $s_{max}^{'}$). In other words, the value of $p_{max}^{'} + 2 \times s_{max}^{'}$ corresponds to the point, where the distribution of the distances between the errors is maximum. This point is reached when the induced model best approximates the current concepts of the considered dataset. The following two thresholds are defined in EDDM:

- $(p_i^{'} + 2 \times s_i^{'})/(p_{max}^{'} + 2 \times s_{max}^{'}) < \alpha$ for the warning level; beyond this level, the examples are stored prior to a possible change of the context.

- $(p_i^{'} + 2 \times s_i^{'})/(p_{max}^{'} + 2 \times s_{max}^{'}) < \beta$ for the drift level; beyond this level, the concept drift is supposed to be true, the model induced by the learning method is reset and a new model is learnt using the examples stored since the warning level was triggered. The values of $p_{max}^{'}$ and $s_{max}^{'}$ are reset too.

In our implementation of EDDM, a minimum of 30 errors are calculated (note that the method may parse a large number of examples before obtaining 30 classification errors). This number was selected because we want to estimate the distribution of the distances between two consecutive errors and compare it with future distributions to find differences. Thus, $p_{max}^{'} + 2 \times s_{max}^{'}$ represents the 95% of the distribution. After obtaining 30 classification errors, the method uses the above-defined thresholds to detect a possible concept drift.

In our experiments, the values of the thresholds α and β were set to 0.95 and 0.90 consecutively. These values were determined after conducting a set of experiments on a synthetic data stream generator (SEA data stream generator). This is done due to the possibility of applying different concept drifts manually to the data stream that allows us to analyse the results by comparing the exact time of introducing the concept drifts and the detected time of concept drift using EDDM method. A good selection of α parameter helps our method to start storing data once a concept drift has started to change the distribution of data. A good selection of β parameter helps our method to detect the concept drifts properly and be able to distinguish between the actual concept drifts and noises throughout the data stream. It is worth mentioning that if the similarity between the actual value of $p_i^{'} + 2 \times s_i^{'}$ and the maximum value $p_{max}^{'} + 2 \times s_{max}^{'}$ increases over the warning threshold, the stored examples are removed and the method returns to normality.

3.6 Summary

This chapter discussed the necessary background and theoretical definitions of the algorithms used as building blocks of the novel methods proposed in this thesis (EACD, RED-GENE and RED-PSO). In particular, the Hoeffding Tree algorithm is employed as the base classification algorithm to build and train the classifiers for the three proposed methods. Furthermore, RD is used in the proposed methods to optimise the number of classifiers in each randomly drawn classification *types*, while GA or PSO are used to optimise the combination of the features for each classification *type* inside the ensemble when a concept drift is detected. Finally, EDDM is employed as the base concept drift detection algorithm to initialise the GA procedure. GA and EDDM are used in EACD (Chapter 5) and RED-GENE (Chapter 6), whereas PSO is utilised in RED-PSO (Chapter 7). The experimental methodology including the details of experimental settings and different datasets used in this thesis is provided in the next chapter.

Chapter 4

Experimental Methodology

In the previous chapter, we reviewed in detail the necessary background and provided theoretical explanations of the algorithms modified and used in the methods proposed in this thesis. This chapter discusses the basic details necessary for conducting the experiments in this thesis. In particular, we describe the experimental settings that include different methods used for comparison, experimental environment and specification, evaluation runs along with different datasets and their characteristics in the experiments.

4.1 Experimental Settings

To evaluate the proposed algorithms, a set of experiments was conducted using nine datasets, including four artificial (synthetic) data stream generators and five real-world data streams. In particular, we compared EACD to the state-of-the-art ensemble methods for non-stationary data stream classification that have shown a good performance and reliable results [45][38], including DWM [42], OAUE [45], OSBoost [41], LevBag [36] and ARF [38]. These methods are introduced in Chapter 2.

The methods proposed in this thesis are implemented in Java programming language using the Massive Online Analysis (MOA) API [79]. All other algorithms are already included in the MOA framework [79], which we used as the experimental environment. MOA is an open-source framework for data stream mining in evolving environments. When running LevBag, ARF, DWM, OAUE and OSBoost, their default parameters as set in MOA were used. The parameters for our proposed methods are listed in Sections 5.3.1 (EACD), 6.3.1 (RED-GENE) and 7.3.1 (RED-PSO).

To ensure a thorough set of experiments with precise results, 10 different variants for every artificial (synthetic) data stream were generated and each method was tested on all variants. These variants were generated by changing different parameters in all artificial streams. The selected parameters for each data stream generator are specified later in Section 4.2. For every real-world data stream, each experiment was repeated 10 times over the same data stream. In other words, for each synthetic data stream, 10 different sets of streams (using different parameters to create the dataset) are generated to conduct the experiments and the result shown for each data stream is the average of those 10 variations. While for each real-world data stream, the same experiment is conducted 10 times over the same set of data and the results shown for each data stream is the average of those 10 repeated experiments over the same data.

We performed two different evaluation runs for each experiment. The first run involved passing one of the chosen datasets through a specific algorithm using the prequential evaluation technique with an immediate access to the real labels of the instances assigned by the system. This evaluation run is called the *immediate setting*. The second run also involved passing each dataset through a specific algorithm using the prequential evaluation; however, the real labels of the instances could be accessed with a delay. This evaluation technique, called the *delayed setting*, can provide more realistic experiments, since the actual labels of streaming data are usually not available immediately in the real world. The classification performance estimates were calculated in the same way for both the immediate and delayed settings. For the delayed setting, the parameter of delay was set to an arbitrary value of $1,000$; hence, the label of each instance was revealed after passing 1,000 instances. The window size (width) of the experiments was set to 1,000 for both the immediate and delayed settings.

Hoeffding trees was used in the experiments as the base classifiers (decision trees). Hoeffding tree, also known as the Very Fast Decision Tree (VFDT) method [44], is an incremental decision tree algorithm capable of learning from massive data streams.

The experiments were performed on a machine equipped with an Intel Core i7-4702MQ CPU @ 2.20GHz and 8.00 GB of installed memory (RAM).

4.2 Datasets

A total of 9 different data streams including 4 artificial (synthetic) data stream generators and 5 real-world datasets are used for conducting the experiments in this thesis. For each of the synthetic data generators namely SEA generator [55], Hyperplane Generator [80], Random Tree Generator [44] and LED generator [81] we have generated 10 different variations by manipulating different parameters such as noise, the magnitude of change, number of attributes/classes and seed for random numbers. All artificial data streams used in this thesis and the way we generated them for our experiments are explained in Section 4.2.1.

For each of the real-world datasets, namely Forest cover-type [82], Electricity [83], Airlines, Poker-Hand and KDDcup99 [84] we have conducted the same experiment 10 times over the same data. This is done to examine the robustness of each method to different variations of each data generator and also to see the effects of using non-deterministic methods that use randomisation approaches. All of the mentioned data streams are introduced in details in Sections 4.2.2 and 4.2.1. All real-world data streams used in this thesis are introduced in Section 4.2.1.

Table 4.1 shows the properties of each data stream used in this thesis (the type of dataset, number of features, number of records, number of classes and imbalance ratio). The Imbalance Ratio (IR) is calculated as $IR = \frac{P_{majority}}{P_{minority}}$, where $P_{majority}$ is the population size of the majority class and $P_{minority}$ is the population size of the minority class. For example, if we have 100, 400, 2000, 5000, 10000 as the population size of a 5 class dataset, then the imbalance ratios would be 1:4:20:50:100 which can be simplified to 1: ... :100. The population sizes of all the classes of each data stream are illustrated in Sections 4.2.1 and 4.2.2. It is worth mentioning that the imbalance ratios calculated for synthetic data stream in Table 4.1 are approximate values as we have created 10 different variations for each synthetic data stream and those values may slightly change in different variations.

4.2.1 Artificial Data Streams

The following four artificial (synthetic) data stream generators were employed to simulate data for the experiments: SEA generator, Hyperplane generator, Random Tree Generator (RTG) and LED generator. Ten different stream variants were created for each of the considered data generators using their respective parameters to examine the performance of the tested algorithms depending on the type of the concept drifts. In case of the SEA generator, the variants were built by changing the random seed along with the type of manually added concept drifts. For the Hyperplane generator, different variants were built by tweaking the number of drifting attributes and magnitude of changes in data. For the RTG, the random seed number, along with the number of attributes and classes, were changed. Finally, for the LED generator,

TABLE 4.1: Properties of each data stream used in the experiments conducted in this thesis.

Dataset	Type	# records	# features	# classes	Imbalance Ratio (IR)
SEA	synthetic	1,000,000	3	2	1:1.8
Hyperplane	synthetic	1,000,000	10	2	1:1
RTG	synthetic	1,000,000	10 to 18	2 to 6	1:1.3:1.9:3.5
LED	synthetic	1,000,000	24	10	1: ... :1
Forest	real-world	99,940	54	7	1: ... :30.90
Electricity	real-world	45,312	8	2	1:1.36
Airlines	real-world	539,383	7	2	1:1.24
Poker	real-world	829,200	10	10	1: ... :200K
KDDcup99	real-world	494,021	41	23	1: ... :113K

different variants were built by tweaking the number of drifting attributes and random seed number.

SEA Generator

The SEA generator [55] is a synthetic data stream generator that aims to simulate concept drifts over time. It generates random points in a three-dimensional feature space; however, only the first two features are relevant. The concept drifts in this data stream are generated by changing the relevant features throughout the stream. Figure 4.1 illustrates the population size of different classes in the generated data streams using SEA generator.

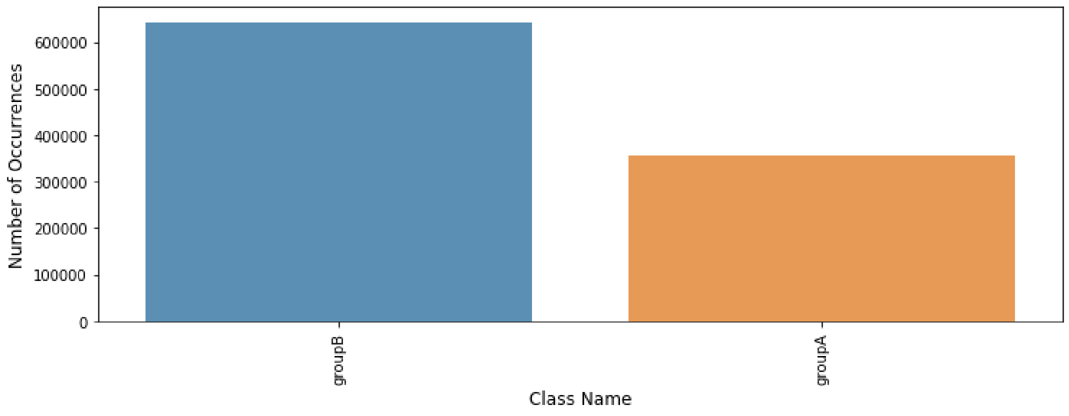


FIGURE 4.1: Number of instances for each class in the data streams generated using SEA generator.

Each variant of the SEA generator was set to include one million instances. In addition, different concept drifts were manually chosen to happen in the instance numbers 200K, 400K, 600K and 800K. For the first five variants, two abrupt concept drifts with a width (width of concept drift change) of one were added at the instance numbers 200K and 400K, and two recurrent concept drifts with the same width were added at the instance numbers 600K and 800K. For the remaining five variants, two gradual concept drifts with a width of 10,000 were added at the instance numbers 200K and 400K, and two recurrent concept drifts with the same width were added at the instance numbers 600K and 800K.

Hyperplane Generator

The Hyperplane generator [80] is an artificial data stream with drifting concepts based on hyperplane rotation. It simulates concept drifts by changing the location of the hyperplane. The smoothness of drifting data can be

TABLE 4.2: The number of drifting attributes and magnitude of change selected for different stream variants of the Hyperplane generator.

Variant	No. of drifting att.	Mag. of change	Variant	No. of drifting att.	Mag. of change
1	2	0.01	2	2	0.02
3	3	0.01	4	3	0.02
5	4	0.01	6	4	0.02
7	5	0.01	8	5	0.02
9	6	0.01	10	6	0.02

changed by adjusting the magnitude of the changes. Figure 4.2 illustrates the population size of different classes in the generated data streams using Hyperplane generator.

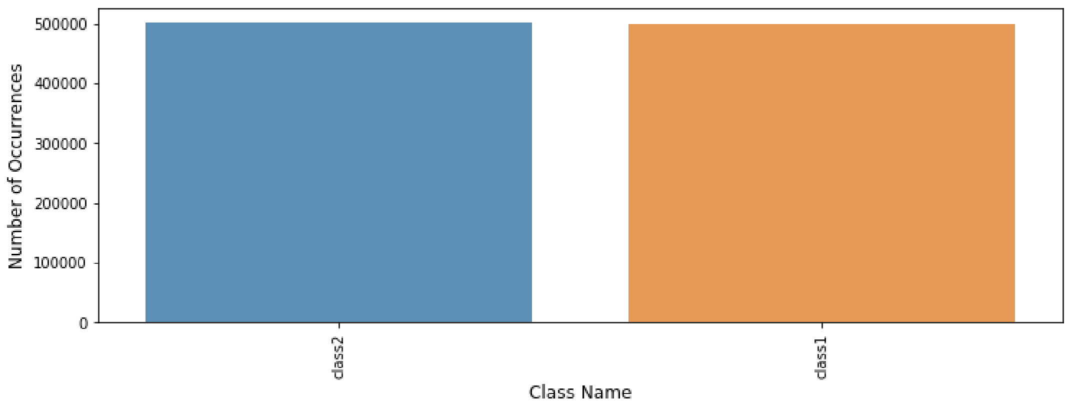
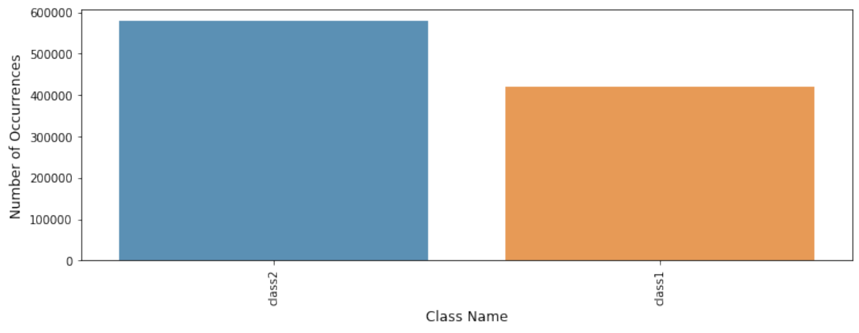


FIGURE 4.2: Number of instances for each class in the data streams generated using Hyperplane generator.

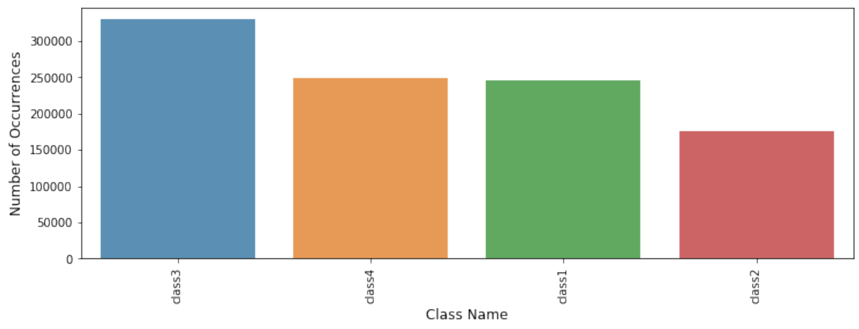
In the presented experiments, abrupt concept drifts are added to the data stream by increasing the magnitude of change parameter in the Hyperplane generator. The number of classes and attributes were set to two and ten, respectively, and the number of drifting attributes and magnitude of changes were set as indicated in Table 4.2. The number of instances in each stream was set to one million.

Random Tree Generator

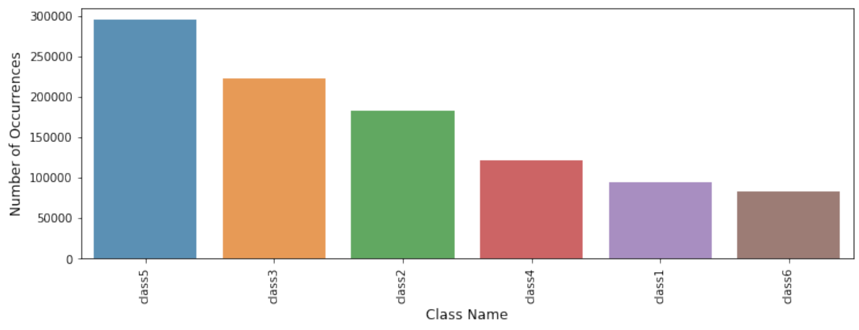
The Random Tree Generator (RTG) [44] builds a decision tree by randomly selecting attributes as split nodes and assigning random classes to them. After the tree is built, new instances are obtained through the assignment of uniformly distributed random values to each attribute. The leaf reached after a traverse of the tree determines its class value according to the attribute values of an instance. The RTG allows customising the number of nominal and numeric attributes, as well as the number of classes. Figure 4.3 illustrates the population size of different classes in the generated data streams using RTG generator.



(A) 2-class variations.



(B) 4-class variations.



(C) 6-class variations.

FIGURE 4.3: Number of instances for each class in the data streams generated using RTG generator.

In the experiments, the number of classes, number of features and random seed number were chosen as indicated in Table 4.3.

LED Generator

LED [81] is a well-known data stream generator. The goal here is to predict the next digit to be displayed on the LED display. The generator contains 24

TABLE 4.3: Total number of attributes, number of classes and random seed number of different stream variants of the RTG.

Variant	Attributes	Classes	Seed No.	Variant	Attributes	Classes	Seed No.
1	10	2	1	2	10	2	2
3	12	3	1	4	12	3	2
5	14	4	1	6	14	4	2
7	16	5	1	8	16	5	2
9	18	6	1	10	18	6	2

Boolean features, 17 of which are irrelevant and the remaining seven features correspond to each segment of a seven-segment LED display. Each feature has a 10% chance of being inverted. In our experiments, the LED generator was used to simulate concept drifts by swapping four of its features resulting in ten different stream variants. Figure 4.4 illustrates the population size of different classes in the generated data streams using LED generator.

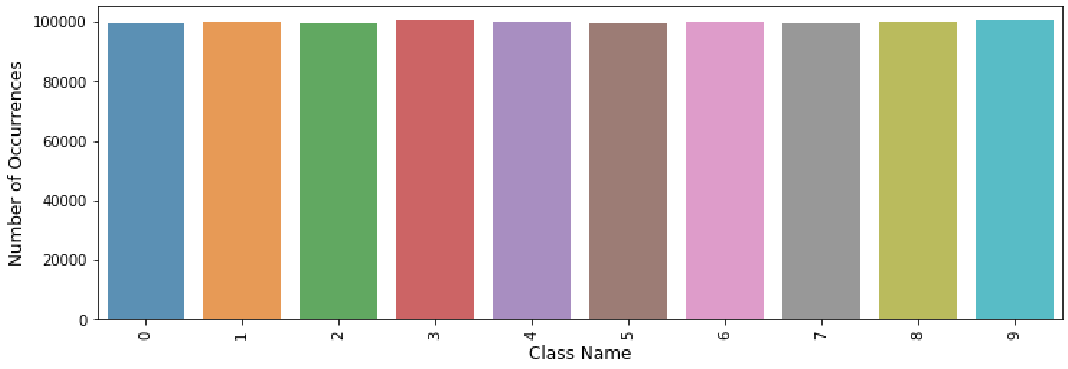


FIGURE 4.4: Number of instances for each class in the data streams generated using LED generator.

For the first five variants, the number of drifting attributes were chosen to be 1, 2, 3, 4 and 5, respectively. For the next five variants, only the random seed was changed, while the drifting attributes remained the same as in the first five variants.

4.2.2 Real World Data Streams

Forest Cover-type Dataset

The Forest Cover-type data stream [82] is a real-world dataset from the UCI Machine Learning Repository [1]. It contains forest cover types of 30 $\times$ 30

[1]http://archive.ics.uci.edu/ml

meter cells obtained from the US Forest Service (USFS). It consists of 99,940 instances and 54 attributes. The goal in this case is to predict the forest cover type from cartographic variables. Figure 4.5 demonstrates the population size of each class in this dataset.

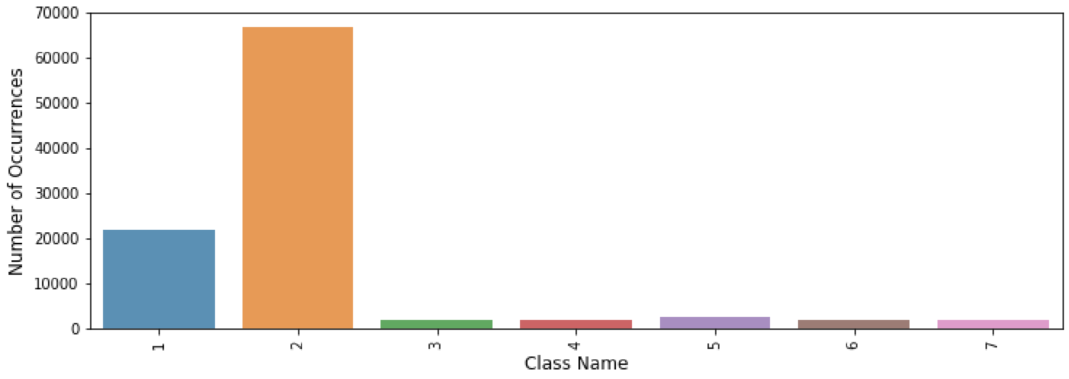


FIGURE 4.5: Number of instances for each class in Forest Covert-type dataset.

Electricity Dataset

Electricity is a widely used dataset by [83] collected from the Australian New South Wales electricity market. In this market, prices are not fixed and affected by demand and supply. The Electricity dataset contains 45,312 instances. Each instance contains eight attributes, and the target class specifies the change of the price (whether it goes up or down) according to its moving average over the last 24 hours. Figure 4.6 demonstrates the population size of each class in this dataset.

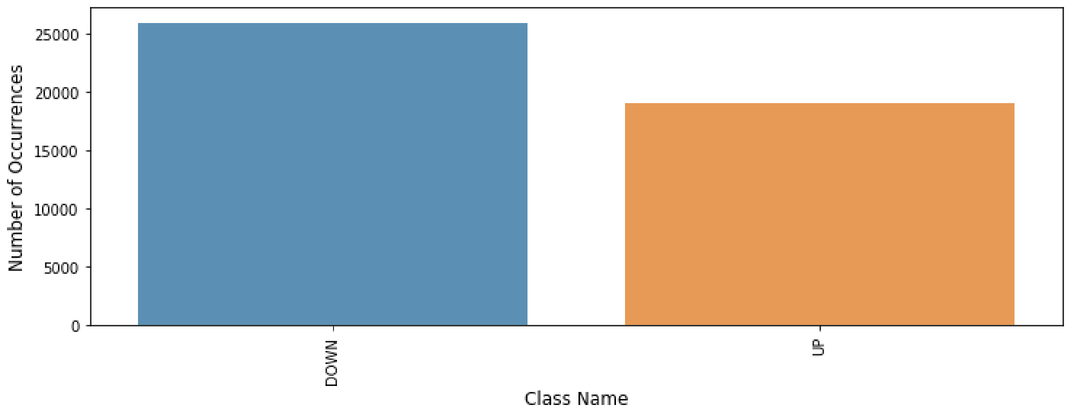


FIGURE 4.6: Number of instances for each class in Electricity dataset.

Airlines Dataset

Airlines[2] is a non-stationary classification dataset. The task is to predict whether a flight will be delayed providing the information on its scheduled departure. This dataset has two classes (whether a flight is delayed or not) and contains 539,383 records with seven attributes (three numeric and four nominal). Figure 4.7 demonstrates the population size of each class in this dataset.

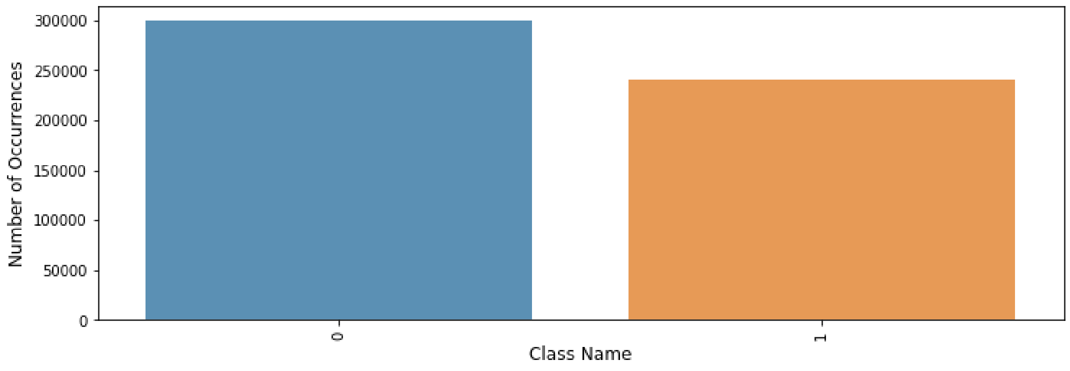


FIGURE 4.7: Number of instances for each class in Airlines dataset.

Poker-Hand Dataset

The Poker-Hand dataset from the UCI Machine Learning Repository[3] consists of 829,200 instances and 10 attributes. Each record of the Poker-Hand dataset is an example of a hand consisting of five playing cards drawn from a standard deck of 52. The total number of classes in this dataset is 10 that shows the poker hand. Figure 4.8 demonstrates the population size of each class in this dataset.

KDDcup99

KDDcup99 [84] is the dataset used in the Third International Knowledge Discovery and Data Mining Tools Competition. The competition task was to build a network intrusion detector – a predictive model capable of distinguishing between "bad" connections (intrusions or attacks) and "good" (normal) connections. KDDcup99 contains a standard set of data to be audited, which includes a wide variety of intrusions simulated in a military

[2] *http : //kt.ijs.si/elena¡konomovska/data.html*

[3] *https : //archive.ics.uci.edu/ml/datasets/Poker + Hand*

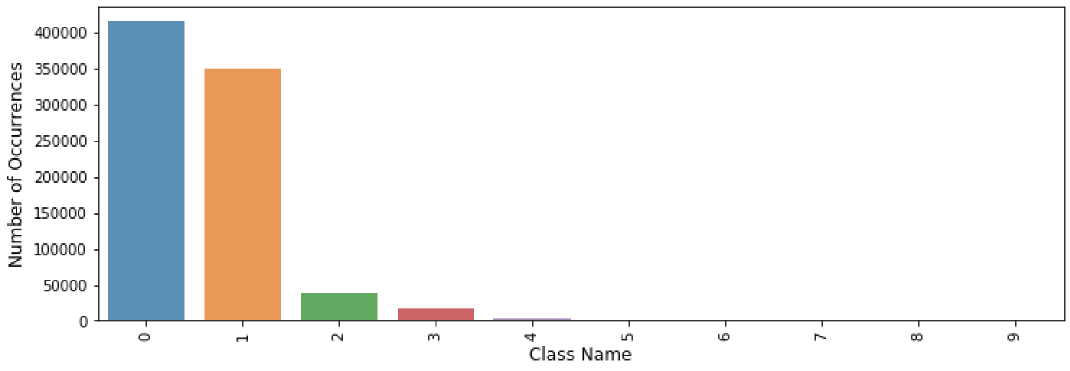


FIGURE 4.8: Number of instances for each class in Poker-hand dataset.

network environment. The preprocessed version of this dataset which contains 494,021 records, 41 attributes and 23 classes. Figure 4.9 demonstrates the population size of each class in this dataset.

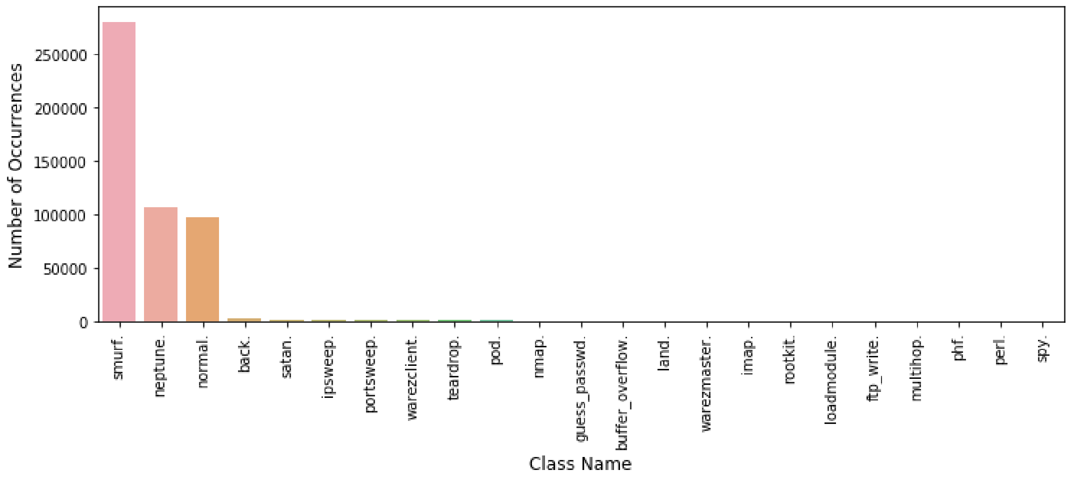


FIGURE 4.9: Number of instances for each class in KDDcup99 dataset.

4.3 Summary

This chapter discussed the necessary details needed for conducting the experiments in this thesis. In particular, we described the experimental settings and introduced different methods and datasets involved in the experiments. The experimental settings and datasets introduced in this chapter are used for running the experiments to compare our proposed methods to other state-of-the-art methods in non-stationary data stream classification tasks. The details of the proposed methods and the results of the experiments are discussed in the next chapters.

Chapter 5

EACD: Evolutionary Adaptation to Concept Drifts in Data Streams

In the previous chapter, we reviewed the necessary details for conducting our experiments throughout the thesis by describing experimental settings and introducing different datasets along with their characteristics. In this chapter, we propose a novel ensemble learning method using Hoeffding Trees, RD, GA, and EDDM to address the second objective of this thesis as outlined in Section 1.4: **use evolutionary algorithms to propose a novel ensemble learning technique for concept drift adaptation in data-streams**. The work discussed in this chapter is published as a full paper titled "EACD: evolutionary adaptation to concept drifts in data streams" in the "Data Mining and Knowledge Discovery" journal [21].

5.1 Introduction

A considerable effort of recent research has focused on data stream classification tasks in non-stationary environments [15]. The main challenge in this research area concerns the adaptation to *concept drifts*, that is, when the data distribution changes over time in unforeseen ways. Concept drifts occur in different forms and can be divided into four general types: abrupt (sudden), gradual, incremental and recurrent (reoccurring). In abrupt (sudden) concept drifts, the data distribution at the time *t* suddenly changes to a new distribution at the time *t+1*. Incremental concept drifts occur when the data distribution changes and stays in the new distribution after going through some new, unstable, median data distributions. In gradual concept drifts, the proportion of new probability distribution of incoming data increases, while the proportion of data that belong to the former probability distribution decreases over time. Recurring concept drifts happen when the same old probability distribution of data reappears after some time of a different distribution.

Ensemble learning has proved its superiority over other techniques for stream classification, especially in non-stationary environments [14], [13]. Ensemble learning is a machine learning approach, in which predictions of individual classifiers are combined using a combination rule to predict incoming instances more accurately. The advantage of using ensemble learning techniques for non-stationary data stream classification lies in their ability to update swiftly according to the most recent data instances. This is usually achieved by training the existing classifiers in the ensemble and changing their weights according to their performance: adding new, well-performing classifiers and removing outdated, poor-performing classifiers. Applications requiring the analysis of non-stationary data streams include spam filtering systems, stock market prediction systems, fraud detection in banking networks, weather forecasting systems, data analysis in IoT networks, traffic and forest monitoring systems, among many others. The extensive range of applications makes the task of non-stationary data stream classification even more challenging, as various applications seek diverse purposes and have different conditions.

To ensure a versatile yet robust ensemble approach in this context, the following main aspects should be taken into consideration: (1) improving the average **accuracy** of the ensemble; (2) improving the **efficiency** of the underlying algorithm by minimising its learning time and computational complexity; and (3) minimising the rate of misclassification and recovery time of **adaptation** upon different types of concept drifts.

The majority of the existing ensemble methods are either focused on one or two of the aforementioned factors, or concentrate on a specific type of data streams. For instance, some approaches do not remove old classifiers [60], [52]; hence, the number of classifiers is unbounded in this case, which can cause a low efficiency in terms of the time and memory usage. Other approaches are designed to cope with recurring concept drifts only [37]; therefore, such algorithms are only suitable for a limited number of applications and environments.

To overcome these limitations, we propose a novel ensemble learning method for data stream classification in non-stationary environments called *EACD*. It uses random selection of features and two evolutionary algorithms, namely, RD and GA. According to EACD, an ensemble of different classification *types* consisting of randomly drawn features (subspaces) of the target data stream are trained first. These randomly drawn subspaces are then optimised using GA to allow the ensemble to cope with different concept drifts over time. Training of the ensemble is performed on sequential data blocks in the stream. EACD allows a dynamic set of classification *types* to take action over time. The number of decision trees in a classification *type* (subspace) depends on the performance of this *type* on the most recent data. Hence, well-performing *types* increase in size, while poor-performing *types* decrease in size.

In summary, our solution allows the ensemble to handle different types of concept drifts by employing two different evolutionary techniques. RD is used to continuously determine well and poorly performing *types* and expand or shrink them accordingly. GA is used to compose new, improved *types* out of the existing ones by iterating over the most recent data.

The rest of this chapter is organised as follows. Section 5.2 describes the proposed EACD method in detail and provides its theoretical justification. Section 5.3 outlines the experimental setup and results of comparing (i) different variations of EACD; and (ii) the best performing variation to other state-of-the-art methods. Section 5.3.5 comprehensively discusses the results of the experiments. Finally, conclusions and the summary of this chapter are presented in Section 5.4.

5.2 EACD Description

In the proposed novel ensemble learning algorithm suitable for non-stationary data stream classification, data come as continuous data blocks. For the study described in this chapter, each data block consists of 1,000 samples. While this number was selected arbitrarily, it can be set to any other value as required. The algorithm comprises of two different layers called the *base layer* and *optimisation layer*. Each layer has a set of classifiers that classify the incoming data independently. The base layer is always active, whereas the optimisation layer is only active when GA has made its generations and the *types* are mature enough. The classifiers making up the second (genetic) layer are set up to have more weight than the classifiers making up the base layer to achieve optimality of the types.

The base layer is built using a random selection of features, which is then extended using RD. The optimisation layer is built by applying GA to the set of features randomly selected from the base layer and introduces a new set of classification *types* optimised using recent instances stored in a buffer. The base and optimisation layers are detailed in the following subsections, while Figure 5.1 illustrates how EACD works.

The rationale behind the proposed architecture is as follows. The main problem with the existing explicit methods (the ones that use a concept drift detection mechanism) is their sensitivity to false alarms. In addition, detecting some types of concept drifts (especially gradual and incremental) is a hard task. Hence, the detection mechanisms employed in explicit methods might not detect such drifts or detect them with a delay. In this scenario, RD offers a smooth yet effective way to improve the performance of the ensemble by increasing and reducing the number of trees in the classification *types*. Furthermore, the main problem with the existing implicit algorithms (the ones without a concept drift detection mechanism) is their slowness in coping with concept drifts since they do not have an immediate reaction to drifts. This is the reason for using a concept drift detection algorithm along with GA to immediately react to concept drifts and optimise the combination of the features in classification *types*. Overall, by combining RD with concept drift detection methods and GA, it is feasible to have the advantages of both explicit and implicit methods as discussed in Chapter 2.

5.2.1 Base Layer

As illustrated in Figure 5.1, the base layer of EACD uses a random selection of features (subspaces) to create a variety of classification *types* in the ensemble, which ensures the ensemble diversity. RD is then applied to make the proposed method compatible with non-stationary environments and able to seamlessly adapt to the most current types of data and concepts. In other words, RD is used to increase the number of well-performing classification trees and reduce the number of unhelpful ones.

The base layer is built using the following steps. First, p percent of all features are randomly selected from the pool of data features (attributes) of the target data stream. This phase is called *random subspace*. In other words, the total number of features to be selected randomly from the pool of features is established as in Equation 5.1.

$$n = \frac{p}{100} \times f, \tag{5.1}$$

where n denotes the total number of features to be selected, p is an arbitrary number ($0 < p < 100$) showing the percentage of the features that should be selected randomly and f denotes the total number of features of the target

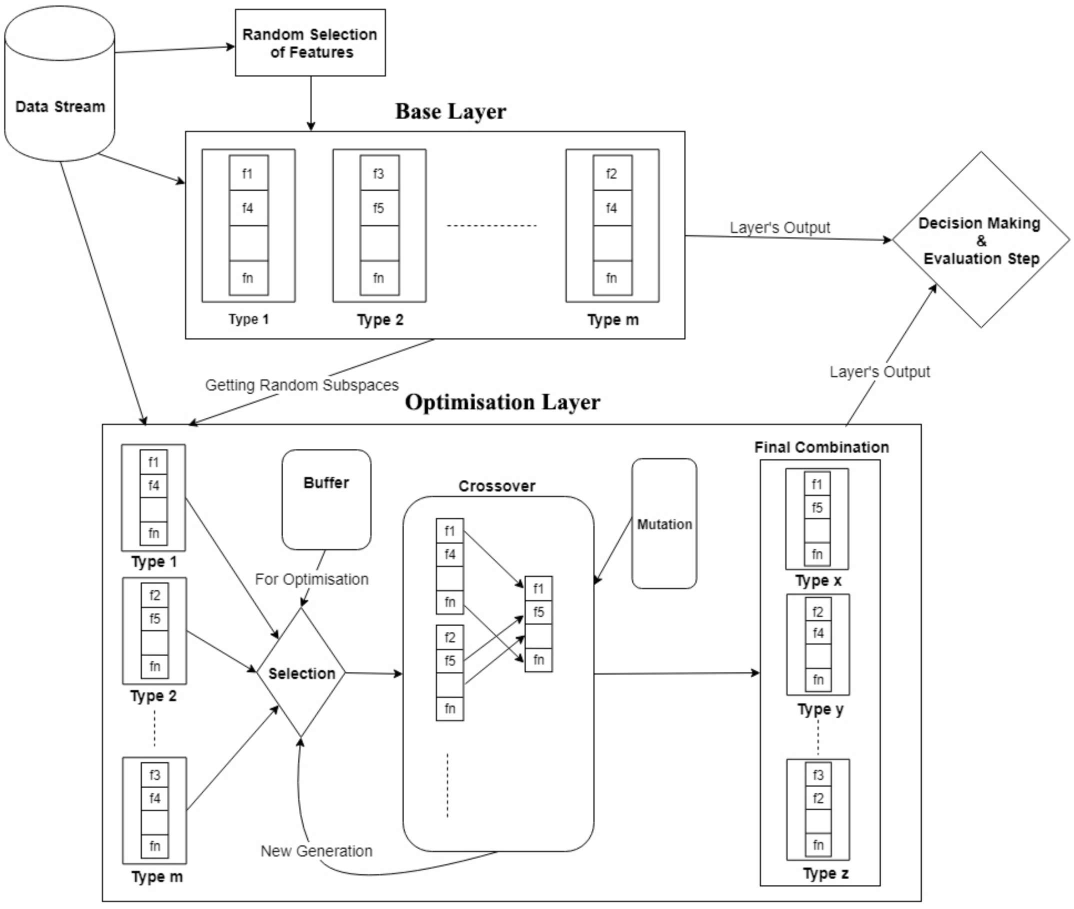


FIGURE 5.1: Architecture of EACD.

data stream. Each iteration of this step produces a set of randomly selected features (subspace) from the pool of features that we call a *type*. This step is repeated m times; hence, there are m independent classification *types* at the end of this step. Note that m is a parameter of our proposed model referring to the total number of classification *types* in the ensemble and is chosen depending on the total number of features of the target stream; there should be a balance between the number of *types* (m) and the number of features in each *type* ($p \times f$).

Next, a decision tree is built per every classification *type* (subspace) when the first block of data (samples) is received by the system. Given the maximum number of classifiers for each *type* max, this step is repeated for the first $\frac{max}{2}$ data blocks received by the system for the *types* to shape and reach a specific maturity level. This phase is called the *initial training*, during which, an average number of classifiers for every *type* in the ensemble is built. Note that for every data block received by the ensemble, all decision trees classify the instances and majority voting then determines the ensemble's output. This is called the *voting* step.

Once the initial training phase is completed, each decision tree is evaluated after classifying incoming instances. The accuracy (a) of each decision tree in a *type* is calculated as in Equation 5.2.

$$a_i = \frac{c_i}{db}, \tag{5.2}$$

where c_i denotes the number of correctly classified instances in i^{th} data block and db denotes the total number of instances in each data block. The total accuracy of each *type* is the average accuracy of its related decision trees. The accuracy of the whole ensemble can be determined similar to Equation 5.2. This phase is called the *evaluation* phase.

Next, the *RD* stage is applied. This is when each *type's* accuracy (the average accuracy of all related trees) is taken into consideration and assessed with an expected payoff (explained previously in Chapter 3). The expected payoff for the study presented in this chapter is set to the average accuracy of all *types* in each data block. However, it can be determined in any other way, such as assigning a fixed number. The *types* with a higher payoff (accuracy) than the expected payoff get a new decision tree (i.e. a new decision tree is built for such data *types* based on the last block's samples), whereas the *types* with a lower payoff (accuracy) than the expected payoff lose a decision tree. In other words, as in the conditions in Equation 5.3.

$$\begin{cases} a(t_i) \geq \frac{\sum_{i=1}^{m} a(t_i)}{m} \Rightarrow grow \\ \\ a(t_i) < \frac{\sum_{i=1}^{m} a(t_i)}{m} \Rightarrow shrink \end{cases} , \tag{5.3}$$

where $a(t_i)$ denotes the accuracy of the ith *type* and m denotes the total number of *types*.

Finally, every decision tree in the ensemble is trained with the samples from a newly received data block in the *retraining* phase. The purpose of this phase is to have a more updated ensemble, especially when a concept drift happens. In this situation, retraining can lead to a fast adaptation since all classifiers are trained with the newly evolved data.

To limit the size of the ensemble, an upper bound for the number of decision trees (classifiers) in a *type* is assigned. When the maximum size of a *type* is exceeded, the least performing decision tree of that specific *type* is removed. The upper bound (max) for the number of classifiers for the study presented in this chapter is set to an arbitrary value of $max = 20$. Furthermore, a lower bound (min) is assigned to all *types* to prevent the *types* from complete removal. The minimum size of all *types* is set to $min = 1$ in this study. Hence, a tree is not removed upon poor performance if it is the only one decision tree related to a type left.

Algorithm 7 shows how the base layer is built and works. In this algorithm, t_j denotes the j^{th} *type* of the ensemble ($1 \leq j \leq m$) and $a(t_j)$ denotes the accuracy of this *type*. The following functions are used in the presented algorithm:

- *Classify()*: the ensemble classifies data using majority voting;
- *Evaluate()*: evaluate the accuracy of all *types* in the ensemble using Equation 5.2;
- *Grow()*: add a new classifier (decision tree) to the specified *type* (if Equation 5.3 stands);
- *Shrink()*: remove one classifier (decision tree) from the specified *type* based on the ensemble's removal mechanism (if Equation 5.3 stands); if this *type* has only one classifier, then do nothing;
- *Train()*: train all classifiers using the samples from a newly received data block.

Algorithm 7: EACD BASE LAYER

```
Input: A continuous block of data, $DB = \{db_1, db_2, .., db_n\}$
$n$: number of features that should be selected in each type
$m$: total number of types
$max$: maximum number of classifiers in each type.
Output: Classified Samples
1  $i := 1$
2  for $t := 1$ to $t := m$ do
3      Randomly select $n$ features
4  while data stream is not empty do
5      if $i \leq \frac{max}{2}$ then
6          Classify($db_i$)
7          Grow(T) for all the types
8      else
9          Classify($db_i$)
10         Evaluate()
11         if $a(t_j) \geq \frac{\sum_{j=1}^{n} a(t_j)}{m}$ then
12             Grow($t_j$)
13         else
14             Shrink($t_j$)
15     Train()
16     $i := i + 1$
```

In the presented algorithm, lines 2 and 3 refer to the *random subspace* phase. Lines 5, 6 and 7 are the *initial training* phase. The *evaluation phase* is implemented in line 10, the *RD* phase is in lines 11 to 14, and finally, the *retraining* phase is in line 15. Decision trees are removed based on their performance; the tree that performs the worst in the specified *type* is removed.

5.2.2 Optimisation Layer

As demonstrated in Figure 5.1, this layer is built using the existing classification *types* of the base layer. GA takes all randomly drawn classification *types* (subspaces) as its input and tries to form the best possible combination of the features in each *type*. This is achieved by iterating over a fixed data that has been received by the system recently (buffer). The optimisation layer is different from the base layer only in this part (i.e. the combination of classification *types*), whereas the classification, training and updating mechanisms are the same as explained for the first layer.

Algorithm 8 shows how the optimisation layer is being built. First, the set of randomly drawn subspaces is taken from the base layer and considered as the first GA population. Note that in this algorithm, each classification *type* is considered as an *individual* in GA, and each feature inside a *type* is a *chromosome* of this *individual*.

The buffer always keeps the most recently labelled instances received by the system. It serves as a search space for the GA optimisation task. Whenever GA starts or restarts, it copies the data inside the buffer into the memory and uses them for its procedures, i.e. the *selection* stage and fitness function.

Selection stage: for every GA iteration, the classification *types* with a better accuracy than the overall average accuracy of all *types* over the search space are selected for the crossover stage. Hence, the GA fitness function is the *types*′ average accuracy over the search space. The "Selection()" function in Algorithm 8 corresponds to the Selection stage.

Crossover stage: the *types* selected in the selection stage are chosen for GA breeding purposes. This lets the *types* with a better accuracy to pair with other well-performing *types* to make offspring. The "Crossover()" function in Algorithm 8 corresponds to the Crossover stage.

Mutation stage: the mutation rate of 5% applies upon breeding of the *types*. Hence, there is a 5% chance for an offspring to get a random feature from the pool of features instead of getting all of them from its parents. The "Mutation()" function in Algorithm 8 corresponds to the Mutation stage.

When the maximum number of generations is achieved, the resulting classification *types* form a new set of classifiers that starts to be trained and evaluated with incoming data. The new ensemble model is said to be mature enough when its performance on the latest data block is better than the average performance of the algorithm. As mentioned before, the base layer is always active, whereas the optimisation layer is active when the GA has done its job and the layer has reached its maturity level. All classifiers inside the base layer of the proposed algorithm are given an arbitrary weight of one ($W_b = 1$), whereas all classifiers inside the optimisation layer are given an arbitrary weight of two ($W_g = 2$). This intensifies the effect of the optimisation layer on the algorithm given the optimality of the types.

Once a new data block is received by the system, it goes to both layers, and the classifiers inside each layer classify the instances and send their predictions to the decision making part of the algorithm independently (Figure 5.1). The decision maker then considers all predictions received from the active classifiers and performs the voting procedure according to the weight of each prediction. This decision maker also tracks and keeps the average accuracy of each layer. Whenever GA is due to restart its procedures, the optimisation layer is deactivated and cleared to make room for a new set of *types*. To determine when to start a new set of GA generations (i.e. reset the optimisation layer), one implicit and one explicit mechanisms are proposed in this chapter.

In the implicit mechanism, GA starts resetting the optimisation layer when the base layer has proved to have a better average accuracy over the last arbitrarily set number of data blocks (we used 10 data blocks). This evaluation part is calculated continuously by the decision maker part mentioned previously in this section. In the implicit variants, the buffer inside the optimisation layer stores the last data block received by the system.

In the explicit mechanism, a concept drift detection method is utilised to specify when to reset the optimisation layer. When the concept drift detector signals a drift, GA starts to rebuild its layer. In this study, we used EDDM [78] as the explicit mechanism; however, any concept drift detection method can be used as the drift detector. EDDM is especially designed to improve the detection in the presence of gradual concept drifts compared to other drift detection methods. The basic idea of EDDM is to consider the distance between two consecutive errors instead of only the error values in the classification process. In the explicit variants of EACD, the buffer inside the optimisation layer starts storing the instances once the concept drift detector signals a *warning*. Hence, when the drift detector signals a *drift*, the instances inside the buffer represent the new concept. The "DriftDetector()" function in Algorithm 8 corresponds to the *concept drift detection* stage.

Algorithm 8: EACD OPTIMISATION LAYER

```
Input: Buffer
g: Maximum number of generations
Resetting mechanism: [implicit/explicit]
Randomly drawn subspaces (types) from the base layer, $T_B$ ={$t_1$,$t_2$,..,$t_m$}
Output: New set of classification types, $T_G$ ={$t_1\prime$,$t_2\prime$,..,$t_m\prime$}
1  for $i := 1$ to $i := g$ do
2      Selection()
3      Crossover()
4      Mutation()
5  if Resetting mechanism=Implicit then
6      repeat
7          Evaluate($T_B$) /*Evaluates base layer over the last 10 data
            blocks*/
8          Evaluate($T_G$) /*Evaluates the optimisation layer over the last 10
            data blocks*/
9      until Average accuracy ($T_G$) $\leq$ Average accuracy ($T_B$)
10     Reset(GA) /*Clear the optimisation layer and restart GA*/
11 else
12     repeat
13         DriftDetector()
14     until DriftDetector() = Drift /*when the detector signals a drift*/
15     Reset(GA) /*Clear the optimisation layer and restart GA*/
```

5.2.3 Theoretical Justification

In the literature of mining non-stationary data streams, there is no deterministic method that can guarantee to find the global optima. This is due to the evolving nature of the data that come in the form of a stream. Hence, a single classifier of a data stream that is optimal in a specific environment can become the worst classifier once the data has evolved in the same data stream. By adding randomisation to create different classification *types* in the first layer of the proposed method, it is feasible to have a variety of classifiers in the ensemble. This leads to a diverse set of available solutions to quickly cope with an occurring concept drift. However, having different classification *types* can also cause problems such as degrading the accuracy in case of using one or more poor *types*. This problem is tackled by employing RD to increase the number of well-performing *types* and reduce the number of poor-performing ones in the base layer of the proposed algorithm.

Furthermore, "stochastic search and optimisation pertains to problems where there is random noise in the measurements provided and/or there is injected randomness in the algorithms itself" [85]. Hence, GA is used in the second (genetic) layer to create new classification *types* to optimise the combination of features of the random *types* used in the first (base) layer. GA is a powerful and broadly applicable stochastic optimisation technique [86] that can be used in dynamic environments (e.g. generating data streams) after adding a few changes to its mechanism as explained in this chapter.

5.3 Experimental Study

This section provides the details of the experiments conducted to investigate the performance of the proposed method in this chapter. We introduce different variations of EACD along with their parameters and computational complexity. Different variations are then compared with each other and the best performing variation is compared with the state-of-the-art algorithms in non-stationary data stream classification tasks. Eventually, the results are statistically analysed and discussed thoroughly. It is worth mentioning that the details necessary for conducting the experiments in this thesis (e.g. experimental settings, compared methods, evaluation runs, datasets, etc.) are discussed in Chapter 4.

5.3.1 EACD Variations

Eight different variations of the proposed algorithm were implemented and compared in the experiments to evaluate the impact of each EACD characteristic and discuss the effect of employing different parameters in the EACD algorithm. The *base* variations only use the base layer of the proposed algorithm, while GA optimisation is not applied; only *base*4 variation uses the concept drift detector to restart the layer upon drifts. The implicit (Imp) variations use an implicit mechanism, whereas the explicit (Exp) variations use an explicit mechanism to specify when the optimisation layer should be restarted (as explained in Section 5.2.2). The specific parameters of the eight proposed variations are as follows:

- $EACD_{base}$: $p = 60\%$ and $m = 0.6 \times f$;

- $EACD_{base2}$: $p = 30\%$ and $m = 0.3 \times f$;
- $EACD_{base3}$: $p = 60\%$ and $m = 0.3 \times f$;
- $EACD_{base4}$: $p = 60\%$, $m = 0.6 \times f$ and restarting the ensemble upon drifts;
- $EACD_{Imp}$: $g = 15, z = 5\%, p = 60\%$ and $m = 0.6 \times f$;
- $EACD_{Imp2}$: $g = 15, z = 0\%, p = 60\%, m = 0.6 \times f$;
- $EACD_{Exp}$: $g = 15, z = 5\%, p = 60\%, m = 0.6 \times f$;
- $EACD_{Exp2}$: $g = 15, z = 0\%, p = 60\%, m = 0.6 \times f$;

where p denotes the number of features in each classification *type*, m denotes the number of classification *types* in the layer, f denotes the total number of features in the data stream, g denotes the total number of generations for each GA iteration and z denotes the mutation rate of GA.

5.3.2 Computational Complexity

Assuming the number of classes c, number of attributes in each classification *type* p, values per attribute v and maximum number of trees in the ensemble k, no more than p attributes can be considered in a single Hoeffding tree [44]. Each attribute at a node requires computing v values. Since calculating information gain requires c arithmetic operations, the cost of k Hoeffding trees at each time-step in the worst case scenario is O($kcpv$). Given the number of classification *types* in the ensemble m and the fact that RD uses m arithmetic operations to calculate payoffs, the cost of applying RD to the ensemble is only O(m). Hence, the time complexity of deploying the base variations of the proposed method ($EACD_{base}$, $EACD_{base2}$ and $EACD_{base3}$) is O($m + (kcvp)$).

Assuming the size s of the GA population and total number of generations g, the cost of GA optimisation is O(sg) at each time when the optimisation layer needs to be restarted. Hence, the time complexity of deploying the implicit variations of the proposed method ($EACD_{Imp}$ and $EACD_{Imp2}$) is O($m + (kcvp) + (sg)$).

Finally, given d as the number of instances in each data block and f as the total number of features in the dataset, the EDDM drift detection method, which uses J48 (C4.5) decision tree as its learning mechanism, requires O(df^2) of time. Hence, the time complexity of deploying the explicit variations of the proposed method ($EACD_{Exp}$ and $EACD_{exp2}$) is $O(m + (kcvp) + (sg) + (df^2))$. Note that the cost of running evolutionary methods is minimised providing the variations applied to EACD as previously discussed.

5.3.3 Results

The considered algorithms are compared using standard criteria, namely, the classification accuracy and overall execution time (including the time of making predictions, training and evaluating the ensemble). There are two settings for each experiment (immediate and delayed).

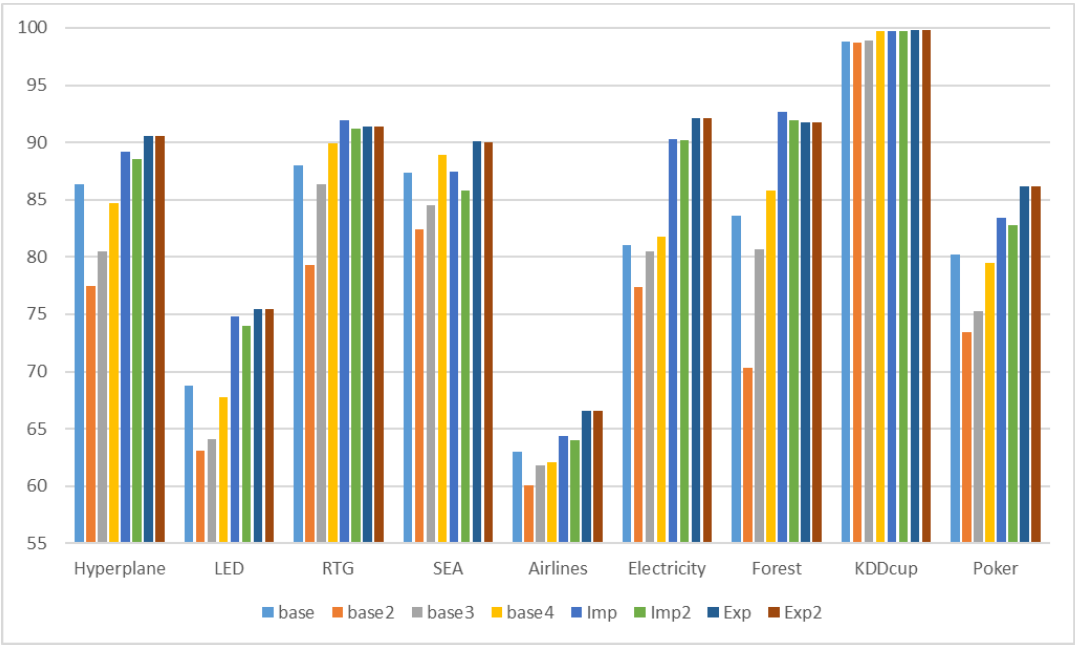


FIGURE 5.2: Accuracy (%) of the EACD variations over the considered nine datasets in the immediate setting.

Figures 5.2 and 5.3 (Tables A.1 and A.2 in Appendix A) illustrate the average accuracy for the proposed EACD variations over the mentioned nine datasets in the immediate and delayed settings, respectively. As can be noticed from the tables, $EACD_{Exp}$ has the best average accuracy over the Hyperplane, LED, SEA, Airlines, Electricity and Poker-Hand datasets. It also has the best overall average accuracy in both the immediate and delayed settings. $EACD_{Imp}$ has the best average accuracy over the Forest Cover-type and RTG datasets, whereas $EACD_{Exp2}$ has the best average accuracy over the KDDcup99 dataset.

Figure 5.4 illustrates the average accuracy (throughout all datasets) of all EACD variations in both immediate and delayed settings. It is clear that the

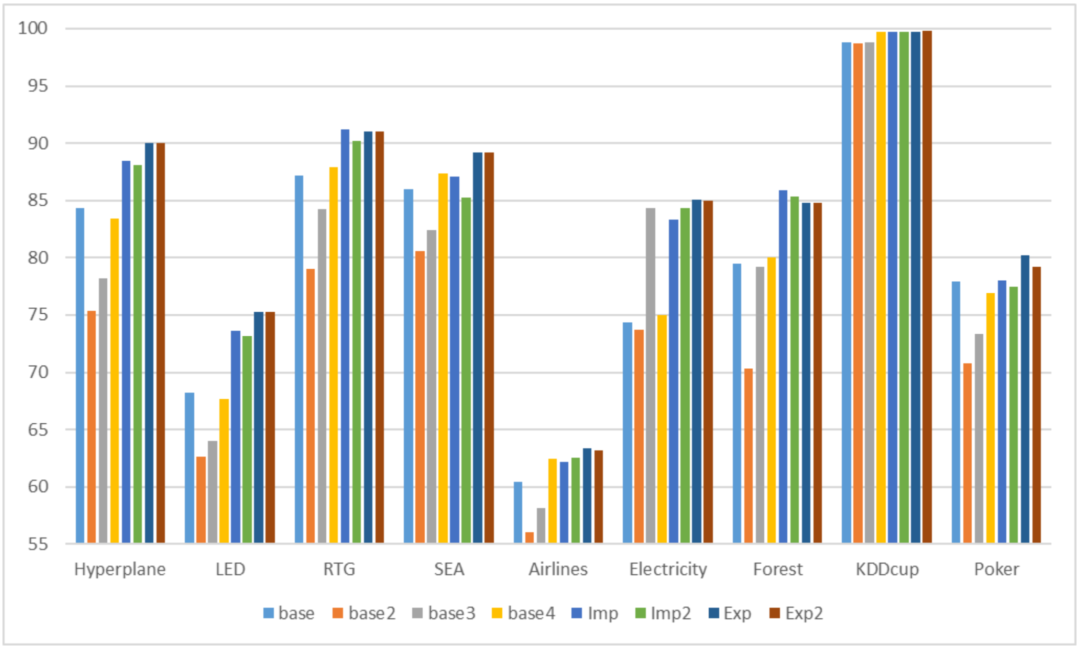


FIGURE 5.3: Accuracy (%) of the EACD variations over the considered nine datasets in the delayed setting.

$EACD_{Exp}$ variation has the best average performance in both immediate and delayed settings.

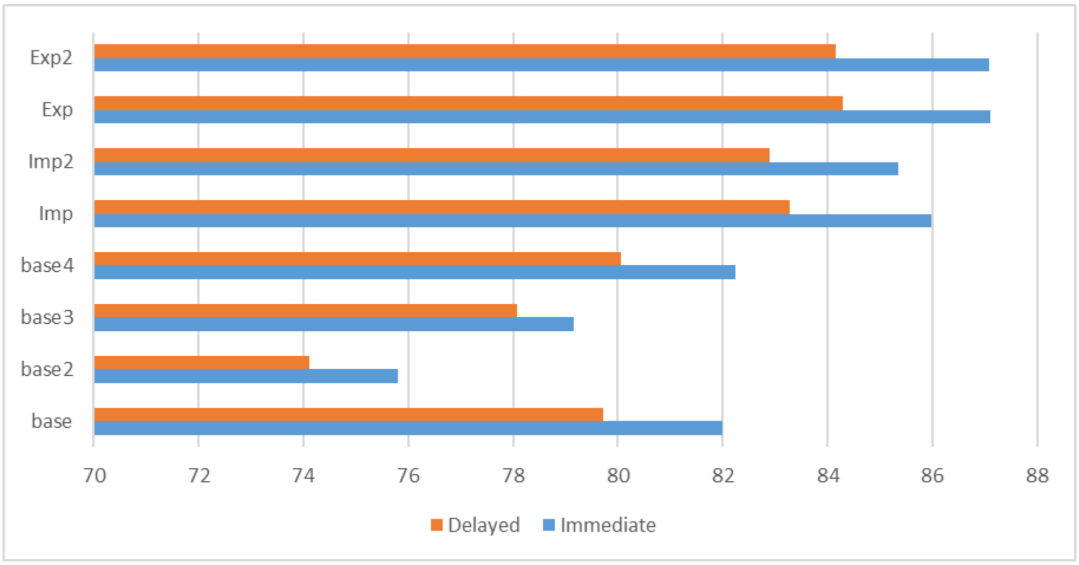


FIGURE 5.4: Average accuracy (%) of the EACD variations in the immediate and delayed settings.

As the difference between $EACD_{Imp}$ and $EACD_{Imp2}$ is in their number of generations used in each GA iteration, their accuracy is not significantly different, and $EACD_{Imp}$, which has a higher number of generations (15), performs better over all datasets. It is clear that the execution time of $EACD_{Imp2}$

is less than that of $EACD_{Imp}$ since GA performs faster on 10 generations compared to 15 generations. Similarly, as the difference between $EACD_{Exp}$ and $EACD_{Exp2}$ is in their GA mutation rate parameter, they both have comparable accuracy and execution time, and only $EACD_{Exp}$ accuracy is slightly better for the majority of the datasets.

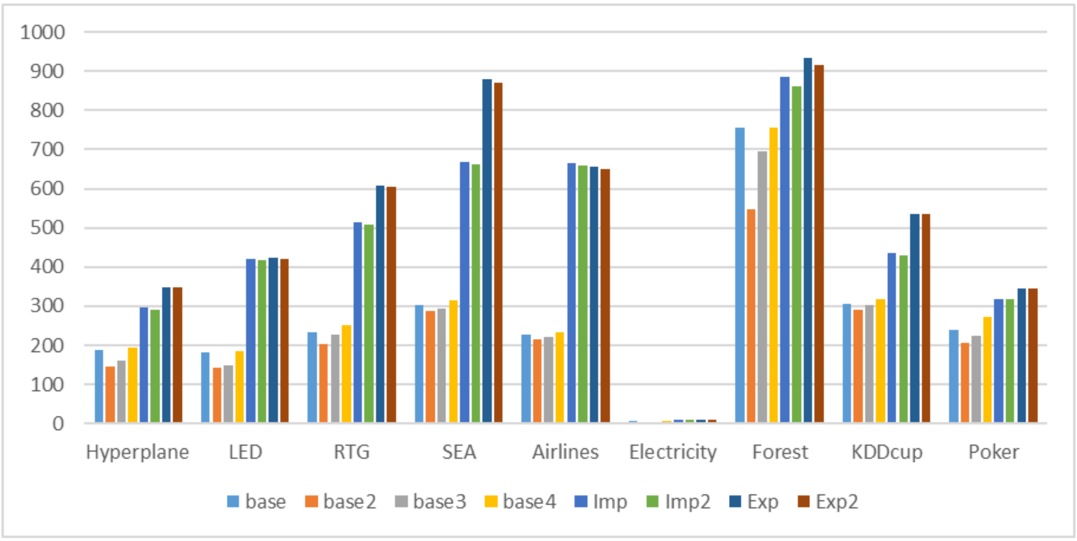


FIGURE 5.5: Average time (in seconds) of executing the EACD variations over the considered nine datasets in the immediate setting.

Figure 5.5 (Table A.3 in Appendix A) shows the overall execution time of the proposed EACD variations in seconds. It is clear that $EACD_{base2}$, which does not use the optimisation layer and has the lowest values of both p and m parameters, is the fastest variation. $EACD_{Imp}$ and $EACD_{Imp2}$ are slightly less time-consuming compared to $EACD_{Exp}$ and $EACD_{Exp2}$ because they do not use a concept drift detection algorithm. Finally, the execution times of the explicit variations of EACD ($EACD_{Exp}$ and $EACD_{Exp2}$) are similar as their only difference is in the GA mutation rate, which does not affect the times severely. Note that the execution times do not have significant difference in the immediate and delayed settings; hence, only the execution times of the immediate setting are listed.

Tables 5.1 and 5.2 show the average, minimum and maximum accuracy along with the standard deviation of the proposed $EACD_{Exp}$ method compared to the other state-of-the-art methods over the nine datasets in the immediate and delayed settings, respectively. Note that the minimum, maximum, average and standard deviation of the accuracies are based on the 10 different evaluation runs (each evaluation run has its specific accuracy) that

TABLE 5.1: Accuracy (%) of $EACD_{Exp}$ compared to the state-of-the-art methods in the immediate setting. Bold values indicate the best performance for each dataset.

Dataset	Criteria	ARF	DWM	LevBag	OAUE	OSBoost	$EACD_{Exp}$
Hyper.	Ave.	88.17	89.64	91.03	**91.42**	85.85	90.59
	σ	1.90	0.83	1.60	1.46	3.01	1.95
	Min	85.96	88.45	88.92	89.66	81.80	87.67
	Max	91.31	90.94	93.54	93.63	89.87	93.76
LED	Ave.	74.05	75.05	74.22	73.99	74.15	**75.45**
	σ	0.31	3.10	0.31	0.10	0.11	1.99
	Min	73.58	73.86	73.93	73.89	74.05	71.04
	Max	74.45	83.83	74.52	74.09	74.26	78.50
RTG	Ave.	78.35	59.35	90.78	88.88	**93.40**	91.42
	σ	8.12	8.87	2.26	3.26	1.45	2.82
	Min	65.86	48.26	87.38	83.39	90.63	86.31
	Max	88.03	73.86	93.72	92.56	95.24	94.56
SEA	Ave.	88.67	87.72	87.59	88.69	85.56	**90.08**
	σ	0.58	0.57	1.67	0.58	0.35	2.94
	Min	88.40	87.18	85.41	88.13	85.25	86.54
	Max	89.55	88.26	89.28	89.25	85.86	93.74
Airlines	Ave.	63.53	63.97	59.42	64.02	61.98	**66.61**
	σ	1.23	0	0.73	0	0	3.10
	Min	62.08	63.97	58.45	64.02	61.98	60.34
	Max	65.46	63.97	60.62	64.02	61.98	70.23
Elec.	Ave.	92.17	75.73	92.09	91.60	88.02	**92.14**
	σ	0.94	0	1.48	0	0	1.76
	Min	90.45	75.73	89.56	91.60	88.02	89.56
	Max	93.19	75.73	93.70	91.60	88.02	94.72
Forest	Ave.	**93.57**	83.75	92.73	90.70	84.45	91.73
	σ	1.58	0	2.10	0	0	3.10
	Min	91.11	83.75	89.45	90.70	84.45	88.34
	Max	95.09	83.75	95.40	90.70	84.45	95.12
KDDcup	Ave.	99.81	99.04	**99.82**	99.80	99.74	99.78
	σ	0.06	0	0.01	0	0	0.10
	Min	99.74	99.04	99.80	99.80	99.74	99.54
	Max	99.91	99.04	99.83	99.80	99.74	99.85
Poker	Ave.	84.19	74.37	**88.52**	80.74	84.31	86.21
	σ	4.55	0	3.34	0	0	2.37
	Min	80.08	74.37	84.67	80.74	84.31	82.34
	Max	90.06	74.37	93.56	80.74	84.31	89.34

are conducted for each method and (each setting). More information on different evaluation runs for each data stream is provided in Section 4.2.

The $EACD_{Exp}$ variant is chosen among all considered variants because it performs the best over the majority of considered datasets (see Tables A.1, A.2 in Appendix A and 5.3). Note the best results for each dataset are highlighted in bold. In the immediate setting (Table 5.1), $EACD_{Exp}$ has the best average accuracy over four datasets, LevBag performs the best over two datasets, while OAUE, OSBoost and ARF achieve the best accuracy over one dataset. In the delayed setting (Table 5.2), $EACD_{Exp}$ has the best average accuracy over five datasets, OAUE achieves the best performance over two datasets, while OSBoost and LevBag achieve the best accuracy over one dataset.

Figure 5.6 illustrates the average accuracy of EACD ($EACD_{Exp2}$) compared to the other state-of-the-art methods over all datasets in both the immediate and delayed settings. As can be observed from this figure, EACD has the best average accuracy in both the immediate and delayed settings.

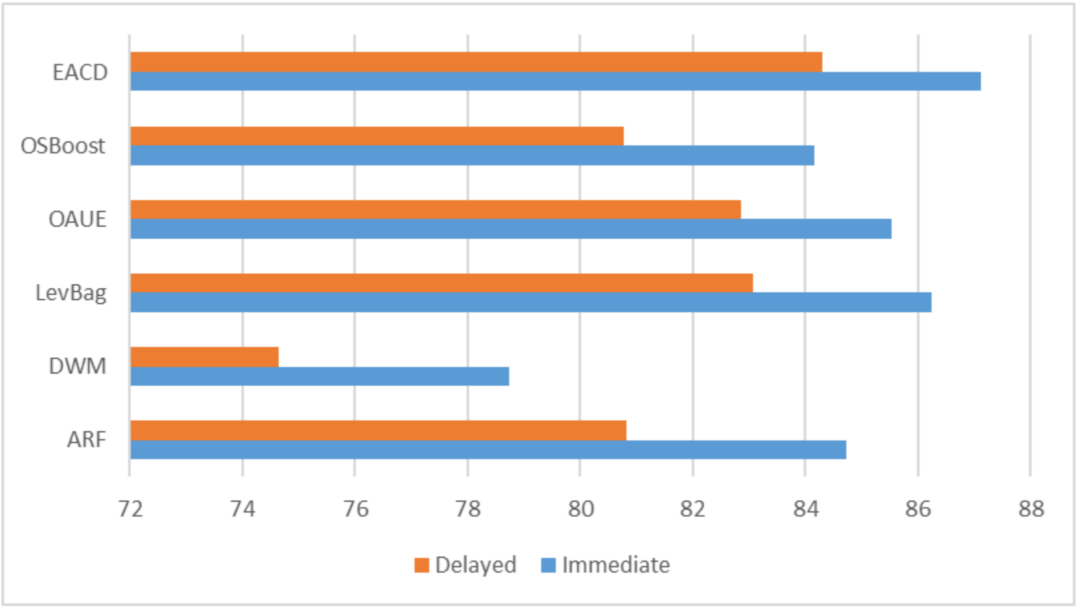


FIGURE 5.6: Average accuracy of the $EACD_{Exp2}$ and other state-of-the-art methods in the immediate and delayed settings.

Figure 5.7 (Table A.4 in Appendix A) shows the overall execution time of $EACD_{Exp}$ compared to the other state-of-the-art methods. For the majority of the datasets, DWM and OSBoost achieve the shortest execution time by far, while EACD has the longest execution time for the majority of the datasets.

Figure 5.8 demonstrates the behaviour of the proposed $EACD_{Exp}$ method along with the other considered methods over the SEA data stream upon

TABLE 5.2: Accuracy (%) of $EACD_{Exp}$ compared to the state-of-the-art methods in the delayed setting. Bold values indicate the best performance for each dataset.

Dataset	Criteria	ARF	DWM	LevBag	OAUE	OSBoost	$EACD_{Exp}$
Hyper.	Ave.	88.05	89.41	90.77	**91.10**	85.74	90.02
	σ	2.02	0.95	1.71	1.59	3.06	2.01
	Min	85.56	88.25	88.60	89.21	81.70	86.64
	Max	91.35	90.86	93.37	93.55	89.78	92.95
LED	Ave.	74.00	74.14	74.21	74.06	74.13	**75.26**
	σ	0.40	0.16	0.15	0.14	0.04	1.33
	Min	73.62	73.99	74.07	73.93	74.10	72.94
	Max	74.49	74.30	74.36	74.19	74.17	77.04
RTG	Ave.	78.24	59.49	90.91	88.72	85.53	**91.05**
	σ	8.06	8.67	2.48	5.13	2.90	3.75
	Min	65.81	49.16	86.94	82.17	81.17	84.64
	Max	87.92	73.73	93.69	93.47	88.70	94.56
SEA	Ave.	88.94	87.48	88.70	88.54	85.31	**89.22**
	σ	0.59	1.02	1.45	0.70	0.42	2.43
	Min	88.28	86.01	86.89	87.81	84.92	86.04
	Max	89.51	88.21	90.32	89.21	85.91	91.89
Airlines	Ave.	61.42	60.57	58.49	62.73	61.80	**63.35**
	σ	1.12	0	0.89	0	0	3.78
	Min	61.22	60.57	57.03	62.73	61.80	59.06
	Max	63.32	60.57	59.65	62.73	61.80	68.34
Elec.	Ave.	83.51	67.43	81.78	80.20	79.04	**85.03**
	σ	1.19	0	0.88	0	0	2.50
	Min	81.78	67.43	80.54	80.20	79.04	80.45
	Max	84.80	67.43	83.00	80.20	79.04	88.85
Forest	Ave.	85.65	74.93	86.22	**86.84**	74.47	84.83
	σ	02.60	0	2.72	0	0	2.36
	Min	83.67	74.93	84.30	86.84	74.47	81.45
	Max	90.49	74.93	84.30	86.84	74.47	88.23
KDDcup	Ave.	99.80	99.12	**99.81**	99.78	99.74	99.76
	σ	0.07	0	0.01	0	0	0.11
	Min	99.72	99.12	99.79	99.78	99.74	99.48
	Max	99.90	99.12	99.83	99.78	99.74	99.84
Poker	Ave.	67.95	59.31	76.78	73.81	**81.23**	80.21
	σ	2.92	0	3.72	0	0	2.01
	Min	64.94	59.31	70.51	73.81	81.23	76.35
	Max	73.29	59.31	79.34	73.81	81.23	83.24

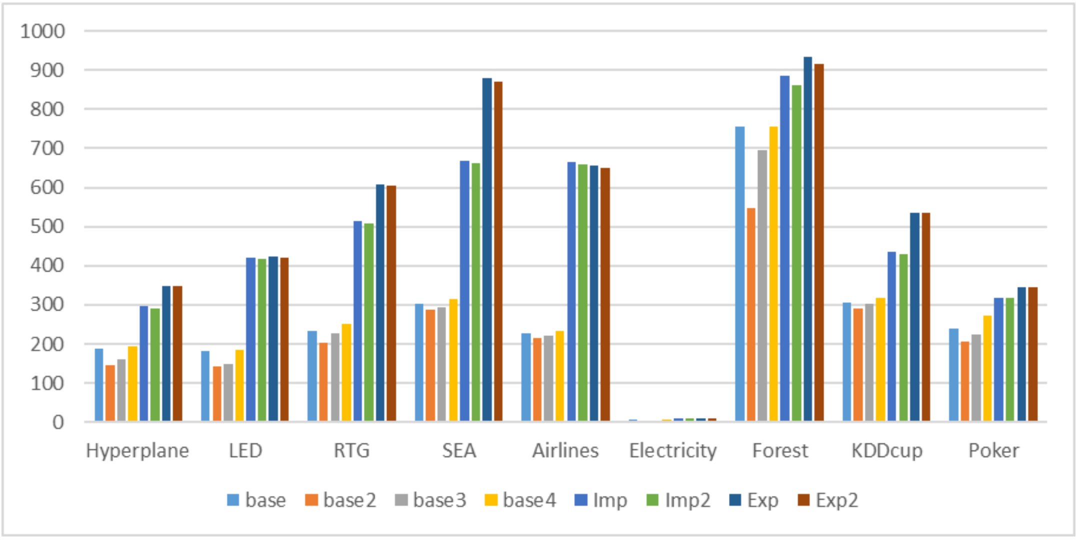


FIGURE 5.7: Average time (in seconds) of executing the $EACD_{Exp2}$ and other state-of-the-art methods over the considered nine datasets in the immediate setting.

different concept drifts (abrupt, gradual and recurrent) added manually to different stages of the data stream (instance numbers 200K, 400K, 600K and 800K) in the immediate (Figures 5.8(a, c, b and g)) and delayed (Figures 5.8(b, d, f and h)) settings. In Figures 5.8(a) and 5.8(b), an abrupt concept drift centred in the instance number 200K is added with a width of one. In Figures 5.8(c) and 5.8(d), a recurrent concept drift centred in the instance number 600K is added with a width of one. In Figures 5.8(e) and 5.8(f), a gradual concept drift centred in the instance number 400K is added with a width of 10,000. And finally in Figures 5.8(g) and 5.8(h), a recurrent concept drift centred in the instance number 800K is added with a width of 10,000.

5.3.4 Statistical Analysis

The Friedman test [87] is a non-parametric statistical test similar to the parametric repeated measures ANOVA (Analysis of Variance). It is used to detect differences across several algorithms in multiple test attempts (datasets). For this test, we need to demonstrate that the Null-hypothesis (stating that there is no significant difference between different algorithms) is rejected [88].

The Friedman test is distributed according to Equation 5.4 with $k-1$ degrees of freedom:

$$\chi_F^2 = \frac{12N}{k(k+1)}\left[\sum_{j=1}^{k} R_j^2 - \frac{k(k+1)^2}{4}\right], \tag{5.4}$$

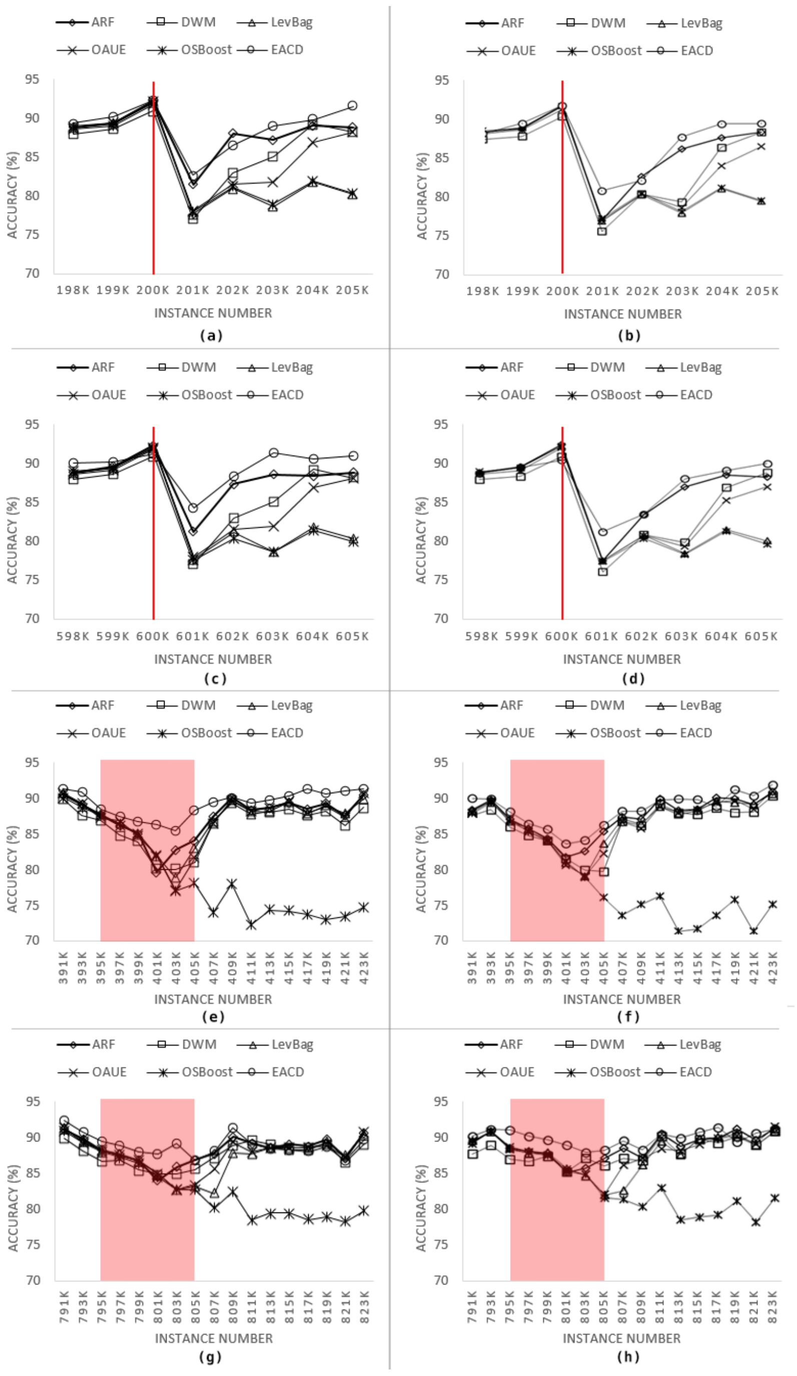


FIGURE 5.8: Behaviour of the $EACD_{Exp2}$ and other state-of-the-art methods upon different concept drifts added to the SEA dataset in the immediate setting (left column; a,c,e and g) and delayed setting (right column; b,d,f and h). The red boxes indicate the location and length of the added concept drifts.

TABLE 5.3: Average rank of the $EACD_{Exp}$ and other state-of-the-art methods according to the Friedman test.

Setting	ARF	DWM	LevBag	OAUE	OSBoost	$EACD_{Exp}$
Immediate	3.33	4.78	2.89	3.44	4.44	**2.11**
Delayed	3.78	5.11	2.67	3	4.44	**2**

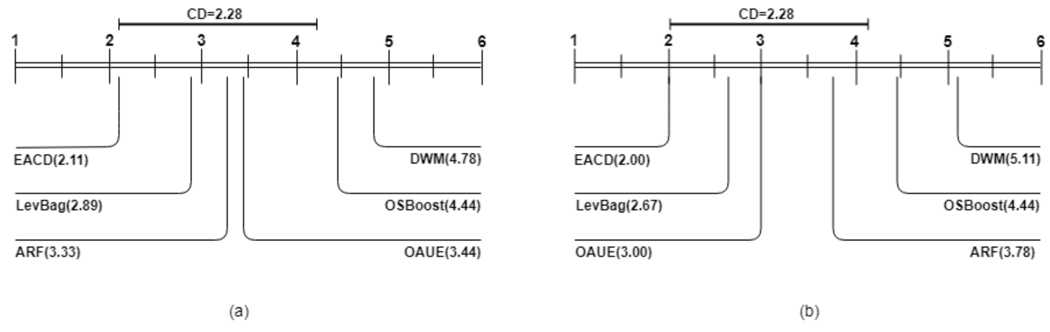


FIGURE 5.9: Comparison of the $EACD_{Exp}$ and other state-of-the-art methods using the Nemenyi test with 90% confidence level for the (a) immediate and (b) delayed settings.

where R_j denotes the rank of the j-th of k algorithms and N denotes the number of datasets. Table 5.3 shows the average rank of each method included in the experiments in both the immediate and delayed settings.

Note that for each setting, $k = 6$ and $N = 9$, as there are six methods and nine different datasets. Providing the value of the Friedman test statistic is $\chi_F^2 = 12.49$ for the immediate setting and $\chi_F^2 = 17.38$ for the delayed setting with 5 $(k - 1)$ degrees of freedom, and the critical value for the Friedman test given $k = 6$ and $N = 9$ is 10.78 at the significance level $\alpha = 0.05$, we can conclude that the accuracy values of the studied methods are significantly different in both settings as their χ_F^2 values (12.49 and 17.38) are greater than the critical value (10.78).

Now that the Null-hypothesis is rejected, we can proceed with a post-hoc test. The Nemenyi test [89] can be used when several classifiers are compared to each other [88]. The performance of two classifiers is significantly different if their corresponding average ranks differ by at least the critical difference (CD).

The critical value in our experiments with $k = 6$ and $\alpha = 0.10$ is $q_{0.10} = 2.28$. As a result, the accuracy of the proposed $EACD_{Exp}$ method is significantly different from that of DWM and OSBoost, whereas it is not significantly different from LevBag, ARF and OAUE. Figure 5.9 graphically represents the comparison of the methods in both settings based on the Nemenyi test.

5.3.5 Discussion

As can be observed from Figures 5.2, 5.3 and 5.5, the average accuracy values of the explicit variations ($EACD_{Exp}$ and $EACD_{Exp2}$) are slightly better than those of the implicit variations ($EACD_{Imp}$ and $EACD_{Imp2}$). Furthermore, the accuracy values of the variations that use the GA optimisation technique are significantly better than those of the base variations for all datasets. By looking at the results of $EACD_{base}$, $EACD_{base4}$ and $EACD_{Exp}$, it can be concluded that using a concept drift detection mechanism alone cannot improve the results significantly, whereas using the concept drift detector along with a stochastic optimiser (GA) improves the accuracy significantly.

Among the variations that use only the base layer of the proposed algorithm, those that use a higher number of *types* and a higher number of features in each *type* ($EACD_{base4}$ and $EACD_{base}$) perform better compared to the other variations in the majority of the experiments. This is because the former variations create more classifiers on each time-step, with each classifier covering more features. This also justifies why they are more time consuming compared to the other base-layer variations. Furthermore, when using a concept drift detection mechanism along with the base layer in $EACD_{base4}$ variation, it fails to improve the accuracy significantly compared to the variation with the same parameters but without using a concept drift detector in $EACD_{base}$ (improving only by 0.25% in the immediate setting and by 0.33% in the delayed setting). The explanation for this might be that while concept drift detectors can be very helpful for achieving a fast reaction to evolving data, they can also be destructive upon false alarms, especially when trained classifiers are removed immediately upon concept drifts.

While the average accuracy of the explicit variations is significantly better than that of the base variations, their execution time is significantly longer than that of the base variations in all experiments. This is because the base variations use only the first layer of the proposed architecture and not the optimisation layer, unlike the implicit and explicit variations that use both layers. Furthermore, since the combination of the features in random subspaces (*types*) in the base variations is not optimised during the run, and only the number of classifiers in each subspace is changed, the overall accuracy depends greatly on the initial selection of the features. However, in the implicit and explicit variations, the combination of the features in each subspace is reconstructed by GA when needed.

The difference between the implicit and explicit variations of the proposed method is in the time it takes them to decide when to let GA start optimising a set of subspaces using the buffer of recently stored instances. Since the average accuracy of $EACD_{Exp}$ is about 1.13% higher than that of $EACD_{Imp}$ in the immediate setting and 1.02% higher in the delayed setting, we can conclude that one of the most challenging parts of the proposed architecture is to decide when GA needs to reconstruct the combination of classification *types* in the optimisation layer.

When looking at the results presented in Tables 5.1 and 5.2, it can be noticed that DWM, OAUE and OSBoost have the same standard deviation of zero for all real-world datasets, whereas RD3+GA, LevBag and ARF have different standard deviation values. This is because the latter algorithms use randomisation in their procedures, whereas the former do not. Since the experiments over the real-world datasets are repeated 10 times over the same data, the results obtained from all deterministic algorithms in all iterations are the same.

It can be further noticed from Tables 5.1 and 5.2 that for the artificial datasets, the standard deviation values for OSBoost, OAUE and DWM vary greatly throughout the experiments, reaching the value of about 8% for the RTG dataset. At the same time, the standard deviation values for LevBag, ARF and $EACD_{Exp}$ do not vary a lot, hardly reaching the value of 3.78%. This might be because the first three methods (OSBoost, OAUE and DWM) are implicit and do not use any concept drift detection mechanisms, whereas the other methods (LevBag, ARF and $EACD_{Exp}$) are explicit and use concept drift detection mechanisms. As explicit methods have an immediate reaction to concept drifts, their accuracy does not drop for a long time throughout the experiments.

From Table 5.7, it can be noticed that DWM has the lowest execution time over four datasets, OSBoost – over three datasets, whereas ARF and OAUE – over one dataset. The main drawback of the $EACD_{Exp}$ variation of the proposed algorithm is its execution time, which is the longest for the majority of the datasets (six out of nine). The main reason for this is that this variation uses two different evolutionary algorithms (RD and GA) along with a concept drift detection method (EDDM). However, the other variations of the proposed method offer slightly shorter execution times in $EACD_{Imp}$ and $EACD_{Imp2}$, and significantly shorter times in $EACD_{base}$, $EACD_{base2}$ and $EACD_{base3}$. This is because the implicit variations of the EACD algorithm use both evolutionary algorithms but no concept drift detection method, while the base variations use only one evolutionary algorithm (RD) with no drift detection method.

From Figure 5.8(a) featuring an abrupt concept drift, it can be noticed that the $EACD_{Exp}$ and ARF methods coped with the drift better than the other methods with almost similar reactions (DWM, OAUE and LevBag coped with the drift more slowly, while OSBoost failed to adapt to it in a good time). The same good performance of $EACD_{Exp}$ and ARF can be observed for this drift in the delayed setting (Figure 5.8(b)); however, the accuracy drop upon the drift is more drastic in ARF compared to $EACD_{Exp}$. Using different random *types* in the base layer of $EACD_{Exp}$ results in a more robust performance, especially over drifting data, when the data distribution is not known in advance. At the same time, both the $EACD_{Exp}$ and ARF methods use explicit strategies allowing them to detect concept drifts as soon as they occur and use their recovery mechanisms. Furthermore, abrupt concept drifts are generally easier to detect for methods using drift detectors since the data distribution changes suddenly in such drifts.

From Figures 5.8(c) and 5.8(d) featuring a recurrent concept drift with a width of one at the instance number 600K, it can be noticed that the accuracy of all methods dropped, with $EACD_{Exp}$ taking less time to adapt to the new data distribution and gain its average accuracy back again in both the immediate and delayed settings. This might be because the proposed method uses two different mechanisms to cope with new environments: one (RD) weights the classification *types* based on their performance, while the other (GA) optimises the combination of the attributes of these *types*.

From Figures 5.8(e) and 5.8(f) featuring a gradual concept drift with a width of 10,000 and centred in the instance number 400K, it is clear that $EACD_{Exp}$ coped with this concept drift in a more robust manner compared to the other methods in both settings. In the situations when a concept drift happens gradually, the time of detecting the drift plays an important role in how the drift is addressed since the majority of explicit methods start their adaptation procedure at that time. Hence, failing to detect a gradual drift on time can cause the methods to suffer from a late adaptation. In the proposed method, the adaptation to drifts can be divided into two stages: (1) before the drift is detected, when the algorithm tries to seamlessly adapt to the drift using RD; and (2) after the drift is detected, when GA starts to optimise the combination of the attributes in the optimisation layer. This justifies the better performance of the proposed method, especially upon gradual concept drifts.

From Figure 5.8(g) featuring a recurrent concept drift with a width of 10,000 in the immediate setting, it can be seen that the accuracy of all methods dropped within the same rate. However, $EACD_{Exp}$ took less time to adapt compared to the other methods. In Figure 5.8(h), where the the same drift is shown in the delayed setting, the behaviour of all methods except OSBoost is relatively similar; however, the accuracy of $EACD_{Exp}$ degrades less than that of the other methods during the drifting period (shown by the red box). In both settings, OSBoost fails to continue improving its performance for at least 14,000 instances from the instance number 805K. This behaviour of OSBoost is similar to its results upon abrupt and gradual concept drifts, which shows that the method lacks a sound adaptation mechanism over different types of concept drifts.

Overall, the main advantage of the proposed $EACD_{Exp}$ method is its accuracy; it has the **best average rank compared to the other state-of-the-art methods** used in the experiments (as shown in Table 5.3). It also proved to have **the fastest reaction over evolving data** on most occasions, especially upon abrupt, gradual and recurrent concept drifts, as shown in Figure 5.8.

While EACD is specifically designed to cope with non-stationary environments, it is possible to use it in stationary environments. However, the main limitation in this case would be the unnecessary overhead that the method puts on the ensemble since EACD always builds classifiers over different time-stamps of the target data stream, while there is no need to do that when a data stream does not evolve.

5.4 Summary

In this chapter, we proposed a novel method for non-stationary data stream classification capable of seamlessly adapting to different types of concept drifts. This method, called EACD, employs a two-layer architecture with a set of classifiers in each layer. The first layer (*base layer*) is constructed by creating a randomly drawn set of subspaces (classification *types*) from the pool of features of the target data stream. Each *type* is the basis for building decision trees (classifiers) in a layer. To allow EACD to seamlessly adapt to concept drifts, RD is used to increase or reduce the number of trees in each *type* according to their recent performance in the data stream. The second layer (*optimisation layer*) uses randomly drawn subspaces from the first layer as the first population for GA employed to optimise the classification *types* with the most recent instances. The process of creating new and training the current classifiers in this layer is the same as in the base layer. For the optimisation layer, two different mechanisms are proposed to determine when to restart GA. The first mechanism is based on comparing the performance of the two layers (implicit EACD), whereas the second one uses a concept drift detection method to check when a new concept drift occurs (explicit EACD).

To test the proposed method and its variations, a set of experiments with five real-world and four artificial datasets was conducted. First, the performance of different variations of the proposed method was compared; then, the best performing variation was compared to the state-of-the-art methods proposed in the literature. All experiments were executed in two different settings: *immediate prequential* and *delayed prequential*. The results showed that the proposed EACD method (namely, its $EACD_{exp}$ variant) achieves the highest average accuracy and best average rank among all compared methods in both settings. However, the overall execution time of the proposed method is the longest in six out of nine datasets, which makes the execution time to be the main drawback of EACD.

Using the Friedman statistical test, it was demonstrated that the accuracy values of the studied methods are significantly different. According to the Nemenyi test (which is a post-hoc test of the Friedman test), the accuracy of the proposed EACD method is significantly different from that of DWM and OSBoost, while it is not significantly different from that of ARF, LevBag and OAUE.

To improve the execution time and average performance of EACD, we propose the REplicator Dynamics & GENEtic algorithm (RED-GENE) method in the next chapter, which satisfies the third objective identified in this thesis (see Section 1.4). RED-GENE extends EACD by proposing three different modifications of RD employed in its *base layer*.

Chapter 6

RED-GENE: Efficient Replicator Dynamics & Genetic Algorithm Approach to Adaptive Data Stream Classification

In the previous chapter, we described a novel method called Evolutionary Adaptation to Concept Drifts (EACD) addressing the second objective of this thesis (see Section 1.4). This chapter introduces the REplicator Dynamics & GENEtic algorithm (RED-GENE) method aiming at improving the execution time and average performance of EACD, which satisfies the third objective of this thesis, namely, **to propose efficient ways of deploying evolutionary algorithms for non-stationary data stream classification problems**. In particular, we propose three novel modifications of RD over its version employed in the *base layer* of EACD. The work presented in this chapter is published as a full paper titled "RED-GENE: An Evolutionary Game Theoretic Approach to Adaptive Data Stream Classification" in the "IEEE Access" journal [25].

6.1 Introduction

The majority of the state-of-the-art ensemble methods for data stream classification are focused on either a specific type of concept drifts or a specific type of applications. In this chapter, we propose a novel ensemble learning method called RED-GENE that performs well regardless of the concept drift type or application. As an improvement of EACD, RED-GENE employs the same approach to creating different classification *types* and GA optimisation as described in Chapter 5. However, in contrast to the most basic modified version of RD employed in EACD, three novel *evolutionary game-theoretic* strategies based on RD are proposed in RED-GENE to retain the classification accuracy upon different concept drifts. As such, in addition to the contributions of Chapter 5, this chapter offers the following further contributions: (1) three novel modifications of RD to accelerate the concept drift adaptation process and advance the way RD is employed; (2) improving the classification accuracy of EACD for the majority of the considered experimental cases; and (3) reducing the running time of EACD by generating a lower number of classifiers.

The rest of this chapter is organised as follows. Section 6.2 details the proposed RED-GENE method. Section 6.3 outlines the experimental setup and presents the results of comparing (i) different variations of RED-GENE; and (ii) the best performing variation to other state-of-the-art methods. Finally, conclusions and the summary of the chapter are provided in Section 6.4.

6.2 RED-GENE Description

This section introduces a novel ensemble learning method for data stream classification tasks in non-stationary environments, namely, the RED-GENE method. The architecture of RED-GENE comprises two layers, the *base layer* and *optimisation layer*, described in Sections 6.2.1 and 6.2.2, respectively. In the *base layer*, several classification *types* are created based on randomly selected features (subspaces) of the target data stream to form an ensemble. These classification *types* are trained using one of the three proposed RD strategies. The combination of features inside each *types* is then optimised in the *optimisation layer* similar to the optimisation layer of EACD as outlined in Section 5.2.2.

6.2.1 Base Layer

As a first step of the RED-GENE algorithm, p percent of data features (attributes) are randomly selected from a pool of features of the target data stream as $n = \frac{p}{100} \times f$, where p denotes the percentage of the attributes (features) to be selected randomly ($p \in (0, 100)$), f denotes the total number of attributes of the target dataset and n denotes the total number of attributes to be chosen in this stage. Each iteration of this step creates a single classification *type* in the ensemble. Hence, it should be repeated m times to generate m different classification *types*, each having n randomly selected attributes. In the proposed method, m is a parameter representing the maximum number of classification *types* to be selected for the ensemble and should be chosen based on the total number of attributes in each *type*.

In this study, we assume that data received by the system are in the form of data blocks of 100 instances. After receiving the true labels of all instances inside each data block, m new classifiers (decision trees in this case, one for each *type*) are trained. These classifiers are then used to classify the next instances inside the data stream. In particular, the majority voting is performed, and the class with the biggest number of votes is considered as the final output of the *base layer* of the ensemble.

Every classification *type* is evaluated once the true labels of instances in a specific data block are available. The accuracy of each classification *type* is assumed to be the average accuracy of all classifiers built using the same set of features (i.e. classification *type*), namely, $a_i = \frac{c_i}{db}$, where c_i denotes the number of correctly classified instances in the i^{th} data block and db denotes the total number of instances in each data block. To allow the *types* to reach a maturity level, the classifiers are not evaluated during the first $\frac{max}{2}$ data blocks, where max denotes the maximum number of classifiers allowed in each classification *type*.

Next, the *replicator dynamics* step is applied (see Algorithm 9), where each *type's* accuracy is taken into consideration and assessed in terms of its expected payoff as in Equation 6.1.

$$\begin{cases} a(t_i) \geq \frac{\sum_{i=1}^{m} a(t_i)}{m} \Rightarrow grow \\ a(t_i) < \frac{\sum_{i=1}^{m} a(t_i)}{m} \Rightarrow shrink, \end{cases} \tag{6.1}$$

where $a(t_i)$ denotes the accuracy of i^{th} *type* and m denotes the total number of *types*. In RED-GENE, a classification *type* comprising randomly selected features of data acts as a single replicator according to the *replicator equation* explained in Section 3.2. The expected payoff for each replicator is set to the average accuracy of all replicators (classification *types*), while a replicator's payoff is the average accuracy of the classifiers built using the same classification *type*.

Once the *types* to be grown or shrunk are identified using the *replicator dynamics* step, the pool of classifiers is reformed based on one of the proposed strategies. In particular, the following three novel RD-based strategies employed in RED-GENE are among the main contributions of this thesis.

RD1: Weighted Trees

According to the original RD algorithm, the number of trees to be added to or removed from the ensemble is specified dynamically as in Equation 6.2.

$$\begin{aligned} T_a(i) &= \lfloor (a(t_i) - \frac{\sum_{i=1}^{m} a(t_i)}{m}) \times T_n(i) \rfloor, \\ T_r(i) &= \lfloor (\frac{\sum_{i=1}^{m} a(t_i)}{m} - a(t_i)) \times T_n(i) \rfloor, \end{aligned} \tag{6.2}$$

where $T_a(i)$ denotes the number of trees to add, $T_r(i)$ denotes the number of trees to remove, $a(t_i)$ denotes the accuracy of subspace i being processed, m denotes the total number of *types* and $T_n(i)$ denotes the total number of trees currently being inside classification *type* i. However, adding more than one classifier (at one time-step) is not possible in an online environment, where one-time processing is applied to incoming data (prequential evaluation). Therefore, a higher weight is assigned to a decision tree (classifier) instead of training more than one classifier in the first proposed RD variation. This is achieved using the original Equation 6.2, where $T_a(i)$ is assumed to denote the weight to be assigned to the newly built classifier. In other words, assigning a *weight* of k to a classifier means that the same classifier gets k votes instead of only one according to the voting mechanism. The removing mechanism in this strategy is similar to that of the original RD algorithm and is based on the performance of the trees; e.g. when the number of trees to remove is k, the k least accurate trees (over the last data block) are removed from the ensemble. Path 1 in Figure 6.1 depicts the flowchart of this strategy.

RD2: Voting without Considering Poor-performing *Types*

According to the second proposed RD variation, a subspace is temporarily eliminated from the ensemble's voting mechanism when it is recognised as poor-performing based on its accuracy (i.e. identified as to be shrunk according to Equation 6.1). The evaluation of such subspaces continues as normal and they are re-activated once they start to perform better (and thus grow in size) again. Furthermore, the number of trees to be added or removed is fixed to $T_a(i) = 1$ and $T_r(i) = 1$, respectively. Therefore, in case of growing/shrinking, only one tree is supposed to be added/removed. Path 2 in Figure 6.1 shows the flowchart of RD2.

RD3: Weighted Trees + Voting without Considering Poor-performing *Types* (RD1 + RD2)

The third proposed RD variation combines the first two strategies (RD1 and RD2). Hence, the number of trees to add or remove is set dynamically based on Equation 6.2, and poor-performing *types* are eliminated from the ensemble's voting system unless they start to grow again. Figure 6.1 depicts the flowchart of this strategy.

Note that all classifiers inside the ensemble are updated (retrained) using all incoming data blocks. This can lead to a faster adaptation to concept drifts in the evolving data stream. To restrict the size of the ensemble and avoid adding overhead to the system, an upper bound is assigned to the number of classifiers of each classification *type* (in this study, the upper bound was set to an arbitrary value of $max = 20$). When the maximum size of a classification *type* is exceeded, a classifier should be removed from the same *type* before adding a new classifier. This is achieved by removing the least accurate classifier for the last data block. Furthermore, a lower bound for each *type* is set to a minimum number of $min = 1$ to prevent the *types* to be completely removed. Hence, when only one decision tree related to a type remains, it is not being removed upon a poor performance.

6.2.2 Optimisation Layer

Similar to EACD (see Chapter 5), GA is used in RED-GENE to optimise the combination of attributes in each classification *type* (subspace) using the existing *types* drawn randomly in the first phase of the proposed algorithm. In particular, features act as individuals or genes, while classification *types* act as chromosomes according to GA. Therefore, in each iteration of GA, the related operations (selection, crossover and mutation) are applied to the population, and the fitness of each *type* is calculated over a fixed set of data stored in the system as a buffer keeping the most recent data records and their corresponding labels. This buffer acts as a search space for the GA algorithm to perform the most up-to-date evaluation. Note that the buffer is kept fixed during each iteration of the GA algorithm.

GA is applied to all randomly drawn classification *types* (from the base layer). It performs optimisation once a drift detector verifies a concept drift. The *optimisation layer* is different from the *base layer* only in this part (i.e. recombining *types* upon concept drifts), while the classification, training and

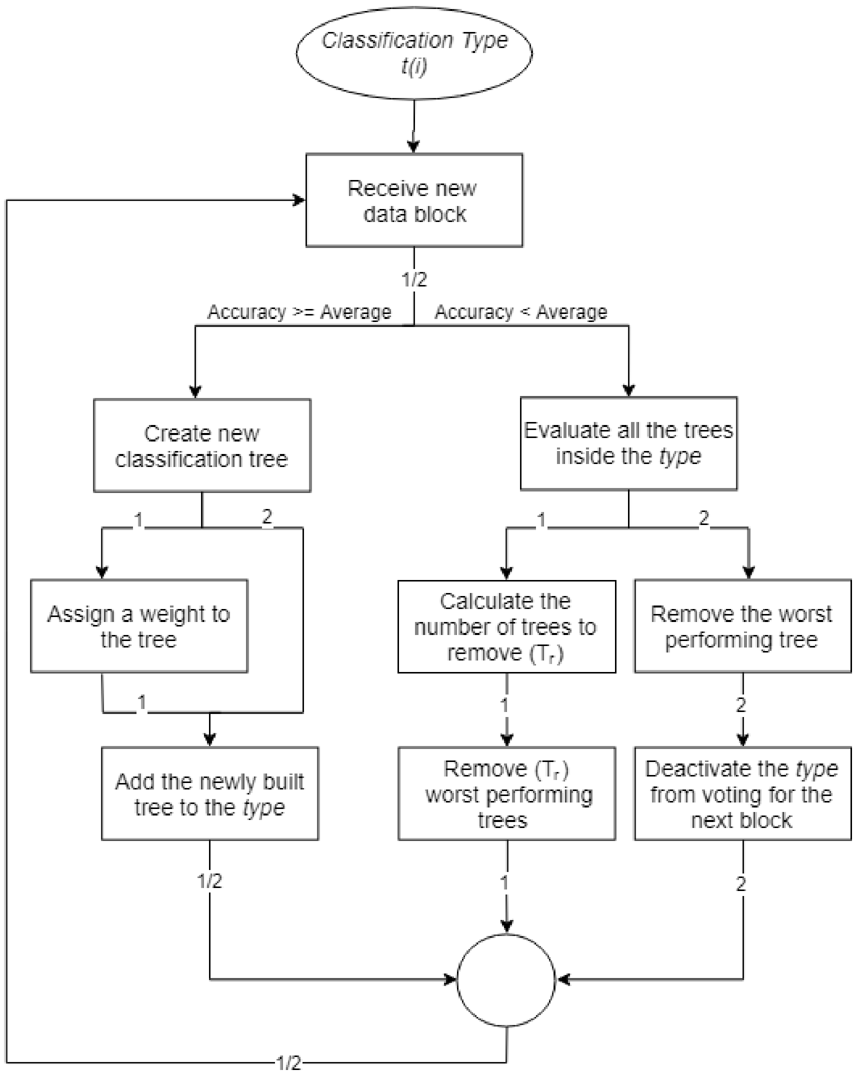

Classification Type t(i)
Receive new data block
1/2
Accuracy >= Average
Accuracy < Average
Create new classification tree
Evaluate all the trees inside the type
1
2
Assign a weight to the tree
Calculate the number of trees to remove (Tr)
Remove the worst performing tree
Add the newly built tree to the type
Remove (Tr) worst performing trees
Deactivate the type from voting for the next block
1/2

FIGURE 6.1: Illustration of the proposed variations of Replicator Dynamics; RD1: Path 1, RD2: Path 2, RD3: Paths 1 and 2.

Algorithm 9: PROPOSED RD STRATEGIES

```
Input: Continuous block of data $DB = \{db_1, db_2, \ldots, db_n\}$
$n$: number of features to be selected in each type
$m$: total number of types
$max$: maximum number of classifiers in each type.
Output: Classified samples
1  $i := 1$
2  for $t := 1$ to $t := m$ do
3      Randomly select $n$ features
4  while data stream is not empty do
5      if $i \leq \frac{max}{2}$ then
6          Classify($db_i$)
7          Grow(T) for all types
8      else
9          Classify($db_i$)
10         Evaluate()
11         if $a(t_j) \geq \frac{\sum_{j=1}^{n} a(t_j)}{m}$ then
12             Grow($t_j$) /* Based on the strategy */
13         else
14             Shrink($t_j$) /* Based on the strategy */
15     Train()
16     $i := i + 1$
```

updating mechanisms are the same as explained for the *base layer* (see Section 6.2.1).

Algorithm 10 outlines the process of the *optimisation layer*.

Selection stage: selection of chromosomes (classification *types*) in this stage is based on their performance over the search space (i.e. data inside the buffer). The *types* with a higher accuracy than the overall average accuracy are selected as the candidates for the crossover stage. The "Selection()" function in Algorithm 10 corresponds to the Selection stage.

Crossover stage: the classification *types* selected in the Selection stage are paired with each other to produce new offspring. In other words, the attributes (Genes) of well-performing *types* (Chromosomes) are mixed with each other to produce new *types*. The "Crossover()" function in Algorithm 10 corresponds to the Crossover stage.

Mutation stage: to prevent different *types* from getting too similar to each other (i.e. using the same features), the mutation step is applied during the breeding procedure. For this purpose, the mutation rate, which is a parameter of the proposed method, should be specified. In this study, the mutation rate was set to the arbitrary value of 5%. The "Mutation()" function in Algorithm 10 corresponds to the Mutation stage.

The proposed GA modification is iterated over the fixed data inside the buffer until the maximum number of generations are produced. The resulting classification *types* are grown by training new classifiers over the incoming data blocks. The newly built classifiers are then evaluated over the incoming data. Once their average performance reaches the average accuracy of the ensemble, they are used in its voting mechanism.

In RED-GENE, all classifiers inside the *base layer* are assigned with an initial weight of one ($W_b = 1$), while those inside the *optimisation layer* – two ($W_g = 2$). This intensifies the effect of the *optimisation layer* on the algorithm given the optimality of the *types*.

In general, the proposed RED-GENE method works as follow. All instances inside a new data block are classified by classifiers inside both layers of the ensemble (base and optimisation layers). The predictions of active classifiers are then summarised by the decision making part of the ensemble according to their weight and output. A concept drift detection mechanism is employed to determine when to start/restart the GA procedure. When GA is due to start its procedure, the classifiers inside its layer (i.e. optimisation layer) are removed from the ensemble to make room for a new collection of classification *types* due to be built.

Once the concept drift detector confirms a drift inside the data stream, GA starts to iterate the necessary operations over the fixed data inside the buffer. We use EDDM [78] specifically designed to improve the detection in the presence of gradual concept drifts. At the same time, any other drift detection methods can be employed at this stage. When the concept drift detector signals a *warning*, the buffer starts storing the incoming data. Once the drift is confirmed by the detector (signalling a *drift*), GA starts its procedures with all data inside the buffer being fixed for the fitness calculation process. The "DriftDetector()" function in Algorithm 10 corresponds to the *Concept Drift Detection* stage of RED-GENE.

Using concept drift detectors can sometimes lead to false positives and false negatives, resulting in an accuracy drop in the classification process. False positives that is when the concept drift detector wrongly signals a drift, should not negatively affect the accuracy of RED-GENE since the new set of classification *types* is based on the most recent data, and only an extra overhead would be required to perform additional GA optimisation. False negatives that do not trigger the optimisation layer can lead to a delay in adapting to a concept drift only if the concept drift is abrupt and significant. On the other hand, the base layer always grows well-performing types and shrinks poor-performing types according to the most recent data, which would help the system to adapt to a concept drift even if it is not detected.

Algorithm 10: GA OPTIMISATION

```
Input: Buffer
g: Maximum number of generations
Resetting mechanism: [implicit/explicit]
Randomly drawn subspaces (types) from the base layer, $T_B = \{t_1, t_2, .., t_m\}$
Output: New set of types, $T_G = \{t_1\prime, t_2\prime, .., t_m\prime\}$
1 for $i := 1$ to $i := g$ do
2     Selection()
3     Crossover()
4     Mutation()
5 repeat
6     DriftDetector()
7 until DriftDetector() = Drift /*When the detector signals a drift*/
8 Reset(GA) /*Clear optimisation layer and restart GA*/
```

6.3 Experimental Study

This section provides the details of the experiments conducted to investigate the performance of the proposed method in this chapter. We introduce different variations of RED-GENE along with their parameters and computational complexity. Different variations are then compared with each other and the best performing variation is compared with the state-of-the-art algorithms in non-stationary data stream classification tasks. Eventually, the results are statistically analysed and discussed thoroughly. It is worth mentioning that the details necessary for conducting the experiments in this thesis (e.g. experimental settings, compared methods, evaluation runs, datasets, etc.) are discussed in Chapter 4.

6.3.1 RED-GENE Variations

The following nine variations of RED-GENE were compared to evaluate the effects of the proposed RD strategies and their parameters:
RD1 uses RD with weighted trees (see Section 6.2.1) and the following parameters: $p = 60\%$ and $m = 0.6 \times f$;
RD2 uses RD without weighted trees and does not consider poor-performing types in voting, as explained in Section 6.2.1, with $p = 60\%$ and $m = 0.6 \times f$;
RD3 uses RD with weighted trees and does not consider poor-performing types in voting, with $p = 60\%$ and $m = 0.6 \times f$;
RD1Lite is a lite version of RD1 with $p = 40\%$ and $m = 0.4 \times f$;
RD2Lite is a lite version of RD2 with $p = 40\%$ and $m = 0.4 \times f$;
RD3Lite is a lite version of RD3 with $p = 40\%$ and $m = 0.4 \times f$;
RD1+GA is a version of RD1 optimised using GA with the number of generations (g) set to 15 and the mutation rate set to 5% (other parameters: $p = 60\%$, $m = 0.6 \times f$);
RD2+GA is a version of RD2 optimised using GA with the number of generations (g) set to 15 and the mutation rate set to 5% (other parameters: $p = 60\%$, $m = 0.6 \times f$);
RD3+GA is a version of RD3 optimised using GA with the number of generations (g) set to 15 and the mutation rate set to 5% (other parameters: $p = 60\%$, $m = 0.6 \times f$).

6.3.2 Computational Complexity

Assuming the number of classes c, number of attributes in each classification *type* p, values per attribute v and maximum number of trees in the ensemble k, no more than p attributes can be considered in a single Hoeffding tree [44]. Each attribute at a node requires computing v values. Since calculating information gain requires c arithmetic operations, the cost of k Hoeffding trees at each time-step in the worst case scenario is O($kcpv$).

Given the number of classification *types* in the ensemble m and assuming the proposed RD modifications use m arithmetic operations to calculate payoffs, the cost of applying RD to the ensemble is only O(m). Hence, the time complexity of deploying the RED-GENE variations without GA optimisation (RD1, RD2, RD3, RD1Lite, RD2Lite and RD3Lite) is O($m + (kcvp)$).

Assuming the size s of the GA population and total number of generations g, the cost of GA optimisation is O(sg) at each time when the optimisation layer is restarted. Furthermore, given d as the number of instances in each data block and f as the total number of features in the dataset, the EDDM drift detection method, which uses J48 (C4.5) decision tree as its learning mechanism, requires O(df^2) of time. Hence, the time complexity of deploying the RED-GENE variations that use GA optimisation (RD1+GA and RD2+GA) is O($m + (kcvp) + (sg) + (df^2)$).

6.3.3 Results and Discussion

The considered methods are compared based on standard criteria, namely, the classification accuracy and overall execution time (including the time of making predictions, training and evaluating the ensemble). Figures 6.2 and 6.3 (Tables B.1 and B.2 in Appendix B) show the average accuracy scores for all variations of RED-GENE and $EACD_{Exp}$ (explained in Chapter 5) over the nine datasets in the immediate and delayed settings, respectively. The GA-optimised variations (RD1+GA, RD2+GA, RD3+GA and $EACD_{Exp}$) perform better than the RD-only variations (RD1, RD2 and RD3) and significantly better than the RDLite variations (RD1Lite, RD2Lite and RD3Lite). This is because the former variations include a concept drift detection mechanism enabling the system to optimise the combination of features in every *type* of the ensemble, especially upon concept drifts. RD3+GA has the best average accuracy over seven out of nine datasets in the immediate setting, five out of nine datasets in the delayed setting and the best overall average accuracy in both the immediate and delayed settings. Furthermore, since the same parameters were employed for both $EACD_{Exp}$ and the RD3+GA variation, we can analyse the impact of applying the proposed RD variations. Consequently, it is evident that the proposed RD strategies improve the classification accuracy compared to the basic RD variation used in $EACD_{Exp}$.

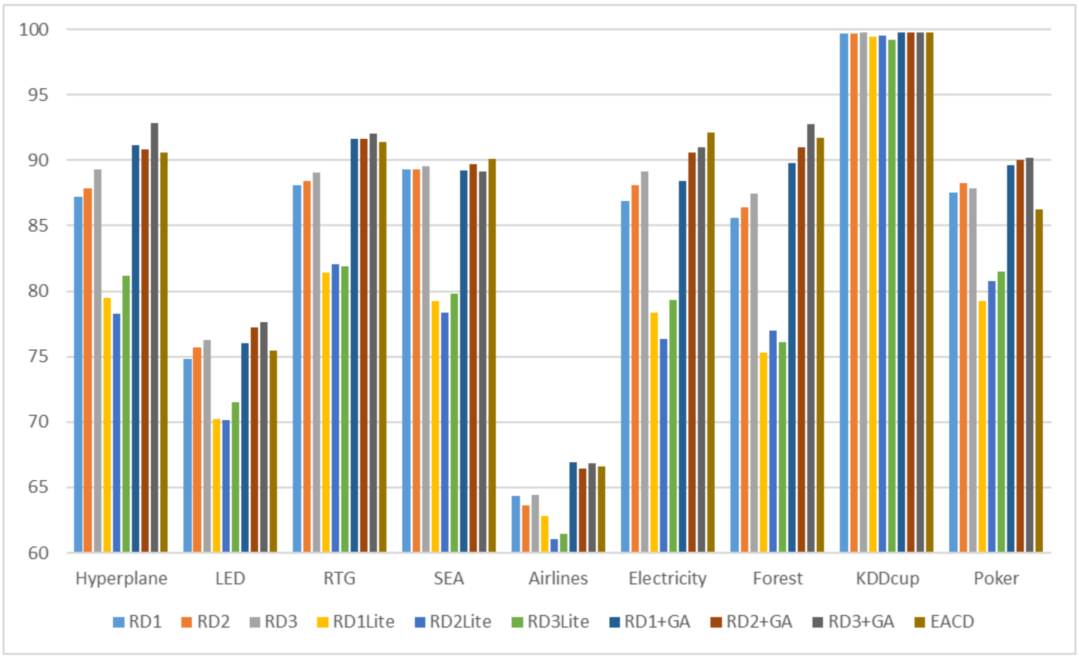


FIGURE 6.2: Accuracy (%) of the RED-GENE variations and $EACD_{Exp}$ over the considered nine datasets in the immediate setting.

Figure 6.4 illustrates the average accuracy (throughout all datasets) of all EACD variations in both immediate and delayed settings. It is clear that the $RD3 + GA$ variation has the best average performance in both immediate and delayed settings.

Tables 6.1 and 6.2 list the average accuracy scores and their standard deviation, minimum and maximum values of the RD3+GA variation (as the best performing RED-GENE variation) and considered state-of-the-art methods in the immediate and delayed settings, respectively. Note that the minimum, maximum, average and standard deviation of the accuracies are based on the 10 different evaluation runs (each evaluation run has its specific accuracy) that are conducted for each method. More information on different evaluation runs for each data stream is provided in Section 4.2.

It can be noticed from the tables that RD3+GA has the best average accuracy over Hyperplane, LED, SEA, Airlines, Electricity and Poker-Hand datasets, as well as the best overall average accuracy in both the immediate and delayed settings. It can also be noticed that DWM, OAUE and OSBoost have the same standard deviation of zero for all real-world data sets, while

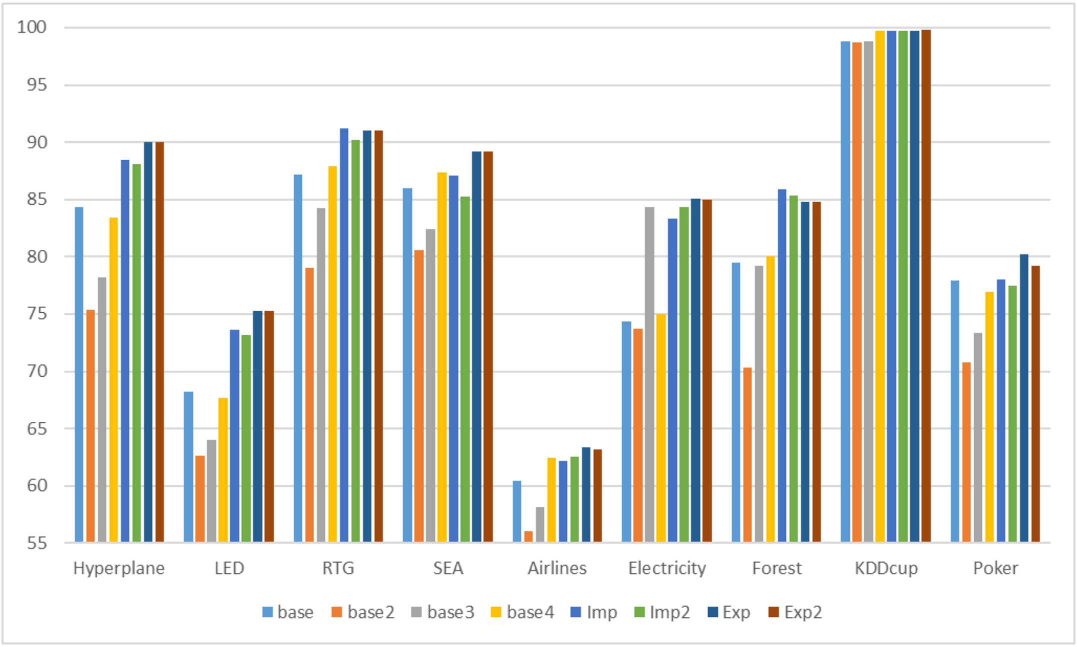


FIGURE 6.3: Accuracy (%) of the RED-GENE variations and $EACD_{Exp}$ over the considered nine datasets in the delayed setting.

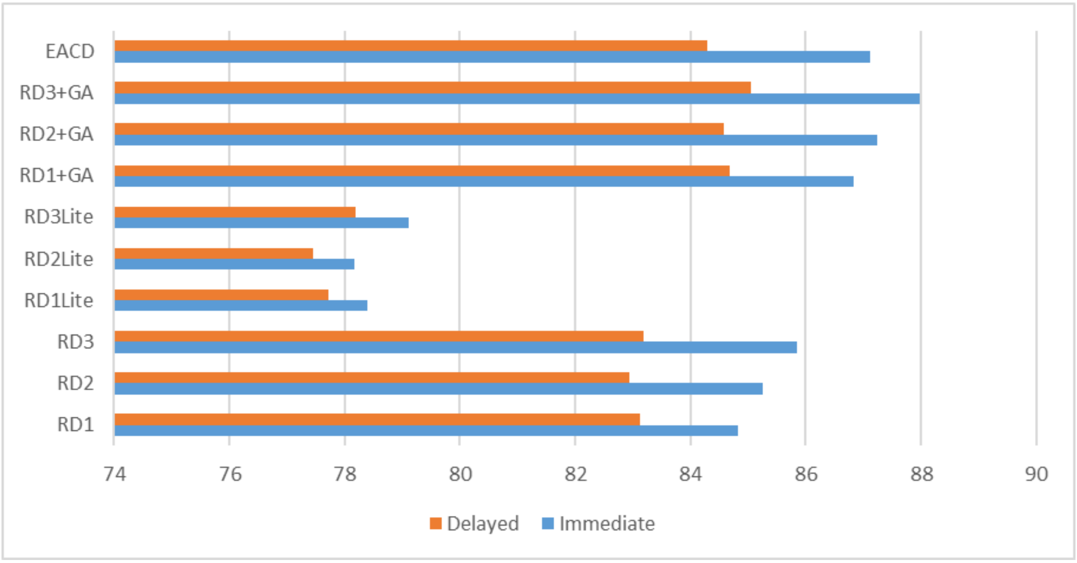


FIGURE 6.4: Average accuracy (%) of the RED-GENE variations and $EACD_{Exp}$ in the immediate and delayed settings.

RD3+GA, LevBag and ARF have different standard deviation values. This is because the latter algorithms use randomisation in their procedures, while the former do not.

Figure 6.5 illustrates the average accuracy (throughout all datasets) of EACD ($EACD_{Exp}$) method compared to other state-of-the-art methods in both immediate and delayed settings. As can be observed from this figure,

TABLE 6.1: Average accuracy (%) scores and their standard deviation, minimum and maximum values of RED-GENE and the compared methods in the immediate setting. RD3+GA achieves the highest average accuracy in five out of 9 datasets. Bold values indicate the best performance for each dataset.

Dataset	Criteria	ARF	DWM	LevBag	OAUE	OSBoost	RD3+GA
Hyper.	Ave.(%)	88.17	89.64	91.03	91.42	85.85	**92.84**
	σ(%)	1.90	0.83	1.60	1.46	3.01	2.02
	Min(%)	85.96	88.45	88.92	89.66	81.80	89.91
	Max(%)	91.31	90.94	93.54	93.63	89.87	95.24
LED	Ave.(%)	74.05	75.05	74.22	73.99	74.15	**77.65**
	σ(%)	0.31	3.10	0.31	0.10	0.11	2.45
	Min(%)	73.58	73.86	73.93	73.89	74.05	73.34
	Max(%)	74.45	83.83	74.52	74.09	74.26	81.24
RTG	Ave.(%)	78.35	59.35	90.78	88.88	**93.40**	92.03
	σ(%)	8.12	8.87	2.26	3.26	1.45	2.00
	Min(%)	65.86	48.26	87.38	83.39	90.63	89.31
	Max(%)	88.03	73.86	93.72	92.56	95.24	94.66
SEA	Ave.(%)	88.67	87.72	87.59	88.69	85.56	**89.14**
	σ(%)	0.58	0.57	1.67	0.58	0.35	2.43
	Min(%)	88.40	87.18	85.41	88.13	85.25	85.20
	Max(%)	89.55	88.26	89.28	89.25	85.86	91.11
Airlines	Ave.(%)	63.53	63.97	59.42	64.02	61.98	**66.88**
	σ(%)	1.23	0	0.73	0	0	1.43
	Min(%)	62.08	63.97	58.45	64.02	61.98	64.34
	Max(%)	65.46	63.97	60.62	64.02	61.98	68.32
Electricity	Ave.(%)	**92.17**	75.73	92.09	91.60	88.02	91.03
	σ(%)	0.94	0	1.48	0	0	3.14
	Min(%)	90.45	75.73	89.56	91.60	88.02	87.54
	Max(%)	93.19	75.73	93.70	91.60	88.02	93.16
Forest	Ave.(%)	**93.57**	83.75	92.73	90.70	84.45	92.73
	σ(%)	1.58	0	2.10	0	0	2.83
	Min(%)	91.11	83.75	89.45	90.70	84.45	88.92
	Max(%)	95.09	83.75	95.40	90.70	84.45	94.24
KDDcup	Ave.(%)	99.81	99.04	**99.82**	99.80	99.74	99.78
	σ(%)	0.06	0	0.01	0	0	0.14
	Min(%)	99.74	99.04	99.80	99.80	99.74	99.60
	Max(%)	99.91	99.04	99.83	99.80	99.74	99.84
Poker	Ave.(%)	84.19	74.37	88.52	80.74	84.31	**90.15**
	σ(%)	4.55	0	3.34	0	0	1.08
	Min(%)	80.08	74.37	84.67	80.74	84.31	88.24
	Max(%)	90.06	74.37	93.56	80.74	84.31	92.40

EACD method has the best average accuracy in both immediate and delayed settings.

Figure 6.6 (Table B.3 in Appendix B) demonstrates the execution time (that includes the time of making predictions, training and evaluating the ensemble) of the RED-GENE variations and $EACD_{Exp}$ method in seconds in the immediate setting. It is clear that the variations without GA optimisation are faster than the ones with it due to the time complexity of GA. Furthermore, all RDLite variations are faster than the other variations as they have fewer

TABLE 6.2: Average accuracy (%) scores and their standard deviation, minimum and maximum values of RED-GENE and the compared methods in the delayed setting. RD3+GA achieves the highest average accuracy in five out of nine datasets. Bold values indicate the best performance for each dataset.

Dataset	Criteria	ARF	DWM	LevBag	OAUE	OSBoost	RD3+GA
Hyper.	Ave.(%)	88.05	89.41	90.77	91.10	85.74	**92.03**
	σ(%)	2.02	0.95	1.71	1.59	3.06	2.00
	Min(%)	85.56	88.25	88.60	89.21	81.70	89.85
	Max(%)	91.35	90.86	93.37	93.55	89.78	94.45
LED	Ave.(%)	74.00	74.14	74.21	74.06	74.13	**76.73**
	σ(%)	0.40	0.16	0.15	14.00	0.04	2.63
	Min(%)	73.42	73.99	74.07	73.74	74.02	72.15
	Max(%)	74.49	74.30	74.36	74.19	74.17	80.21
RTG	Ave.(%)	78.24	59.49	**90.91**	88.72	85.53	90.24
	σ(%)	8.06	8.64	2.48	5.13	2.90	2.41
	Min(%)	65.81	47.16	86.94	82.17	81.17	85.19
	Max(%)	87.92	73.73	93.49	92.47	88.70	92.90
SEA	Ave.(%)	**88.94**	87.48	88.70	88.54	85.31	88.56
	σ(%)	0.59	1.02	1.45	0.70	0.42	2.41
	Min(%)	88.28	86.01	86.89	87.81	84.92	84.17
	Max(%)	89.51	88.21	90.32	89.21	85.91	90.73
Airlines	Ave.(%)	61.42	60.57	58.49	62.73	61.80	**64.45**
	σ(%)	1.12	0	0.89	0	0	1.90
	Min(%)	61.22	60.57	57.03	62.73	61.80	61.63
	Max(%)	63.32	60.57	59.65	62.73	61.80	67.04
Electricity	Ave.(%)	**83.51**	67.43	81.78	80.20	79.04	83.41
	σ(%)	1.19	0	0.88	0	0	2.84
	Min(%)	81.78	67.43	80.54	80.20	79.04	79.58
	Max(%)	84.80	67.43	83.00	8020	79.04	86.25
Forest	Ave.(%)	85.65	74.93	86.22	86.84	74.47	**87.19**
	σ(%)	2.60	0	2.72	0	0	2.04
	Min(%)	83.67	74.93	83.02	86.84	74.47	85.34
	Max(%)	90.49	74.93	89.50	86.84	74.47	91.04
KDDcup	Ave.(%)	99.80	99.12	**99.81**	99.78	99.74	99.76
	σ(%)	0.07	0	0.01	0	0	0.07
	Min(%)	99.72	99.12	99.79	99.78	99.74	99.59
	Max(%)	99.90	99.12	99.83	99.78	99.74	99.80
Poker	Ave.(%)	67.95	59.31	76.78	73.81	81.23	**83.11**
	σ(%)	2.92	0	3.72	0	0	1.62
	Min(%)	64.94	59.31	70.51	73.81	81.23	81.55
	Max(%)	73.29	59.31	79.34	73.81	81.23	86.09

types and features in each *type*. Figure 6.7 (Table B.4 in Appendix B) illustrates the overall execution times of RED-GENE and the considered state-of-the-art methods in seconds in the immediate setting. DWM and OSBoost have the shortest execution time by far for the majority of the datasets (DWM over SEA, Airlines, Electricity and Poker-Hand datasets, and OSBoost over Hyperplane, LED and Forest Cover-type datasets), while the proposed RED-GENE RD3+GA variation has the longest execution time in most cases.

In summary, the main advantage of RED-GENE is its accuracy providing

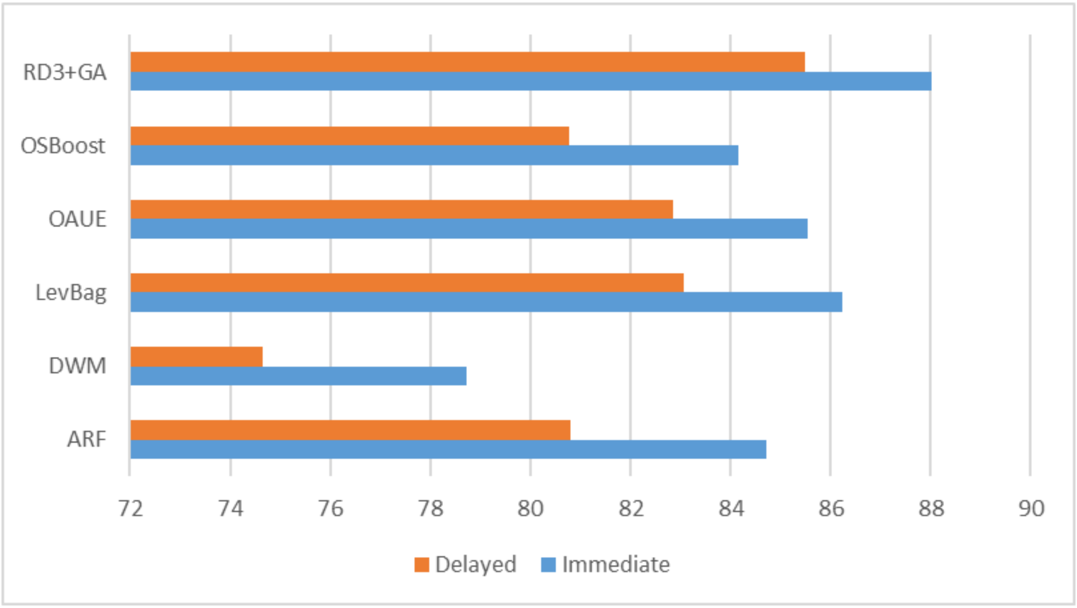


FIGURE 6.5: Average accuracy (%) of the RD+GA RED-GENE variation and other state-of-the-art methods in the immediate and delayed settings.

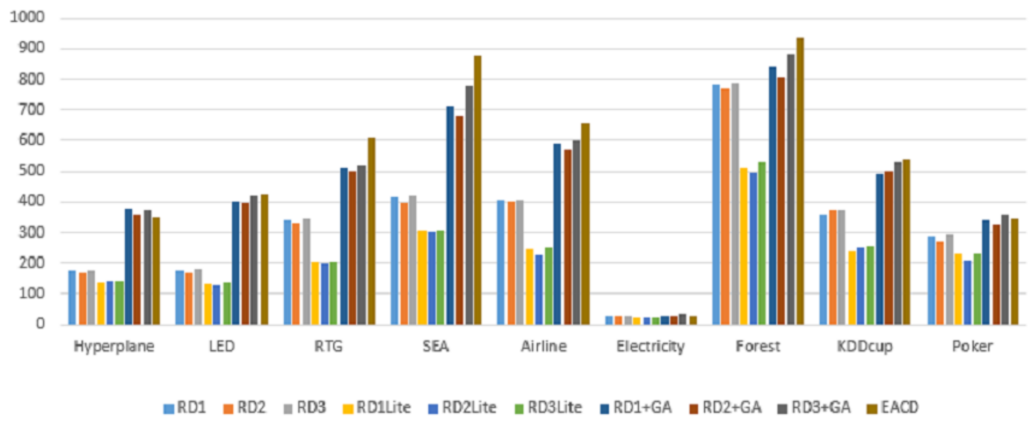


FIGURE 6.6: Average time (in seconds) of executing the RED-GENE variations and $EACD_{Exp}$ over the considered nine datasets in the immediate setting.

the best average rank compared to the state-of-the-art methods and **robust performance in evolving environments with concept drifts**. In addition, the RD strategies proposed in this chapter are more efficient than the basic RD strategy used in our previously proposed EACD method (Chapter 5). The main drawback of the best performing RD3+GA variation of RED-GENE is its longest execution time over six out of nine datasets, which is due to the employment of two different evolutionary algorithms (RD and GA), along with a concept drift detection method (EDDM). At the same time, the RD1, RD2 and RD3 variations of RED-GENE offer slightly shorter execution times,

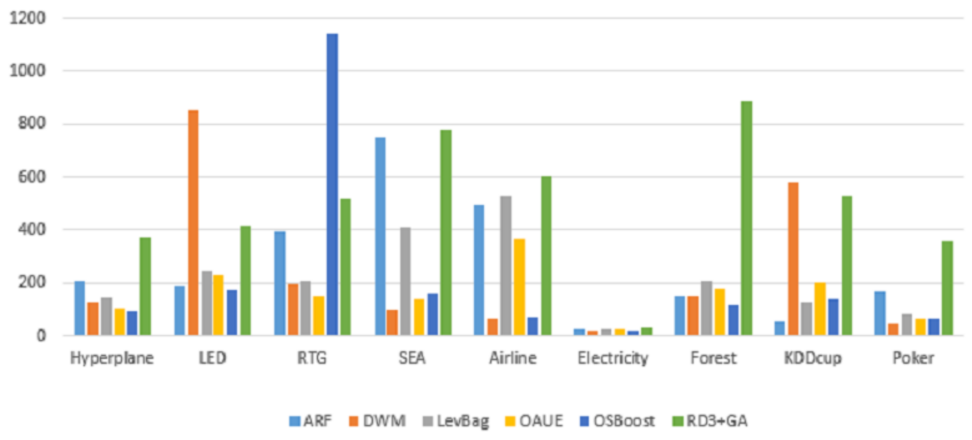


FIGURE 6.7: Average time (in seconds) of executing the RD+GA RED-GENE variation and other state-of-the-art methods over the considered nine datasets in the immediate setting.

TABLE 6.3: Average rank of the RD3+GA and other state-of-the-art methods according to the Friedman test.

Dataset	ARF	DWM	LevBag	OAUE	OSBoost	RD3+GA
R_j	3.500	4.944	2.888	3.333	4.500	**1.833**
R_j^2	12.250	24.447	8.346	11.111	20.250	3.361

and significantly shorter times in their *lite* versions (RD1Lite, RD2Lite and RD3Lite), while maintaining a high accuracy in many cases.

6.3.4 Statistical Analysis

We performed the Friedman [87] and Nemenyi [89] statistical tests to further analyse the results of our experiments regarding the RD3+GA variation of RED-GENE and considered state-of-the-art methods.

Table 6.3 shows the average rank and its squared value for each methods considered in our experiments with $k = 6$ and $N = 18$. Providing that the value of the Friedman test statistic is $\chi_F^2 = 31.80$ with 5 $(k-1)$ degrees of freedom, and the critical value for the Friedman test given $k = 6$ and $N = 18$ is 14.63 at a significance level of $\alpha = 0.01$, we can conclude that the accuracy values of the studied methods are significantly different (31.80 is greater than 14.63).

Now that the Null-hypothesis is rejected, we can proceed with a post-hoc test. The Nemenyi test [89] can be used for comparing classifiers to each other [88]. The performance of two classifiers is significantly different if the corresponding average ranks differ by at least the critical difference (CD).

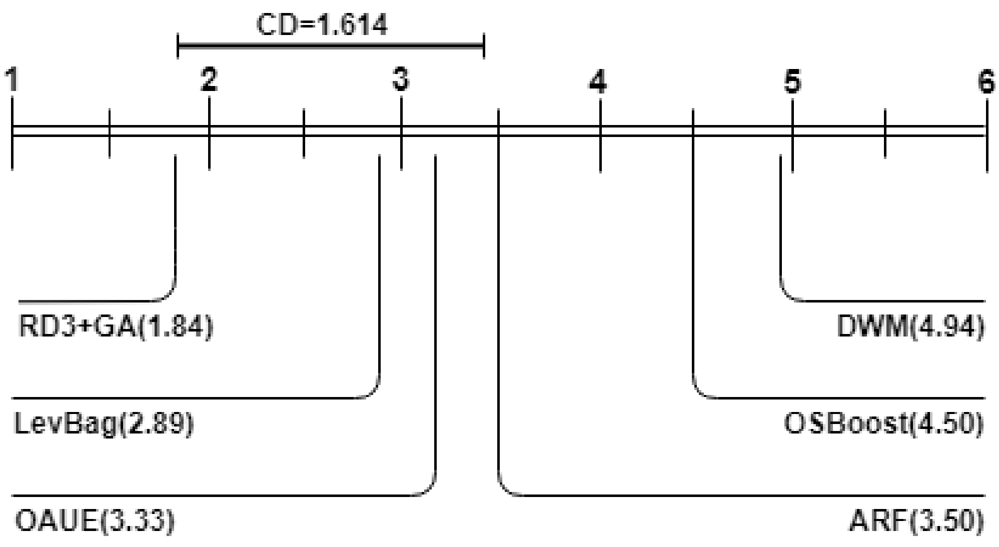


FIGURE 6.8: Comparison of the RD3+GA and other state-of-the-art methods using the Nemenyi test at $\alpha = 0.10$.

The critical value in our experiments with $k = 6$ and $\alpha = 0.10$ is $q_{0.10} = 1.614$. As a result, the accuracy of the proposed $RD3 + GA$ method is significantly different from that of ARF, DWM, and OSBoost methods, while it is not significantly different from that of LevBag and OAUE methods. Figure 6.8 graphically represents the comparison of the different methods employed in our experiments based on the Nemenyi test.

6.4 Summary

In this chapter, we proposed a method, called RED-GENE, capable of coping with different concept drifts when performing data stream classification tasks. RED-GENE combines three novel RD strategies with a GA optimisation technique. RD was used to grow and shrink a randomly drawn set of subspaces (classification *types*) in the ensemble according to the most recent instances of the data stream. GA was employed to optimise the combination of features of the *types* when a concept drift was detected by EDDM.

To test the proposed method, a set of experiments was conducted with four synthetic and five real-world data streams. Nine different variations of RED-GENE were implemented to examine different strategies of adopting RD in online learning and assess the effect of selecting its different parameters. In addition, RED-GENE was compared to the existing state-of-the-art methods and the previously proposed EACD method (Chapter 5). The results of the comparison demonstrated that RED-GENE had the highest average accuracy and best average rank among all compared methods in two different settings (immediate and delayed). However, the execution time of RED-GENE was the longest over the majority of the considered datasets compared to other state-of-the-art methods.

Consequently, using the Friedman statistical test, it was shown that the performance of the methods studied in the experiments was significantly different. Furthermore, using the Nemenyi test, it was proved that the performance of RED-GENE was significantly better than that of ARF, DWM and OSBoost, while it was not significantly better than that of LevBag and OAUE.

RED-GENE presents an improvement over EACD in terms of the *average accuracy score* and *execution time*, which leads to achieving the third objective of this thesis (as outlined in Section 1.4). In the next chapter, we propose the REplicator Dynamics & Particle Swarm Optimisation (RED-PSO) method that uses RD and PSO to tackle the last objective of this thesis by balancing out the exploration and exploitation aspects of the search space.

Chapter 7

RED-PSO: REplicator Dynamics & Particle Swarm Optimisation Approach to Adaptive Data Stream Classification

The previous chapter presented a novel ensemble learning method RED-GENE built upon the EACD method described in Chapter 5. EACD and REG-GENE were proposed to address the second and third objectives of this thesis as stated in Section 1.4, respectively. In this chapter, the REplicator Dynamics & Particle Swarm Optimisation (RED-PSO) method is proposed to address the final objective of this thesis, namely, **to optimise the metaheuristic solutions in non-stationary data stream classification by balancing out the exploration and exploitation of metaheuristics**. To achieve this, RED-PSO uses RD to optimise the number of classifiers in each classification *type* and PSO to optimise the combination of features in each *type*. The findings presented in this chapter are published as a full paper titled "A non-canonical hybrid metaheuristic approach to adaptive data stream classification" in the "Future Generation Computer Systems" journal [26].

7.1 Introduction

An ideal approach to non-stationary data stream classification should satisfy the following objective: achieve the least possible misclassification rate while minimising the computational complexity and quickly adapting to possible concept drifts. At present, there is a lack of a comprehensive approach satisfying this objective. To address this gap, this chapter introduces a novel method, called RED-PSO, that is based on a modified metaheuristic algorithm, namely, PSO. In particular the classical PSO algorithm is extended by taking into consideration both the exploration (local optima) and exploitation (global optima) aspects of the search space (combination of features).

The architecture of RED-PSO includes three layers. Each layer is initially assigned some predefined classification *types* randomly created from a pool of features of the target data stream. Similar to EACD and RED-GENE, RED-PSO employs RD to seamlessly cope with smooth (i.e. gradual or incremental) concept drifts. In particular, RD allows the classification *types* with a good performance to grow and those with a poor performance to shrink in size. The combination of features in all types is then optimised using a modified version of PSO for each layer individually. This helps the method to cope with more sudden (i.e. recurring or abrupt) concept drifts. PSO allows the types in each layer to move towards local (within the same type) and global (in all types) optimums with a specified velocity.

RED-PSO was evaluated using the four synthetic data stream generators and five real-world data streams introduced in Section 4.2. Different types of concept drifts were added to the synthetic datasets to examine how the proposed method adapts to different concept drifts compared to the state-of-the-art methods. Many real-world data streams can be categorised into completely and partially labelled. Complete labelling can be achieved when the true labels of respective instances are completely accessible either instantly or after a delay, presenting no or small overhead to the system. This is the case in many forecasting tasks such as weather forecasting, stock market analysis, forest monitoring, airline predictions and bill estimation. Partial labelling is performed when the real labels are accessible with an extra overhead to the system via a third party (usually human). This is the case when the real labels are retrieved after an analysis of the related data in the stream (e.g. medical diagnosis in health data and anomaly/fraud detection in credit card transactions). RED-PSO was designed to deal with completely labelled data streams. Hence, the experimental part of this chapter assumes that the true labels of instances are completely accessible either instantly or after a specified delay. In this respect, each dataset was processed twice: once in the delayed setting (where the actual labels of instances are accessible after a specified time) and once in the immediate setting (where the actual labels are accessible instantly).

The rest of this chapter is organised as follows. Section 7.2 details the proposed RED-PSO method. Section 7.3 outlines the experimental setup and results of comparing (i) different RED-PSO variations; and (ii) the best performing variation to other state-of-the-art methods. Finally, Section 7.4 presents conclusions and outlines the summary of this chapter.

7.2 RED-PSO Description

In this section, we present RED-PSO, a novel ensemble learning method for non-stationary classification using the RD and PSO techniques. RED-PSO trains an ensemble of classifiers comprising three categories (layers) of different classification *types* generated by randomly selecting features (subspaces) from a set of attributes of the target dataset. To make the proposed method effective in evolving environments and ensure its seamless adaptation to the most recent types of the data distribution, we apply RD to all layers of the ensemble to grow well-performing *types* and shrink poor-performing ones in size. Since the original RD is designed specifically for static datasets, we modify it to be compatible with streaming data in the same manner as described for EACD (Chapter 5) and RED-GENE (Chapter 6). Unlike EACD and RED-GENE, where GA is applied after RD, in the next step of RED-PSO, the randomly drawn classification *types* (subspaces) are optimised using a modified version of PSO applied to each layer individually to ensure fast adaptation to different concept drifts.

PSO is a metaheuristic algorithm [27] inspired by the social behaviour of the movement of organisms in a bird flock or fish school. This algorithm is introduced in Chapter 3 in detail. The main goal of the PSO algorithm is finding the global minimum of a function using both exploration and exploitation aspects of the search space. While PSO does not guarantee an optimal solution, it is shown to have promising results in various applications [28].

A canonical PSO algorithm is designed to iterate over a static data, where there is only one possible optimal solution. In contrast, in data stream classification tasks, data arrive in an online manner, and the optimal solution is subject to change with time. Therefore, we propose a non-canonical version of PSO as described in the next section to enable the classical PSO algorithm to work in streaming environments.

7.2.1 RED-PSO Algorithm

First, three different layers are created, with each layer containing a predefined number of *types* and each classification *type* covering a predefined percentage of features from a pool of features. There is a negative linear correlation between the percentage of features in each *type* and the number of *types* in each layer. In other words, the bigger the number of *types* inside each layer is, the smaller the number of features each *type* is required to cover in the same layer, which can be represented as in Equation 7.1.

$$\frac{m}{f} = 1 - \frac{n}{f}, \tag{7.1}$$

where m denotes the total number of *types*, n denotes the total number of features that are to be selected for each *type* and f denotes the total number of features of the target data stream.

Once the parameters of each layer are specified, n_l features (attributes) are randomly selected from the set of all attributes, where n_l is nth parameter for the layer number l (l=1, 2 or 3). This step is repeated m_l times for each layer, where m_l is mth parameter for the layer number l. As a result, m_l independent *types* are obtained for each layer at the end of this step.

When all layers are created using a random subspace of features in each layer, the ensemble is trained using RD. According to the original RD algorithm, the number of trees to be added to a subspace is specified dynamically as in Equation 7.2.

$$T_a(i) = \lfloor (a(t_i) - \frac{\sum_{i=1}^{m} a(t_i)}{m}) \times T_n(i) \rfloor, \tag{7.2}$$

$$T_r(i) = \lfloor (\frac{\sum_{i=1}^{m} a(t_i)}{m} - a(t_i)) \times T_n(i) \rfloor, \tag{7.3}$$

where $T_a(i)$ denotes the number of trees to add, $T_r(i)$ denotes the number of trees to remove, $a(t_i)$ denotes the accuracy of the subspace i being processed, m denotes the total number of *types* and $T_n(i)$ denotes the total number of trees currently present inside the subspace i. When the number of trees to be added to a subspace is more than one, different trees are created using the bootstrap aggregating (bagging) model. However, prequential evaluation is impossible in this case (i.e. when incoming data are used for testing and then training) since only one tree can be created at each time step. Hence, instead of creating different trees in a subspace at each time step, only one tree is created but with a higher weight assigned to it for the voting purpose.

For example, a tree with $weight = 2$ has two votes in the voting mechanism (i.e. having a higher impact) instead of only one. The removing mechanism in this strategy is based on the performance of the classifiers; e.g. when the number of trees to remove is 2, two least accurate trees (in the last data block) are eliminated from the ensemble.

Once the first data block is obtained by the ensemble, a classifier is built for every *type* (subspace) inside all layers. Given *max* as the maximum number of decision trees for each *type*, this step is repeated for the first $\frac{max}{2}$ blocks of data to shape the *types* and allow them to reach a specific maturity level. This phase is called the *initial training* and performed to build an average number of classifiers for every *type* in the ensemble. Note that for every data block received by the ensemble, all decision trees classify the instances, and the majority voting is applied to determine each layer's output. Then the ensemble's output is determined by combining the output of each layer using their weights obtained as in Equation 7.4.

$$W_i = W_{i-1} + \alpha(P_{i-1} - A_{i-1}), \tag{7.4}$$

where W_{i-1} denotes the weight of a layer at $(i-1)^{th}$ data block, P_{i-1} denotes the accuracy of the same layer over $(i-1)^{th}$ data block, A_{i-1} denotes the average accuracy of all layers over $(i-1)^{th}$ data block and α denotes the coefficient for recent data, which is an arbitrary parameter of the proposed algorithm ($\alpha > 1$). This phase is called the *voting* step.

Once the initial training phase is completed, each decision tree in a *type* is evaluated after classifying incoming instances by calculating its accuracy as in Equation 7.5.

$$a_i = \frac{c_i}{db}, \tag{7.5}$$

where c_i denotes the number of correctly classified instances in ith data block and db denotes the total number of instances in each data block. The accuracy of each *type* is the average accuracy of its constituent decision trees. The accuracy of the whole ensemble can be determined similarly to Equation 7.5. This phase is called the *evaluation* step.

Next, the *RD* step is applied similar to that in EACD (see Chapter 5), where each *type's* accuracy is taken into consideration and assessed in terms of the expected payoff calculated for each layer individually, as in the conditions in Equation 7.6.

$$\begin{cases} a(t_i) \geq \frac{\sum_{i=1}^{m} a(t_i)}{m} \Rightarrow grow \\ \\ a(t_i) < \frac{\sum_{i=1}^{m} a(t_i)}{m} \Rightarrow shrink, \end{cases} \tag{7.6}$$

where $a(t_i)$ denotes the accuracy of ith *type* and m denotes the total number of *types*.

The expected payoff in this study is set to the average accuracy of all *types* in each layer. Algorithm 11 shows how the *RD* step works. In this algorithm, t_j denotes the jth *type* of the ensemble ($1 \leq j \leq m$) and $a(t_j)$ denotes the accuracy of this *type*. The following functions are used in the presented algorithm:

- *Classify()*: classifying data using the majority voting;
- *Evaluate()*: evaluating the accuracy of classifiers/*types* in the ensemble using Equation 7.5;
- *Grow()*: adding a new classifier (decision tree) to the specified *type* (if Equation 7.6 stands);
- *Shrink()*: removing one, the worst performing classifier (decision tree) from the specified classification *type* (if Equation 7.6 stands); if this *type* has only one classifier, then do nothing;
- *Train()*: training all classifiers of the ensemble using the samples from a newly received data block.

To set a boundary for the number of classifiers (decision trees), an arbitrary upper bound of $max = 10$ was set in this study for all *types* in the ensemble. Hence, once the number of classifiers of a *type* is exceeded, the weakest performing decision tree of the same *type* is removed to make room for the newly built classifier. Furthermore, to prevent the *types* from complete removal, a lower bound of $min = 1$ was assigned to all *types*.

7.2.2 PSO Optimisation

The following non-canonical version of PSO is used in RED-PSO to optimise the combination of features of the classification *types* inside each layer. Initially, the new PSO algorithm takes all randomly drawn *types* as its input and

Algorithm 11: MODIFIED RD ALGORITHM

```
Input: Continuous block of data $DB = \{db_1, db_2, .., db_n\}$
$l$: layer number ($1 \leq l \leq 3$)
$n_l$: number of features to be selected in each type for the $l^{th}$ layer
$m_l$: total number of types for the $l^{th}$ layer
$max$: maximum number of classifiers in each type.
Output: Updated state of classification types
1  $l := 1$
2  $t := 1$
3  for $l := 1$ to $l := 3$ do
4      for $t := 1$ to $t := m_l$ do
5          Randomly select $n_l$ features
6          $t := t + 1$
7      $l := l + 1$
8  $i := 1$
9  while data stream is not empty do
10     if $i \leq \frac{max}{2}$ then
11         Classify($db_i$)
12         Grow(T) for all types
13     else
14         Classify($db_i$)
15         Evaluate()
16         if $a(t_j) \geq \frac{\sum_{j=1}^{n} a(t_j)}{m}$ then
17             Grow($t_j$) /* According to Equation 7.6 */
18         else
19             Shrink($t_j$) /* According to Equation 7.6 */
20     Train()
21     $i := i + 1$
```

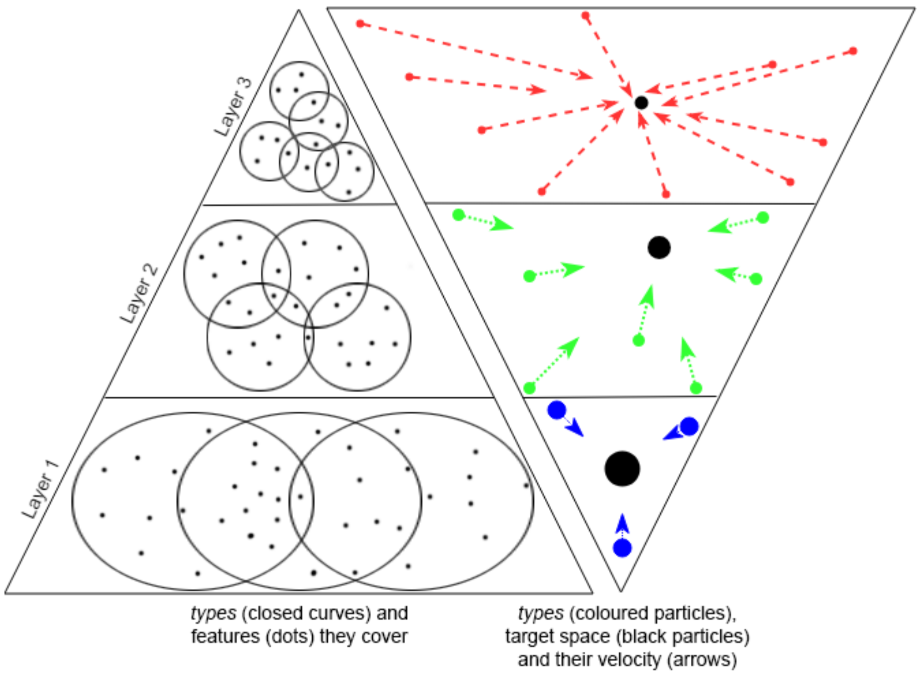

Layer 3
Layer 2
Layer 1
types (closed curves) and features (dots) they cover
types (coloured particles), target space (black particles) and their velocity (arrows)

FIGURE 7.1: Illustration of the two-layer architecture of RED-PSO. The left triangle represents the relationship between the number of classification *types* and the number of features they contain depending on the layer: the first layer has a smaller number of classification *types* with more features in each *type*, whereas the third layer has a larger number of classification *types* with less features in each *type*. The right triangle represents the different velocity at which each classification *type* moves towards global and local optimums in each layer: the lower the number of features a classification *type* has, the faster it moves towards optimums (i.e. has a higher velocity).

moves them towards the global best (*gbest*) and local best (*pbest*) solution *types* with a predefined velocity in each iteration (when a new data block is received). 'Moving a particle to a specific space with a velocity' means reforming the combination of features of a particle (classification *type*) to resemble a specific subspace of features with a defined proportion (velocity).

Let the *gbest* subspace include a set of features G, *pbest* include a set of features L and the current subspace include a set of features T. Then, the space towards which a *type* can move is in $S = [(G - T) \cap (L - T)]$. This set has features in the two 'best' subspaces that are not in the current subspace (i.e. S is the set of features the current particle can add to resemble (1) the best performing previous location of the same particle and (2) the currently best performing particle in the whole swarm). If the space is empty, the particle would move to the space $S = [(G - T) \cup (L - T)]$. To compensate for any newly added features from S, the algorithm removes a set $B = [T - (G \cap$

$L)]$ of features in T that are not among the good particles and may have contributed to a lower performance (i.e. B is a set of features that are not in the well-performing particles (*types*) and thus may be irrelevant to the current concept).

The velocity of each particle (i.e. how fast a particle moves towards local and global optima) is calculated as follows. If the difference (distance) between a particle (classification *type*) and its corresponding space S is greater than $x\%$, then the maximum velocity of the particle is set to $\beta \times 100\%$, where β $(0 < \beta \leq 1)$ is a constant value (coefficient) different for each layer and used to add more diversity to the layers (i.e. prevent classification *types* from being similar in terms of their features). If the distance between a particle (classification *type*) and its corresponding space S is less than $x\%$, then the particle is moved according to how far it is from the space S $(\beta \times \frac{d}{x})$, where d is the distance between each particle and the space S. If $|S| > |B|$, only B is used out of S just to maintain the same number of features in a *type* (i.e. when the number of features that can be added to S is greater than the number of features that can be removed from B; B is set to be the upper bound on the number of features to be added, maintaining the same number of features from one iteration to the next). In the experiments presented below, the values for the constant β were arbitrarily set to be 1, 0.7 and 0.4 for the layers 3, 2 and 1, respectively. Hence, the maximum velocity in layer 3 having the lowest number of features in each *type* was set to 100%. Similarly, the maximum velocities for layers 2 and 1 were set to 70% and 40%, respectively.

Algorithm 12 shows how the modified PSO stage works. In addition to the functions previously introduced in this section, the following functions are used in this algorithm:

- *Update()*: update the weight of each layer according to Equation 7.4;
- *Update-optimal()*: update the layer's global optimal *gbest* and each type's previous optimal (*pbest*) using the evaluated accuracy for each *type*;
- *Move()*: move each *type* inside the layer towards its *gbest* and *pbest* with a specified velocity based on its distance (in accuracy) of each *type* to the average accuracy of optima (as explained earlier in this section).

PSO is performed in each layer of RED-PSO individually and iterated in every data block received by the ensemble. As a result, particles (*types*) inside each layer of the ensemble move towards *gbest* and *pbest* of the same layer with the specified velocity calculated based on their performance over the last data block with known true labels. Note that each iteration in Algorithm 12 starts only after the same data block is processed by Algorithm 11.

Figure 7.1 illustrates the main concepts of RED-PSO. The left triangle in the figure shows *types* (closed curves) in each layer and features (dots) in each *type*. The right triangle demonstrates how *types* (particles) move towards a specific particle ($gbest \cap pbest$) with a specific velocity (arrows). The smaller the classification *types* are (i.e. having a small number of features), the faster they move towards optimums (i.e. having a high velocity).

In summary, RED-PSO uses an evolutionary method (RD) to seamlessly adapt to the concept drifts that are more smooth in nature (such as gradual and incremental concept drifts) by increasing the size of well-performing *types* and reducing the number of trees of poor-performing *types*. At the same time, it uses a modified version of PSO to optimise the combination of features in each layer individually, which is suitable for coping with more immediate concept drifts (such as abrupt and recurring concept drifts) that usually take longer to adapt to using the state-of-the-art methods. Finally, having three layers of different *types* built using different parameters, such as the number of *types* (m) and number of features (n) in each *type*, can help the algorithm to be less sensitive to drifting features by minimising their effects on each layer.

Algorithm 12: MODIFIED PSO OPTIMISATION

```
Input: Continuous block of data $DB = \{db_1, db_2, .., db_n\}$
Randomly drawn subspaces (types) for each layer, $T_l = \{t_1, t_2, .., t_m\}$
$W = \{w_1, w_2, w_3\}$ /* initial weight for each layer*/
Output: New set of types, $T_l\prime = \{t_1\prime, t_2\prime, .., t_m\prime\}$
1  $i := 1$ while data stream is not empty do
2      Classify($db_i$)
3      for $l := 1$ to $l := 3$ do
4          Evaluate($T_l$)
5          Update-optimal($gbest$)
6          for $j := 1$ to $j := m$ do
7              Evaluate($t_j$)
8              Update-optimal($pbest$)
9              Move($t_j$) /*move to $gbest/pbest$ according to its velocity */
10         Update($w_l$) /*update the weight of each layer*/
```

7.3 Experimental Study

This section provides the details of the experiments conducted to investigate the performance of the proposed method in this chapter. We introduce different variations of RED-PSO along with their parameters and computational complexity. Different variations are then compared with each other and the best performing variation is compared with the state-of-the-art algorithms in non-stationary data stream classification tasks. Eventually, the results are statistically analysed and discussed thoroughly. It is worth mentioning that the details necessary for conducting the experiments in this thesis (e.g. experimental settings, compared methods, evaluation runs, datasets, etc.) are discussed in Chapter 4.

In addition to the methods mentioned in Chapter 4 as the comparing methods and providing the fact that RED-PSO is designed to deal with all types of concept drifts, two more state-of-the-art methods, namely ADOB [40] and Learn++.NSE [60] are also included in the experimental study for this chapter.

7.3.1 RED-PSO Variations and Parameter Tuning

We compared the following five different variations of RED-PSO to evaluate the effect of its different characteristics and discuss the impact of employing different parameters.

RED1 uses the proposed three-layer architecture, along with RD to grow well-performing *types* and shrink poor-performing ones; it does not use PSO. The following parameters were employed for each layer in this variation: $Layer1 \Rightarrow m = 0.3 \times f$ and $n = 0.7 \times f$; $Layer2 \Rightarrow m = 0.5 \times f$ and $n = 0.5 \times f$; $Layer3 \Rightarrow m = 0.7 \times f$ and $n = 0.3 \times f$.

RED2 is the same as RED1 but with the following parameters: $Layer1 \Rightarrow m = 0.5 \times f$ and $n = 0.5 \times f$; $Layer2 \Rightarrow m = 0.3 \times f$ and $n = 0.7 \times f$; $Layer3 \Rightarrow m = 0.1 \times f$ and $n = 0.9 \times f$.

RED-PSO1 is the same as RED1 but also includes PSO and the following additional parameter for the threshold of the maximum velocity (x): $x = 2\%$.

RED-PSO2 is the same as RED2 but also includes PSO and the following additional parameter for the threshold of the maximum velocity (x): $x = 4\%$.

RED-PSO3 is the same as RED2 but also includes PSO and the following additional parameter for the threshold of the maximum velocity (x): $x = 2\%$.

Other general parameters for all variations of RED-PSO tested in the experiments are listed in Table 7.1, where D denotes the number of instances in

TABLE 7.1: General parameters used in the experiments for all variations of RED-PSO.

D	min	max	β_1	β_2	β_3	W_1	W_2	W_3
1000	1	10	1.0	0.7	0.4	1	2	4

each data block, min/max denotes the minimum/maximum number of classifiers (trees) in each classification *type*, β_i denotes the diversity coefficient used for ith layer and w_i denotes the initial weight assigned for ith layer.

7.3.2 Computational Complexity

Assuming the number of classes c, number of features in each classification *type* p, maximum number of values per feature v and maximum number of trees in the ensemble k, no more than p features are considered in a single Hoeffding tree [44]. Each feature at a node requires computing v values. Since calculating information gain requires c arithmetic operations, the cost of k Hoeffding trees at each time-step (iteration) in the worst case scenario is O($kcpv$). Given the number of all classification *types* in the ensemble m and the fact that RD uses m arithmetic operations to calculate payoffs, the cost of applying RD to the ensemble is only O(m). Hence, the computational complexity of RED1 and RED2 variations (deploying Hoeffding trees along with RD) is O($m + (kcvp)$). Furthermore, assuming that the number of classification *types* in each layer of the ensemble is m_i ($1 \leq i \leq 3$) and the number of iterations for each data block is 1, the PSO optimisation procedure uses $m_1 + m_2 + m_3$ arithmetic operations in each time-step. Therefore, the computational complexity of RED-PSO variations (RED-PSO1, RED-PSO2 and RED-PSO3) of the proposed framework is O($m + (kcvp) + m_1 + m_2 + m_3$).

In summary, the procedure of reading, processing and classifying data in RED-PSO is as follows. Once a new block of data has been filled up to its full capacity ($d = 1000$ instances in this case), the system starts reading and classifying the instances in a synchronous manner using all current classifiers in the ensemble. Since each processing layer of RED-PSO has a specific weight assigned to its classifiers, the majority voting takes place to determine the predicted class of each instance. Once the real labels of all instances in a data block are received, the classifiers inside each layer start moving based on the local and global optimum values according to the optimisation procedure described above. Finally, in the evaluation step, the weights of all layers are updated based on their performance over the last data block.

7.3.3 Results and Discussion

All algorithms are compared using standard criteria, namely, the prequential accuracy in the immediate and delayed settings, Kappa M (comparison with a majority-class classifier) and overall execution time (including the time of making predictions, training and evaluating the ensemble). The first set of experiments is related to comparing different RED-PSO variations to understand the effects of using different mechanisms and parameters such as the modified versions of RD and PSO.

Comparison of the Different RED-PSO Variations

Figures 7.2 and 7.3 (Tables C.1 and C.2 in Appendix C) illustrate the average accuracy of all RED-PSO variations over the nine datasets in the immediate and delayed settings. It is clear that the PSO-optimised variations (RED-PSO1, RED-PSO2 and RED-PSO3) perform better than the RD-only variations (RED1 and RED2). This is because the former variations include an additional optimisation technique to move all *types* in each layer towards the best possible spaces (*gbest* and *pbest*) in the same layer. This enables the system to optimise the combination of features in every *type* according to the time and most recent data received by the system, especially upon concept drifts. RED-PSO3 has the best average accuracy over the Hyperplane, RTG, SEA, KDDcup, Airlines and Forest Cover-type datasets, and the best overall average accuracy in both the immediate and delayed settings.

Figure 7.4 illustrates the average accuracy (throughout all datasets) of all RED-PSO variations in both the immediate and delayed settings. It is clear that the RED-PSO3 variation has the best average accuracy in both the immediate and delayed settings.

Figure 7.5 (Table C.3 in Appendix C) shows the execution time of different RED-PSO variations in the immediate setting (the variability of the execution times between the immediate and delayed settings is negligible). It is clear that the variations without PSO optimisation are faster than the ones with it due to the time complexity of PSO being added to the ensemble. The RED2 variation has the shortest execution time over seven out of nine datasets compared to the other variations used in the experiments, while RED-PSO2 and RED-PSO3 have the longest execution time for the majority of the datasets.

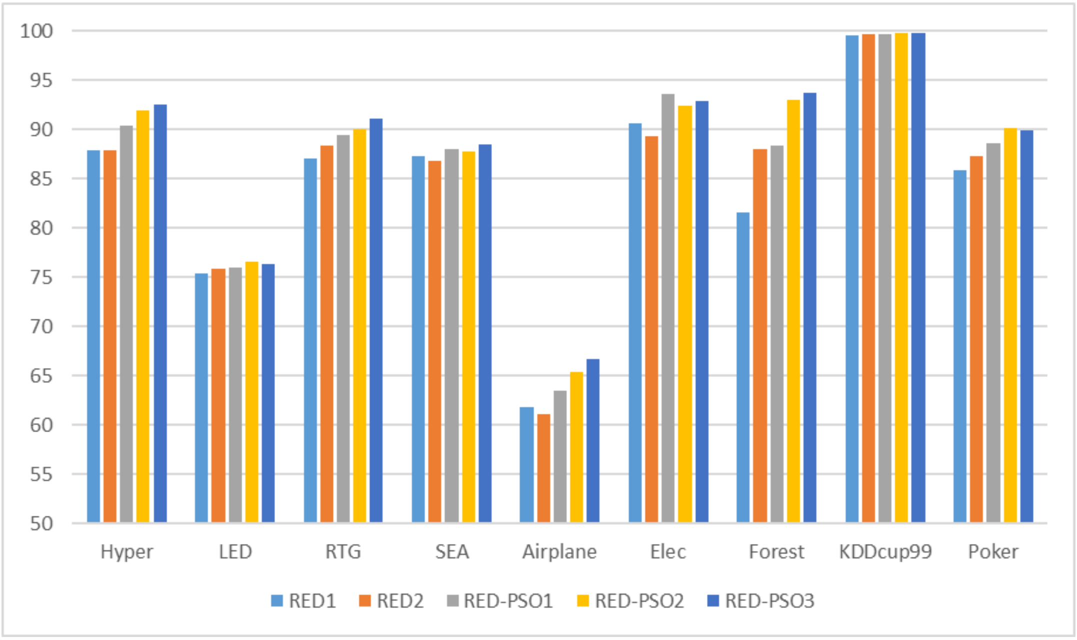


FIGURE 7.2: Average accuracy of the different RED-PSO variations over the considered nine datasets in the immediate setting.

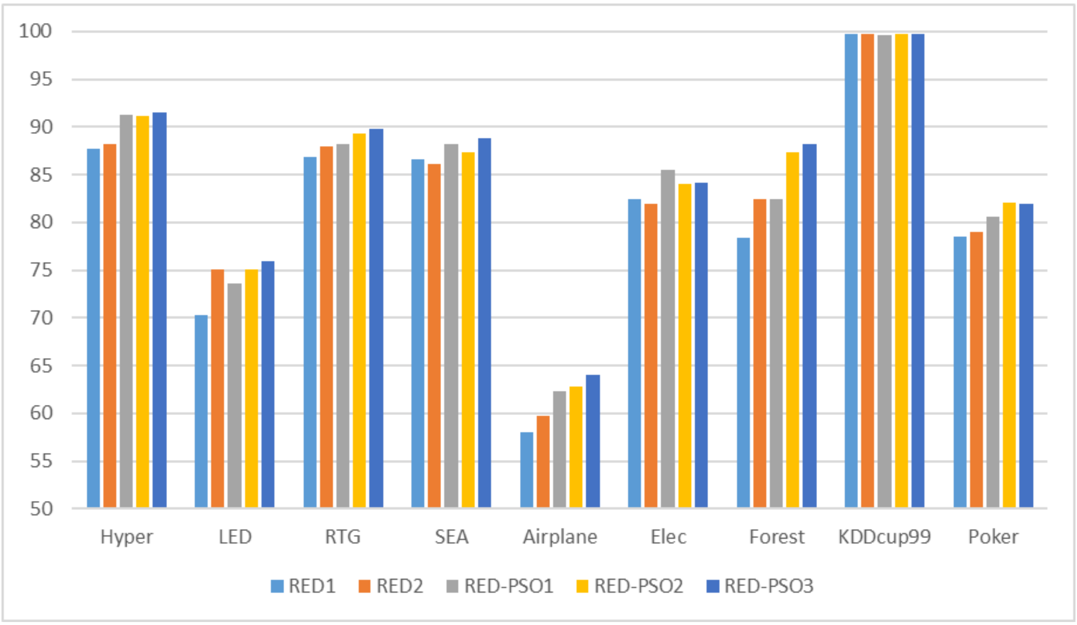


FIGURE 7.3: Average accuracy of the different RED-PSO variations over the considered nine datasets in the delayed setting.

Comparison With Other Methods

The second set of experiments involves comparing seven different state-of-the-art methods to the best performing RED-PSO variation, namely, RED-PSO3. Tables 7.2 and 7.3 list the average accuracy scores and their standard deviation, minimum and maximum values of RED-PSO3 compared to that

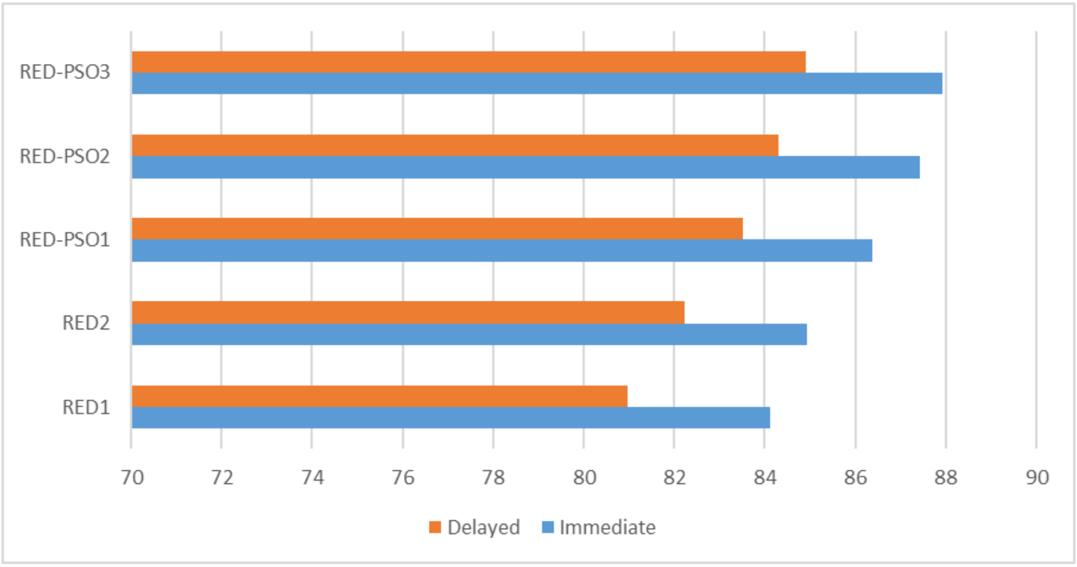


FIGURE 7.4: Average accuracy (%) of the RED-PSO variations in the immediate and delayed settings.

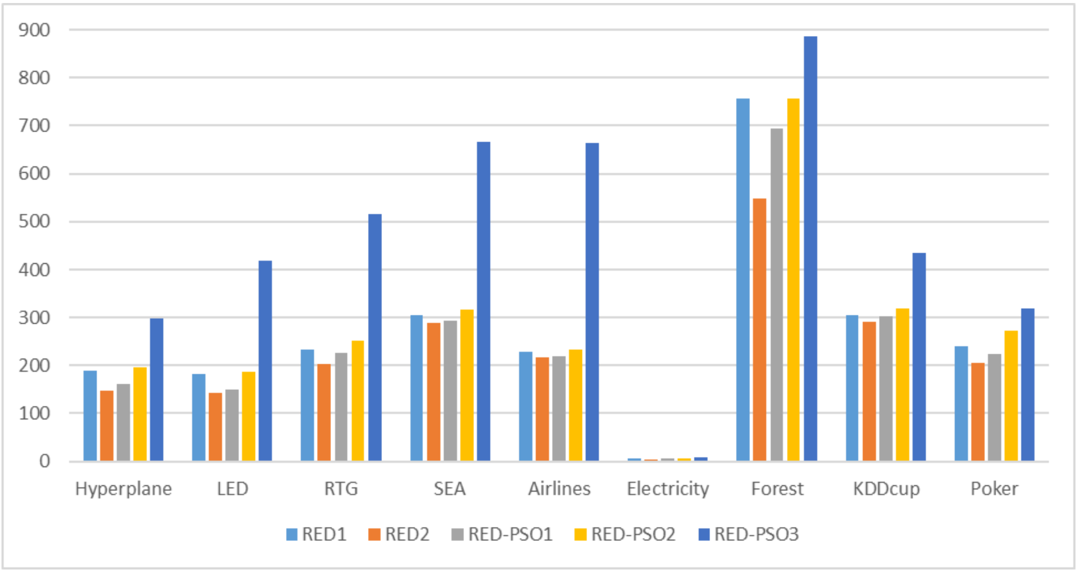


FIGURE 7.5: Average time (in seconds) of executing the RED-PSO variations over the considered nine datasets in the immediate setting.

of the considered state-of-the-art methods over all datasets. Note that the minimum, maximum, average and standard deviation of the accuracies are based on the 10 different evaluation runs (each evaluation run has its specific accuracy) that are conducted for each method. More information on different evaluation runs for each data stream is provided in Section 4.2.

It can be noticed that RED-PSO3 has the best average accuracy over the Hyperplane, LED, Airlines, Electricity and Poker-hand datasets in both the

TABLE 7.2: Average accuracy (%) scores and their standard deviation, minimum and maximum values of RED-PSO3 and the compared methods in the immediate setting. Bold values indicate the best performance for each dataset.

Dataset	Criteria	ARF	DWM	LevBag	OAUE	OSBoost	Learn++	ADOB	RED-PSO3
Hyper.	Ave.(%)	88.17	89.64	91.03	91.42	85.85	87.2	49.54	**92.54**
	σ(%)	1.90	0.83	1.60	1.46	3.01	2.81	7.39	2.56
	Min(%)	85.96	88.45	88.92	89.66	81.80	86.51	53.37	89.60
	Max(%)	91.31	90.94	93.54	93.63	89.87	89.72	47.32	95.01
LED	Ave.(%)	74.05	75.05	74.22	73.99	74.15	68.28	11.70	**76.29**
	σ(%)	0.31	3.10	0.31	0.10	0.11	1.05	3.53	3.09
	Min(%)	73.58	73.86	73.93	73.89	74.05	67.58	9.28	74.01
	Max(%)	74.45	83.83	74.52	74.09	74.26	69.05	13.55	79.87
RTG	Ave.(%)	78.35	59.35	90.78	88.88	**93.40**	56.68	92.40	91.09
	σ(%)	8.12	8.87	2.26	3.26	1.45	8.26	1.84	1.59
	Min(%)	65.86	48.26	87.38	83.39	90.63	52.46	90.77	89.92
	Max(%)	88.03	73.86	93.72	92.56	95.24	60.05	93.61	93.51
SEA	Ave.(%)	88.67	87.72	87.59	88.69	85.56	74.90	71.75	**89.50**
	σ(%)	0.58	0.57	1.67	0.58	0.35	1.84	3.18	0.80
	Min(%)	88.40	87.18	85.41	88.13	85.25	73.00	68.91	88.95
	Max(%)	89.55	88.26	89.28	89.25	85.86	76.73	73.65	90.45
Airlines	Ave.(%)	63.53	63.97	59.42	64.02	61.98	60.50	46.98	**66.98**
	σ(%)	1.23	0	0.73	0	0	0	0	2.15
	Min(%)	62.08	63.97	58.45	64.02	61.98	60.50	46.98	64.11
	Max(%)	65.46	63.97	60.62	64.02	61.98	60.50	46.98	68.86
Electricity	Ave.(%)	**92.17**	75.73	92.09	91.60	88.02	71.53	72.70	**92.96**
	σ(%)	0.94	0	1.48	0	0	0	0	0.86
	Min(%)	90.45	75.73	89.56	91.60	88.02	71.53	72.70	92.03
	Max(%)	93.19	75.73	93.70	91.60	88.02	71.53	72.70	93.61
Forest	Ave.(%)	**93.57**	83.75	92.73	90.70	84.45	89.69	21.59	91.71
	σ(%)	1.58	0	2.10	0	0	0	0	2.59
	Min(%)	91.11	83.75	89.45	90.70	84.45	89.69	21.59	89.67
	Max(%)	95.09	83.75	95.40	90.70	84.45	89.69	21.59	93.89
KDDcup	Ave.(%)	99.81	99.04	99.82	99.80	99.74	27.55	**99.88**	99.80
	σ(%)	0.06	0	0.01	0	0	0	0	0.02
	Min(%)	99.74	99.04	99.80	99.80	99.74	27.55	9.88	99.78
	Max(%)	99.91	99.04	99.83	99.80	99.74	27.55	9.88	99.83
Poker	Ave.(%)	84.19	74.37	88.52	80.74	84.31	63.41	69.64	**89.89**
	σ(%)	4.55	0	3.34	0	0	0	0	3.94
	Min(%)	80.08	74.37	84.67	80.74	84.31	63.41	69.64	85.91
	Max(%)	90.06	74.37	93.56	80.74	84.31	63.41	69.64	92.03

immediate and delayed settings and the best accuracy over SEA in the immediate setting.

Figure 7.6 illustrates the average accuracy (across all datasets) of RED-PSO3 compared to other state-of-the-art methods in both the immediate and delayed settings. It is evident that RED-PSO3 has the best average accuracy in both settings. Therefore, it can be concluded that RED-PSO3 has the most consistent performance compared to that of the other methods for different data streams. This is opposed to the ADOB algorithm that has a drastically weak performance over the LED and Forest Cover-type datasets, whereas it has the best performance over the KDDcup99 dataset. One possible explanation for the more stable performance of the proposed method is that in RED-PSO3, different strategies are adopted for different environments and concept drifts using the three layers and their dynamic weights during PSO

TABLE 7.3: Average accuracy (%) scores and their standard deviation, minimum and maximum values of RED-PSO3 and the compared methods in the delayed setting. Bold values indicate the best performance for each dataset.

Dataset	Criteria	ARF	DWM	LevBag	OAUE	OSBoost	Learn++	ADOB	RED-PSO3
Hyper.	Ave.(%)	88.05	89.41	90.77	91.10	85.74	86.84	49.54	**91.48**
	σ(%)	2.02	0.95	1.71	1.59	3.06	2.76	8.06	2.93
	Min(%)	85.56	88.25	88.60	89.21	81.70	83.99	42.80	88.90
	Max(%)	91.35	90.86	93.37	93.55	89.78	89.45	54.29	94.39
LED	Ave.(%)	74.00	74.14	74.21	74.06	74.13	67.80	10.10	**75.91**
	σ(%)	0.40	0.16	0.15	14.00	0.04	1.51	0.89	2.74
	Min(%)	73.42	73.99	74.07	73.74	74.02	65.00	9.03	72.27
	Max(%)	74.49	74.30	74.36	74.19	74.17	69.59	12.61	77.32
RTG	Ave.(%)	78.24	59.49	**90.91**	88.72	85.53	56.28	87.80	89.81
	σ(%)	8.06	8.64	2.48	5.13	2.90	6.90	2.32	2.40
	Min(%)	65.81	47.16	86.94	82.17	81.17	49.61	84.53	86.29
	Max(%)	87.92	73.73	93.49	92.47	88.70	59.98	90.19	92.01
SEA	Ave.(%)	**88.94**	87.48	88.70	88.54	85.31	74.75	70.75	88.80
	σ(%)	0.59	1.02	1.45	0.70	0.42	1.91	2.12	1.61
	Min(%)	88.28	86.01	86.89	87.81	84.92	71.03	67.55	86.24
	Max(%)	89.51	88.21	90.32	89.21	85.91	76.86	74.43	90.01
Airlines	Ave.(%)	61.42	60.57	58.49	62.73	61.80	50.58	46.98	**64.54**
	σ(%)	1.12	0	0.89	0	0	0	0	2.51
	Min(%)	61.22	60.57	57.03	62.73	61.80	50.58	46.98	60.96
	Max(%)	63.32	60.57	59.65	62.73	61.80	50.58	46.98	67.91
Electricity	Ave.(%)	**83.51**	67.43	81.78	80.20	79.04	56.60	65.81	**84.18**
	σ(%)	1.19	0	0.88	0	0	0	0	1.11
	Min(%)	81.78	67.43	80.54	80.20	79.04	56.60	65.81	81.79
	Max(%)	84.80	67.43	83.00	8020	79.04	56.60	65.81	86.57
Forest	Ave.(%)	85.65	74.93	86.22	86.84	74.47	**87.42**	16.59	86.19
	σ(%)	2.60	0	2.72	0	0	0	0	1.99
	Min(%)	83.67	74.93	83.02	86.84	74.47	87.42	16.59	83.70
	Max(%)	90.49	74.93	89.50	86.84	74.47	87.42	16.59	88.37
KDDcup	Ave.(%)	99.80	99.12	99.81	99.78	99.74	31.68	**99.88**	99.77
	σ(%)	0.07	0	0.01	0	0	0	0	0.02
	Min(%)	99.72	99.12	99.79	99.78	99.74	31.68	99.88	99.74
	Max(%)	99.90	99.12	99.83	99.78	99.74	31.68	99.88	99.81
Poker	Ave.(%)	67.95	59.31	76.78	73.81	81.23	63.87	55.11	**81.89**
	σ(%)	2.92	0	3.72	0	0	0	0	2.16
	Min(%)	64.94	59.31	70.51	73.81	81.23	63.87	55.11	79.15
	Max(%)	73.29	59.31	79.34	73.81	81.23	63.87	55.11	83.03

optimisation.

The next step in our comparison is the evaluation of the classification performance according to $Kappa_M$ statistic, which is a robust inter-rater agreement for qualitative items. Table 7.4 lists the average $Kappa_M$ values for the RED-PSO3 and other state-of-the-art methods in the delayed setting. The proposed RED-PSO3 variation has the best performance over the Hyperplane, SEA, Airline and Poker-hand datasets and the highest average $Kappa_M$ value. Taken together, these results demonstrate that the majority of the agreements in the ensemble did not occur by chance.

Figure 7.8 (Table C.4 in Appendix C) shows the execution time of RED-PSO compared to that of the other state-of-the-art methods in the immediate setting. Since the execution time values in the delayed setting are similar to those in the immediate setting, respectively, we only show the results for

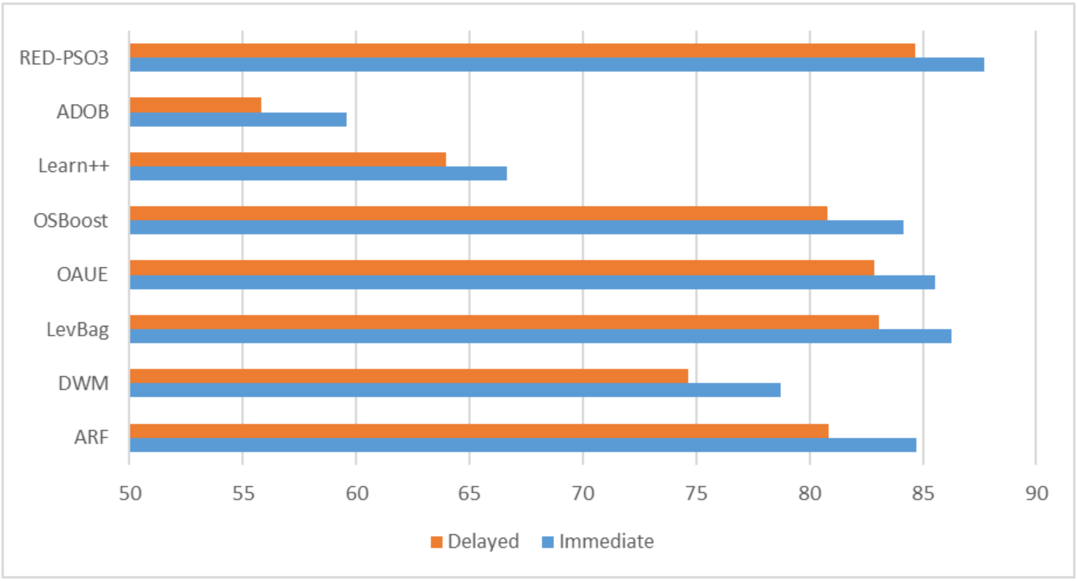


FIGURE 7.6: Average accuracy of the RED-PSO3 and other state-of-the-art methods over the considered nine datasets in the immediate (blue bars) and delayed (orange bars) settings.

the immediate setting. According to the experimental results, DWM and OSBoost algorithms have the shortest execution time by far for the majority of the datasets (DWM over the SEA, Airlines, Electricity and Poker-Hand datasets, and OSBoost over the Hyperplane and LED datasets), while the ADOB method has the longest execution time over four out of nine datasets (RTG, Airlines, Electricity, Forest Cover-type and Poker-Hand). According to the overall execution times of different methods, ADOB and Learn++ are the most time consuming methods by far comparing to the other six methods (by more than 10,000 s). Apart from the high execution times of Learn++ and ADOB, the proposed method (RED-PSO3) has the longest overall execution time compared to that of the other methods (by about 1000 s compared to the ARF method which is the next slowest method). It seems possible that this is due to having three different layers of optimisation running in parallel in RED-PSO. It is worth noting that given the design of our proposed classification system, the three optimisation layers can greatly benefit from parallel processing, as they operate independently when optimising the classification *types*. This can potentially provide a multi-fold speed-up of the system.

Performance Over Different Types of Concept Drifts

We added different types of concept drifts to the datasets generated by the SEA data stream generator and compared the prequential performance of the considered algorithms in each case. Details on how different concept drifts

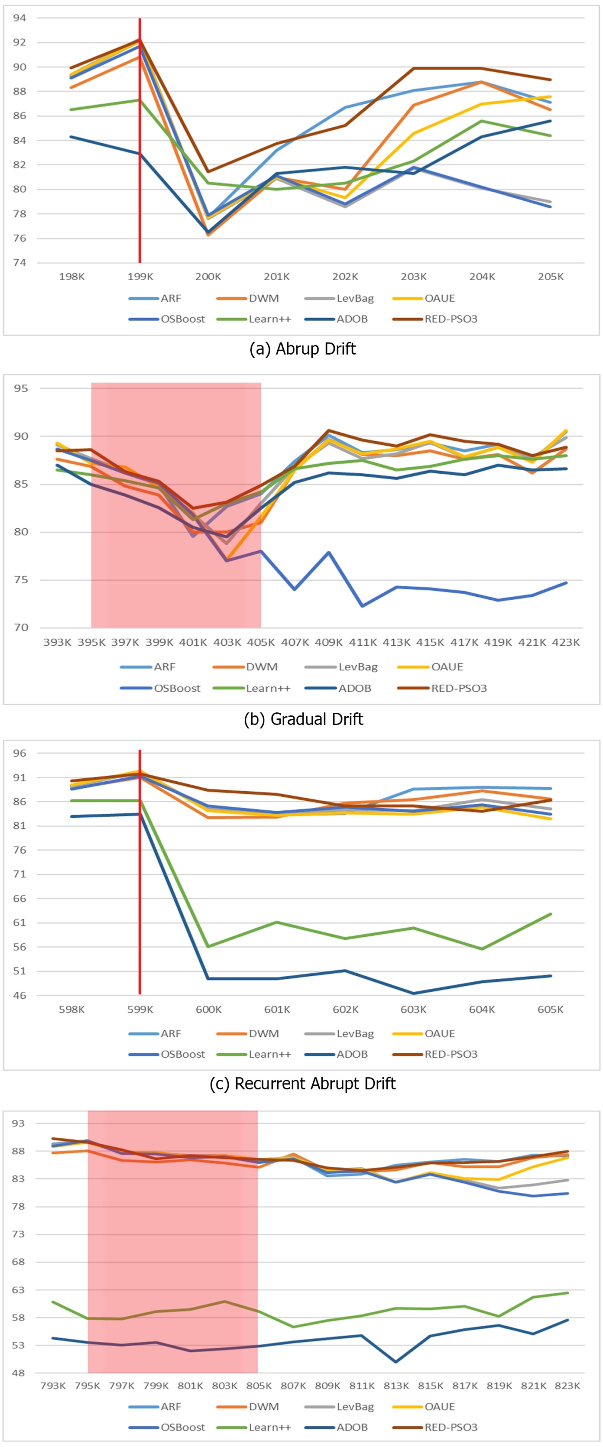


FIGURE 7.7: Classification accuracy of the RED-PSO3 and other state-of-the-art methods over the SEA dataset upon different types of concept drifts in the delayed setting. The red boxes indicate the length and location of the added concept drifts.

TABLE 7.4: $Kappa_M$ statistic of the RED-PSO3 and other state-of-the-art methods in the delayed setting. Bold values indicate the best performance for each dataset.

Dataset	ARF	DWM	Lev-Bag	OAUE	OS-Boost	Learn-++	ADOB	RED-PSO3
Hyper.	75.45	78.24	81.03	81.7	70.69	72.63	-4.87	**81.19**
LED	**71.03**	70.85	70.86	70.76	70.84	64.71	-2.06	70.99
RTG	66.89	35.84	85.10	81.98	76.87	40.52	**82.89**	80.98
SEA	72.01	68.57	70.81	71.95	60.55	43.78	36.61	**73.11**
Airlines	13.14	9.06	4.4	13.96	11.91	-13.60	-24.27	**15.80**
Elec.	**67.44**	30.19	60.38	57.60	55.03	7.06	26.77	65.21
Forest	46.92	0.58	35.68	**65.51**	6.72	32.25	-341.3	32.49
KDDcup	99.45	97.99	99.59	99.50	99.41	-57.12	**99.72**	99.43
Poker	24.63	-7.06	51.15	36.47	54.38	5.99	-11.77	**57.3**

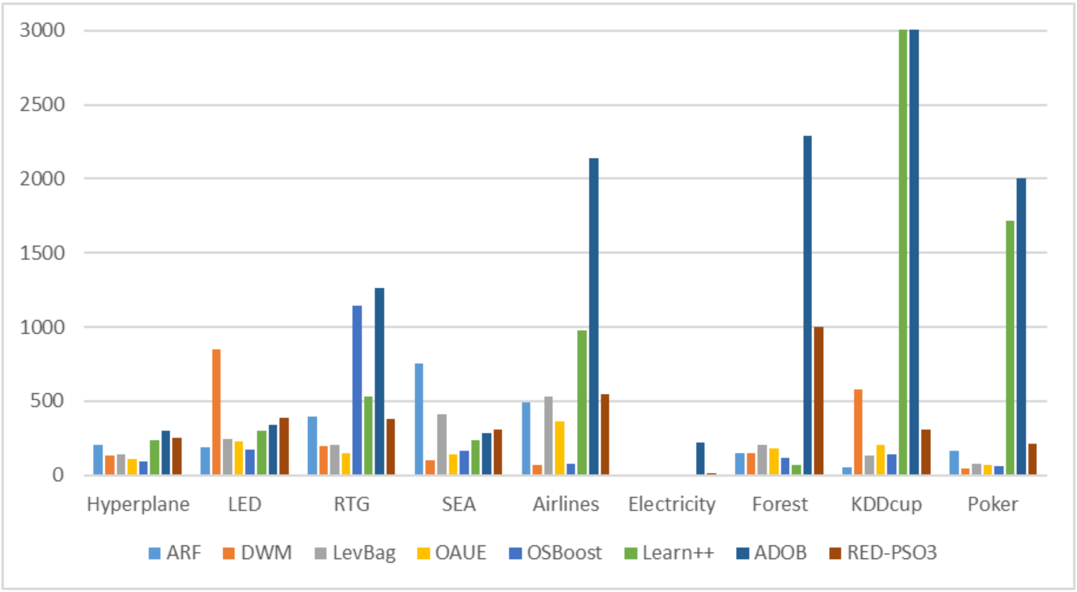


FIGURE 7.8: Average time (in seconds) of executing the RED-PSO3 and other state-of-the-art methods over the considered nine datasets in the immediate setting.

were added to the data streams are discussed in Chapter 4.2. The reason for choosing a synthetic dataset for these experiments is that the exact time and actual type of concept drifts present in real-world data streams often remain unknown.

Figure 7.7(a) illustrates how different methods adapt to an abrupt drift with a width of 1 at an instance number of 200K. It is evident that RED-PSO3 has the highest accuracy value right after the concept drift happens. Furthermore, RED-PSO3 recovers from the introduced concept drift faster than the other methods, whereas LevBag and OSBoost fail to recover from the concept drift in a timely manner. Figure 7.7(b) illustrates the case when a gradual drift was added with a width of 10K. It can be noticed from the figure that all methods except OSBoost fully recovered right after the drift is over. This result suggests that in the case of gradual concept drifts, OSBoost may take a long time to have a full recovery. Figure 7.7(c) illustrates the behaviour of the methods upon a recurrent drift with a width of 1. Again, RED-PSO3 has the lowest accuracy drop compared to the other methods. At the same time, it struggles to fully recover from the drift initially. In this case, the ARF method adapts to the introduced concept drift faster than the other methods. Furthermore, ADOB and Learn++ demonstrate a drastic decrease in accuracy and fail to recover from the drift in a timely manner. Finally, Figure 7.7(d) illustrates the case when a recurrent drift was added with a width of 10K at an instance number of 795K. It is evident that the accuracy of all algorithms did not change drastically upon the drift. It is worth noting that both ADOB and Learn++ still struggled to adapt to the previous concept drift introduced at an instance number of 600K. A possible explanation may be that their concept drift detectors fail to detect the drift.

In summary, according to the experimental results, the main advantage of RED-PSO3 is its **accuracy** and **robust performance**. In particular, the proposed method demonstrated the **best average rank and consistent performance** compared to the other state-of-the-art methods in both the immediate and delayed settings and upon introducing different types of concept drifts. The main drawback of RED-PSO3 is its execution time. While the overall execution time of the proposed method is not the longest among all considered state-of-the-art methods, it is relatively long, especially over the datasets with a high number of features such as Forest Cover-type. This is due to the fact that the number of classifiers in the ensemble increases with the number of features in the target data stream. Furthermore, the adoption of an evolutionary algorithms (RD) along with a bio-inspired optimisation algorithm (PSO) in RED-PSO leads to a high computational complexity of the system. To overcome these limitations, other variations of RED-PSO such as RED1 and RED2 can be used in case of high-dimensional and time-restricted applications. Furthermore, parallel processing can be applied to speedup the execution time of RED-PSO.

TABLE 7.5: Average rank of the RED-PSO3 and other state-of-the-art methods according to the Friedman test.

Dataset	ARF	DWM	Lev-Bag	OAUE	OS-Boost	Learn-++	ADOB	RED-PSO3
R_j	3.833	5.333	3.167	3.528	4.889	6.722	6.500	**2.028**
R_j^2	14.694	28.444	10.028	12.445	23.901	45.188	42.250	4.111

7.3.4 Statistical Analysis

The Friedman test [87] is a popular non-parametric statistical test that can be used to detect differences across several algorithms in multiple test attempts (e.g. datasets).

Table 7.5 shows the average rank of each algorithm considered in our experiments and their squared values with $k = 8$ and $N = 18$ since the total number of methods is eight and the total number of datasets in both the immediate and delayed settings is 18 $(9 + 9)$. Providing that the value of the Friedman test statistic is $\chi_F^2 = 57.18$ with 7 $(k - 1)$ degrees of freedom, and the critical value for the Friedman test given $k = 8$ and $N = 18$ is 18.48 at a significance level of $\alpha = 0.01$, we can conclude that the accuracy values of the studied methods are significantly different (57.18 is greater than 14.63).

Now that the Null-hypothesis is rejected, we can proceed with a post-hoc test. The Nemenyi test [89] can be used when all classifiers are compared to each other [88].

The critical value in our experiments with $k = 8$ and $\alpha = 0.10$ is $CD_{0.10} = 1.805$. As a result, the accuracy of the proposed RED-PSO3 method is significantly different from that of the DWM, OSBoost, Learn++ and ADOB methods, while it is not significantly different from that of the LevBag, OAUE and ARF methods. Figure 7.9 illustrates the statistical comparison of the methods considered in the experiments based on the Nemenyi test.

7.4 Summary

In this chapter, we proposed a novel ensemble learning method called RED-PSO to seamlessly adapt to different concept drifts when performing classification tasks in non-stationary data streams. RED-PSO is based on a three-layer architecture to produce classification *types* of different size created by randomly selecting features from a pool of features of the target data stream. RD is used in RED-PSO to seamlessly adapt to different concept drifts, while

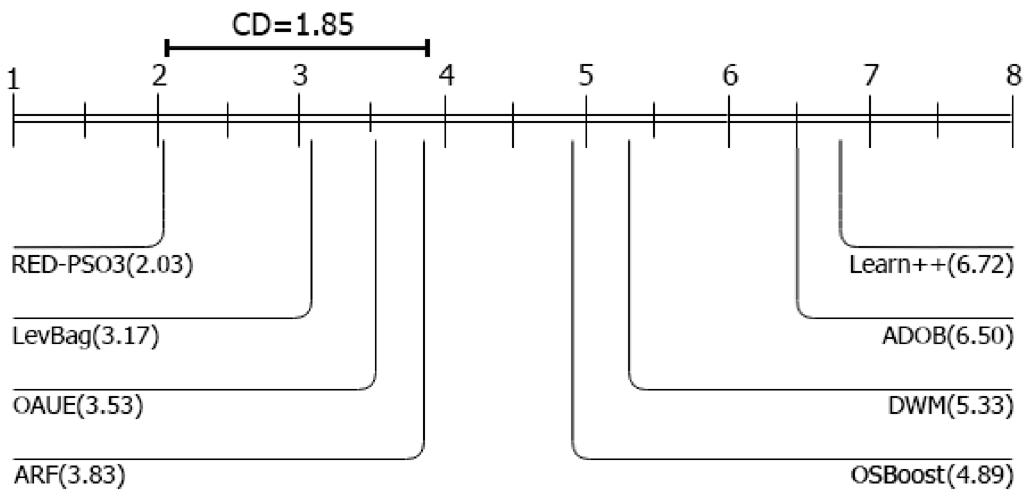


FIGURE 7.9: Comparison of the RED-PSO3 and other state-of-the-art methods using the Nemenyi test at $\alpha = 0.10$.

a modified version of PSO is applied to optimise the combination of features in each classification *type* in all layers.

A set of experiments was conducted to compare the performance of the different RED-PSO variations and that of the best-performing variation (RED-PSO3) to some state-of-the-art algorithms over five real-world and four synthetic data streams using the immediate and delayed prequential evaluation methods. According to the experimental results, RED-PSO3 has the lowest rank and highest average accuracy compared to those of the other RED-PSO variations and considered state-of-the-art methods. Using the Friedman statistical test, it was shown that the accuracy values of the studied methods were significantly different. Furthermore, according to the Nemenyi test, the accuracy of RED-PSO3 was significantly different from four out of seven compared methods (DWM, Learn++, ADOB and OSBoost), while it was not significantly different from the other three methods (LevBag, OAUE and ARF). The main drawback of RED-PSO is its long overall execution time in the case when the target data stream has a high number of features, which can be addressed in the future by parallelising the optimisation of the three layers providing that they operate independently.

By presenting RED-PSO in this chapter, we address the final objective of this thesis (see Section 1.4). Providing that all four objectives are addressed in Chapters 2, 5, 6 and 7, respectively, the next chapter summarises the work presented in this thesis and draws conclusions. It contains a brief description of the novel methods proposed in this thesis and points out several important directions for future research.

Chapter 8

Conclusion and Perspectives

This thesis studies the impact of applying metaheuristic methods to address the main challenges in non-stationary data stream classification tasks. The aims and objectives identified in the thesis are mainly concerned with studying the state-of-the-art methods and proposing novel metaheuristic methods to improve the performance of the existing approaches, offer efficient ways of deploying metaheuristic algorithms and optimise the adaptation process by balancing out the exploration and exploitation of metaheuristics.

Chapter 1 presents an introduction to data stream mining tasks and the main challenges in this area of research. This is followed by stating the identified research gaps, as well as the aims and objectives recognised for this thesis. Chapter 2 addresses the first objective outlined in Section 1.4. In particular, the chapter discusses the existing ensemble-based methods for non-stationary data stream classification tasks and presents an analytical study on ensemble dynamics of these classification methods. After a thorough analysis of the existing methods, it has become clear that there is lack of a comprehensive approach that is (1) able to cope with different types of concept drifts; (2) resistant to noise and false alarms; and (3) has a fast adaptation mechanism (reaction) to all types of concept drifts. Hence, this thesis focuses on overcoming these limitations.

Chapter 3 reviews the necessary background and theoretical explanation of the algorithms employed in the novel methods proposed in this thesis. Chapter 4 discusses the necessary details for conducting the experiments in this thesis. In particular, the experimental settings are described and different methods and datasets involved in the experiments are introduced along with a description of how each evaluation run is conducted.

The second objective of this thesis is addressed in Chapter 5 by proposing the EACD method using a modified version of two evolutionary algorithms, namely, RD and GA. This approach is extended in Chapter 6 resulting in a novel algorithm called RED-GENE to achieve the third objective outlined in this thesis. The final objective of this thesis is addressed in Chapter 7 by proposing a three-layer architecture and employing modified versions of two metaheuristic algorithms, namely, RD and PSO. A reflection on how the objectives of this work have been achieved is provided in the following section.

Overall, we conclude the thesis by presenting (1) a summary of the research carried out in this thesis; (2) a comparison of the novel algorithms proposed to achieve the objectives of the thesis; and (3) a discussion of some future directions and perspectives of the contributions of this thesis.

8.1 Summary

Data stream mining has been a growing area of research in recent years due to the advancements of modern digital technologies such as the IoT, social media networks, fraud detection and predictive systems and World Wide Web. The main challenge in this area of research is the ability to cope with the changing behaviour of data in real-world applications. This changing behaviour is caused by the presence of concept drifts, which can be defined as unforeseen changes in the data distribution of a data stream. Ensemble learning techniques using a pool of classifiers have proved superiority over single classifier techniques when addressing this challenge [13].

Despite the current progress in solving non-stationary data stream classification problems, to the best of our knowledge, there is a lack of a ubiquitous classification algorithm meeting the following requirements:

- performing effectively over various data streams with different number of features;
- seamlessly adapting to different types of concept drifts such as gradual, abrupt, incremental and recurrent in a timely manner;
- combining exploitation and exploration approaches when dealing with the dynamic nature of data streams;
- offering different variations of the same processing method in the presence of certain limitations such as time or memory constraints.

To understand different mechanisms developed so far for adapting to concepts drifts and identify a research gap within the area of stream mining, the state-of-the-art ensemble learning techniques were first comprehensively studied to achieve the first objective of this thesis: **"study the state-of-the-art algorithms for non-stationary data stream classification tasks and explore different mechanisms to cope with current challenges in such tasks"**. This study then inspired the core idea of this thesis to use metaheuristic methods for analysing non-stationary data streams. In particular, we believe that metaheuristic methods can offer a natural solution to the changing behaviour of non-stationary data streams due to the known characteristics of these methods such as self-replication, self-tolerance, reproduction, evolution, adaptation, learning and growth [18]. However, in their canonical form, metaheuristic methods can only work with stationary datasets collected in advance. Hence, their adaptation to processing evolving data streams arriving continuously in real-time is required.

Consequently, in Chapter 5, we proposed our first novel method called EACD to address the second objective of this thesis: **"use evolutionary algorithms to propose a novel ensemble learning technique for concept drift adaptation in data streams"**. EACD has a two-layer architecture and employs ensemble learning to train an ensemble of different classification *types* consisting of randomly drawn features (subspaces) of the target data stream. These randomly drawn subspaces are grown and shrunk in the first layer of EACD using a modification of the RD algorithm. Furthermore, the combination of features selected for each classification *type* is optimised in the second layer of EACD using a modification of GA to cope with different concept drifts over time. This solution allows the ensemble to handle different types of concept drifts: RD helps dealing with more gradual (hard to detect) concept drifts, while GA helps dealing with more abrupt (easy to detect) concept drifts.

Next, RED-GENE was presented in Chapter 6 to address the third objective of this thesis, namely, **"propose efficient ways of deploying evolutionary algorithms for performing non-stationary data stream classification tasks"**. RED-GENE was built based on the EACD method. Both methods employ the same approach to creating different classification *types* and a GA optimisation technique. However, in contrast to RED-GENE, only the most basic modified version of RD proposed for processing streaming data was used in EACD. RED-GENE improves the EACD method by offering the following further contributions: (1) proposing three different modified versions of RD to accelerate the concept drift adaptation process; (2) improving the classification accuracy for the majority of the considered experimental cases; and (3) reducing the running time of the algorithm by generating a lower number of types while improving the total accuracy.

Finally, the last objective of this thesis, that is, **"optimise the metaheuristic solutions for non-stationary data stream classification by balancing out the exploration and exploitation of metaheuristics"** was achieved in Chapter 7 by proposing a novel RED-PSO method. RED-PSO has a three-layer architecture; each layer is initially assigned some predefined classification *types* randomly created from a pool of features of the target data stream. Similar to EACD and RED-GENE, RED-PSO employs RD to seamlessly adapt to smooth (i.e. gradual or incremental) concept drifts. However, unlike the first two proposed methods, RED-PSO further applies a modified version of PSO to each of its layers individually to optimise the combination of features in all classification *types*. In this way, both exploration and exploitation are practised in the search space by moving particles (classification *types*) towards local and global optima.

8.2 Comparison among the Proposed Methods

The proposed EACD, RED-GENE and RED-PSO methods are individually compared to the state-of-the-art methods for non-stationary data stream classification in Chapters 5, 6 and 7, respectively. It was demonstrated that the proposed method offer significant contributions to the area of data stream mining as they allow to efficiently adapt to all types of concept drifts. In this section, we compare the results of EACD, RED-GENE and RED-PSO to analyse and discuss how each algorithm addressed the identified objectives of this thesis. The experimental settings for the analysis presented in this section are detailed in Section 5.3 and the adopted parameters for each algorithm are the same as discussed in each of the respective chapter.

Figure 8.1 shows the average accuracy (in %) of the three proposed methods over the nine datasets specified in Section 4.2 in the immediate setting. According to the results, RED-GENE performs the best over the Hyperplane, LED, RTG, Forest cover-type and Poker-hands datasets, while RED-PSO performs the best over the Arilines, Electricity and KDDcup99 datasets and EACD performs the best over the SEA dataset.

Figure 8.2 shows the average accuracy (in %) of the three proposed methods over the same datasets in the delayed setting. In can be noticed from the figure that RED-GENE performs the best over the Hyperplane, LED, Forest cover-type and Poker-hand datasets, RED-PSO performs the best over the

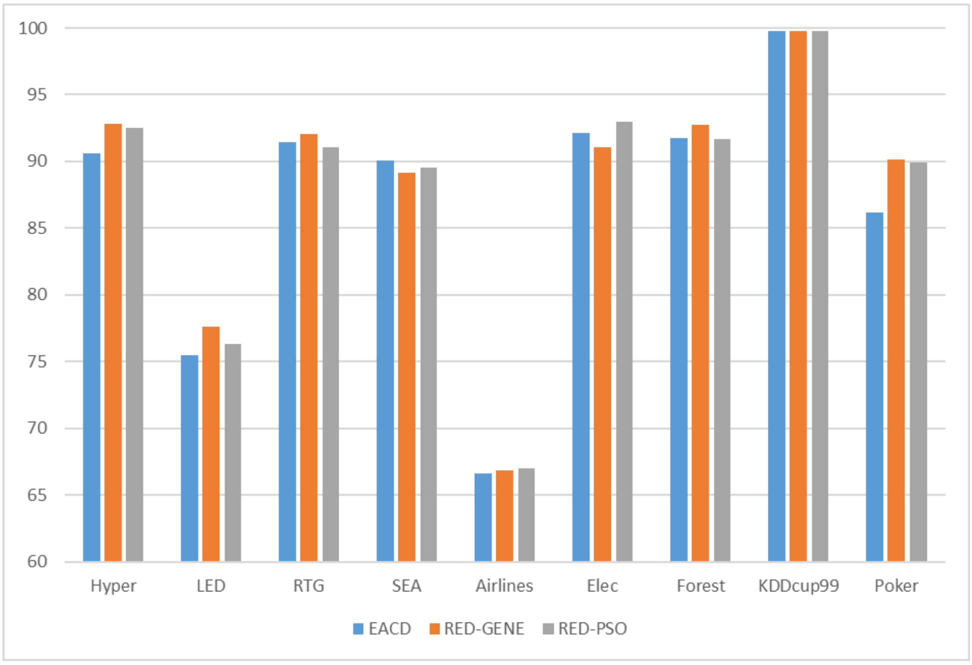


FIGURE 8.1: Accuracy of the proposed methods in the immediate setting.

Airlines and KDDcup99 datasets and EACD performs the best over the SEA, RTG and Electricity datasets.

Figure 8.3 shows the overall time (in seconds) of executing the proposed methods over the same nine datasets in the immediate setting. It is clear that RED-PSO is the most time-efficient method compared to the other two methods, with RED-GENE being more time-efficient than EACD.

8.2.1 Discussion

As can be observed from Figures 8.1, 8.2 and 8.3, each of the proposed methods in this dissertation has its strengths towards concept drift adaptation in non-stationary data stream classification. The advantage of each method can be studied further by taking into consideration the characteristics of data streams used in the experiments. The datasets used in this thesis are introduced in Section 4.2.

The concept drifts in the SEA dataset are manifested in having some features become irrelevant. As can be seen from Figures 8.1 and 8.2, EACD can outperform any other methods over this dataset, because of its active addition and deletion of classifiers periodically. This is needed for a quick recovery of this type of concept drifts because irrelevant features can be taken off the new classifiers (decision trees) and replaced by classifiers that do not model any irrelevant features.

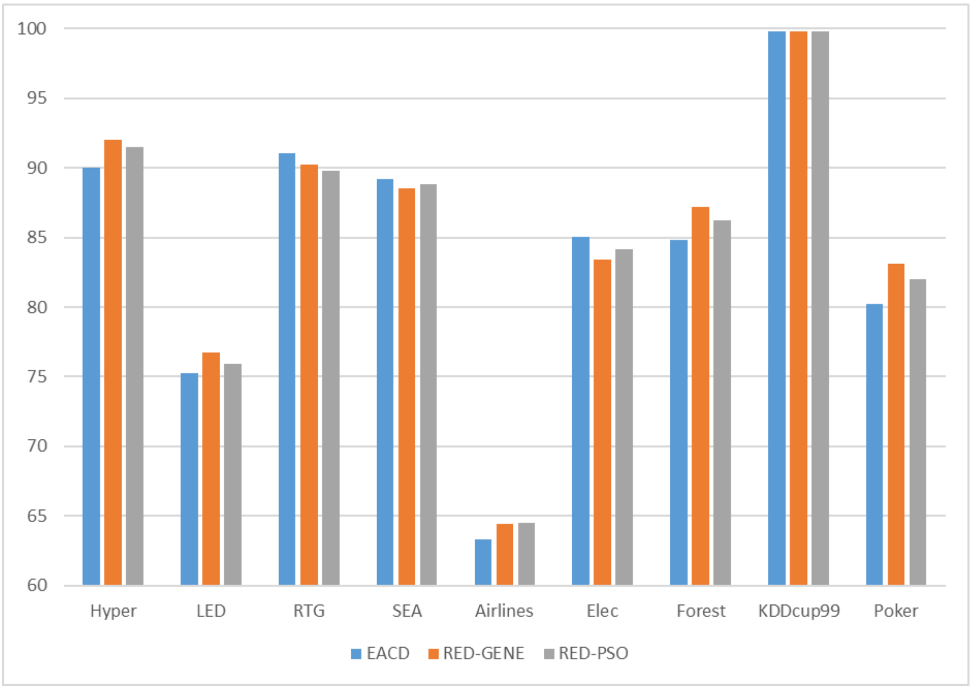


FIGURE 8.2: Accuracy of the proposed methods in the delayed setting.

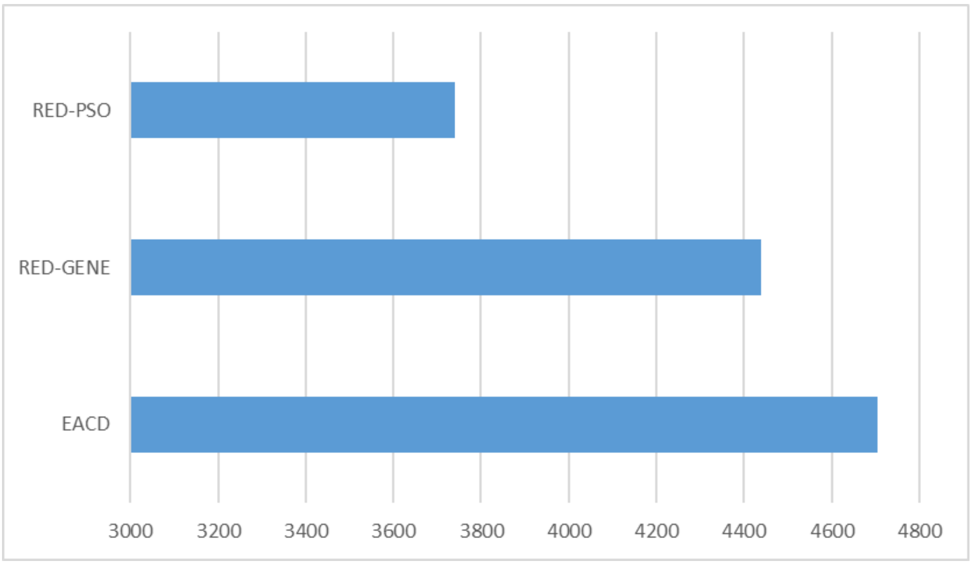


FIGURE 8.3: Overall time (in seconds) of executing the proposed methods over the considered nine datasets in the immediate setting.

Furthermore, in the Hyperplane dataset that the concept drifts are mostly abrupt with various severity, the RED-GENE method outperforms other methods in both immediate and delayed settings. As this method uses a concept drift detection mechanism and abrupt concept drifts can be easily detected by concept drift detectors, the RED-GENE allows a quick adaptation procedure using GA and its dynamic RD modifications.

Moreover, when the target data stream contains more noisy data such as in Airlines dataset, RED-PSO can outperform any other methods as shown in Figures 8.1 and 8.2. The reason for this is due to its capability of investigating both exploration and exploitation aspects of the search space and also its implicit mechanism towards concept drifts that makes this method resistant to false alarms.

By comparing the results of the proposed methods in the immediate and delayed settings, it can be concluded that while EACD performs the best over only one dataset in the immediate setting (over SEA dataset), it performs the best over three of the datasets (SEA, RTG and Electricity) in the delayed setting. This demonstrates that the late arrival of instances has less impact on the accuracy of EACD than it does on the accuracy of RED-GENE and RED-PSO.

According to the results, RED-GENE being built upon EACD slightly improves its average accuracy for the majority of the datasets. At the same time, RED-GENE takes less time to execute compared to EACD (about 35 seconds less on average for each dataset). This makes RED-GENE more efficient than EACD and justifies the proposed three different modifications of RD employed in RED-GENE to satisfy the third objective of this thesis.

Furthermore, it is clear that the RED-PSO method consumes significantly less time compared to EACD and RED-GENE. This is because RED-PSO is an implicit classification method that does not use any concept drift detection methods, while both EACD and RED-GENE are explicit algorithms that use a concept drift detection method. Moreover, the combinations of features in RED-PSO are optimised using PSO, which is less complex compared to GA employed in RED-GENE and EACD. Since the PSO algorithm takes into consideration both local optima (exploration aspect) and global optima (exploitation aspect) of the search space (the combination of features), it can naturally deal with recurrent concept drifts (when an old concept drift reappears after some time).

TABLE 8.1: Suggested method(s) to utilise for different applications or environments.

When? (applications/environments)	What? (suggested method)
Feature irrelevance is the main cause of concept drift	EACD
Concept drifts are mostly abrupt	RED-GENE
Concept drifts are mostly gradual or recurrent	RED-PSO
The number of features of the target data stream is higher than 10	RED-GENE
Systematic noise is added to the synthetic data streams	RED-PSO
The true label of each record is released to the system with a delay	EACD & RED-GENE
Time- and memory-constrained applications	RED-PSO
Imbalanced data streams (where the ratio of minority and majority classes differs significantly)	RED-GENE

In summary, each method proposed in this thesis makes a novel contribution towards concept drift adaptation in non-stationary data stream classification tasks. In particular, EACD improves the performance of the state-of-the-art ensemble learning methods and suggests the use of evolutionary algorithms as a natural way for adapting to concept drifts. The different variations of RD in RED-GENE make novel contributions to both evolutionary game theory and efficient adaptation to concept drifts in data stream classification. Finally, RED-PSO makes a novel contribution to mining evolving data streams by taking into consideration the historical information of the search carried out for each classification *type* in the search space (local optima), as well as the information on the best performing classification *type* (global optima). This approach helps the ensemble to cope with recurrent concept drifts in a timely manner and optimise the combination of features in all classification *types*.

Based on the results of the experiments presented in Chapters 5, 6 and 7, along with the method comparison presented in this chapter, a set of suggestions can be formulated regarding when to use each of the proposed methods. Table 8.1 lists different conditions and suggested method(s) to use for each one of them.

8.3 Future Directions

The methods proposed in this thesis open the door to new developments that should be theoretically analysed and practically tested in the future. The following ideas for future work can be put forward, to mention some:

- detecting the classification *types* that have not been useful for a long time in different environments to remove them and eventually make a room for new, better performing *types* to be added;
- applying other metaheuristic algorithms such as Simulated annealing (SA), Ant Colony Optimisation (ACO) and Harmony Search (HS) to seamlessly adapt to different concept drifts in non-stationary data stream classification;
- using a different removal mechanism when the maximum number of trees for a classification *type* is reached and a classifier (decision tree) should be removed; e.g. removing the oldest classifier inside the *type* instead of removing the worst performing one as proposed in this thesis;
- conducting a more in-depth analysis on real-world data streams regarding how different noises/concept drifts effect the performance of the proposed methods methods.
- proposing a novel concept drift detection algorithm by analysing the behaviour of the classification *types* in the RD method.

In conclusion, this thesis takes a step forward and opens up new opportunities for solving non-stationary data stream classification tasks by proposing hybrid metaheuristic methods that offer a natural solution to this problem.

Appendix A

Results of Experiments Presented in Chapter 5

TABLE A.1: Average accuracy (%) of the EACD variations in the immediate setting. Bold values indicate the best performance for each dataset.

Dataset	$base$	$base2$	$base3$	$base4$	Imp	$Imp2$	Exp	$Exp2$
Hyper.	86.31	77.47	80.52	84.74	89.23	88.54	**90.59**	90.53
LED	68.78	63.05	64.12	67.75	74.78	74.02	**75.45**	75.42
RTG	88.03	79.34	86.34	89.93	**91.89**	91.23	91.42	91.41
SEA	87.35	82.43	84.56	88.90	87.43	85.78	**90.08**	90.00
Airlines	62.97	60.09	61.78	62.08	64.37	63.98	**66.61**	66.60
Elec.	81.01	77.34	80.45	81.76	90.30	90.23	**92.14**	92.10
Forest	83.56	70.34	80.67	85.83	**92.64**	91.94	91.73	91.73
KDDcup	98.76	98.67	98.89	99.76	99.76	99.76	99.78	**99.79**
Poker	80.24	73.45	75.23	79.51	83.45	82.78	**86.21**	86.17
Overall Average	82.00	75.80	79.17	82.25	85.98	85.36	**87.11**	87.08

TABLE A.2: Average accuracy (%) of the EACD variations in the delayed setting. Bold values indicate the best performance for each dataset.

Dataset	*base*	*base2*	*base3*	*base4*	*Imp*	*Imp2*	*Exp*	*Exp2*
Hyper.	84.35	75.34	78.23	83.40	88.43	88.05	**90.02**	89.98
LED	68.17	62.67	64.00	67.69	73.60	73.14	**75.26**	75.25
RTG	87.16	79.02	84.23	87.92	**91.24**	90.20	91.05	91.01
SEA	85.94	80.56	82.43	87.38	87.06	85.24	**89.22**	89.20
Airlines	60.45	56.07	58.12	62.48	62.18	62.56	**63.35**	63.14
Elec.	74.35	73.67	84.35	75.01	83.32	84.35	**85.03**	84.97
Forest	79.45	70.34	79.23	80.05	**85.90**	85.34	84.83	84.80
KDDcup	98.76	98.67	98.84	99.75	99.75	99.76	99.76	**99.77**
Poker	77.92	70.78	73.37	76.90	78.03	77.45	**80.21**	79.24
Overall Average	79.73	74.12	78.09	80.06	83.28	82.90	**84.30**	84.15

TABLE A.3: Average time (in seconds) of executing the EACD variations. Bold values indicate the best performance for each dataset.

Dataset	*base*	*base2*	*base3*	*base4*	*Imp*	*Imp2*	*Exp*	*Exp2*
Hyper.	189	**147**	162	195	297	290	349	349
LED	183	**143**	149	186	419	417	423	420
RTG	233	**202**	227	251	515	509	607	606
SEA	304	**289**	293	316	667	663	880	870
Airlines	228	**216**	220	232	665	659	657	651
Elec.	5.8	**4.7**	5.1	6.0	9.5	9.5	10.4	10.4
Forest	756	**547**	694	756	887	860	935	917
KDDcup	305	**291**	303	319	435	430	536	536
Poker	240	**205**	225	273	319	317	346	346

TABLE A.4: Average time (in seconds) of executing the EACD and other state-of-the-art methods in the immediate setting. Bold values indicate the best performance for each dataset.

Dataset	ARF	DWM	LevBag	OAUE	OSBoost	$EACD_{Exp}$
Hyperp.	208	130	144	107	**93**	349
LED	188	851	246	227	**174**	423
RTG	394	195	207	**148**	1141	607
SEA	751	**98**	409	139	162	880
Airlines	495	**66**	531	366	74	657
Elec.	7.73	**1.48**	5.12	3.05	2.06	10.45
Forest	153	148	206	180	**114**	935
KDDcup	**56**	581	130	204	138	536
Poker	167	**46**	81	66	64	346

Appendix B

Results of Experiments Presented in Chapter 6

TABLE B.1: Average accuracy (%) of the RED-GENE variations and $EACD_{Exp}$ method in the immediate setting over different datasets. Bold values indicate the best performance for each dataset.

Dataset	RD1	RD2	RD3	RD1Lite	RD2Lite	RD3Lite	RD1+GA	RD2+GA	RD3+GA	EACD
Hyperplane	87.21	87.84	89.26	79.46	78.27	81.20	91.15	90.84	**92.84**	90.59
LED	74.83	75.69	76.23	70.24	70.16	71.54	76.04	77.24	**77.65**	75.45
RTG	88.06	88.42	89.05	81.43	82.08	81.93	91.67	91.63	**92.03**	91.42
SEA	89.30	89.34	89.54	79.21	78.36	79.83	89.23	89.71	89.14	**90.08**
Airlines	64.32	63.63	64.39	62.81	61.06	61.45	**66.89**	66.45	66.88	66.61
Electricity	86.91	88.06	89.14	78.35	76.32	79.34	88.45	90.56	**91.03**	92.14
Forest	85.61	86.43	87.45	75.32	77.02	76.13	89.78	91.03	**92.73**	91.73
KDDcup99	99.72	99.71	99.75	99.45	99.51	99.19	99.75	99.78	**99.78**	**99.78**
Poker	87.49	88.24	87.82	79.24	80.74	81.51	89.65	90.05	**90.15**	86.21
Overall Ave.	84.83	85.26	85.85	78.39	78.16	79.12	86.84	87.25	87.98	87.11

TABLE B.2: Average accuracy (%) of the RED-GENE variations and $EACD_{Exp}$ method in the delayed setting over different datasets. RD3+GA achieves the highest average accuracy over five out of nine data sets. Bold values indicate the best performance for each dataset.

Dataset	RD1	RD2	RD3	RD1Lite	RD2Lite	RD3Lite	RD1+GA	RD2+GA	RD3+GA	EACD
Hyperplane	86.94	87.23	88.86	79.53	78.14	80.04	90.52	90.92	**92.03**	90.02
LED	75.15	75.69	75.23	71.34	70.25	70.93	76.01	76.54	**76.73**	75.26
RTG	88.06	87.42	87.05	81.30	81.54	81.32	89.44	89.08	90.24	**91.05**
SEA	87.30	88.34	87.52	78.46	78.03	79.24	88.34	**89.24**	88.54	89.22
Airlines	63.85	62.03	62.39	61.67	60.78	61.06	**65.49**	64.34	64.45	63.35
Electricity	81.91	81.20	81.42	75.35	74.20	77.25	83.56	82.90	83.41	**85.03**
Forest	84.61	85.43	85.83	75.19	76.24	75.50	86.42	86.50	**87.19**	84.83
KDDcup99	99.73	99.72	99.73	99.46	99.49	99.50	99.74	99.73	**99.76**	**99.76**
Poker	80.49	79.53	80.56	77.16	78.42	78.92	82.65	82.10	**83.11**	80.21
Overall Ave.	83.12	82.95	83.18	77.71	77.45	78.19	84.69	84.58	85.05	84.30

TABLE B.3: Average time of executing the RED-GENE variations and $EACD_{Exp}$ in the immediate setting over different datasets. Bold values indicate the best performance for each dataset.

Dataset	RD1	RD2	RD3	RD1Lite	RD2Lite	RD3Lite	RD1+GA	RD2+GA	RD3+GA	EACD
Hyperplane	181	180	181	**134**	136	136	360	352	358	349
LED	186	185	188	132	**130**	134	400	398	418	423
RTG	338	335	339	200	**198**	200	510	505	512	607
SEA	404	398	411	303	**299**	302	708	680	776	880
Airlines	402	399	401	245	**229**	249	596	584	598	657
Electricity	10.32	10.48	10.36	10.01	10.02	**10.00**	10.44	10.45	10.46	10.45
Forest	787	777	791	502	**499**	512	841	803	885	935
KDDcup99	352	359	359	**240**	245	246	498	500	530	536
Poker	291	274	297	232	**202**	232	367	336	356	346

TABLE B.4: Average time (in seconds) of executing the RED-GENE and other state-of-the-art methods in the immediate setting. Bold values indicate the best performance for each dataset.

Dataset	ARF	DWM	LevBag	OAUE	OSBoost	RD3+GA
Hyperplane	208	130	144	107	**93**	358
LED	188	851	246	227	**174**	418
RTG	394	195	207	**148**	1141	512
SEA	751	**98**	409	139	162	776
Airlines	495	**66**	531	366	74	598
Elec.	7.73	**1.48**	5.12	3.05	2.06	10.46
Forest	153	148	206	180	**114**	885
KDDcup	**56**	581	130	204	138	530
Poker	167	**46**	81	66	64	356

Appendix C

Results of Experiments Presented in Chapter 7

TABLE C.1: Average accuracy (%) of the RED-PSO variations in the immediate setting over different datasets. Bold values indicate the best performance for each dataset.

Dataset	RED1	RED2	RED-PSO1	RED-PSO2	RED-PSO3
Hyperp.	87.83	87.94	90.43	91.92	**92.54**
LED	75.41	75.88	76.01	**76.54**	76.29
RTG	87.06	88.32	89.44	89.98	**91.09**
SEA	87.30	86.78	88.01	87.74	**88.50**
Airlines	61.85	61.06	63.49	65.34	**66.68**
Electricity	90.61	89.34	**93.56**	92.43	92.86
Forest	81.56	88.01	88.42	92.99	**93.71**
KDDcup	99.53	99.67	99.62	99.79	**99.80**
Poker	85.89	87.34	88.56	**90.10**	89.89

TABLE C.2: Average accuracy (%) of the RED-PSO variations in the delayed setting over different datasets. Bold values indicate the best performance for each dataset.

Dataset	RED1	RED2	RED-PSO1	RED-PSO2	RED-PSO3
Hyperp.	87.74	88.14	91.32	91.18	**91.48**
LED	70.29	75.12	73.61	75.03	**75.91**
RTG	86.89	87.90	88.16	89.26	**89.81**
SEA	86.66	86.14	88.14	87.34	**88.80**
Airlines	58.01	59.76	62.29	62.87	**64.54**
Electricity	82.45	81.90	**85.51**	84.00	84.18
Forest	78.34	82.34	82.42	87.32	**88.19**
KDDcup	99.73	99.70	99.60	99.76	**99.77**
Poker	78.54	79.05	80.56	**82.06**	81.98

TABLE C.3: Average time (in seconds) of executing the RED-PSO variations in the immediate setting. Bold values indicate the best performance for each dataset.

Dataset	RED1	RED2	RED-PSO1	RED-PSO2	RED-PSO3
Hyper.	184	**172**	279	230	253
LED	204	**189**	400	379	388
RTG	281	**243**	405	358	379
SEA	**163**	176	296	320	308
Airlines	398	**365**	528	494	543
Elec.	9.3	**8.9**	17.8	16.1	16.4
Forest	741	**722**	10871	1167	1334
KDDcup	269	**229**	349	301	308
Poker	**152**	159	201	228	210

TABLE C.4: Average time (in seconds) of executing the RED-PSO3 and other state-of-the-art methods in the immediate setting. Bold values indicate the best performance for each dataset.

Dataset	ARF	DWM	Lev-Bag	OAUE	OS-Boost	Learn-++	ADOB	RED-PSO3
Hyper.	208	130	144	107	**93**	239	298	253
LED	188	851	246	227	**174**	301	340	388
RTG	394	195	207	**148**	1141	531	1261	379
SEA	751	**98**	409	139	162	240	284	308
Airlines	495	**66**	531	366	74	977	2140	543
Elec.	7.73	**1.48**	5.12	3.05	2.06	5.9	221	16.4
Forest	153	148	206	180	114	**67**	2292	1334
KDDcup.	**56**	581	130	204	138	9819	5979	308
Poker	167	**46**	81	66	64	1720	2006	210
Overall	2419	2116	1959	**1440**	1942	13899	14821	3409

Bibliography

[1] S. Chakrabarti, M. Ester, U. Fayyad, J. Gehrke, J. Han, S. Morishita, G. Piatetsky-Shapiro, and W. Wang, "Data mining curriculum: A proposal (version 1.0)", *Intensive Working Group of ACM SIGKDD Curriculum Committee*, vol. 140, 2006.

[2] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*, 10. Springer series in statistics New York, 2001, vol. 1.

[3] C. M. Bishop, *Pattern recognition and machine learning*. Springer Science+ Business Media, 2006.

[4] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited, 2016.

[5] C. C. Aggarwal, *Data classification: algorithms and applications*. CRC press, 2014.

[6] R. Kohavi *et al.*, "A study of cross-validation and bootstrap for accuracy estimation and model selection", in *Ijcai*, Montreal, Canada, vol. 14, 1995, pp. 1137–1145.

[7] M. M. Gaber, A. Zaslavsky, and S. Krishnaswamy, "Mining data streams: A review", *ACM Sigmod Record*, vol. 34, no. 2, pp. 18–26, 2005.

[8] H. Kargupta, R. Bhargava, K. Liu, M. Powers, P. Blair, S. Bushra, J. Dull, K. Sarkar, M. Klein, M. Vasa, *et al.*, "Vedas: A mobile and distributed data stream mining system for real-time vehicle monitoring", in *Proceedings of the 2004 SIAM International Conference on Data Mining*, SIAM, 2004, pp. 300–311.

[9] M. PhridviRaj and C. GuruRao, "Data mining–past, present and future–a typical survey on data streams", *Procedia Technology*, vol. 12, pp. 255–263, 2014.

[10] Z. Miller, B. Dickinson, W. Deitrick, W. Hu, and A. H. Wang, "Twitter spammer detection using data stream clustering", *Information Sciences*, vol. 260, pp. 64–73, 2014.

[11] W. Fan, Y.-a. Huang, H. Wang, and P. S. Yu, “Active mining of data streams”, in *Proceedings of the 2004 SIAM International Conference on Data Mining*, SIAM, 2004, pp. 457–461.

[12] V.-D. Ta, C.-M. Liu, and G. W. Nkabinde, “Big data stream computing in healthcare real-time analytics”, in *2016 IEEE International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*, IEEE, 2016, pp. 37–42.

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

[14] H. M. Gomes, J. P. Barddal, F. Enembreck, and A. Bifet, “A survey on ensemble learning for data stream classification”, *ACM Computing Surveys (CSUR)*, vol. 50, no. 2, p. 23, 2017.

[15] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, “A survey on concept drift adaptation”, *ACM Computing Surveys (CSUR)*, vol. 46, no. 4, p. 44, 2014.

[16] M. M. Gaber, “Advances in data stream mining”, *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 2, no. 1, pp. 79–85, 2012.

[17] L. N. De Castro and F. J. Von Zuben, *Recent developments in biologically inspired computing*. Igi Global, 2005.

[18] C. Teuscher, D. Mange, A. Stauffer, and G. Tempesti, “Bio-inspired computing tissues: Towards machines that evolve, grow, and learn”, *Biosystems*, vol. 68, no. 2-3, pp. 235–244, 2003.

[19] M. L. McHugh, “Interrater reliability: The kappa statistic”, *Biochemia medica: Biochemia medica*, vol. 22, no. 3, pp. 276–282, 2012.

[20] H. Ghomeshi, M. M. Gaber, and Y. Kovalchuk, “Ensemble dynamics in non-stationary data stream classification”, in *Learning from Data Streams in Evolving Environments*, Springer, 2019, pp. 123–153.

[21] ——, “Eacd: Evolutionary adaptation to concept drifts in data streams”, *Data Mining and Knowledge Discovery*, vol. 33, no. 3, pp. 663–694, 2019.

[22] I. M. Bomze, “Lotka-volterra equation and replicator dynamics: A two-dimensional classification”, *Biological cybernetics*, vol. 48, no. 3, pp. 201–211, 1983.

[23] J. Hofbauer and K. Sigmund, “Evolutionary game dynamics”, *Bulletin of the American Mathematical Society*, vol. 40, no. 4, pp. 479–519, 2003.

[24] K. Fawgreh, M. M. Gaber, and E. Elyan, “A replicator dynamics approach to collective feature engineering in random forests”, in *Research and Development in Intelligent Systems XXXII*, Springer, 2015, pp. 25–41.

[25] H. Ghomeshi, M. M. Gaber, and Y. Kovalchuk, “Red-gene: An evolutionary game theoretic approach to adaptive data stream classification”, *IEEE Access*, 2019.

[26] ——, “A non-canonical hybrid metaheuristic approach to adaptive data stream classification”, *Future Generation Computer Systems*, vol. 102, pp. 127–139, 2020.

[27] J Kennedy and R Eberhart, *Particle swarm optimization, proceedings of ieee international conference on neural networks (icnn’95) in*, 1995.

[28] R. Poli, “An analysis of publications on particle swarm optimization applications”, *Essex, UK: Department of Computer Science, University of Essex*, 2007.

[29] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, “Models and issues in data stream systems”, in *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, ACM, 2002, pp. 1–16.

[30] F. Chu and C. Zaniolo, “Fast and light boosting for adaptive mining of data streams”, in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, Springer, 2004, pp. 282–292.

[31] L. Breiman, “Bagging predictors”, *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.

[32] Y. Freund and R. E. Schapire, “A decision-theoretic generalization of on-line learning and an application to boosting”, *Journal of computer and system sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[33] 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*, ACM, 2009, pp. 139–148.

[34] N. C. Oza, “Online bagging and boosting”, in *Systems, man and cybernetics, 2005 IEEE international conference on*, IEEE, vol. 3, 2005, pp. 2340–2345.

[35] A. Bifet and R. Gavalda, "Learning from time-changing data with adaptive windowing", in *Proceedings of the 2007 SIAM International Conference on Data Mining*, SIAM, 2007, pp. 443–448.

[36] A. Bifet, G. Holmes, and B. Pfahringer, "Leveraging bagging for evolving data streams", *Machine Learning and Knowledge Discovery in Databases*, pp. 135–150, 2010.

[37] P. M. Gonçalves Jr and R. S. M. De Barros, "Rcd: A recurring concept drift framework", *Pattern Recognition Letters*, vol. 34, no. 9, pp. 1018–1025, 2013.

[38] H. M. Gomes, A. Bifet, J. Read, J. P. Barddal, F. Enembreck, B. Pfharinger, G. Holmes, and T. Abdessalem, "Adaptive random forests for evolving data stream classification", *Machine Learning*, pp. 1–27, 2017.

[39] L. Breiman, "Random forests", *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[40] S. G. T. de Carvalho Santos, P. M. G. Júnior, G. D. dos Santos Silva, and R. S. M. de Barros, "Speeding up recovery from concept drifts", in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, Springer, 2014, pp. 179–194.

[41] S.-T. Chen, H.-T. Lin, and C.-J. Lu, "An online boosting algorithm with theoretical justifications", *arXiv preprint arXiv:1206.6422*, 2012.

[42] J. Z. Kolter and M. A. Maloof, "Dynamic weighted majority: An ensemble method for drifting concepts", *Journal of Machine Learning Research*, vol. 8, no. Dec, pp. 2755–2790, 2007.

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

[44] P. Domingos and G. Hulten, "Mining high-speed data streams", in *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2000, pp. 71–80.

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

[46] G. Jaber, "An approach for online learning in the presence of concept change", PhD thesis, Citeseer, 2013.

[47] H. M. Gomes and F. Enembreck, "Sae: Social adaptive ensemble classifier for data streams", in *Computational Intelligence and Data Mining (CIDM), 2013 IEEE Symposium on*, IEEE, 2013, pp. 199–206.

[48] ——, "Sae2: Advances on the social adaptive ensemble classifier for data streams", in *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, ACM, 2014, pp. 798–804.

[49] G. Folino, C. Pizzuti, and G. Spezzano, "An adaptive distributed ensemble approach to mine concept-drifting data streams", in *Tools with Artificial Intelligence, 2007. ICTAI 2007. 19th IEEE International Conference on*, IEEE, vol. 2, 2007, pp. 183–188.

[50] ——, "Gp ensembles for large-scale data classification", *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 5, pp. 604–616, 2006.

[51] P. Vivekanandan and R. Nedunchezhian, "Mining data streams with concept drifts using genetic algorithm", *Artificial Intelligence Review*, vol. 36, no. 3, pp. 163–178, 2011.

[52] S. Ramamurthy and R. Bhatnagar, "Tracking recurrent concept drift in streaming data using ensemble classifiers", in *Machine Learning and Applications, 2007. ICMLA 2007. Sixth International Conference on*, IEEE, 2007, pp. 404–409.

[53] K. O. Stanley, "Learning concept drift with a committee of decision trees", *Informe técnico: UT-AI-TR-03-302, Department of Computer Sciences, University of Texas at Austin, USA*, 2003.

[54] J. Z. Kolter and M. A. Maloof, "Using additive expert ensembles to cope with concept drift", in *Proceedings of the 22nd international conference on Machine learning*, ACM, 2005, pp. 449–456.

[55] 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*, ACM, 2001, pp. 377–382.

[56] 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*, AcM, 2003, pp. 226–235.

[57] J. Rushing, S. Graves, E. Criswell, and A. Lin, “A coverage based ensemble algorithm (cbea) for streaming data”, in *Tools with Artificial Intelligence, 2004. ICTAI 2004. 16th IEEE International Conference on*, IEEE, 2004, pp. 106–112.

[58] K. Nishida and K. Yamauchi, “Adaptive classifiers-ensemble system for tracking concept drift”, in *Machine Learning and Cybernetics, 2007 International Conference on*, IEEE, vol. 6, 2007, pp. 3607–3612.

[59] M. Deckert, “Batch weighted ensemble for mining data streams with concept drift”, in *International Symposium on Methodologies for Intelligent Systems*, Springer, 2011, pp. 290–299.

[60] R. Elwell and R. Polikar, “Incremental learning of concept drift in nonstationary environments”, *IEEE Transactions on Neural Networks*, vol. 22, no. 10, pp. 1517–1531, 2011.

[61] H.-L. Nguyen, Y.-K. Woon, W.-K. Ng, and L. Wan, “Heterogeneous ensemble for feature drifts in data streams”, *Advances in Knowledge Discovery and Data Mining*, pp. 1–12, 2012.

[62] M. Woźniak, “Application of combined classifiers to data stream classification”, in *Computer Information Systems and Industrial Management*, Springer, 2013, pp. 13–23.

[63] A. Ortíz Díaz, J. del Campo-Ávila, G. Ramos-Jiménez, I. Frías Blanco, Y. Caballero Mota, A. Mustelier Hechavarría, and R. Morales-Bueno, “Fast adapting ensemble: A new algorithm for mining data streams with concept drift”, *The Scientific World Journal*, vol. 2015, 2015.

[64] S. R. Safavian and D. Landgrebe, “A survey of decision tree classifier methodology”, *IEEE transactions on systems, man, and cybernetics*, vol. 21, no. 3, pp. 660–674, 1991.

[65] I. Rish *et al.*, “An empirical study of the naive bayes classifier”, in *IJCAI 2001 workshop on empirical methods in artificial intelligence*, vol. 3, 2001, pp. 41–46.

[66] S. Dreiseitl and L. Ohno-Machado, “Logistic regression and artificial neural network classification models: A methodology review”, *Journal of biomedical informatics*, vol. 35, no. 5-6, pp. 352–359, 2002.

[67] C. Cortes and V. Vapnik, “Support-vector networks”, *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[68] Y. Liao and V. R. Vemuri, “Use of k-nearest neighbor classifier for intrusion detection”, *Computers & security*, vol. 21, no. 5, pp. 439–448, 2002.

[69] R. A. Fisher, “The use of multiple measurements in taxonomic problems”, *Annals of eugenics*, vol. 7, no. 2, pp. 179–188, 1936.

[70] J. R. Quinlan, “Induction of decision trees”, *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[71] J. R. Quinlan, *C4. 5: programs for machine learning*. Elsevier, 2014.

[72] L. Breiman, *Classification and regression trees*. Routledge, 2017.

[73] W. Hoeffding, “Probability inequalities for sums of bounded random variables”, in *The Collected Works of Wassily Hoeffding*, Springer, 1994, pp. 409–426.

[74] R. B. Myerson, *Game theory*. Harvard university press, 2013.

[75] E. Elyan and M. M. Gaber, “A genetic algorithm approach to optimising random forests applied to class engineered data”, *Information sciences*, vol. 384, pp. 220–234, 2017.

[76] A. Mantri, S. N. S. Kendra, G. Kumar, and S. Kumar, *High Performance Architecture and Grid Computing: International Conference, HPAGC 2011, Chandigarh, India, July 19-20, 2011. Proceedings*. Springer, 2011, vol. 169.

[77] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, “Learning with drift detection”, in *Brazilian symposium on artificial intelligence*, Springer, 2004, pp. 286–295.

[78] M. Baena-García, J. del Campo-Ávila, R. Fidalgo, A. Bifet, R. Gavaldà, and R. Morales-Bueno, “Early drift detection method”, 2006.

[79] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, “Moa: Massive online analysis”, *Journal of Machine Learning Research*, vol. 11, no. May, pp. 1601–1604, 2010.

[80] G. Hulten, L. Spencer, and P. Domingos, “Mining time-changing data streams”, in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2001, pp. 97–106.

[81] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and regression trees*. CRC press, 1984.

[82] J. A. Blackard and D. J. Dean, “Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables”, *Computers and electronics in agriculture*, vol. 24, no. 3, pp. 131–151, 1999.

[83] M. Harries and N. S. Wales, “Splice-2 comparative evaluation: Electricity pricing”, 1999.

[84] K. Cup, "Data (1999)", *URL: http://kdd.ics.uci.edu/databases/kddcup99*, 1999.

[85] J. C. Spall, *Introduction to stochastic search and optimization: estimation, simulation, and control*. John Wiley & Sons, 2005, vol. 65.

[86] M. Gen and R. Cheng, *Genetic algorithms and engineering optimization*. John Wiley & Sons, 2000, vol. 7.

[87] M. Friedman, "A comparison of alternative tests of significance for the problem of m rankings", *The Annals of Mathematical Statistics*, vol. 11, no. 1, pp. 86–92, 1940.

[88] J. Demšar, "Statistical comparisons of classifiers over multiple data sets", *Journal of Machine learning research*, vol. 7, no. Jan, pp. 1–30, 2006.

[89] P. Nemenyi, "Distribution-free multiple comparisons", in *Biometrics*, INTERNATIONAL BIOMETRIC SOC 1441 I ST, NW, SUITE 700, WASHINGTON, DC 20005-2210, vol. 18, 1962, p. 263.