ADAPTIVE ALGORITHMS FOR CLASSIFICATION ON HIGH-FREQUENCY DATA STREAMS: APPLICATION TO FINANCE

by

Andrés León Suárez Cetrulo

A dissertation submitted by in partial fulfillment of the requirements for the degree of Doctor of Philosophy in

COMPUTER SCIENCE AND TECHNOLOGY

Universidad Carlos III de Madrid

Advisor(s):

Dr. Alejandro Cervantes Rovira
Dr. David Quintana Montero

Tutor:

Dr. David Quintana Montero

Leganés. June, 2022

Part of this Ph.D. dissertation was written during a research stay in Ireland's Centre for Applied Artificial Intelligence (CeADAR), in the School of Computer Science, University College Dublin (UCD), Ireland

Please feel free to reach the author at: andres.suarez-cetrulo@ucd.ie

This page is intentionally left blank.

Acknowledgements

The last five years of my life have been a continuous challenge. I've been juggling to have time to work in industry, perform this part-time Ph.D. and spend time with friends and my partner. For this reason, in the first place, I want to thank her, Azucena Blanco, for all the continuous support and patience during this long period, which accounts for millions of rows of data at the second level in my thesis. This is a living example of how love is a system that needs to perform at scale. Structural breaks can be observed in behavioural finance and a part of our lives, which is a continuous dynamic environment. Azucena has been next to me during these five years, including the COVID-19 pandemic and recurring lockdowns. She deserves a special mention as her routines and living style have helped me finish this Ph.D.

This Ph.D. started as a hobby, as a way to improve my knowledge in Artificial Intelligence, and as a way to pursue a research passion that I discovered thanks to one of my supervisors, Dr. Alejandro Cervantes. Ten years ago, I was working on my Bachelor's degree thesis with him, and he made me fall in love with his research topic at the time, online machine learning for data streams. Having him as a supervisor during this long time has entirely conditioned my life and career for the better. I also want to thank Dr. David Quintana for his continuous patience during this Ph.D., allowing me to perform it part-time and always be a great mentor to listen and provide valuable and constructive feedback.

During this time, I've also had numerous managers that have inspired and supported me. When I started my Ph.D., I was working as a Data Scientist in Deloitte Ireland on a project for a multinational financial institution. My mentor, Dr. Michael Bridges, became a role model for me and made me understand Data Science as a way to think and live, approaching problems that can be at the code or life level. My directors and partners in Deloitte always supported me in doing a Ph.D. and gave me a study leave during the second year to progress with my research. I want to thank especially Mrs. Freda Cunningham and Mr. Simon Murphy for that. Thanks as well to Dr. Richard Southern for putting together such a great team of professionals in the Deloitte Advanced Analytics team and believing in me when my English was not great. Hopefully, the little I could say, I said it right.

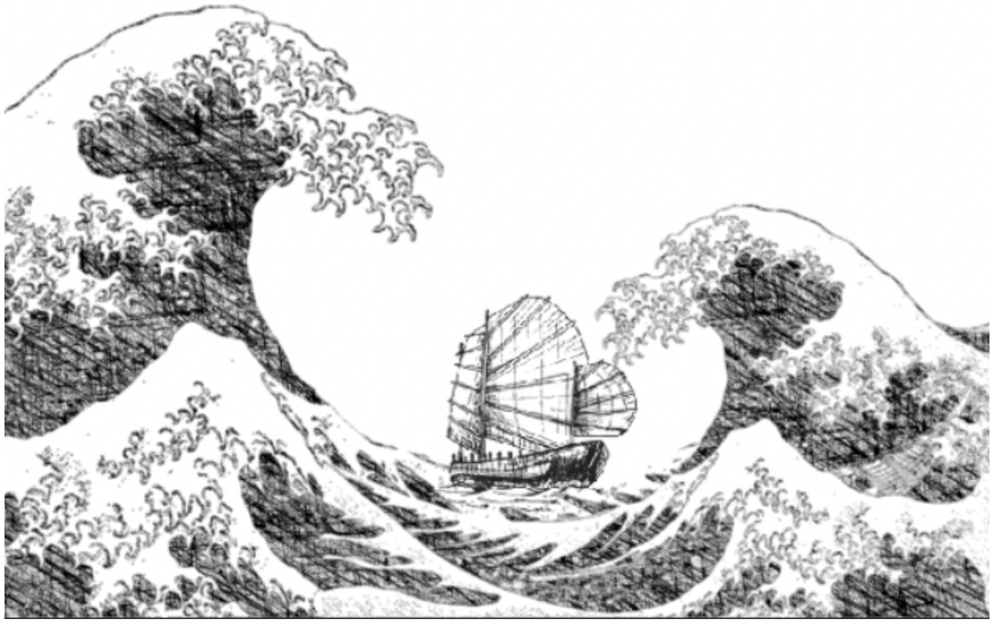
Without Richard sending me to a three-year project in a financial institution, I would never have started this Ph.D. In this financial institution, I worked on a confidential project called P11. That's all I can say. I worked with brilliant colleagues that later became friends. I want to thank Mrs. Puneetha Kucherova for believing in me. You wanted to see what could I do in my thesis. Here you have it! I want to thank Dr. Marcelo Beckmann for his guidance during my Ph.D's initial phase. Thanks to all members of P11 since I learned things from each of you that, sooner or later, I would apply to this thesis.

After P11, I was lucky to get hired to work in Ireland's Centre for Applied AI (CeADAR). The experience developed in this research centre part of UCD has helped me improve my knowledge across different fields in AI. I want to express my gratitude with Dr. Edward McDonnell and Dr. Ricardo Simón Carbajo for their continuous support and flexibility since I joined CeADAR, allowing me to perform this Ph.D. part-time in parallel to my work in innovation projects. I must also thank members of the Innovation of Development Group and the Applied Research Group for their support and lessons that have helped me bring this thesis to the next level.

And last but not least, thanks to everyone who thought I would not make it. You have been a good motivation.

En mi lengua natal, quiero agradecer a mi familia sus continuos ánimos todos estos años. A mis padres Virgilio y Nazarena por enseñarme la ley del esfuerzo. Me hubiese sido imposible hacer este doctorado a tiempo parcial sin la educación que ellos me dieron. A mis abuelos Abu y Mamina, y a mis padrinos Dani y Nanda, por introducirme desde pequeño en la informática, la escritura y el pensamiento crítico. Y al resto de mi familia, tanto la Suárez como la Cetrulo, por poner un granito de arena, cada uno, en lo que me ha traído hasta aquí a día de hoy.

Por último, gracias a Josh por su magnífico diseño para la portada física de esta tesis. En el boceto de abajo, un antiguo barco de *vela japonés* navega a través de una *corriente* entre grandes olas. El barco representa modelos entrenados con datos financieros utilizados en esta tesis (precios), que reciben los datos como *streams* para evitar oleadas mayores.



This page is intentionally left blank.

Published Content

As part of the body of this Ph.D. thesis, the candidate has published two articles indexed by the *Journal Citation Report* (Q1 and Q2).

Suárez-Cetrulo, Andrés L and Cervantes, Alejandro.

Title: An online classification algorithm for large scale data streams: iGNGSVM [328].

Date: Nov 2017.

Journal: *Neurocomputing*.

Volume: 262.

Pages: 67-76.

Doi: 10.1016/j.neucom.2016.12.093

Url: www.sciencedirect.com/science/article/pii/S0925231217309852

Publisher: Elsevier.

Contribution: Software, Investigation, Visualization, Writing - original draft, Writing - review and editing.

Wholly included in this thesis. The material from this source included in this thesis is not singled out with typographic means and references.

Section 4.1: Using GNG to Model Data Distributions in Streams.

Suárez-Cetrulo, Andrés L and Cervantes, Alejandro and Quintana, David.

Title: Incremental Market Behavior Classification in Presence of Recurring Concepts [329].

Date: Jan 2019.

Journal: *Entropy*.

Number: 1.

Volume: 21.

Pages: 25.

Doi: 10.3390/e21010025

Url: https://www.mdpi.com/1099-4300/21/1/25

Publisher: Multidisciplinary Digital Publishing Institute.

Contribution: Software, Investigation, Visualization, Writing - original draft, Writing - review and editing.

Wholly included in this thesis. The material from this source included in this thesis is not singled out with typographic means and references.

Section 4.2: Changes over Adaptive Random Forest for Recurrences.

Other Research Merits

Beyond the research presented in this thesis, we have considered the work performed during these years important. During the prosecution of his Ph.D., the candidate has been a full-time staff in a research centre in Ireland[1], participated in hands-on research, proposal writing in different research projects at the national and European level (Horizon Europe), and has been an active member of a task leader team in the H2020 call DT-ICT-05-2020.

Furthermore, the candidate has worked in parallel projects that overlap with his research area and has collaborated in the following articles; one of them is a journal article indexed by the *Journal Citation Report* (Q1).

Alonso-Monsalve, Saúl and Suárez-Cetrulo, Andrés L and Cervantes, Alejandro and Quintana, David.

Title: Convolution on neural networks for high-frequency trend prediction of cryptocurrency exchange rates using technical indicators [9].

Date: July 2020.

Journal: *Expert Systems with Applications.*

Doi: 10.1016/j.eswa.2020.113250

Publisher: Elsevier.

[1] Centre for AI & Applied Data Analytics (CeADAR) https://ati.ec.europa.eu/technology-centre/ceadar-irelands-centre-ai-and-applied-data-analytics

Suárez-Cetrulo, Andrés L and Kumar, Ankit and Miralles-Pechuán, Luis.

Title: Modelling the COVID-19 virus evolution with Incremental Machine Learning [330].

Date: March 2022.

Pages: 12.

Urn: https://nbn-resolving.org/urn:nbn:de:0074-3105-7

Conference: The 29th Irish Conference on Artificial Intelligence and Cognitive Science 2021 (AICS 2021), Dublin, Republic of Ireland, December 9-10, 2021.

Publisher: CEUR Workshop Proceedings.

Suárez-Cetrulo, Andrés L and Burnham-King, Lauren and Haugthon, David and Simón Carbajo, Ricardo.

Title: Wind Power Forecasting using Ensemble Learning for Day-ahead Energy Trading [326].

Date: 2022.

Journal: *SSRN Electronic Journal* [Preprint].

Pages: 32.

Doi: https://doi.org/10.2139/ssrn.4015233

Url: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4015233

Publisher: Elsevier.

Abstract

In recent years, the problem of concept drift has gained importance in the financial domain. The succession of manias, panics and crashes have stressed the non-stationary nature and the likelihood of drastic structural changes in financial markets. The most recent literature suggests the use of conventional machine learning and statistical approaches for this. However, these techniques are unable or slow to adapt to non-stationarities and may require re-training over time, which is computationally expensive and brings financial risks.

This thesis proposes a set of adaptive algorithms to deal with high-frequency data streams and applies these to the financial domain. We present approaches to handle different types of concept drifts and perform predictions using up-to-date models. These mechanisms are designed to provide fast reaction times and are thus applicable to high-frequency data. The core experiments of this thesis are based on the prediction of the price movement direction at different intraday resolutions in the SPDR S&P 500 exchange-traded fund. The proposed algorithms are benchmarked against other popular methods from the data stream mining literature and achieve competitive results.

We believe that this thesis opens good research prospects for financial forecasting during market instability and structural breaks. Results have shown that our proposed methods can improve prediction accuracy in many of these scenarios. Indeed, the results obtained are compatible with ideas against the efficient market hypothesis. However, we cannot claim that we can beat consistently buy and hold; therefore, we cannot reject it.

Keywords: Meta Learning, Adaptive Learning, Data Streams, Prototype Generation, Recurring Concepts, Concept Drift, Stock Price Direction Prediction, Structural Breaks, Regime Changes, Computational Finance

This page is intentionally left blank.

Contents

*

This page is intentionally left blank.

List of Figures

List of Tables

This page is intentionally left blank.

Chapter 1

Introduction

In this chapter, we introduce the work performed in this Ph.D. thesis. In Section 1.1, we start describing its research context, located in the intersection between the domains of machine learning for large scale data and finance.

This is followed by the motivation of this thesis and the description of the main problems that we aim to address in Section 1.2. To solve these problems,a we work under a set of hypotheses which we describe in Section 1.3. For this whole process, we follow a set of research objectives and a research methodology that is documented in Sections 1.4 and 1.5 respectively. After this, in Section 1.6 we explain the scientific impact that we expect from the outputs of this research work.

Finally, Section 1.7 describes the structure of the remainder of this thesis.

1.1 Context

This Ph.D. thesis involves research work in two subfields of computer science: artificial intelligence (AI) [314] and computational finance [255]. Nowadays, AI is applied to numerous domains; computational finance can involve different applications in quantitative investing, or in algorithmic trading and high-frequency trading (HFT) [90]. Both AI and computational finance are very broad fields that are described in more depth in Chapter 2. To be more specific, our work takes place in the overlap of two research areas: online machine learning for data streams [48, 175] and regime changes (RCs) in financial markets [19].

The overall hierarchy of areas studied is illustrated in Figure 1.1. Computational finance and artificial intelligence are subfields of computer science that overlap with other disciplines, such as finance, statistics and mathematics. These are better described and reviewed in Chapters 2 and 3.

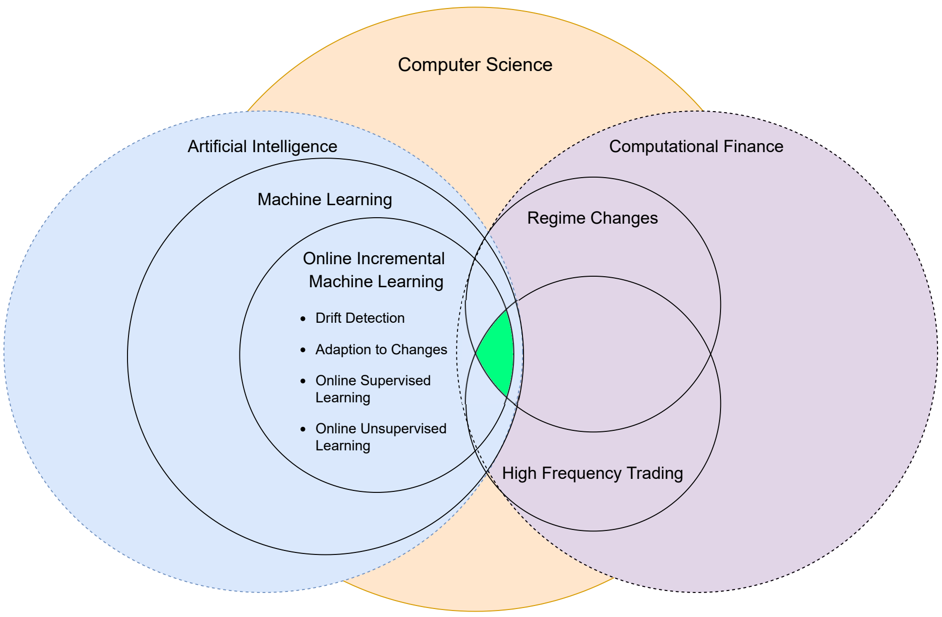


Figure 1.1: *Context of this Ph.D. Thesis. The green area represents the current work.*

This thesis will explore different online incremental learning techniques for data stream classification (supervised learning). We will also focus on the areas of unsupervised online machine learning, as well as on approaches to handle and measure concept drifts. These concepts will be applied to predict ups and downs in stock market prices during RCs at high frequencies (HFT). A list of the areas studied, their subfields, and a brief explanation of each can be seen below. In any case, formal definitions for these are given in Chapter 2.

1. Online incremental machine learning: this field focuses on machine learning algorithms (a subfield of AI) to predict and train over continuous data streams on the fly. Online learning algorithms should always be up to date and available to predict. Here we will focus on many aspects:

    - *Supervised learning*: supervised machine learning approaches train an algorithm to recognise a target feature or label given a set of attributes or features. They are considered *supervised* since the input data needs to be labelled prior to training tasks.

- *Unsupervised learning*: conversely, these techniques do not need labelled data for training. They group the incoming data based on the similarity of the feature vectors [386].
- *Handling of concept drifts*: there are different mechanisms to detect or adapt to changes in non-stationary data or between states of stationary data [37, 118, 290, 386]. A machine learning algorithm for data streams must be able to adapt to these without the need of retraining [154]. This thesis explores passive adaption and drift detection techniques to deal with these changes over time.

2. Computational finance: in general, AI is applied to finance for prediction of future prices or trends in financial assets, optimisation of investment portfolios, and sentiment analysis of news regarding assets or companies [129]. The main application area of this thesis is price trend prediction at high frequencies. The final purpose of our system is to serve as an indicator for HFT. We will focus on:

    - *Price trend classification at high frequencies*: we will consider the literature on stocks price movement classification [179] to prepare real and synthetic datasets for predicting stock price direction;
    - *Regime changes*: this thesis will focus on the prediction under structural breaks. An example of changes at low frequencies can be the end of a financial bubble or a financial crisis. However, changes can also occur at the intraday level due to any change in the behaviour of the investors.

Online machine learning and computational finance are fields that are closely related since financial time series can be interpreted as continuous data streams, especially at high frequencies (HFT) [90] with data arriving at fractions of a second. In this context, RCs can be seen as shifts in the generative process of these series. We will parse the incoming streams of data following the relevant literature for financial data and machine learning [179]. From this point, incoming data will be considered a data stream. Our research will focus on the incrementalisation of different algorithms, on the online scheme for predicting using up to date models, dealing with changes in the generative process, and improving classification accuracy.

1.2 Motivation

In the last decade, the digitalisation of different industry sectors has accelerated the growth of information to be processed and stored. Laney [215] settled the base for a term *big data* in the early 2000's that now has become a norm in most data-driven companies altogether with the usage of distributed computing techniques [75]. In data matured organisations [94], this evolution is transforming business analysis processes with automated data pipelines and AI models to support decision-making processes. Despite this fact, the industry continues relying on batch techniques for the application of AI as the de-facto standard. Even in continuous scenarios where sequential deep learning models are used, there is a frequent need for retraining strategies at some point in time [193, 292, 303]. Most of these techniques are, in general, unable to deal efficiently with data updates, as well as its evolution in stationary or non-stationary domains where a hidden context [367] may influence the predictive model behaviour over time in unforeseen ways. Moreover, many of these techniques are not scalable for a continuous learning setup [328].

The financial domain is characterised by data intensity, noise, unstructured nature, a high degree of uncertainty, and hidden relationships [184]. Financial markets are an evolutionary and nonlinear dynamical complex system [2, 329]. Albeit the standard in the financial domain is to make forecasts using traditional statistical methods, these tend to assume that the underlying data has been created by a linear process [79]. Another line of work to make financial predictions is to use machine learning. These algorithms have surprised financial experts [179, 329] because of their success mapping nonlinear relationships without prior knowledge [29]. For example, deep learning algorithms (neural networks) and ensembles have been some of the techniques obtaining the best results for stock trend prediction [36, 58, 212, 271, 275].

Recently, the problem of concept drift [350] has gained importance in computational finance [329]. Different crises, recessions and bubbles, such as the COVID-19 pandemic, or pumps and dumps in crypto markets [25], have stressed the non-stationary nature and the presence of drastic structural changes in financial markets. During these periods, mean returns, volatilities and correlations in assets tend to change in short periods [19]. Thus, many recent research works point out that financial assets or companies present different states that may repeat or not overtime or evolve due to inflation, deflation, or changes in supply and demand [104, 162, 182, 259, 319, 347, 360].

Online incremental machine learning algorithms are scalable for continuous learning scenarios and able to deal with non-stationarities, shifts, and drifts in the data [124]. However, stationary scenarios in machine learning for data streams (namely recurring concepts) are still a subject of study [7, 156, 159, 329]. Even in scenarios where models previously trained may become relevant again, most of the current algorithms need to relearn previous instances as these are forgotten due to the stability-plasticity dilemma [251]. This need for retraining results in a waste of computational resources, longer training times, and more significant prediction errors while the model is not up-to-date. Some authors of the literature on machine learning for data streams have started to consider stationary scenarios in their algorithms [118, 144, 151, 290, 365]. However, in the financial field, there is a small number of research works considering recurring concept drifts [286, 287, 329].

In finance, a change in the collective behaviour of market participants and their reactions is called a regime change. As covered by the marked efficiency hypothesis [128], we cannot observe the individual behaviour of a trader or its intentions. Instead, we can only observe changes in the price dynamics and macro or microeconomic variables and extrapolate the changes that make them modify their behaviour. The execution of these strategies is the actual generative process of these time series. The estimation of the hidden processes driving the market into different regimes is often approached using regime-switching models, a type of time series model where parameters can have different values in different cycles [282]. Hamilton [167] proposed a regime-switching model to measure these shifts in economic variables. These models are able to capture fat tails, skewness, and time-varying correlations during different periods [19, 347]. However, they are not suitable for online scenarios since the number of regimes needs to be defined in advance.

Despite the fact that artificial intelligence has recently become a trend and even a buzzword in many industries, this has not become the main trend yet for trading systems. This is mainly to the hard explainability and complexity of these models [174], a must for stakeholders and decision-makers in this sector [63]. Instead, traders tend to identify directional changes in the market state using different popular indicators tailored according to their needs. Traditionally, the literature has used static methods to interpret patterns based on the meaning of these indicators and their historical correlation to future prices. However, this correlation may vary over time. Behavioural shifts of investors changing continuously with a hidden context can also be observed through the change in sell versus buy volumes, in differences between local minima and local maxima over time, and through different moving averages at different time frames depending on the granular detail observed (frequencies) at intraday, daily or weekly levels. Changes in financial markets challenge traders and investors since most of their models rely on previous patterns. Hence, a way to recognise these changes is handy as a competitive advantage since it allows to change trading strategies ahead of other investors [209]. Detecting concept shifts also helps lower the risks of financial exposure in HFT.

In this thesis, we try to bring to the academic community's attention how machine learning for data streams techniques can help in terms of scalability and liaise with changes in regimes that apply to financial markets and other domains. Our goal is to leverage the benefits of modern machine learning algorithms that work in continuous scenarios and deal in real-time with any changes that may arise. For this purpose, we will explore the incrementalisation of machine learning algorithms, strategies to keep the model up to date, and concept drift detection techniques. Finally, we will apply our proposal to predict stock price movements. We expect to find strategies to improve prediction accuracy during times of change, avoiding model retraining. Hence, we aim to efficiently increase the potential profits of any organisation using these algorithms, avoiding the computational burden of model retraining and benefiting from always having an up-to-date model.

1.3 Hypotheses

This research work has been carried out under a set of assumptions listed below:

- Traditional static machine learning techniques do not scale and thus are not suitable for high frequency and non-stationary data that needs models to be up to date with the latest trends.
- It is possible to measure and typify market states (regimes) in intraday financial data. It is also possible to model these states and simulate a scenario where the ground truth is known.
- Adaptive and incremental machine learning algorithms allow learning in near-real-time. Thus, predictions can be made with a model trained with the most recent data, avoiding extra computational costs or bottlenecks.

- The use of adaptive techniques will improve prediction accuracy, especially during concept drifts or changes in the underlying high-frequency data.

1.4 Objectives

This section describes the different goals of this thesis. The main research objectives of this work are listed below:

1. Development of new machine learning algorithms to improve the state of the art regarding forecasting of price trend in financial markets. This development should be driven by the combination of the study of the state of the art and the experimental work performed in this thesis.

2. Modification of techniques from the relevant literature to predict structural change in high-frequency data. Different techniques based on online incremental machine learning for data streams are reviewed for this purpose. This objective can be subdivided into the next three:

    - Detection of RCs through concept drift detection techniques.
    - Once these structural breaks are detected, our next objective is to find the structural patterns in the market state dynamics to detect recurrences. Our research's primer objective is generating specialised models for scenarios, such as more volatile markets, or for up or downtrends may be more effective.
    - Our final goal in this regard is to reuse effectively previously trained models when a recurring market state is detected.

3. Adaption of algorithms and creation of methodologies for the identification of market states, which involves modelling and simulation of these high-frequency time series to adapt and react to structural changes.

4. Application of the techniques proposed to the financial domain in the intraday market and prediction using high-frequency price series.

1.5 Methodology

To achieve the objectives mentioned in the last section, we have performed the following research methodology.

1. Initial review of the state of the art and relevant literature. The goal of this is to find a problem to be addressed in this thesis and, thus, propose a solution or improvements for it. This involves the review of financial and machine learning literature to address high-frequency data and learning at scale.

2. Introduction of a method to solve the initial working hypotheses. This comprises the development of new machine learning algorithms for large scale and high-frequency data and their documentation to enable reproducibility by the academic community.

3. Exhaustive evaluation of algorithms and tools created during the development stage, and comparison of such algorithms with other relevant techniques from the literature to benchmark our proposal.

4. Validation of the achievement of the research objective mentioned in the previous section and of the aforementioned working hypotheses.

5. Exhaustive analysis of the results, concluding about this work contributions and introduction of future research in this area.

1.6 Contribution

This work has contributed in the following way to the scientific community:

- Improvement of the state of the art with new online incremental machine learning algorithms.

- Enrichment of the state of the art with a benchmark of techniques of the state of the art of online machine learning for data streams and application to high-frequency data to the financial domain. Learning and study of how some of these techniques perform in these scenarios.

- Research bridge created between the literature of RCs in time series and concept drifting data streams.
- Introduction of a methodology to model and simulate financial time series.
- Introduction of a framework to measure adaption of a model regarding the structural change in financial time series once the ground truth changes are known.

We expect these contributions and, thus, this thesis to positively impact future research in both machine learning and the financial domain. Since we hope this work brings different fields from the literature together, we believe this will be the start of a new area in the application of concept drift related literature to financial markets.

1.7 Structure of the Document

The rest of this document is structured as follows:

- *Chapter 2* describes the theoretical background that is the base of this thesis, including the basics of online incremental learning for data streams, concept drifts, and an introduction to structural changes in the financial domain.
- *Chapter 3* reviews the relevant state of the art in the relevant research topics. It covers relevant works on concept drift detection, supervised and non supervised learning on data streams, and related research works in the financial domain being our research context. This chapter approaches the research area of regime changes beyond time series approaches and aims to connect it with the data stream mining literature. So far, we are not aware of any state-of-the-art report or survey bridging these two fields; hence, this section provides a comprehensive review.
- *Chapter 4* covers preliminary studies performed in this work before our main proposal. Our first approach, iGNGSVM, is an incremental algorithm for classification tasks in large scale data streams. The second approach, RCARF, deals actively with concept drifting scenarios and is applied to real-world financial data to predict ups and downs in stock prices.

- *Chapter 5* describes and validates our proposal, namely GroCH, over synthetic datasets where ground truth changes are known. This chapter describes a framework to model and simulate synthetic sets that behave as high-frequency financial data. In such a framework, we control structural changes and drifts. At the end of this chapter, we benchmark our proposal over its main competitors in the relevant literature.

- *Chapter 6* applies our main proposal to a real-world scenario. GroCH is used in real-world financial data to predict ups and downs in market prices of different periods and frequencies.

- *Chapter 7* provides the conclusions of our work and the state of the working hypotheses. After the end of the body of the thesis, we include several appendices that give extra information and results obtained during our research.

- *Appendix A* describes the parameter exploration, the algorithm design process in our main proposal, and experimental results in more detail.

- *Appendix B* defines the synthetic data streams used in our proposal.

- *Appendix C* describes the hardware used for research tasks during this thesis.

- *Glossary* is a space where we include a compilation of definitions, explanations of terms and expressions about topics covered in this document, and a brief description of these sorted alphabetically.

- *Acronyms* lists all acronyms and abbreviations used in this thesis.

- *Bibliography* is a section that includes all research works cited in this document.

Chapter 2

Theoretical Background

This chapter serves as an introduction to the theory necessary to understand the work performed in this thesis. First, in Section 2.1, we will cover the field of forecasting in financial applications. We will describe problems and challenges in the area and explain the task of stock market prediction in detail. This section explains different ways to approach this depending on the data available and the model chosen.

Section 2.2 will describe machine learning as a subfield of AI and provide an overview of supervised algorithms and unsupervised algorithms as well as distance metrics used in this study. This section will also discuss performance metrics in this field and different evaluation schemes.

Section 2.3 will provide an overview of the subfield of online incremental machine learning for data streams. The concept of data stream will be explained, as well as the problem of concept drift and types of drifts. Different mechanisms (active and passive) and model reuse to deal with shifts and drifts will be reviewed. Finally, we will explain many of the base algorithms used in this thesis, analyse the computational costs associated with reusing models in continuous learning scenarios, and list relevant software for online machine learning tasks.

Finally, Section 2.5 provides a summary of this chapter.

2.1 Forecasting in Financial Applications

2.1.1 Overview

Early studies from the financial literature claim that financial markets are efficient [128] and, as a result, that asset prices follow a random walk [238]. This research, which claims that financial markets cannot be consistently beaten on a risk-adjusted basis and that their prices cannot be anticipated, has always been a source of controversy in the literature. Many research works have pointed to different markets being predictable using different sources of information [30, 83, 146, 179, 229, 239].

Forecasting in the financial domain can be characterised by a non-stationary and unstructured nature and by hidden relationships [110, 184]. Economic, social and political factors within countries and international impact and add uncertainty to financial markets [4, 26, 53, 105, 117, 223, 291]. Hence, markets can be considered an evolutionary and non-linear complex system [2]. The financial literature has covered different approaches to predict market prices using statistical and, more recently AI-based methods. This section will focus on the first group as part of the domain-specific background of our work.

In recent years, different events like the COVID-19 pandemic or the bankruptcy of Lehman Brothers in 2008 have led to periods with changes in mean, volatilities and correlations in stock market returns [19], stressing the non-stationary nature and the existence of drastic structural changes in financial markets [104, 182, 259, 319, 360].

In the financial literature, changes in the price behaviour of financial markets that go beyond their normal price fluctuations receive the name of regime changes [106, 168, 347] or business cycles shifts [54]. We will cover them in Subsection 2.1.3. In order to model these regime changes, one of the most popular techniques is the regime switching model [19], which was first applied by Hamilton [166] as a technique to deal with cycles of different economic activity such as recessions and market expansions.

This thesis will assume that, as the latest research suggests, some markets can be modelled and forecasted using different statistical and AI-based methods. We will leverage previous research where regime changes are the result of the shifts in the behaviour of the traders, and that can be observed indirectly from market price returns and movements [347]. This work will focus on the prediction of ups and downs in stock market prices at high frequencies (minutes and seconds). Since market movements are non-stationary, we will use online incremental machine learning techniques that adapt to any changes in the data distribution. Adapting to the new market behaviour after structural breaks will allow predictive models to diminish any decrease in predictive accuracy during or after these changes in the market dynamics. Finally, our main proposal will reuse older models if a previous regime reoccurs to deal with regime changes. In Subsection 2.1.4.2 we will explain some of the time-series-based techniques used to model and simulate regime changes in synthetic datasets used later in this thesis.

2.1.2 Market Efficiency

As introduced in the previous section, the early financial literature claims that financial markets follow a random walk [238]. This random walk means that changes in stock prices are independent of each other, and thus the market is an stochastic process, not predictable with previous information. Another related early assertion in this field was that stock markets are efficient, made by the economist Eugene Fama in 1965 [127]. The *efficient market hypothesis* (EMH) maintains all stocks trade at their fair value because all available information about the market is already incorporated into its prices. The hypothesis, revisited by Fama in 1970 [128], considers three degrees of efficiency.

- *Strong efficiency*: this states that all information of stock prices is accounted for (both public and private information). In this case, there would not be under or overvalued assets, and the market would not be beatable on a risk-adjusted basis.
- *Semi-strong efficiency*: this states that all public information is factored into stock prices. In this scenario, the only reliable strategy with be to use private information.
- *Weak efficiency*: this states that stock prices reflect all past prices. Under the weak form, investors might obtain excess returns exploiting new information (public and private) different to prices, such as news or fundamentals [55,339,378]. This will be discussed in further detail in Subsection 2.1.4.

Conversely, in an *inefficient market*, assets may not reflect their true value. This may occur for different reasons [54, 315]. The EMH has been challenged for a number of reasons. It assumes that all investors perceive and react to the market similarly. The idea of efficient markets expects all investors to have the same knowledge, background and tools to analyse the markets. It also speculates with them to value assets similarly and in parallel. In other words, it assumes that all investors share their judgment, investment goals and targets at the same point in time. An example of this failing in the real world are discrepancies between retail investment influencers in different social media channels [194, 378]. For instance, in 2021, there was a boom in cryptocurrency retail investment. Once passed the Bitcoin price all-time high in the first half of the year, different channels and newspapers started forecasting opposite directions for its price in the near future [213].

Financial markets can behave efficiently in many cases despite the inefficient behaviour of investors and firms due to market adjustments [323]. Nevertheless, the latent repetition of events as market crashes and bubbles in the financial history reveals a degree of inefficiency. In any case, in the rest of this thesis, we will not hypothesise about the degree of efficiency of any market. We will rather leverage recent research showing that certain assets at specific points in time can be predicted using previous prices [28, 30, 83, 146, 179, 179, 229, 239].

2.1.3 Regime Changes

In financial markets, there are periods of time with different degrees of efficiency and predictability. There can be moments where, due to the market-wide sentiment given by political or economic circumstances, the behaviour of investors may change towards a bear, bull lateral market, and periods or time frames with different levels of volatility [54]. As anticipated in the overview of this section, changes in the price behaviour of financial markets that go beyond their normal price fluctuations receive the name of regime changes (RC) in the financial literature [106, 168, 347]. At the macroeconomic level, RC are often related to abrupt breaks in long-term cycles like the break of bubbles or economic crises [167]. Changes of market regimes could be driven as well by investor expectations [19]. The financial literature identifies two types of regimes easy to recognise: steady and highly volatile regimes usually linked to economic growth or deflation periods, respectively. Figure 2.1, by Tsang and Chen [347], illustrates an example of these breaks during the Great Recession.

Regime changes challenge investors, making them change their trading strategies as the collective trading behaviour of the market changes. Different examples of RC have been covered in the recent literature. Davies in [106] analysed different cases and consequences of regime changes in the Great Recession that impacted several asset classes such as equities, bonds, commodities and currencies at micro and macroeconomic levels. Hamilton in [168] observed alternating patterns between steady and turbulent periods since the Second World War and subsequent recessions by looking at US unemployment rates. Ang and Timmerman in [19] identified cyclic changes in the behaviour of asset prices and mean, volatility and correlation patterns in stock returns during the Great Recession and the 1973 oil crisis. Kritzman et al. [209] discovered that investors could benefit from having different asset allocation strategies in different market regimes to minimise losses.

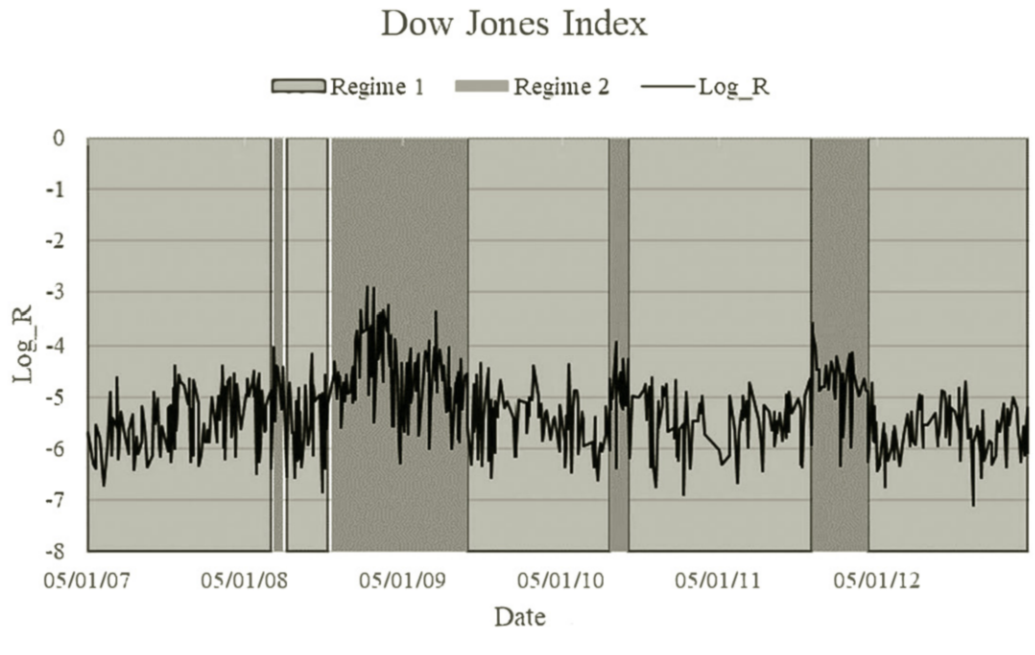


Figure 2.1: *Regime Changes in the DJIA Index (indicator of the United States economy) based on daily logarithmic returns. Extracted from [347].*

Many other studies consider these drastic changes an intrinsic characteristic of financial data that might be caused by significant events, and thus, these will be observable not only in prices and economic variables but also in other kinds of public information [18, 102, 145, 166–168]. Hamilton proposed in [166] a time-series based approach [282] to capture non-linear effects like RC, identify market breaks and hidden changes in economic cycles known as the *regime switching model* [167]. This model, also known as the Markov-switching model, is fitted to observations following different patterns in different periods and is mainly applied to recognise low volatility regimes with economic growth vs high volatility periods with economic contractions [21]. Ang and Timmermann in [19] applied these models to predict interest rates and equity and foreign exchange returns. They discussed how to model RCs for these time series models. This thesis is not focused on traditional time series models, and thus these approaches will not be part of the scope.

We propose to deal with regime changes as shifts in the data generative processes, also known as concept drifts, which will be explained in Subsection 2.3. From a financial point of view, these shifts could still be due to changes in trading behaviour among the investors, causing trained models to underperform in terms of predictive accuracy. Techniques to model raw market data for prediction in the financial domain will be explained in Subsection 2.1.4.

Finally, this thesis will use time series modelling approaches to characterise different generative processes. Transitions between generative processes are based on the work from Shaker and Hüllermeier in [312], which will be covered briefly in Section 3. With this in mind, time series and other forecasting techniques used to simulate synthetic raw data will be introduced in Subsection 2.1.4.2. This simulation will allow us to have a controlled scenario as ground truth and build a mechanism to deal with structural breaks in the financial domain, enabling our approaches to react rapidly and minimise the risk of misclassifying during changes, which maps to financial losses in its application area.

2.1.4 Stock Market Prediction

2.1.4.1 Fundamental and Technical Analysis

In the previous two sections, we covered the EMH and the concept of RC in the financial domain. Assuming that financial markets can be predicted in different periods, there are two different approaches to predict stock market prices and trends; fundamental and technical analysis [43]. Nti et al. in [269] provided a systematic review of predictions using either or both of these approaches. Other surveys bridging the literature using either fundamental or technical analysis and providing future research directions are [29, 79, 179].

- *Fundamental analysis* studies economic factors and public information that may impact the market, such as revenue, income statements, assets, sales, liabilities and dividends of an organisation and related news to predict future prices [55,339, 378]. Its underlying idea, more suited for long-term investment, is that investors should learn fundamental aspects of a company, like its capitalisation and its number of employees [357], before investing in it. For this, analysts and investors use quantitative tools such as marketing strategies, product innovation, financial ratios and other factors like market trends, changes in the legislation, financial news and social networks [214].
- *Technical analysis* relies on the extraction of patterns from previous and current performance of stocks to forecast the future price trend direction [79]. Technical analysis practitioners argue that these trends are caused by the difference between stock supply and demand [62]. The idea of technical analysis is to understand the behaviour of other participants in the market using mathematical formulas called indicators to track price and volume change patterns [338]. Technical indicators can be seen as snapshots of the market price over a specific time window. Murphy in [260] presents a typical list of these indicators, including relative strength index, moving averages, on balance volumes, directional movement indicators, momentum and rate of change.

Technical analysis ignores the fact that fundamentals may also influence market movements separately [128]. Nevertheless, several research works claim that movements in market prices are not random [230] and that trading strategies relying on technical indicators can produce excess returns [70]. Technical indicators are widely used for short-term prediction of the stock market by researchers, investors, financial economists, and brokerage firms [269, 297, 336] since these simplify the problem of predicting market movements to a mere pattern recognition problem [338].

In this thesis, we do not aim to propose new measures or indicators to predict financial markets. Due to their widespread usage in the literature, we will focus on technical indicators to predict movements at high frequencies in the very short term. One of our goals is to use online incremental machine learning to adapt to any changes from any private or fundamental public information not explicitly present as a feature in our datasets. The pre-processing steps and indicators that we will use will be described in Section 3.6.

2.1.4.2 Traditional Time Series Techniques

In mathematics, a time series is a sequence of numerical observations gathered sequentially in time [71, 348]. Each of these observations is usually called *lag* feature. In the traditional financial literature, data for different use cases tend to be modelled in this way to feed predictive models [28, 79]. Some example applications are asset closing prices such as different stocks, interest rates, or unemployment levels at different frequencies over time [122]. Time series allow the identification and extraction of trends, cycles, outliers, and seasonalities [97], use them for producing trading signals buying and maximise then investment profits [260]. There is also a trend in the financial literature to use time series as a framework to show dramatic breaks in market behaviours [167, 347].

The problem of predicting data with temporal dependence can be formulated as $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, with $\mathbf{x}_i \in X$ being an observation (stock price or return at a given time) and $y_i \in \mathbb{R}$ being the predicted value [375]. Models are then fitted with past data (X, y) [338] based on the belief that the history tends to repeat itself [260, 338]. In a time series though, the target feature y_i is predicted from its previous values $(y_1, y_2, \ldots, y_i)$. Time series are considered stationary if their statistical properties do not change over time. Hence, $(y_1, y_2, \ldots, y_i)$ is stationary if y_i does not depend on i, or if patterns are repeated across time. Trends or seasonalities in time series are non-stationary as these affect the predicted feature y at different points in time. This tends to be the case even if these trends or seasonalities occur during cycles, as in real-world datasets, cycles are not of a fixed length. Figure 2.2, extracted from the book of Hyndman and Athanasopoulos [187], illustrates examples of non-stationary versus stationary time series. Subfigures a, c, e, f and i (this last with increasing variance over time) illustrate non-stationary trends. b and g are examples of stationary time series according to the authors [187]. d, h and i are examples of time series with seasonality.

The predictive performance of these time series models depends on the level of (auto) correlation of the target feature to its previous values. Hence, for this reason, many applications in this domain use auto-regressive models, where earlier values of y or different technical indicators are used as feature sets [179]. Two of the most popular time series forecasting methods are *autoregressive integrated moving average* (ARIMA) and *generalised autoregressive conditional heteroskedasticity* (GARCH). The books of Eatwell et al. in [122] and Hyndman in [187] describe time series techniques and models such as ARIMA and other of the autoregressive models usually applied to finance.

Andersen et al., in their book [13] provided an in-depth description of GARCH models and variants and other time series regression techniques used to forecast stock market volatilities. Atsalakis in [28] provided a literature review of these and other derived time-series based forecasting applied to stock markets. However, predicting volatility of financial time series or the conditional mean of a process due to volatility are different challenges [13]. Many research works applied to econometrics have previously [176,335] claimed that financial time series could be modelled by the combination of autoregressive and GARCH methods. One of these combined methods, namely ARMA-GARCH, will be used later in this thesis to model different market states.

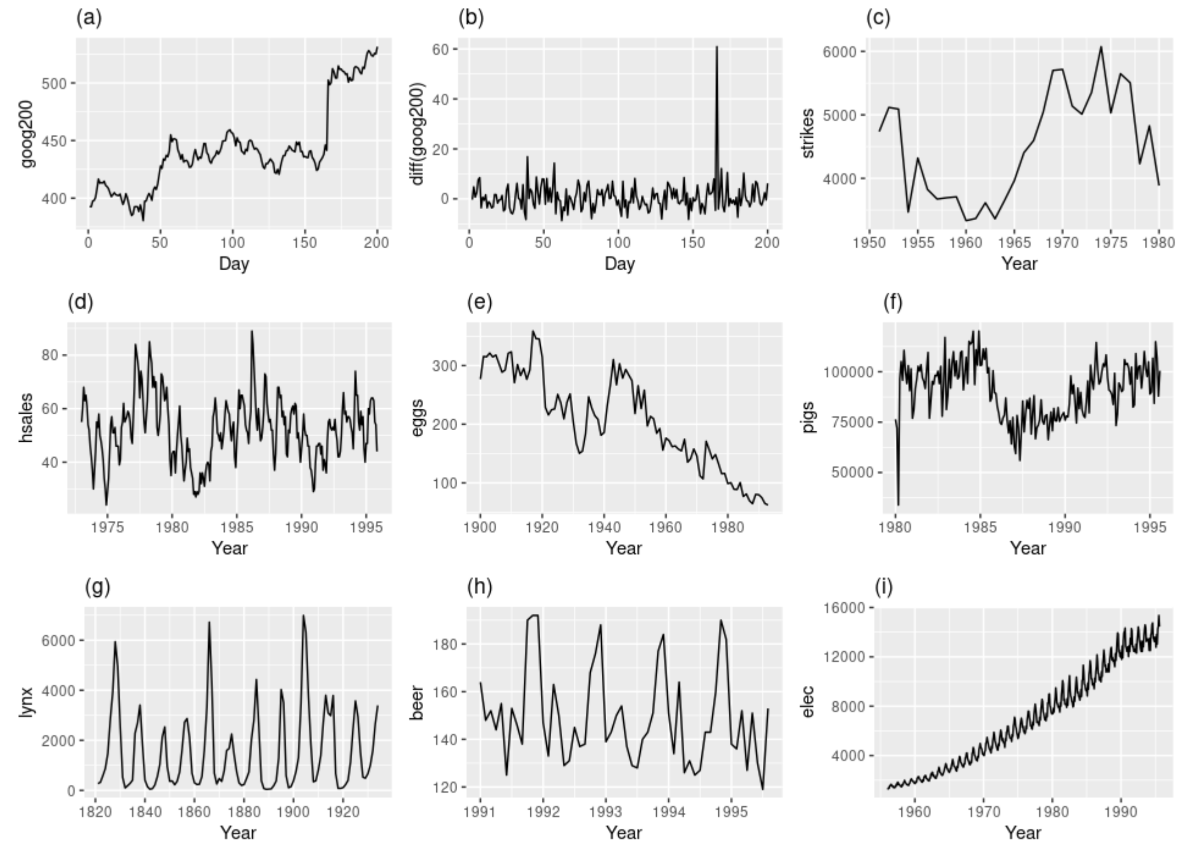


Figure 2.2: *Example of nine different time series extracted from [187].*

Autoregressive moving average (ARMA) models were initially proposed for univariate forecasting on stationary time series [61]. The ARIMA model, previously mentioned, is a generalisation of ARMA that adds and integration component. ARMA models are the combination of two well known approaches for time series prediction called autoregressive and moving average models.

Autoregresive models (AR), as their name indicates, predict the target variable y_i of the time series through a linear combination (regression) of its past values $(y_1, y_2, \ldots, y_i)$. An autoregressive model of order p, hence an AR(p) model, can be written as:

$$y_i = c + \phi_1 y_{i-1} + \phi_2 y_{i-2} + \cdots + \phi_p y_{i-p} + \epsilon_i \tag{2-1}$$

where i represents the current time step and ϵ_i white noise. The model fits values of ϕ depending on the correlation of previous time steps to y_i. For instance, values of ϕ close to 0 and 1 tend to generate white noise or random walks, respectively, as a result [187].

The *moving average model* (MA) is another regression-like method usually used for time series, but without observed (previous) values of the target feature y. A moving average model of order q, hence an MA(q) model, can be written as:

$$y_i = c + \epsilon_i + \theta_1 \epsilon_{i-1} + \theta_2 \epsilon_{i-2} + \cdots + \theta_q \epsilon_{i-q} \tag{2-2}$$

Besides the good results obtained with autoregressive in financial use cases, these methods only model the mean of a time series, being thus challenged over time in the presence of periods exhibiting different levels of volatility. These divergences in the behaviour of the time series may cause a non-constant variance of errors across observations in autoregressive models, which can be defined as *heteroskedasticity*. Engle in [126] demonstrated how to model mean and variance in a time series at the same time by proposing the autoregressive conditional heteroskedastic (ARCH) model. This approach was initially proposed to validate the conjecture of Friedman [133] about the unpredictability of inflation due to business cycles and its impact on the investor's behaviour due to uncertainty [125].

ARCH was applied to financial modelling soon after its proposal due to the need of many of its applications to predict volatility [57]. ARCH was expanded by Bollerslev and, separately, by Taylor in 1986 [56, 337]. They proposed a generalised version of ARCH (GARCH), which allowed the expectation of conditional variance to be modelled with ARMA [176]. In econometrics, the term *conditional variance* refers to the potential variance of a random variable given the values of its correlated features. The most popular GARCH model in econometrics is GARCH$(1, 1)$ model, that is, $p = q = 1$ [13], where p and q are the orders of σ^2 (time-dependent standard deviation) and ϵ^2 (white noise) respectively. A GARCH process, following the notation of the original paper except for i and k (for consistency with previous equations), referred to as t and i respectively in the original paper [56], is given by the equations that follow:

$$y_i = x_i' b + \epsilon_i \tag{2-3}$$

$$\epsilon_i \mid \psi_{i-1} \sim \mathcal{N}\left(0, \sigma_i^2\right) \tag{2-4}$$

$$\sigma_i^2 = \omega + \alpha_1 \epsilon_{i-1}^2 + \cdots + \alpha_q \epsilon_{i-q}^2 + \beta_1 \sigma_{i-1}^2 + \cdots + \beta_p \sigma_{i-p}^2 = \omega + \sum_{k=1}^{q} \alpha_k \epsilon_{i-k}^2 + \sum_{k=1}^{p} \beta_k \sigma_{i-k}^2 \tag{2-5}$$

where ω, α and β represent the coefficients of the GARCH process. A deeper mathematical explanation can be found in Andersen's Handbook [13].

The autocorrelations of certain asset returns are incompatible with GARCH models [13]. While the relevant literature has demonstrated the good results of GARCH model predicting volatility [12], some research works claim that it obtains a suboptimal forecasting performance [64, 98, 130, 191, 192, 245]. With this in mind, many researchers have mixed the goodnesses of GARCH forecasting volatilities with the good performance of autoregressive and moving average models predicting mean returns by combining them as AR-GARCH and ARMA-GARCH models. Wong et al. in [369], and later Tang et al. in [335] presented the mixture models AR-GARCH and ARMA-GARCH, for exchange rates and stock price prediction, respectively. These mixture models improved GARCH in financial prediction. ARMA-GARCH, as an improvement of AR-GARCH, further enhanced its forecasting results. According to Atsalakis in his literature review of statistical approaches to predict stock market movements in [28], GARCH based models outperform in many cases other forecasting techniques such as Autoregressive or ARIMA models.

As explained earlier in this chapter, this thesis will use ARMA-GARCH processes to model different market states with different mean returns and different levels of volatility. The volatility aspect is crucial since this can be the differential factor between normal and abnormal regimes, as already covered in Subsection 2.1.3. Shifts between states will be modelled as a change in the generative process as further explained in Subsection 2.3 and Chapter 3.

2.2 Machine Learning

2.2.1 Overview

In Section 2.1 we introduced the domain of forecasting in the financial domain. We presented challenges, different indicators used for decision-making and classical forecasting algorithms used in econometrics. Most trading strategy systems in this domain are based on wired rules that rely on trader assumptions or traditional time series forecast models that are uni-modal as presented in Subsection 2.1.4.2. These traditional statistical methods tend to model and predict future data based on the assumption that the time series under study is generated from a linear process with features normally distributed [349]. This presents challenges since financial data is characterised by non-linearity and non-stationarity besides a high level of uncertainty and noise [357].

A different approach to perform financial forecasts is with the use of machine learning techniques. Several literature reviews show the benefits of these techniques against traditional methods [150,179,300,302], surprising practitioners by contradicting early theories like the random walk and EMH [128, 179, 346] described in Section 2.1. Machine learning algorithms can handle non-linear relationships without prior knowledge [29], outperforming traditional time series methods [36, 58, 212, 275]. Different research works from the economic literature have either used technical indicators or raw prices and returns. As covered in Section 2.1, technical indicators are able to show behavioural patterns among traders and thus provide an extra level of signal to predictive models. These can be valuable to automate the behaviour of short-term traders despite being an underestimated technique as well as the inefficiency of financial markets [179]. In any case, most of the economics literature has focused on linear processes that may not have been able to extract relevant information nor infer complex relationships among technical indicators where some new machine learning methods could [9].

Machine Learning is a branch of AI that finds patterns in previous data (training vectors) and fits an algorithm to predict future events. In this branch of AI, a data point or observation is called an instance or example. Data instances are represented by a set of variables called a feature set or set of attributes. Machine Learning algorithms can be traditionally classified into two types: unsupervised and supervised. supervised machine learning algorithms are the ones that use labelled data to learn. This label refers to the ground-truth value that the algorithm should target to reach a perfect forecast. Figure 2.1 illustrates an example of a labelled dataset.

x_1	x_2	x_3	x_4	x_5	y
3	36	22	23	12	uptrend
3.5	31	25	28	13	downtrend
8	32	21	22	27	downtrend
6.6	35	23	27	33	uptrend
4.5	36	25	24	40	downtrend

Table 2.1: *Example of labelled dataset; y represents the target (or predicted) feature and all the X variables represent the set of attributes (predicting features).*

Conversely, non-supervised machine learning algorithms do not precise a label for training tasks. Instead, instances are grouped by similarity into clusters . This similarity depends on different distance metrics. We review some of them in Section 2.2.2.2. Unsupervised and supervised algorithms are briefly covered in Section 2.2.2 and Section 2.2.3. The purpose of this is to provide the reader with the theoretical background and basic concepts about these different machine learning tasks and relevant methods that will be used in this thesis.

2.2.2 Unsupervised Learning

The objective of non-supervised machine learning techniques is to discover a set of similar instances within the datasets. Each of these sets is called a *cluster*. Some algorithms need to receive the number of clusters as an input parameter. In contrast, other algorithms can extrapolate the number of these. Unsupervised techniques can be categorised into different sub-types according to the literature reviewed by Nguyen et al. in [265], all present in Table 2.2, extracted from this research paper, illustrating various advantages and limitations of these clustering methods.

One of the oldest and most popular non-supervised machine learning algorithms is the partitioning algorithm K-means [237], described in Subsection 2.2.2.1. K-means will be used in Chapter 5 to visualise the theoretical limitations of the datasets used due to the simplicity of this method. In any case, the core proposal of this thesis will use model-based techniques to group different states of a data stream over time. These are explained and motivated in Subsection 2.3.6.

Type	Advantages	Limitations
Partitioning	Simple and relatively efficient. Terminate at a local optimum.	Need to specify the number of clusters
Hierarchical	Derive more meaningful cluster structures.	Unable to discover non-spherical clusters. High complexity
Density-based	Can find arbitrary-shape clusters. Robust to noises.	Sensitive to the order of the data records. Need many parameters: density and noise thresholds. Difficult to detect clusters with different densities.
Grid-based	Fast and can discover arbitrary-shape clusters. Robust to noises.	The clustering quality depends on the grid granularity. Unsuitable to high-dimensional data.
Model-based	Simple and can include domain knowledge.	Depends strongly on the assumed models.

Table 2.2: *Advantages and limitations of clustering approaches. Extracted from [265].*

Non-supervised machine learning algorithms measure the separation between instances and compute similarity using a distance function. In this regard, Euclidean distance and Mahalanobis distance distances, due to their simplicity and their ability to represent well data clusters , respectively, are some of the preferred metrics in the literature [21]. Both distance metrics will be described in Subsection 2.2.2.2

2.2.2.1 K-means

The k-means algorithm finds k clusters of instances x within a dataset X. It characterises each cluster with its centers, which are known as centroids. The algorithm, as described in [237], starts with a random set of k centroids (μ). Every learning iteration, each data instance x is assigned to their nearest centroid (see equation 2-6). The next step in the learning process is the repositioning of the centroids using the mean of the data instances assigned to them (the nearest ones) (see 2-7). This process iterates until all data instances are assigned to centroids. Figure 2.3 illustrates k-means clusters for $k = \{3, 5, 7\}$ in an example dataset with two dimensions (F_1, F_2). Centroids are marked in yellow.

$$S_i^{(t)} = \left\{x_p : \left|x_p - \mu_i^{(t)}\right|^2 \leq \left|x_p - \mu_j^{(t)}\right|^2 \ \forall j, 1 \leq j \leq k\right\} \tag{2-6}$$

$$\mu_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{x_j \in S_i^{(t)}} x_j \tag{2-7}$$

K-means tries to minimise the square loss, as seen in equations 2-8 and 2-7. A problem in k-means is that equation 2-8 tends to converge to a local minimum depending on the initialisation of the centroids. In this sense, a common approach is to run the algorithm several times and to consider correct those clusters that repeat among most executions.

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} ||x_n - \mu_k||^2 \tag{2-8}$$

$$\text{with } r_{nk} = \begin{cases} 1 & x_n \in S_k \\ 0 & \text{otherwise} \end{cases}$$

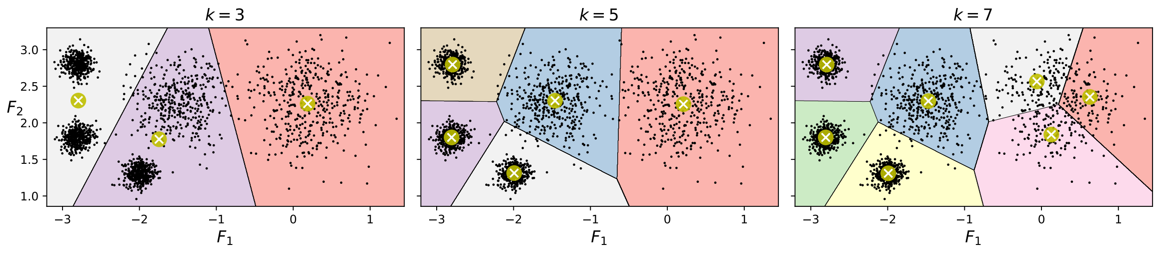


Figure 2.3: *An example of k-means clusters for different values of k and the same input data.*

2.2.2.2 Distance Metrics

2.2.2.2.1 Euclidean Distance One of the most commonly used distance metrics is the Euclidean distance distance due to its computational efficiency and simplicity. Distances between two data samples, x$_i$ and x$_j$ $\epsilon \{x\}_K$ are computed following equation 2-9.

$$d\left(x_i, x_j\right) = ||x_i - x_j|| = \sqrt{\sum_{l=1}^{N} \left(x_{i,l} - x_{j,l}\right)^2} \tag{2-9}$$

The Euclidean distance distance considers all attributes to be equally important [163]. This may create several challenges when dealing with multi-variate signals or indicators of different natures in financial data since attributes with larger scales will significantly impact the overall distance between instances. In order to tackle this, the feature set may need to be normalised to equalise the importance of different features before the computation of Euclidean distance distances. For this purpose, many researchers [21, 68, 347, 373] use Mahalanobis distance distances which normalise features using a correlation matrix.

2.2.2.2.2 Mahalanobis Distance The Mahalanobis distance distance considers the standard deviation of x_i and x_j through a covariate matrix $\boldsymbol{\Sigma}_K$ [21, 101].

$$d\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) = \sqrt{\left(\boldsymbol{x}_i - \boldsymbol{x}_j\right)^T \boldsymbol{\Sigma}_K^{-1}\left(\boldsymbol{x}_i - \boldsymbol{x}_j\right)} \tag{2-10}$$

The covariance matrix of $\{\boldsymbol{x}\}_K$ is represented by $\boldsymbol{\mu}_K = \frac{1}{K}\sum_{i=1}^{K}\boldsymbol{x}_i; \boldsymbol{\Sigma}_K$

$$\boldsymbol{\Sigma}_K = \frac{1}{K}\sum_{i=1}^{K}\left(\boldsymbol{x}_i - \boldsymbol{\mu}_K\right)\left(\boldsymbol{x}_i - \boldsymbol{\mu}_K\right)^T \tag{2-11}$$

The Mahalanobis distance distance can be seen as an extension of the Euclidean distance distance, where features are weights based on a covariance matrix that accounts for the variance of each feature and the correlation among features. This type of distance standardises the dataset to be uncorrelated and then computes Euclidean distance distances. Hence, the Mahalanobis distance distance can become Euclidean distance if the features are already uncorrelated ($\boldsymbol{\Sigma}_K = I$) [21, 68, 190, 373]. One drawback of the Mahalanobis distance distance is the high computational cost of computing the covariance matrix depending on the dataset used [359].

In the main proposal of this thesis we will use Mahalanobis distance distances to consider the correlation between different financial indicators when computing the similarity to look for recurring patterns in the data stream.

2.2.3 Supervised Learning Algorithms

2.2.3.1 Naive Bayes

Naive Bayes (NB) is a probabilistic classification model based on the Bayes theorem [42]. This classifier aims to describe the probability of an event (label) based on previous observations of it. The NB classifier is considered *naive* because it assumes that all attributes are independent for each class, which is called *class-conditional independence* [217]. Despite this strong assumption, NB is able to outperform other models from the relevant literature in many classification tasks [295]. The Bayes theorem is described by the following equation:

$$p\left(A|B\right) = \frac{p\left(B|A\right)p\left(A\right)}{p\left(B\right)} \tag{2-12}$$

A and B represent independent events, and p(A) and p(B) the probability of observing either the event A or B, respectively. p(A|B) is the conditional probability of A if B occurs. p(B|A) represents the opposite scenario.

In this thesis, the algorithm will calculate the probability of ups and downs in market prices. NB will classify an up or downtrend using the label with the most significant probability. In our main proposal, we will use an incremental version of the naive Bayes classifier trained continuously from the *massive online analysis* (MOA) [48].

2.2.3.2 Decision Trees

Decision trees are one of the most used classifiers in both offline and online machine learning. Their popularity is not only due to their good results but also to their degree of explainability. A decision tree is made of a set of nodes. The learning process commences by splitting the initial (root) node based on a separation metric like information gain or the Gini coefficient [67] over the attributes of the feature set used in the learning task. This process creates child nodes that may be divided further during the learning process. Final nodes (without children) are called leaf nodes, which lead to the algorithm's prediction. Figure 2.4 illustrates this process. The accuracy of this algorithm is enhanced in the literature by applying other algorithms such as naive Bayes in the leaf nodes [48] or using them as weak learners in ensemble models [66, 152], as will be covered in Chapter 3. This thesis will use online decision trees in the second experiment and the main proposal; Section 4.2 and Chapter 5. The theoretical background of these online models will be covered in Subsection 2.3.5.

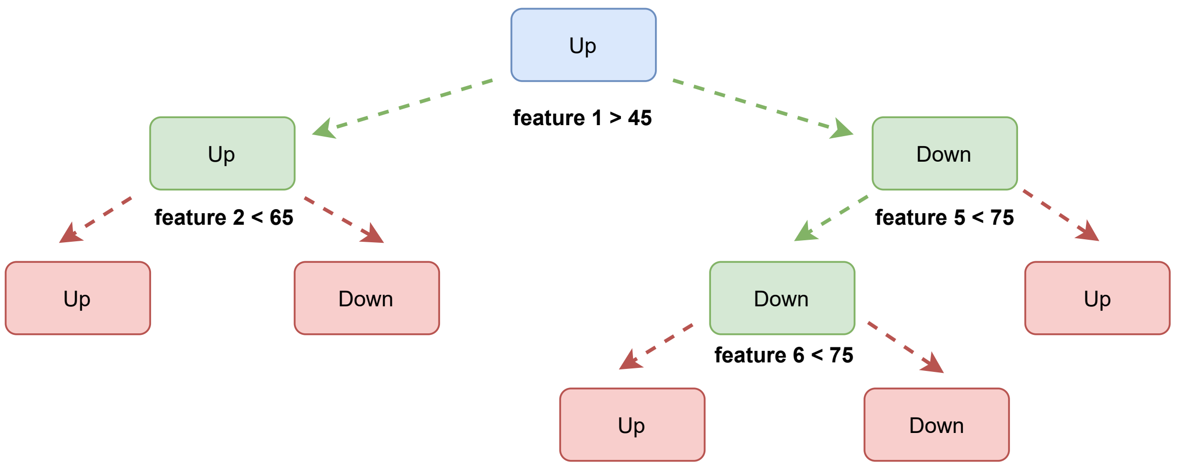


Figure 2.4: *Example decision tree for trend classification (up or down). Root node coloured in blue. Intermediate and leaf nodes are coloured in green and red, respectively.*

2.2.3.3 Ensembles

Ensembles are collections of classifiers such as the previously mentioned decision trees and NB. The motivation to do this is that an ensemble should be more robust and reduce its error by considering the prediction of a set of base models. The predictions of all base models are then combined. In this sense, a common classification combination approach is majority voting. In this case, the ensemble prediction is equal to the prediction of the majority of the classifiers. If there is a different degree of confidence in each of the classifiers, then votes can also be weighted based on the performance of the models on a validating set; this is called *weighted majority voting*.

The overall idea is that the ensemble predictions improve each of the base classifier predictions. For this purpose, its base models should offer different predictions. This is not the case if the algorithm is deterministic and it receives the same data. Thus, to promote diversity, different methods like bagging to train each base classifier in a different subset of data are used.

Bagging, proposed by Breiman [65], generates a set of k training sets using bootstrapping, which is a resampling method proposed by Efron in an early version of their work in [123]. Bootstrapping consists of randomly selecting numbers in a sequence with equal probability to choose each number until N numbers are chosen. In bagging, this method is applied over the random selection of instances in a training set. Then, each of the k training sets generated of length N is used to train a different base classifier. Bagging is generally used in models which outcome can vary largely with any small change in the training data (e.g. neural networks or decision trees).

Ensembles are one of the machine learning techniques obtaining the best results for trend classification in the financial domain [179]. An example of an ensemble technique widely used, and one of the best methods predicting stock market movements is the random forest algorithm [36, 58, 210, 212, 275, 276]. In the second experiment of this thesis, we will propose an ensemble approach based on an incremental version of the random forest (RF) algorithm [66]. Random forest, proposed as well by Breiman, is an ensemble of decision trees that promotes diversity combining the bagging method explained above with the selection of a random subset of features per base classifier. This random selection of features is a method known as *feature bagging* [72], and the part of the RF algorithm name can be attributed to it. The other part (forest) can be attributed to RF having a homogeneous set of tree-based classifiers. RF requires some adaptations to handle structural changes in an online setting. These are covered in Chapter 4.

2.2.3.4 Other Algorithms

2.2.3.4.1 Nearest Neighbours K-nearest neighbours (kNN) is a supervised algorithm that classifies an instance depending on the labels of the closes data examples. Traditionally, the closest neighbours are computed using the Euclidean distance distance described in Section 2.2.2.2. In this thesis, we use the prototype selection technique *Wilson's edited nearest neighbours* (ENN) for noise removal in the first experiment. This algorithm is based on the first (kNN), as it removes any data instance that does not match the label of the majority of its k neighbours. In the literature, traditionally, $k = \{3, 5, 7\}$. The steps followed by ENN are shown in algorithm 2.1. ENN is used during the first experiment in Section 4.1, and can be considered a prototype reduction technique; these methods will be described further in Subsection 2.2.4.

Algorithm 2.1 ENN algorithm according to [69]

```
1. Compute the $k$ nearest neighbours for every data example based on the provided input network topology.

2. For every data example, if most of its neighbours belong to a different class, this is labelled as noise.

3. Once finished with the neighbourhood comparisons for all the data examples, it deletes all the data examples
   labelled as noise.
```

2.2.3.4.2 Support Vector Machines Support vector machines (SVM) classify instances using a hyperplane as a separator that maximises the margin between classes [59, 96]. Although modern versions combine multiple separators, working for multiple classes, they were initially proposed for binary classification. This technique, will be used and covered in more detail in the first experiment in Section 4.1. Algorithm 2.2 describes a linear SVM model for binary classification.

Algorithm 2.2 Linear model for support vector machines according to [171] [120]

1. In a binary classification problem, we have $Y = \{A, B\}$ where Y is the set of both possible classes (class A and class B).
2. A set of vectors $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ where $xi \in R_d$ and $y_i \in A, B$ for $i = 1, \ldots, n$ is separable if there is an hyperplane in R_d able to separate the vectors $X = \{x_1, \ldots, x_n\}$ with class $y_i = A$ from those with class $y_i = B$.
3. Given a separable set, there will be at least an hyperplane $\Pi : w \cdot x + b = 0$ able to separate the vectors $X = x_1, \ldots, x_n$.
4. Once the separator is determined, it is adjusted to more vectors according to Equations Π_1 and Π_2 below.

$$x_i \cdot w + b \geqslant +1 \text{ for } y_i = A \text{ (Equation } \Pi_1)$$
$$x_i \cdot w + b \leq -1 \text{ for } y_i = B \text{ (Equation } \Pi_2)$$

   $+1$ and -1 are labels A and B respectively. w is a normal to the hyperplane.

5. The points for which the equality of equations Π_1 and Π_2 hold are called *support vectors* of its respective class.
6. Π_1 and Π_2 are parallel as they have the same normal (w) and there is no data examples between them. Thus the pair of hyperplanes that maximises the margin between classes can be obtained by minimising $|w|$.
7. The algorithm is formalised as: *minimise* $|w|$ *s.t.* $y_i(w \cdot xi + b) - 1 \geqslant 0, \forall x_i$

2.2.4 Prototype Reduction

This thesis will explore different mechanisms to reduce computational cost and make online learning tasks manageable in high-frequency data streams. We will explore model-based *prototype reduction* techniques, which can be used for different purposes like summarising or cleaning a dataset before machine learning operations. These techniques, which can be either supervised or non-supervised, are traditionally divided into two different approaches [113, 344, 345]: prototype selection (PS) and prototype generation (PG).

- *Prototype Selection* techniques select the data instances considered by a model to represent the initial distribution best. We introduce one of this methods, ENN, in Section 2.2.3.4, and we will use it in Section 4.1.

- *Prototype Generation* techniques create new synthetic data points (prototypes) to summarise the initial distribution. Many of these methods will be introduced in Subsection 2.3.6. In Chapters 5 and 6 we will use one of these methods, GNG, to summarise previous market behaviours.

In Section 4.1 we will use both PG and PS methods to summarise the initial data distribution and noise removal, respectively.

2.2.5 Performance Metrics

In this section, we describe the theoretical principle of the different performance metrics that will be used during the experiments. In order to report prediction performance, we will mainly use accuracy and kappa statistics since these are some of the preferred techniques in the literature of machine learning for data streams. Other metrics such as *f1 score*, *precision* and *recall* have not been considered necessary since the data used is balanced. Regarding the cost of the model, we will mainly report the RAM-hours and the total run time for testing and training (prequentially - as in Subsection 2.2.6.2) tasks.

2.2.5.1 Accuracy and Confusion Matrix

Confusion matrices summarise the success of classification tasks by presenting predictions against ground truth results per class or label. In confusion matrices, one of their axes is dedicated to the label predicted by the model, and the other to the ground truth class. In a confusion matrix, the number of data instances correctly identified positively is the number of true positives (TP). Conversely, true negatives (TN) are the number of negatives correctly classified. False positives (FP) and false negatives (FN) are the numbers of negatives and positives classified incorrectly as positives and negatives, respectively. Table 2.3 illustrates an example confusion matrix for binary classification.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{2-13}$$

$$\text{Error} = \frac{FP + FN}{TP + TN + FP + FN} \tag{2-14}$$

		Predicted	
		+	−
Actual value	+	TP	FN
	−	FP	TN

Table 2.3: *Example of confusion matrix for binary classification.*

The *accuracy* metric can be derived from the confusion matrix as the total number classifications out of all the predictions performed. This thesis will use this as one of the main metrics to report prediction performance in balanced datasets. Equations 2-13 and 2-14 show the definitions of accuracy and its complement, the error rate (1 - accuracy) respectively for binary classification. A more general definitions can be seen in Eqs. 2-15 and 2-16.

$$\text{Accuracy} = \frac{\text{Correct Predictions}}{\text{Total Number of Instances Predicted}} \tag{2-15}$$

$$\text{Error} = \frac{\text{Incorrect Predictions}}{\text{Total Number of Instances Predicted}} \tag{2-16}$$

2.2.5.2 Kappa Statistics

Although the datasets that will be used in this thesis have a balanced class distribution, data streams may present differences in the class balance over time. For this purpose, in the literature of online incremental machine learning for data streams, the convention is to use kappa statistics. In online settings, this metric is considered more appropriate than other metrics, like the area under the ROC curve, due to its computational efficiency [45].

Kappa statistics (κ) is a measure that expresses agreement between observations. This was initially introduced by Cohen [93] and considered the randomness of each class for rating the strength of a classifier. Since there are many definitions of this formula that are not equivalent, in this thesis, we will use the implementation offered as an error metric in the MOA framework [52].

$$\kappa = \frac{P_o - P_c}{1 - P_c} \tag{2-17}$$

κ is defined by equation 2-17. P_o is the observed overall agreement, in other words, the classifier's accuracy; P_c is the mean proportion of agreement expected; it represents the probability of predicting correctly by chance (randomly). $P_o - P_c$ represents the actual degree of agreement obtained. This is normalised by the maximum obtainable agreement $1 - P_c$. $\kappa = 1$ if the classifier always predicts correctly. However, if the classifiers' predictions are only correct on the same measure than what they should be randomly (e.g. due to class distribution), then $\kappa = 0$.

2.2.5.3 Computational Resources

An essential consideration of online incremental machine learning algorithms for data streams is that these are must be prepared to run continuously. In this regard, when designing an online machine learning algorithm, the user must bear in mind the run time of the learning and prediction process for a given data stream and be aware of any bottlenecks that may cause issues in a continuous setup. Hence, in the main proposal of this thesis, we will report the *time* in seconds consumed by the algorithms for learning and prediction tasks to process each data stream.

Another factor to consider is the pricing of computational resources over time. Some of these models nowadays may run on the cloud vendors where there is a cost associated per hour, and pricing in these platforms depends on the size of the machine rented [51]. Thus, there is a cost associated with running machine learning models more CPU, GPU, storage or RAM intensive. For this reason, *RAM-hours* (megabytes of RAM used per hour) has become an important evaluation measure of the resources used by streaming algorithms [51]. RAM-hours can be used to quantify if the cost of the changes is worth it [44]. In this thesis, RAM-hours will be compared to improvements in other evaluation measures such as the run-time, the overall accuracy and kappa statistics.

2.2.6 Evaluation Schemes

2.2.6.1 Holdout

Holdout is one of the typical evaluation schemes in traditional machine learning. However, the main idea of this scheme is that train and test are disjointed sets; data instances do not overlap between them [143]. Hence, training and test sets in a holdout evaluation are independent. The splitting of these sets can be done in different ways to adjust for learning data over time, for instance, through sliding windows where each train and test sets represent different time windows. Figure 2.5 illustrates a simple and sliding window holdout approaches.

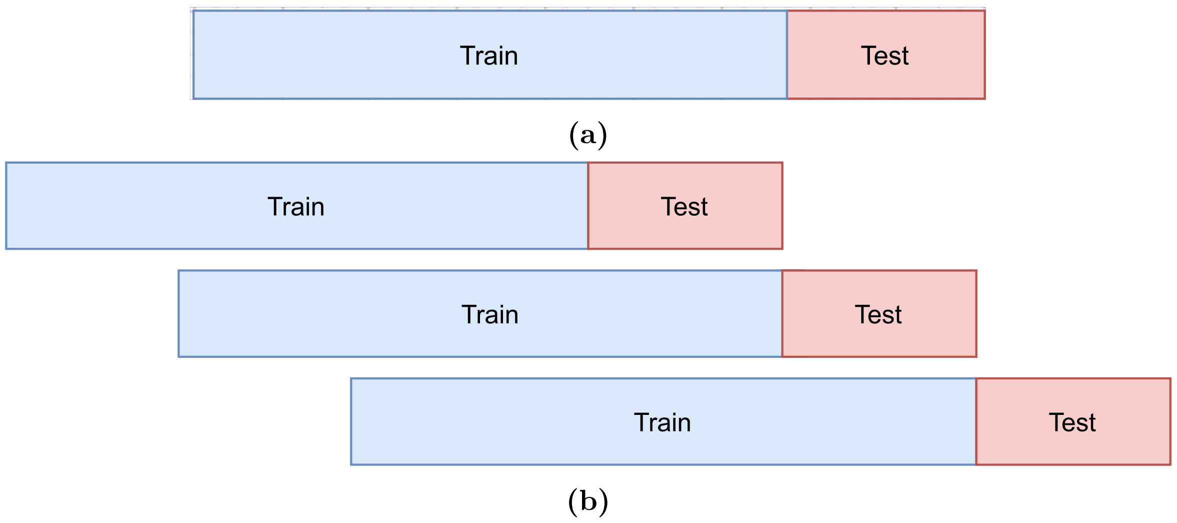


Figure 2.5: *Examples of holdout evaluation with disjointed train and test sets. SubFigure 2.5a refers to a single training batch. In subFigure 2.5b, a sliding window holdout approach is taken.*

One of the different problems associated with this scheme is that it reduces the amount of data available for training and testing tasks due to the splitting. Due to this split, in non-stationary data, a model can overfit the state of the data distribution in a given set of instances and not adapt well to out of sample instances. This split can also create different levels of skewness across the datasets used due to the continuous evolution and temporal feature and class imbalance over time in data streaming scenarios. In this case, cross-validation, where holdout is repeated for different splits, can be a workaround.

2.2.6.2 Prequential or Interleaved Test-then-train

Prequential or interleaved test-then-train evaluation has become popular in machine learning for data streams since it helps monitor the error of an algorithm over time [81, 139, 292, 330, 340].

At the start of the testing process, the model has been trained with few instances, impacting its predictive accuracy. In a prequential evaluation, data is continuously evaluated as soon as it is available, hence being valuable for online learning [81]. Each data instance is used first to predict. Then once the ground truth is known, this instance can be labelled, and the prediction error of the model can be computed. Finally, the model can be updated, using that instance for training. Figure 2.6 illustrates a prequential evaluation as opposed to the holdout shown in Figure 2.5a. The main difference between this approach and holdout is that test and train are not disjointed sets. Thus, the prequential evaluation uses instances more efficiently [81], and it is suitable for online scenarios and incremental machine learning algorithms which can be updated and adapt to new instances. The use of this evaluation scheme avoids the need of retraining the whole model from scratch in non-stationary scenarios [385]. For this reason, the prequential evaluation will be the main scheme used in this document.

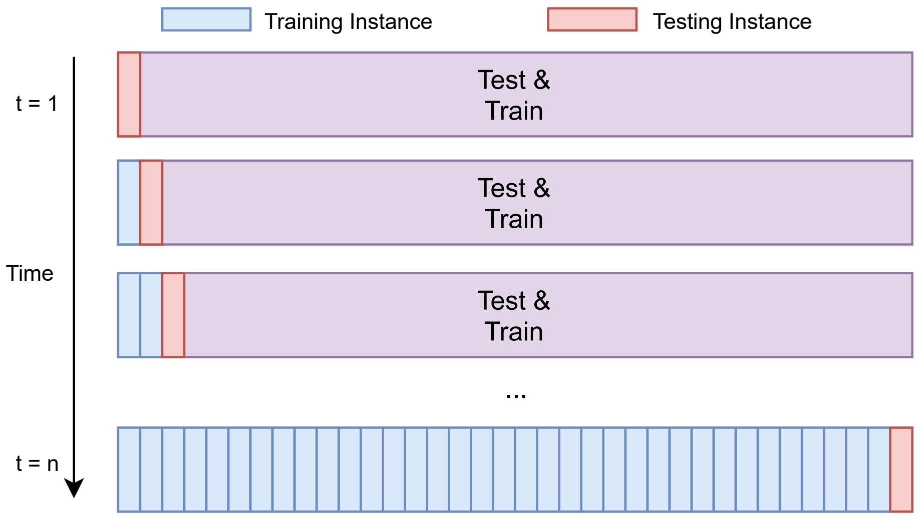


Figure 2.6: *Prequential scheme. Each instance is used first for test, and then to train.*

In a trading context, this continuous training process could run at high frequencies. When a model is put into production, the model may be empty; if a pre-trained model is loaded, it would not be fine-tuned. The performance of the model is expected to improve over time since it will be trained with more instances. When a new sample arrives, a trading system with a continuous learning approach would first predict and generate a trading signal straightway. Then, once a label for this data instance is available, this would be used for training the algorithm. Models under this training task are always up-to-date as data is fed continuously.

2.3 Machine Learning for Data Streams

2.3.1 Overview

Financial data can present different levels of uncertainty, non-linear behaviours, and statistical properties over time [357]. The level of correlation between the attribute space and a predicted feature, as well as the degree of volatilities exhibited by price returns, might present dynamic changes [180]. In the machine learning literature, these changes are known as the problem of concept drift [350]. As mentioned in Chapter 1, this problem has gained importance in the financial domain [104] in recent years as different crises like the Great Recession and the COVID-19 pandemic have stressed the non-stationary nature and the likelihood of drastic structural shifts in markets [104, 162, 182, 259, 319, 360]. These dynamic changes require algorithms to learn on the fly and "*not to have a separate one-off training phase followed by an exploitation phase*" [21]. Online incremental machine learning techniques are used in the academic literature to deal actively or passively [124] with concept drifts and the non-stationary nature of data from different domains [350]. These techniques, explained further in Section 2.3.2, are suitable to learn from continuous data streams of information.

In Section 2.1.4.2 we have discussed the non-stationary nature of financial time series. Despite this fact, the number of contributions using online incremental machine learning for data streams in the economic forecasting domain is still minimal [286, 287, 329]. Rather than online models that receive incremental updates over time, the current trend is to use offline (traditional) machine learning algorithms that need to be retrained at some point in time. These algorithms have a cost in terms of predictive accuracy when the model is not up-to-date and in terms of computational cost at the time of retraining with historical data or fine-tuning. Figure 2.7, extracted from [21], illustrates the above-mentioned differences between offline and online machine learning.

In this thesis, we propose the use of online learning in the stock forecasting domain. Most models presented are constantly updated with the latest labelled data and learn continuously, not needing any separate retraining phase. We explore different types of changes (concept drifts) described in Section 2.3.3, and techniques to deal with these that are described in Section 2.3.4. The online incremental machine learning algorithms covered in this thesis, both supervised and unsupervised, are described in Sections 2.3.5 and 2.3.6 respectively.

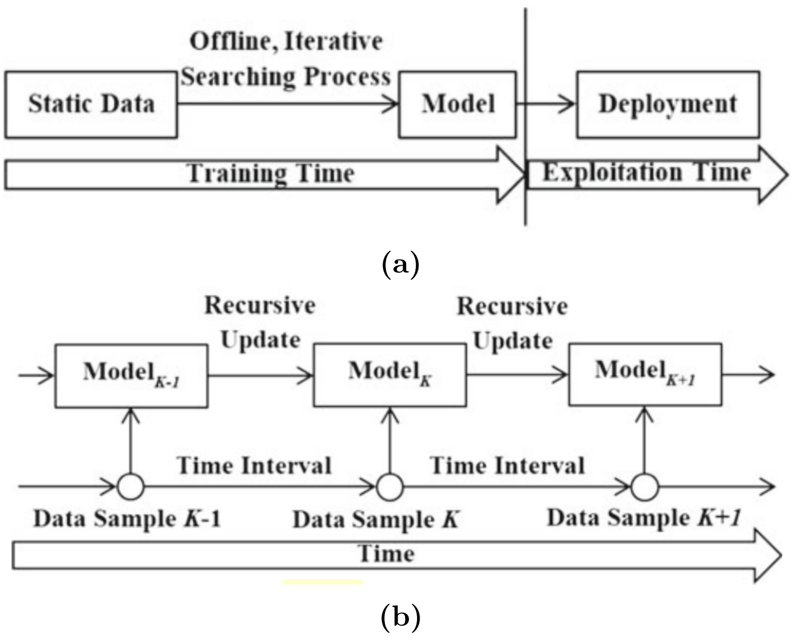


Figure 2.7: *Offline (2.7a) versus online (2.7b) learning. Extracted from [21].*

2.3.2 Data Stream Mining

A data stream is defined in the literature [144, 185, 320] as a continuous data flow of information, in the form of ordered data instances or chunks (sets of instances or batch) arriving at different speeds or time intervals, $\mathcal{DS} = [X_1, X_2, \ldots, X_t]$, where X_t represents the most recent data received. As in offline machine learning, every data instance x_t has a label y_t for supervised settings. Each data instance is made of a set of n attributes, as, e.g., $X_t = [x_{t,1}, \ldots, x_{t,n}]$. As opposed to time-series modelling, setting an specific time order between these attributes is not a requirement of data instances in data stream mining [292].

There is a clear distinction in the literature between instance incremental and batch incremental learning in data streams. In the first one, a machine learning algorithm learns from every individual instance coming from a data stream. The second one performs a learning iteration once it has received a given number of instances. As explained in Section 2.1.4.2 time series is a field focused on modelling in chronological order observations over a specific time interval and where data may arrive in an online fashion. Data stream mining has been recently seen in the literature as an extension of the time series field [292] as these can involve temporal dependence [154] as analysed by Zliobaite in [384].

Data stream learning is a field that faces extra challenges apart from the ones related to temporal dependence and time series. In data streams, instances and the number of them and ranges for values of attributes are not given beforehand but are received sequentially. Since data streams can be infinite, it may not be possible to store the incoming information into memory and learners must be prepared to process the information as it comes [152]. For this reason, there is an emphasis on achieving one-pass algorithms (without epochs or iterations) to process and learn incrementally from data as it comes and the need to avoid high-performance bottlenecks and reduce computational complexity exhibited by offline algorithms [21, 292]. According to Zliobaite in [385], in terms of resources, "*online algorithms should: (1) scale linearly with the incoming data in terms of processing time; (2) use limited memory; and (3) execute adaptation only if the expected utility is sufficient*".

In online learning scenarios and data stream mining, there is no clear separation between testing and training processes as in offline learning since models need to start predicting before the data stream is over (as this point may not exist due to the assumption of infinite streams in this field). Thus, it is necessary to evaluate the model continuously to assess its evolution over time and the health of the models. For this purpose, the convention in data stream mining is to evaluate the models prequentially, as covered in Section 2.2.6.2. In other words, a stream mining learner should always be ready to, first, receive an unlabeled instance and predict its label with an up to date model and, second, receive the true class label for the previous prediction and use it for training [48]. In data stream mining access to the true labels of the instances for supervised training may be delayed. Most of the existing works on data stream classification assume that the true class y_t for X_t becomes available before the next data instance X_{t+1} is received by the learner [152]. While this is not necessarily true in all data streaming scenarios, in this thesis, we work on prediction settings where this assumption is met.

2.3.3 The Problem of Concept Drift

2.3.3.1 Definition

Online machine learning algorithms do not assume stationary and static datasets like offline methods. These receive data instances gradually over time, and, depending on the domain, no assumptions shall be made regarding correlation among features, size, values or order of arrival [37]. Data streams may present non-stationary behaviour over time, leading to changes in their probability distribution and deterioration of the quality of the previously learned models. This phenomenon, which can be easily linked to structural breaks as mentioned earlier in this section and must be handled when learning from data streams to ensure a steady model performance over time, is known in the academic literature as concept drift [350]. Different techniques used for this purpose are covered in Section 2.3.4.1.

A concept (C) can be defined as a set of prior class-conditional probability density functions and class probability distributions [266] as seen in Equation 2-18.

$$C = \bigcup_{y_i \in Y} \{(P[y_i], P[\vec{x} \mid y_i])\} \tag{2-18}$$

For a given data stream $\mathcal{DS}$, each instance produced overtime X_t will be created by a generative process or concept C_t (or ground truth state as presented in the next Subsection). While $C_{t_i} = C_{t_{i-1}}$, the concept of the data stream is considered stable. If at some point $C_{t_i} \neq C_{t_{i-1}}$, then we consider that there has been a shift or drift in the data [49, 144]. Equation 2-19 defines a concept drift as a change of the class-conditional probability density functions and class probability distributions (p) between two timestamps t_0 and t_1.

$$\exists X : p_{t_0}(X, y) \neq p_{t_1}(X, y) \tag{2-19}$$

In data stream learning, there may be periods where concepts are stable (remains the same) and periods with concept drifts of different lengths. For this reason, online learners are required to balance the retention of previously learned knowledge (stability) while adapting to new concepts (plasticity). This tradeoff is known as the stability-plasticity dilemma by the literature of online learning for data streams [144, 222].

2.3.3.2 Our Definition of Ground Truth

In artificial intelligence research, ground truth tends to refer to the fundamental truth in an underlying prediction problem. This can refer to the true class label in supervised learning tasks, the actual position of an object in an image in object recognition, or the absolute number of clusters in unsupervised settings.

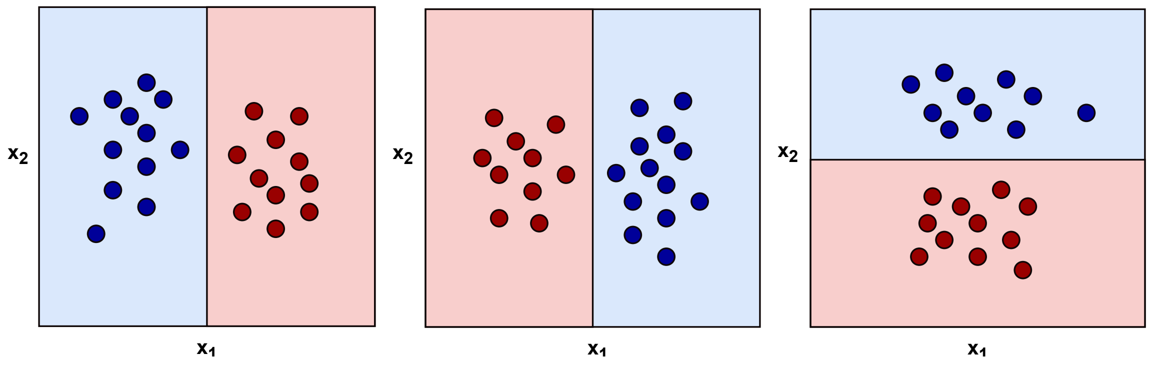
Since part of this work is about change detection, when we talk about ground truth changes in this thesis, we refer to having explicit information or knowledge regarding when a concept drift starts and finishes (ground truth changes). If we mention a ground truth state, then we refer to the actual market state at a point in time.

2.3.3.3 Concept Drift Categorisation

As mentioned in Subsection 2.3.3, concept drifts can be described as transitions between generative processes in a data stream. These transitions can occur with different speeds, severity and distribution [181]. Drifts can be classified differently depending on the impact, interval, distribution or speed of the change. Thus, in the relevant literature [118,151,207] there are various approaches to categorise concept drifts considering these aspects:

- *Impact on the boundaries of the data distribution.* The literature makes a distinction between real and virtual concept drift. The first one affects decision boundaries and deteriorates the performance of the models learned. The second one (virtual) only impacts the conditional probability density function. It does not impact the posterior probabilities as real drifts [290]. Figure 2.8 illustrates a difference between these two types of drift.

- *Distribution and reach of change.* Drifts can occur within a given class or clusters or among many of these [181], hence, being considered local or global drifts, respectively.



(a) *Original data* **(b)** *Virtual Drift* **(c)** *Real Drift*

Figure 2.8: *Types of drift depending on their areas of influence. x_1 and x_2 represent two attributes of the feature set. Classes in blue and red colours.*

- *The interval of occurrence of drifts.* If drifts always occur after the same time interval, these are considered periodic drifts. If the time of occurrence is not fixed, these are considered irregular.

- *Speed of change.* This speed is defined by the number of instances or batches until the shift completes, and the change is considered completed when data is only generated by the new process [254]. Figure 2.9 represents the different types of transitions graphically depending on the speed of change.

  – Sudden or abrupt drifts occur in short periods. Usually in very few data instances. For example, if the generative process of a data stream changes between two consecutive data instances or batches $C_{t_i} \neq C_{t_{i-1}}$, we talk about a sudden drift.

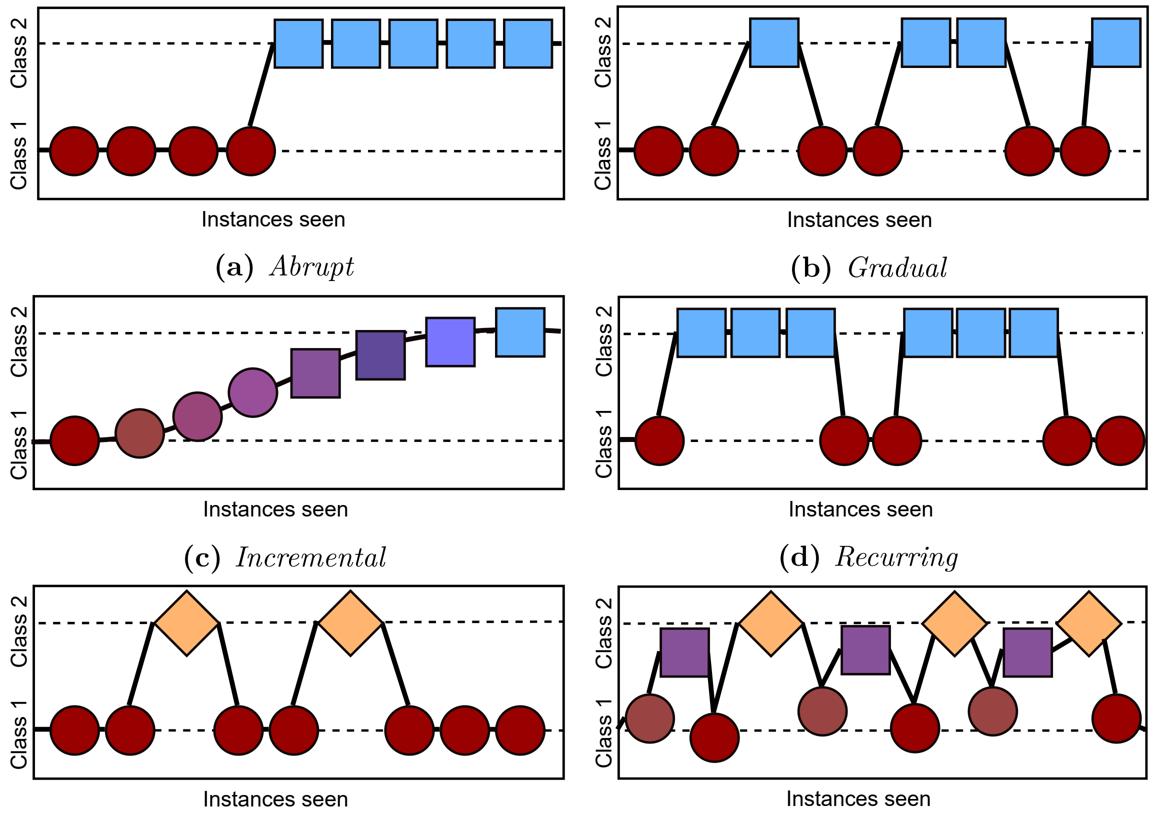
  – Gradual drifts are characterised by a more moderate speed than sudden drifts. These exhibit a longer transition phase and data instances are generated by a mixture of the previous $C_{t_{i-1}}$ and the new C_{t_i} concepts.

  – Incremental drifts are characterised by the slowest speed of change, and differences between data instances in the transition period may not even be statistically significant.

- *Recurrence.* Recurring (or recurrent) drifts are transitions to concepts previously seen. These represent a change leading to stationarity in the data stream. For instance, if the data stream has a set of states $\mathcal{DS} = \{S_1, S_2, \ldots, S_n\}$, where each state S_i is generated by a different generative process C_i, transitions to these know processes previously seen (e.g. S_2) are considered recurring drifts when they occur as a new drift S_n. This is well defined formally in [290] as $\mathrm{S}_{i+1} = \mathrm{S}_{i-k}$, where k represents the k_{th} previous generative process. A recurrent drift can be sudden, gradual or incremental depending on its speed, and periodic or irregular depending on its intervals of repetition.

- *Blips (or outliers) and noise* tend to be ignored in the literature as they may represent random shifts for short timeframes and represent residual concept transitions, respectively.

Several surveys from the state of the art in data stream mining [181, 290] consider abrupt drifts those of length equal to one. Thus, assuming that sudden drifts do not have any sort of transition period. In the financial domain, structural breaks do not tend to be sudden from one data instance to the next one. This is especially at high frequencies since the adoption of different behaviour by investors does not occur simultaneously. In these and other domains, sudden structural breaks can have a transition period. Hence, due to the financial background of this thesis, we will adopt this terminology and consider that sudden drifts could still take more than one time step to complete. Consequently, we will name gradual drifts those changes of considerable longer transition periods than what we will define as abrupt drifts.



(a) *Abrupt* **(b)** *Gradual* **(c)** *Incremental* **(d)** *Recurring* **(e)** *Blips* **(f)** *Noise*

Figure 2.9: *Different types of drifts depending on their speed and sharpness.*

Furthermore, the reader must note that in real-world data streams, drifts and their types are unknown beforehand. In certain cases, a concept drift may appear as a mixture of the types mentioned above, or multiple single drifts may occur concurrently in the data stream. For the sake of simplicity, in this thesis, we assume that drifts occur once at a time (single drifts). For us, as explained earlier in this section and Chapter 1, drifts are changes in the generative process of the data stream instances. We will consider concept drifts of different transition speeds that we will name as abrupt or gradual drifts depending on the sharpness of the change. One of the goals of this thesis is to correctly identify recurring drifts to avoid model retraining if an algorithm has been previously trained for a given generative process. In real-world datasets, we will consider irregular intervals for any drifts. In simulated streams, we will create periodic transitions to simplicity the problem of detecting recurring concept drifts at various speeds. We will mainly deal with real drifts deteriorating classification performance since our final goal will be to obtain the best classification accuracy in stock trend prediction. In this regard, we will not make any distinction between local or global drifts. Blips and noise will not be directly covered, but we will use techniques from the state of the art that should be resilient to these and are introduced in the next section.

2.3.4 Learning under Concept Drift

2.3.4.1 Passive versus Active Drift Handling Approaches

Section 2.3.3 covered the concept drift phenomenon and how drifts can occur unexpectedly or gradually in a data stream in various ways. As shifts can have an impact on the performances of the trained model, the search for strategies to handle changes in data streams is an active area of research [37, 144, 151, 290]. Broadly, there are two different approaches to handle concept drifts.

- *Passive (adaptive) approaches.* Models that react passively to concept drifts incorporate mechanisms to adapt to changes in the latest data instances received. These sorts of approaches work better with incremental or gradual shifts or data streams that change continuously over time.
- *Active or explicit (detection) approaches.* Active drift detectors follow quantitative methods to observe patterns breaking periods of stability and react consequently (e.g. sliding windows based approaches [37, 39, 151]). These can be integrated as part of online algorithms for data stream learning [152] or combined with them to decide when to replace, update or re-train a model [14, 290]. Active approaches are aimed to sudden concept drifts as these can react rapidly to any changes in the data distribution [147] or in the learning accuracy [39].

In this thesis, we will use both passive and active approaches to handle concept drifts. The next section describes the drift detectors that will be used in Chapters 4 and 5.

2.3.4.2 Drift Detection

In this thesis, we will use some of the drift detectors that have proven to achieve better results in the literature [39, 107, 160]. These drift detectors that we will use fall into the category of supervised drift detectors and generally receive a sliding window of (online) prediction errors obtained by a supervised machine learning algorithm to monitor potential changes impacting the learner's performance. These errors indicate if the base classifier predicts the arriving instance correctly or not, decreasing when the learner classifies an instance correctly. Changes in these online errors or other statistics gathered and based on rolling windows are considered a potential drift.

Many of these detectors work in a two-stage setting where a lower confidence level raises a warning that indicates a potential drift [290]. Thus, this setting illustrated by Figure 2.10 raises the following two signals:

- *Warning signal.* This is produced at a lower confidence interval to report a potential drift when it starts. It reports that a change is suspected in the data stream. Hence, concept drifts handling strategies tend to create a new base classier to be trained in parallel to the active model but only with the new data stream instances.
- *Drift signal.* This has the highest confidence level of the two and reports that a change has already occurred. Drift-handling strategies generally replace the active model with a new learner when this signal is raised.

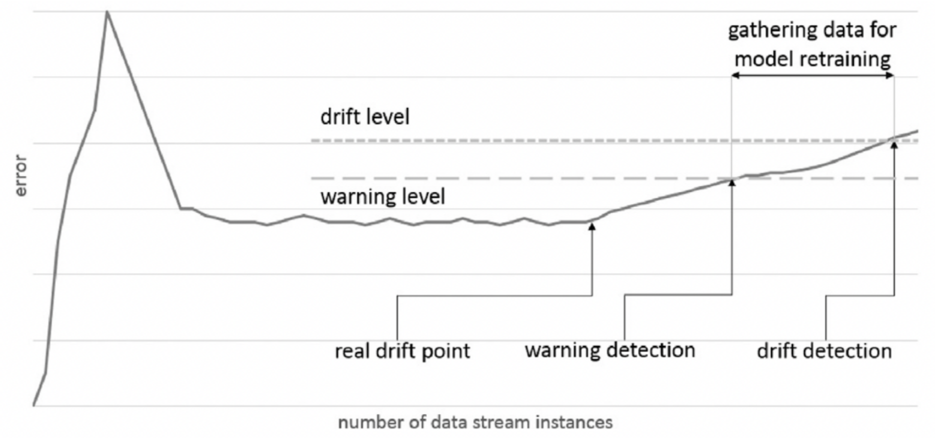


Figure 2.10: *Illustration of a two-stage drift detection mechanism, extracted from [207].*

These supervised detectors need to test learners over a certain number of data instances of the stream to measure the change in classification error and thus raise warning and drift signals. This design implies a delay between the real drift point and any warning or drift signal being produced by the detectors, as illustrated in Figure 2.10. The time between the warning and drift signals in the relevant literature is referred to as warning window [158, 252].

The drift detectors used in this thesis are listed below. For a more detailed description, see [107, 160].

- *Drift detection method* (DDM). Proposed by Gama and Labidi in [142], it uses classification results to compute the online error rate of the base learner. DDM considers that, when the concept changes, the base learner will incorrectly classify the arriving instances that are created by a different generative process. Thus, if the error rate increases up to a certain threshold, it raises a concept drift signal.
- *Early drift detection method* (EDDM). This detector was proposed by Baena-García et al. in [32] as a variant of DDM that analyses the distance between two consecutive misclassifications instead of the number of misclassifications. One advantage of this detector is that it does not have any input parameters.
- *ADaptive WINdowing 2* (ADWIN2). Bifet and Gavalda proposed this drift detector in [46]. It maintains a sliding window divided into two sub-windows representing old and new data and adjusting dynamically. ADWIN2 signals drift if the mean difference between both sub-windows surpasses a threshold. The size of the window decreases in the presence of drift and increases during periods of stability. This detector has recently been used in the literature to detect concept drift using online classification error rates [152]. A separate instance of ADWIN2 needs to be in place with a lower threshold to detect warnings.
- *Reactive drift detection method* (RDDM). This detector was proposed by Barros et al. in [38] as an improvement of DDM, which sensitivity decreased over time in very large concepts. RDDM continuously recomputes the statistics responsible for signalling warnings and drifts. It discards old instances and forces drift in concepts and warnings active for long periods, respectively. RDDM has been demonstrated to be one of the best-supervised drift detectors of the literature for gradual drifts in synthetic datasets [107].

- *Drift detection method based on Hoeffding's inequality* (HDDM). This detector was proposed by Frías-Blanco et al. in [132]. It applies "*probability inequalities that assume only independent, univariate and bounded random variables to obtain theoretical guarantees for the detection of such distributional changes*". HDDM monitors false positive and negative rates, not assuming that the results come given by a Bernoulli distribution. The authors of HDDM propose two different versions:

  - A-test (HDDM$_A$) uses two moving averages to track changes.
  - W-test (HDDM$_W$) uses weighted moving averages for the same purpose.

  HDDM's A-test and W-test are aimed for abrupt and gradual changes respectively [107].

2.3.4.3 Model Reuse

State-of-the-art papers on data stream mining focus on different types of drifts and shifts that can either impact or not in the learning models' error rates. This has been explained in Subsections 2.3.3.3 and 2.3.4. In this context, changes are normally associated with non-stationarities in a data stream. However, stationary data streams can have concept drifts at the time of change between different stationarities or states.

The relevant literature claims that reusing a classifier previously learned for the current state is more efficient than rebuilding a classifier from zero. At least, there should be a shorter time spawn to have a model trained for the current data, and this impacts not only computational cost but also the learning model error rates. The task of reusing a previous model is often seen in the literature as a sort of *transfer learning* task (TL) [165, 292]. However, this term is usually associated with the deep learning field.

However, and especially in data streams with more than two concepts, one of the challenges of the task of reusing a previous model is to identify what is the suitable model to be reused. In certain contexts, there may be dozens or hundreds of previously learned models in a history of saved learners (concept history), and finding the right metrics to select the most appropriate one is already a research branch in data stream mining. Two different metrics used for this purpose in this thesis are conceptual equivalence and concept similarity. These are briefly described on the next page.

- *Conceptual equivalence* was initially proposed by Yang et al. in [376] and assumes that when two classifiers behave similarly predicting during a time window, both describe the same concept. When drift is signalled, all classifiers are given a set of new data instances to predict. If, for a data instance, two classifiers produce the same output, then the conceptual equivalence between the two increases. The total score is the mean conceptual equivalence for all the data instances compared across all classifier pairs. The full algorithm can be found in [376].

- *Concept similarity* was initially proposed by Li et al. in [221] to detect recurring drifts in the absence of labelled data. The approach in [221] aimed to recognise similar concepts using Euclidean distance distances between clusters representing different concepts (namely *concept clusters*). They defined a set of thresholds to decide between the occurrence of drifts, noise and recurrences. The latter (recurrent concepts) are recognised if the deviation between a new and a previous cluster is lower than a certain threshold.

2.3.4.4 Meta-learning

Another challenge for reusing previous models is how to store them in memory. There can be scenarios where novel concepts keep appearing in the data stream, and thus, the stack of historical models will grow infinitely. This leads to research in setting a maximum pool size for historical models and pruning and replacement strategies of old or unused learners. The data stream mining literature uses online machine learning models to manage strategies and operations with classifiers [14, 140, 141]. These models (or frameworks) that manage how, when, and what-to-learn [355], and that are beyond the learner used (algorithm-agnostic), are known as meta-learners and are often used to handle scenarios with recurring concepts [16, 140, 252].

Meta-learning approaches can either decide when to train, when and what model to replace, when to forget (prune) a learner and when to create one [143, 299] by using the evaluation performance metrics of active and historical models (meta-features) [356]. These approaches can also help with model selection for a given dataset. Hence, meta-learning is considered a subfield of *AutoML* despite being an older branch of research [294, 353]. This subfield is inspired by the human learning system that reuses previous knowledge to learn new tasks, not starting from zero every time. In data stream mining, meta-learning algorithms tend to manage a pool of learners and estimate the weights of these training them as an ensemble algorithm [188, 354]. We will use these in Chapter 5 to handle recurring changes in the financial domain.

2.3.5 Supervised Learning in Data Streams

One of the differences between online and offline (batch) supervised learning techniques is their interleaved nature, as explained in Section 2.2.6.2. In an offline setting, the entire dataset is available at the start of the training task, and a whole batch of data is fed to the model, often in multiple iterations or epochs.

Conversely, in online scenarios, data instances will be available incrementally over time and thus should learn from them as soon as these become available [139]. At the same time, the nature of the prediction in an online scenario is to occur at any point in time with the current learning model, which creates a set of challenges [240] as seen along this chapter. Another difference, as covered in the previous section, is the presence of concept drifts that can invalidate the model learned, which gives importance to the adaptive learning process and the interleaved evaluation [152].

This subsection describes Hoeffding trees (HT) and Hoeffding adaptive trees (HAT). The incremental version of naive Bayes that we use is not described as its theoretical background has been already covered in Section 2.2.3.1. Beyond the incrementalisation of other approaches or ensemble models, HT and HAT, which are state-of-the-art one-pass incremental learners for stationary and non-stationary streams, respectively [240] will be the online learners that will be used in this thesis.

2.3.5.1 Hoeffding Trees

The Hoeffding tree (HT) algorithm is a version of CVFDT of the concept-adapting very fast decision tree online learner (CVFDT) proposed by Hulten in [185]. This algorithm starts its learning process using the first data instances received to choose the attribute used to split the root node. All the following instances received are passed to HT until reaching the leaves. Child nodes keep statistics representing the data distribution seen by them, and these statistics are used to decide what nodes to split and what attributes to use as separation boundaries. In its default setting, HT creates a naive Bayes classifier at the leaves if it provides better accuracy than using the majority class. Child nodes are evaluated recursively for further splits [119, 185]. Thus in HT, there are two types of nodes:

- *Learning nodes*: leaves used to predict and gather statistics over time.
- *Filtering nodes*: root and intermediate nodes used to forward examples to the right child.

One of the challenges online trees face is deciding the best time to produce a split. This is because instances are not processed at once, unlike offline trees, and data streams may be infinite. In this regard, HT runs a statistical test called the Hoeffding test over child node statistics [119, 173] to decide when there is enough confidence in the stream to place a decision boundary by splitting a node. The Hoeffding inequality provides upper and lower bounds of the number of examples necessary to produce a split at each node. This algorithm has been studied recently in [240]. Despite being proposed for stationary streams, it can adapt to concept drift (especially abrupt changes). Part of this is because child nodes split in non-stationarities, resetting their previously gathered statistics. This model does not store previous instances, and thus, it does not reconstruct node statistics (forgetting them) after a split. Hence, splits in Hoeffding trees help them to adapt to concept drifts.

This algorithm will be used as a base classifier in Section 4. algorithm 4.3 will refer to Hoeffding trees with a bagging mechanism, which is part of an ensemble model. For more information about HT or to see the full algorithm, see [240].

2.3.5.2 Hoeffding Adaptive Trees

Hoeffding adaptive tree (HAT) is an adaptive version of the Hoeffding tree proposed by Bifet in [47]. It uses ADWIN2 as the default supervised drift detector and monitors changes in the error metrics of different subtrees (branches). HAT replaces them with more accurate branches (background subtrees) if the error of the former one decreases over time. HAT also improves the HT algorithm with a bootstrap sampling strategy which usually is a feature of ensembles.

The main differences of the HAT with respect to HT contributing to fast reaction in case of abrupt changes are the following two actions during training:

- *Creation of new subtrees*: background subtrees are created directly when a drift is detected, not depending on any pre-established number of instances or buffer.
- *Replacement of subtrees*: HAT replaces old branches with the background subtree once the second is more accurate [48].

2.3.6 Unsupervised Learning in Data Streams

Online methods need to take into consideration the potential presence of concept drifts in the incoming data stream. The use of offline techniques implies storing the training datasets for retraining and re-evaluation purposes later. The literature of data stream mining assumes that streams can be unbounded, and thus their storage is not feasible after a certain point [320]. Hence, online methods avoid the computational and storage-related costs by applying incremental updates over time. Online unsupervised learning (or stream clustering) algorithms continuously feed new data instances to existing models. Clustercentres are updated incrementally accordingly to the latest instances seen, and old data can be forgotten over time if this is not relevant to the model (e.g. low weights) [320].

Like traditional data clustering methods explained in Section 2.2.2, stream clustering can be classified into five categories: i) partitioning methods, ii) hierarchical methods, iii) density-based methods, iv) grid-based methods, and v) model-based methods. This work will focus on model-based methods because these aim to "*optimise the likelihood between a dataset and statistic models*" [265], and we will use statistical methods to model ground truth market states. For a deep insight into the relationships between traditional clustering methods and stream clustering methods, see [265].

Model-based clustering methods train their model and grow the list of centroid-like entities (or neurons) based on the model's error. While other methods only look at the data distribution and the error between instances and centroids [111]. We believe that this property of model-based techniques can help us represent in a non-supervised manner a concept or state.

In Chapter 4 we will use model-based techniques to summarise the data distribution. Chapter 5 uses an online unsupervised learning algorithm to represent our interpretation of different financial market states. This algorithm, namely *growing neural gas*, and the algorithms it leverages from are described in the following subsections.

2.3.6.1 Self-Organising Maps

Kohonen proposed the self-organising maps (SOM) algorithm in [202]. This algorithm is a shallow neural network that consists of an input and a competitive layer. The first layer has a set of neurons (also called prototypes in our work) that are connected among them. Each neuron is assigned a vector of weights to be updated during training. An in-depth description of this competitive learning strategy can be found in [204].

After training, SOM splits the feature space into a set of sub-regions that receive the name of *voronoi regions* and assigns neurons as the centres of each of these regions. As these region centers act as centroids, this technique can be categorised as a model-based clustering algorithm. Additionally, SOM projects (preserving the original topology) the input feature space into a lower dimension. It adapts weights of neurons to the topological regions with more density to find clusters in a 2D or 3D space, which allows its use for data visualisation of high-dimensional spaces [226]. This mapping to a low dimensional space is illustrated by Figure 2.11.

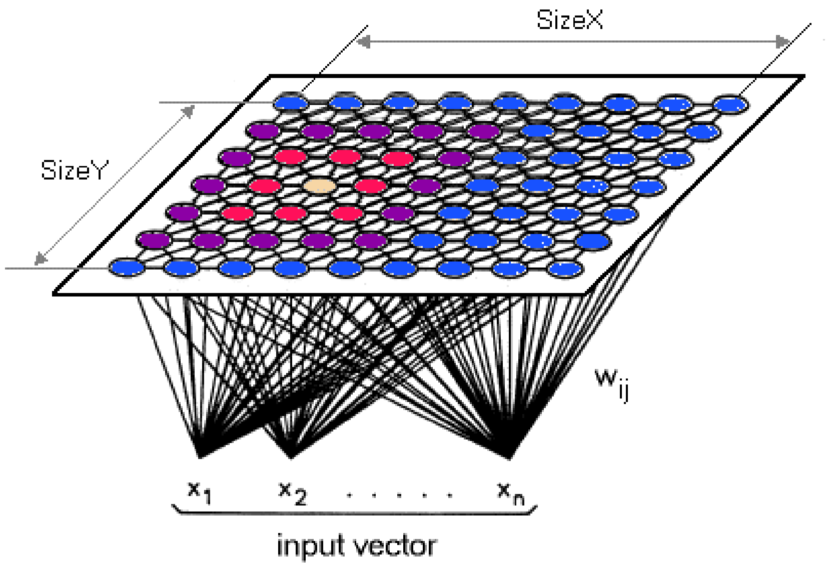


Figure 2.11: *Example of SOM representing multiple clusters as different colors. From [92].*

Some limitations of SOM, according to [226, 322] are that the results obtained can change highly by a set of predefined parameters that are hard to establish before knowing the data distribution. For instance, the size of the network (neurons) is specified before and is immutable during the learning process. Too large or small size can make the model over or under fit to the data distribution respectively. A solution for this, which in any case may not work for complex cluster shapes, is to determine the shape of the distribution in parallel to the desired number of neurons in the network at the training stage. This approach implies multiple training iterations, which makes SOM unsuitable for both online scenarios and dynamic distributions [309, 316].

Different continuous learning alternatives to SOM have been proposed in the literature [136, 250]. One of them, described in Section 2.3.6.3, will be used in Chapters 4 and 5 of this thesis.

2.3.6.2 Neural Gas

Martinetz and Schulten initially proposed the prototype generation *neural gas* (NG) algorithm in [242]. This approach can be categorised as an unsupervised self-generating learning algorithm. The learning process starts with a fixed number of prototypes (or neurons) not connected among themselves. When NG receives instances, the prototypes move in the feature space, and connections are created among the closest prototypes to the instances received. During the learning process, prototypes can be removed if they surpass a maximum age.

This algorithm, which served as a theoretical base to the unsupervised learning algorithm that will be used in this thesis, is covered in more detail in Section 2.3.6.3. This algorithm faces similar limitations to the SOM algorithm presented in the previous section since it is not prepared for online scenarios and depends on predefined parameters.

2.3.6.3 Growing Neural Gas

Growing neural gas (GNG) was proposed by Fritzke [136] as an incremental version of NG. It could also be seen as a modification of the SOM algorithm that does not have a fixed number of prototypes. Instead, the number of prototypes evolves during the competitive learning training process.

The training process of this prototype generation method is made of two stages, the adaption process and the growth of the network. The adaption process selects an input vector depending on a distribution function (neighbourhood function) and the closest prototype (neuron) to each instance received. This adaption mechanism, which can be compared to the adaption process in k-means explained in Section 2.2.2.1, is also used by SOM with the only difference that the distribution function is only applied to neighbours of the closest prototype [256]. GNG starts with two prototypes placed randomly in the feature space. Edges (connections) are created between two prototypes with most activity if there is no connection between them. The network of connections is evaluated iteratively. As data stream instances keep coming, prototypes are created where the local error is larger or pruned following a competitive Hebbian learning fashion if they are older than a certain threshold and do not have connections. This process, illustrated in [135], can be seen in Figure 2.12 and step by step in algorithm 2.3. In Figure 2.12, original instances are coloured in green, topology prototypes represented as blue dots, and neighbourhood connections are represented as edges in the graph.

Algorithm 2.3 Growing neural gas algorithm subtracted from [136]

```
1. Start with two units a and b at random positions $w_a$ and $w_b$ in $\mathbf{R}^n$.
2. Generate an input signal $\xi$ according to $P(\xi)$.
3. Find the nearest unit $s_1$ and the second-nearest unit $s_2$.
4. Increment the age of all edges emanating from $s_1$.
5. Add the squared distance between the input signal and the nearest unit in input space to a local counter variable:
       $\Delta errors_{s1} = |w_{s1} - \xi|^2$
6. Move $s_1$ and its direct topological neighbours towards $\xi$ by fractions $\epsilon_b$ and $\epsilon_n$, respectively, of the total distance:
       $\Delta w_{s1} = \epsilon_b(\xi - w_{s1})$
       $\Delta w_n = \epsilon_n(\xi - w_{sn})$ for all direct neighbours $n$ of $s_1$
7. If $s_1$ and $s_2$ are connected by an edge, set the age of this edge to zero. If such an edge does not exist, create it.
8. Remove edges with an age larger than $a_{max}$. If this results in points having no emanating edges, remove them as well.
9. If the number of input signals generated so far is an integer multiple of a parameter $\lambda$, insert a new unit as follows:
    • Determine the unit $q$ with the maximum accumulated error.
    • Insert a new unit r halfway between $q$ and its neighbour $f$ with the largest error variable:
          $w_r = 0.5(w_q + w_f)$.
    • Insert edges connecting the new unit $r$ with units $q$ and $f$, and remove the original edge between $q$ and $f$.
    • Decrease the error variables of $q$ and $f$ by multiplying them with a constant $\alpha$. Initialise the error variable of $r$ with the new value of the error variable of $q$.
10. Decrease all error variables by multiplying them with a constant $d$.
11. If a stopping criterion (e.g., net size or some performance measure) is not yet fulfilled go to step 1.
```

The GNG algorithm is considered suitable for continuous online learning and dynamic data clustering. As a disadvantage versus SOM, GNG does not perform dimensionality reduction and thus is not suitable for data exploration purposes. This fact is not an issue in our current work, as visualisation of feature maps is not a task in scope. The authors of [186] perform a basic comparison between a dynamic version of SOM and GNG. Unlike in SOM, in GNG, the connections between prototypes are not permanent, which allows the continuous evolution of a concept and the continuous change of the spatial distribution of a class. Finally, conversely to offline algorithms like SOM and k-means, growing neural gas also adapts the size of its topology of prototypes to the data received.

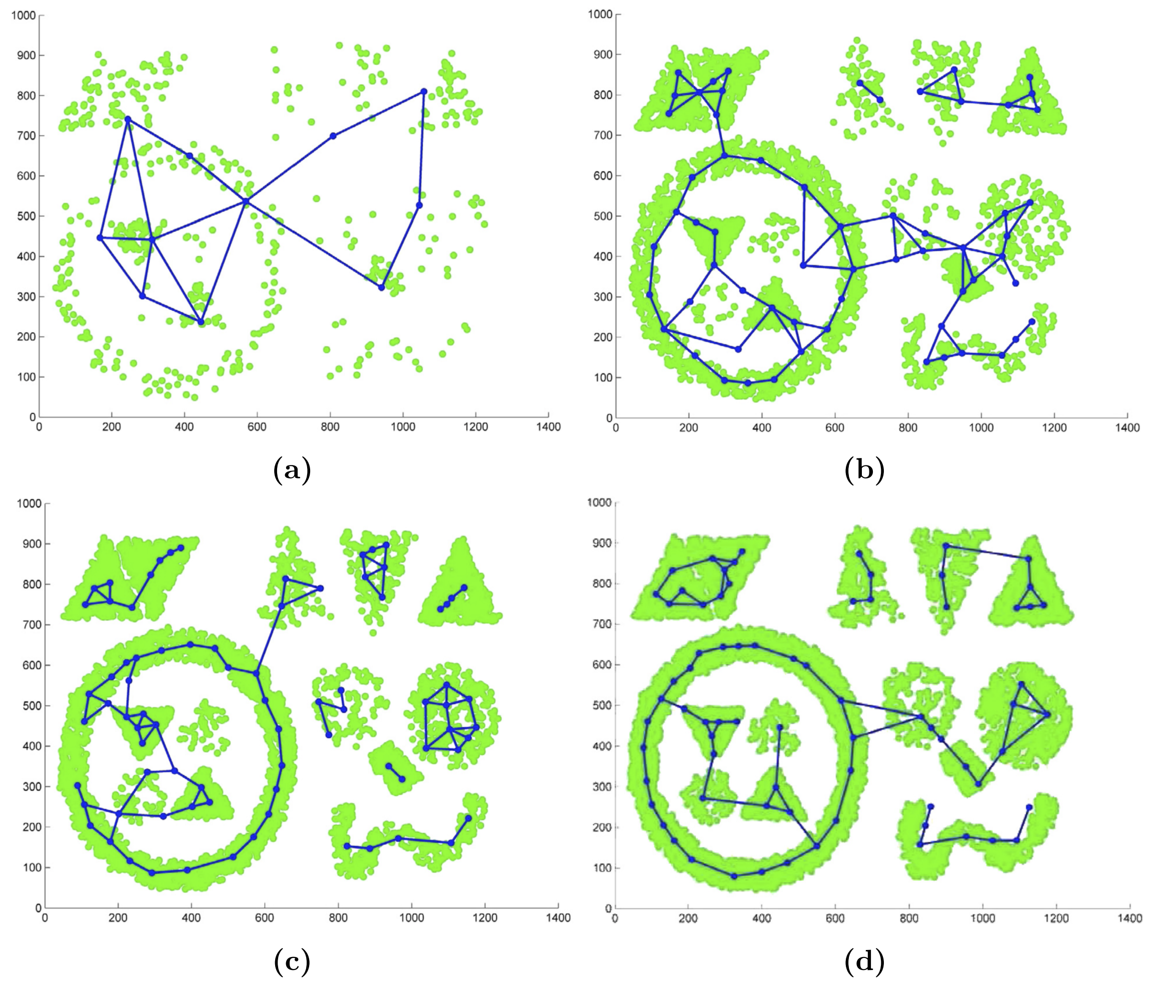


Figure 2.12: *Example of learning process in growing neural gas over time. Behaviour of a variant of GNG extracted from [149].*

2.4 Relevant Software

This thesis will use *massive online analysis* (MOA) [52], a tool developed in Java by the University of Waikato. This tool will be used to simulate data streams and test the models developed in a sort of back-testing fashion. The online incremental machine learning algorithms proposed in this thesis will also be developed for the MOA framework. The reasons to select this framework were: i) its popularity, being the most widespread technology for online learning for data stream mining at the start of this thesis [48], and ii) that the majority of the state-of-the-art algorithms were already implemented in this framework. Experiments in this thesis have been run as bash and Python scripts. Training and evaluation of MOA algorithms, coded in Java, was triggered using bash. The Python programming language has mainly been used for data pre-processing, analysis of results and visualisation.

The reader must note that currently there is a growing number of alternative tools for the simulation (or development in production) of data streams and online machine learning apart from MOA that are gaining popularity among both academics and practitioners. For instance, if we consider stream processing, which is the first step in production environments to produce, receive and parse incoming data flows, there is a number frameworks such as Apache Spark, Flink, S4, Storm, and Kafka that have gained popularity in the last years. Chintapalli et al. in [87] and Lopez et al. benchmark them in [231]. Mehmood and Anees in [247] provide a systematic literature review on the challenges of real-time data processing in big data streams.

In parallel, due to the adoption of AI in industry, frameworks for continuous delivery and integration of machine learning models such as CML [1] are gaining traction to allow retraining strategies and model reuse with offline algorithms, which may require updates over time [34]. Offline learning remains the main trend in the industry. However, there has been an increase in the popularity of continuous deep learning and reinforcement learning algorithms [317,334] that have proved their usefulness in learning from data streams [154]. A selection of these methods will be cited at the end of Subsection 3.2.1. In any case, new frameworks like *River*[2] [257] are appearing to allow developers outside academia to use and develop online machine learning algorithms. Imbrea in [188], and Gomes et al. in [154], mention relevant tools in this domain.

[1] https://github.com/iterative/cml
[2] https://github.com/online-ml/river

2.5 Summary

This thesis introduces new algorithms to make predictions in high-frequency time series under structural changes like intraday stocks prices in the financial domain. In this chapter, we have laid the foundation of this new connection between the online machine learning and the financial literature by introducing their theoretical backgrounds. We have focused on the idea of structural breaks in markets that may present a level of efficiency depending on their traders' perception and geopolitical situations.

First, in Section 2.1 we covered the concept of financial time series. Second, in Section 2.2 we used time series as a framework to present the use of adaptive machine learning techniques in the financial field. Using these approaches to handle structural breaks is beneficial to minimise the prediction error during periods of instability and allow continuous learning, which can lead to a financial advantage in markets that are non-stationary by nature. Section 2.3 covered the field of data stream mining. In this regard, we compared financial time series with incoming data streams of data. We continued comparing structural breaks to the concept drift phenomenon. Then, we introduced the theoretical background of different offline and online machine learning algorithms that will be used in this thesis. Finally, Section 2.4 mentioned relevant software and relevant reviews to guide any reader interested in data stream mining and direct future research.

Chapter 2, attempted to provide the reader with the necessary theoretical base to understand the rest of this work. In Chapter 3, we will cover recent and relevant research to the approaches that will be presented later in this thesis. The application of the data stream mining literature to the financial domain was not widespread at the time of writing this thesis. Thus, in the next chapter, we will provide what for us is the state of the art in this subject.

Chapter 3

State of the Art

Once explained the theoretical background of this thesis in Chapter 2, this chapter will study related works in the two fields approached: i) online incremental machine learning and ii) price stock movement prediction. We aim to bridge the gap between both research topics and the problem of concept drift in data stream mining, which are commonly studied separately. In fact, these similar fields are not connected in the literature since financial experts have not used machine learning (ML) traditionally, and data stream mining has been until now barely explored in industry.

First, in Section 3.1, we will introduce the research field of structural breaks (regime changes) and relevant works. Then, in Section 3.2 we will review contributions regarding concept drift. This will cover different approaches and metrics to handle and measure drifts in the literature. In Section 3.3 we will cover a selection of remarkable supervised learning works in data stream mining. This includes relevant works that automate the ensembling, replacement and reuse of classifiers over time. Section 3.5 will cover research works about the topic of non-supervised learning under concept drift. We will analyse different alternatives to traditional clustering approaches to support the selection of models in continuous learning settings, which can benefit online learning across different market states.

Section 3.6 will study the main contributions regarding price trend classification. This will include previous research works and the main approach that we will use to model financial data. Finally, we will summarise this review of the relevant literature and analyse potential research opportunities in Section 3.7.

3.1 Structural Changes as a Type of Concept Drifts

In computational finance, changes in the behaviour of the market are normally referred to as regime changes or switches [19, 164, 169], structural breaks or changes [17, 109, 280, 280], volatility shifts [31], switching processes [288] or market states [259]. In this kind of data, long periods of stability might be interrupted by short episodes of abrupt changes [169]. Changes that may or not be transitory since a newly adopted behaviour by the market, reflected as part of the mean returns, their volatility or correlation among them, may persist for many periods. Timely recognition of these sudden behavioural changes in markets can significantly lower the risk of financial exposure. This has inspired the materialisation of techniques such as regime switching models in the financial literature. Regime switching models work under the premise that new dynamics of price returns and fundamentals persist for several periods after a change. A key element in these models is if the exact market regimes reoccur over time (e.g. across recessions or periods of economic growth) or if new regimes deviate or have evolved from previous ones [19].

The prediction of future values in financial markets is a common application area for ML. Previous research works report high accuracies forecasting price changes with advanced techniques and the feasibility of making profits using these predictions, which is against the EMH (explained in Chapter 2), that points to unbeatable markets. An alternative theory is the *adaptive market hypothesis* (AMH) [227], introduced by Andrew Lo in 2004. This theory, with empirical evidence in a increasing number of research works [351], combines the EMH with principles from behavioural finance, allowing the ideas of market efficiency and inefficiencies to co-exist. Under the AMH, the efficiency of a market evolves as market participants adapt to an environment that changes continuously. In this regard, participants rely on heuristics to make their investment choice, leading to mostly rational markets under those heuristics (like the EMH). The main difference is at the time of major behavioural shifts in the market participants, as in economic shocks or crises. In this case, the AMH considers a market that evolves, and the initially adaptive heuristics may become static in certain market situations. Consequently, the EMH may not continue under periods of abnormal conditions, stress or abrupt changes in the market. Hence, financial markets may be predictable in certain periods, as demonstrated by Lo [228]. Therefore, convergence to market efficiency is neither guaranteed nor likely to occur. The level of efficiency depends on the market participants and the market conditions at that time.

One of the very few financial studies citing concept drift explicitly, being external to use cases of evolving or adaptive AI research, can be found in Masegosa et al. in [243]. The authors of [243] analysed data of the Great Recession and claimed that economic changes during this period manifest as concept drifts in their generative processes. An intermediate example of trying to predict financial crises using machine learning methods can be found in Samitas et al. in [305], where the authors studied possible contagion risks between financial markets that could trigger financial crises to signal warnings at an early stage.

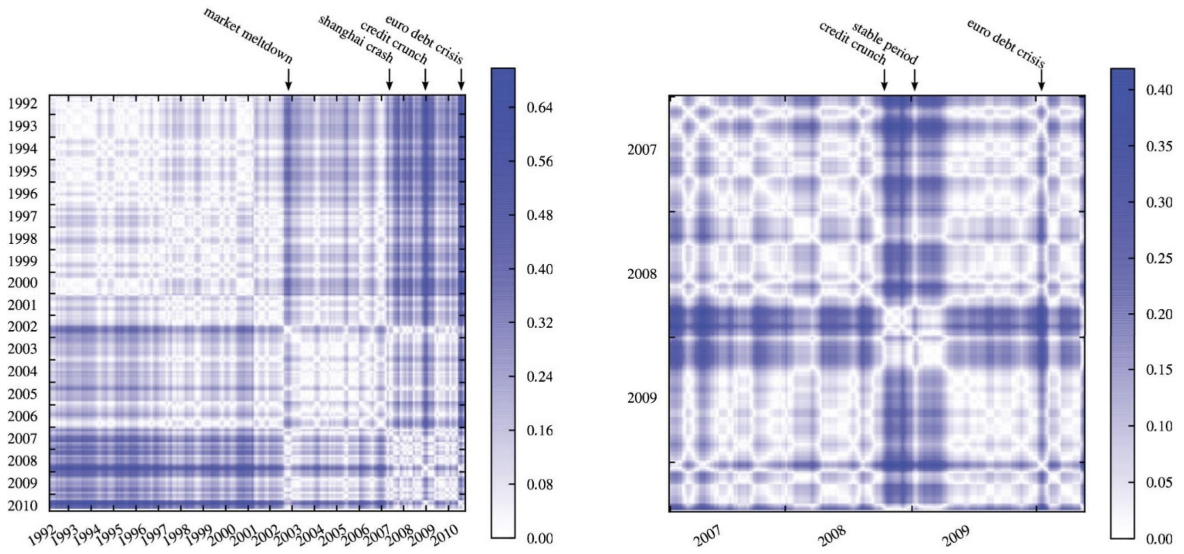


Figure 3.1: *Representation of financial crises (blue areas) versus moments of stability (white) comparing the correlation across assets of the S&P 500. Extracted from [259].*

Two key research pieces in this regard are the works from Tsang [347] and Münnix et al. in [259], which proposed mechanisms to identify points of drastic changes in financial time series. Tsang [347] used statistical-based and traditional machine learning (e.g. naive Bayes) approaches to classify normal versus abnormal regimes. They proposed a framework based on the change speed of price returns and the degree of changes to visualise and discriminate between different market regimes depending on the volatility of their price returns. The book of Tsang [347] provides an insightful review of the literature on the topic of regime changes (RCs) in financial markets. Münnix et al. in [259] visualised differences in the correlation structure of the price returns across assets in the S&P 500 during the Great Recession (see Figure 3.1). They extended the selection to a sample from 1992 to 2010, identifying eight market states repeating behavioural changes over time.

However, most of these previous research works focus on the likelihood of daily or monthly changes, where retraining a model is an achievable task. In this thesis, we focus on structural breaks and changes in intraday trading and at high frequencies. Little research is focused on the seasonalities and changes at the intraday level [134,138, 307,308,366]. Due to the computational cost of ML and statistical methods, and the high amount of data received at these resolutions, there is a need to keep models up to date. For this reason, we will apply the data stream mining literature as a framework to train online models that are always up to date. This approach will allow us to tackle changes in the market structure as concept drifts. In Sections 3.2, 3.3 and 3.5 we cover the most relevant research works in concept drift handling and data stream mining machine learning algorithms.

3.2 Contributions Regarding Concept Drift

3.2.1 Introduction

The problem of concept drift was covered in Chapter 2 as a change in the data distribution and evolution of relationships between attributes and the target feature over time. Chapter 2 explained the difference of passive adaption and active drift detection mechanisms and Subsections 3.2.2 and 3.2.3 will cover relevant research works in this regard. This thesis will also explore the presence of stationarities in financial data. Since in these scenarios previous learned models may become relevant again in the future, we tackle this issue with solutions proposed for the problem of recurring concepts [7, 156, 159], of increasing popularity in the data stream mining literature [118,144,151,290,365]. Considering potential concept recurrences is suitable since, as part of the stability-plasticity dilemma mentioned in Section 2.3.3, online incremental machine learning algorithms may have to relearn previous concepts if these do not have explicit mechanisms to remember them. This process has a high computational burden, as it implies adapting or training a new model from scratch. Thus it causes a lower predictive accuracy while the models are not up to date to the latest state of the data stream.

Wares et al. in [364] described important challenges encountered in concept drifting data streams. Online machine learning models need first to learn the latent space representation of the dataset and then handle changes in the probability distribution and deal with *catastrophic forgetting*. Catastrophic forgetting, a challenge faced in continuous learning and especially in evolving data streams [206, 334], refers to adaptive models forgetting previous knowledge when learning new patterns over time.

An open issue in the data stream mining field is the lack of links with the traditional time series literature [154]. This has started being studied by Jesse Read recently [292, 293], who aimed to unify the concepts of data streams and time series by assessing their definitions in the literature and theoretical formulation. Read in [292] proposed to interpret concepts as temporal sequences to allow continuous adaptation and transfer knowledge to the next concept, as an effective alternative to explicit concept drift detection. He noted that approaches such as stochastic gradient descent would be able to perform this continuous adaption and referred to the literature of transfer learning (TL) [272] and therefore neural networks (either shallow or deep) as a way to deal with concept drifting data streams. The approach suggested by Read was based on the temporal dependence that he perceived in the context of gradual or partial concept drifts, a behaviour also observed by Mello et al. in [108]. Although traditionally, TL has been used in offline settings, requiring the entire training set to be present in memory before training commences, a few recent studies applied it to non-stationary data streaming environments [253]. The authors in [24] presented a way to include information about the current data distribution and its evolution over time into machine learning algorithms.

Several approaches from the deep learning (DL) field (e.g. RNNs), which has obtained state-of-the-art results in some financial applications [218, 271], have also tried to face the problem of concept changes when learning continuously [206, 303, 334, 379]. While its application to high-frequency markets is still an open problem, recent research works using DL [321] claimed that financial data at high-frequencies exhibit stylised facts and may hold learnable stationary patterns over long periods. In [313], the authors reviewed modern ML and DL approaches applied to high-frequency trading at the minute level. In [154] the authors analysed the usefulness of DL and reinforcement learning (RL) methods in data streaming applications and cover further the links between time series and data streams. Both techniques work naturally for prediction in streaming contexts, but these have not been widely approached yet in the data stream mining literature due to different reasons like their difficulty to be trained and their need for reward functions instead of true labels.

Despite the promising potential of DL approaches, this thesis does not focus on these and is instead focused on prototype generation, meta-learning and ensembles. We have done this following the current trend in the literature that will be presented in this chapter. In this regard, a good subset of adaptive approaches in data stream mining tend to use ensemble learners, so each base classifier adapts to a different non-stationary behaviour or scenario that may re-appear in the future. This topic will be covered further in Section 3.3.

As covered in Section 2.3.4, the literature on data stream mining makes a clear distinction on the design of mechanisms to learn new data over time. Both passive and active methods to handle concept drifts present different challenges in the literature. Subsections 3.2.2 and 3.2.3 will cover related works regarding adaption and detection to concept drifts respectively. Finally, Subsection 3.2.4 will touch relevant metrics in the literature to measure the detection of concept changes.

3.2.2 Adaption to Changes

The main difference between incremental and adaptive algorithms is that the second group considers explicit strategies to forget irrelevant information. In fact, according to Gama et al. [144], adaptive learners can be interpreted as "*advanced incremental learning algorithms*" that can adapt to changes in a data stream. The evolving nature and speed of dynamic environments like high-frequency data streams produces a set of issues at the storage and learning stage in machine learning algorithms. In this type of non-stationary data, the underlying generative process of a time series can change over time; models trained on old data instances may reduce their performance under such changes. Hence, a priority in this field is to create mechanisms to handle and adapt to concept drifts [233] while still accounting for periods of stability. The authors in [33] provided a survey discussing research constraints and the state-of-the-art in different supervised and non-supervised learning in data stream mining.

The trade-off between cost efficiency and performance [181, 310, 385] is one of the most significant challenges in data stream mining. Online machine learning algorithms have additional requirements compared to offline learners, such as the need to process instances incrementally to avoid storing data for multiple passes. In terms of passive adaption approaches to concept drift, ensembles have been one of the mechanisms with the greatest predictive and computational performance in the relevant literature.

Ensembles naturally fit the purpose of distributed computing frameworks, being easily scalable to deal with massive data streams. However, the incremental nature of the online learning process, and especially of the techniques to handle concept drifts are purely sequential and become a computational bottleneck for base learners running in parallel. In these scenarios, an adaptive framework can benefit from a mini-batch strategy as proposed in [77]. Mini-batch approaches have been widely used in the literature of data stream mining to port offline state-of-the-art algorithms to work with dynamic environments or non-stationary data. These have also been used extensively in ensemble learning for data streams as will be covered later in Subsection 3.3.2. A drawback of mini-batch techniques is that, depending on the speed of the changes in the generative process of the stream, approaches with fixed-size batch sizes may not react in time to drifts. Thus, tuning the size of the batches becomes vital in these non-purely incremental approaches. Small sizes can help to adapt to abrupt drifts but can impact the predictive performance of the learners negatively during periods of stability and have a higher computational cost [73].

Training a model continuously has become a convention in the data stream mining literature, as this focuses typically on concept drifting data streams. However, continuous learning may only be a requirement in non-stationary scenarios, as other stationary processes could be handled by model reuse. Having said this, in an infinite data stream, there may be small non-stationary learnable patterns inside each state, and stationary states could also evolve continuously over time. Zliobaite in [385] proposed a framework to assess the utility of having adaptive learners in different prediction problems. Furthermore, each use case and domain may need different adaption strategies, but the manual development of a strategy is a time-consuming process. Bakirov et al. in [34] proposed a flexible mechanism to automate the development of adaption strategies. However, this is a very recent proposal; its use is not widespread and does not have enough competitors.

Another research area of interest for classification using online data streams are evolving intelligent systems (EIS) [23, 40, 197], or evolving fuzzy systems [235] [234] [236] [284]. These online and incremental systems are able to adapt themselves to concept drifts of different nature on-the-fly through adaptive fuzzy-rules [22]. EIS are based on fuzzy systems, which have already demonstrated their ability to solve different kinds of problems in various application domains like the financial one [95,225]. These have achieved great results classifying non-stationary time series [162, 286, 287]. Recent EIS approaches can work as ensembles of rules [21] and apply meta-cognitive scaffolding theory for tuning the learned model incrementally in what-to-learn, when-to-learn, and how-to-learn [306]. These have also introduced the ability to deal with recurrent concepts explicitly and have beaten other methods at predicting the S&P 500 [283, 286, 287]. For instance, Pratama, Lu, Lughofer, Zhang, and Anavatti [285], and Pratama, Lu, Lughofer, Zhang, and Er [286] employed an evolving type-2 recurrent fuzzy neural networks to learn incrementally and handle recurring drifts. In any case, there is still a significant gap between EIS and the rest of the literature for data streams classification. Hence, these have been left out of scope of this work.

3.2.3 Drift Detection

While the literature has proved that continuous adaption is a good mechanism to handle gradual or incremental drifts, in case of abrupt changes, incremental learners may need time to adapt due to the model trained already for previous instances of a prior concept. In these cases, the convention in this research field is to monitor either change points (or intervals [41]) in the data distribution or the predictive performance of a learner to quantify or characterise a drift. Different techniques to track these changes and detect drifts able to maintain the performance of a model under unknown changing conditions were presented in Section 2.3.4.2.

When drifts occur, algorithms in this field tend to completely or partially replace a model when a significant change is detected. Many online classifiers use drift detectors as a solution embedded in their approach. For instance, HAT [47], uses the detector ADWIN2 at each node of the tree, cutting branches if drift is detected, and these will grow again when learning new data instances from a new concept. J. Lu and A. Liu et al. in [233] reviewed the state-of-the-art in concept drift detection and adaption, unifying the general framework being used by most works in the literature for this purpose. This, made of four different stages, is illustrated in Figure 3.2.

1. Retrieval of data stream instances (both historical and new).
2. Data pre-processing and modelling.
3. Test statistic calculation.
4. Hypothesis post-hoc tests.

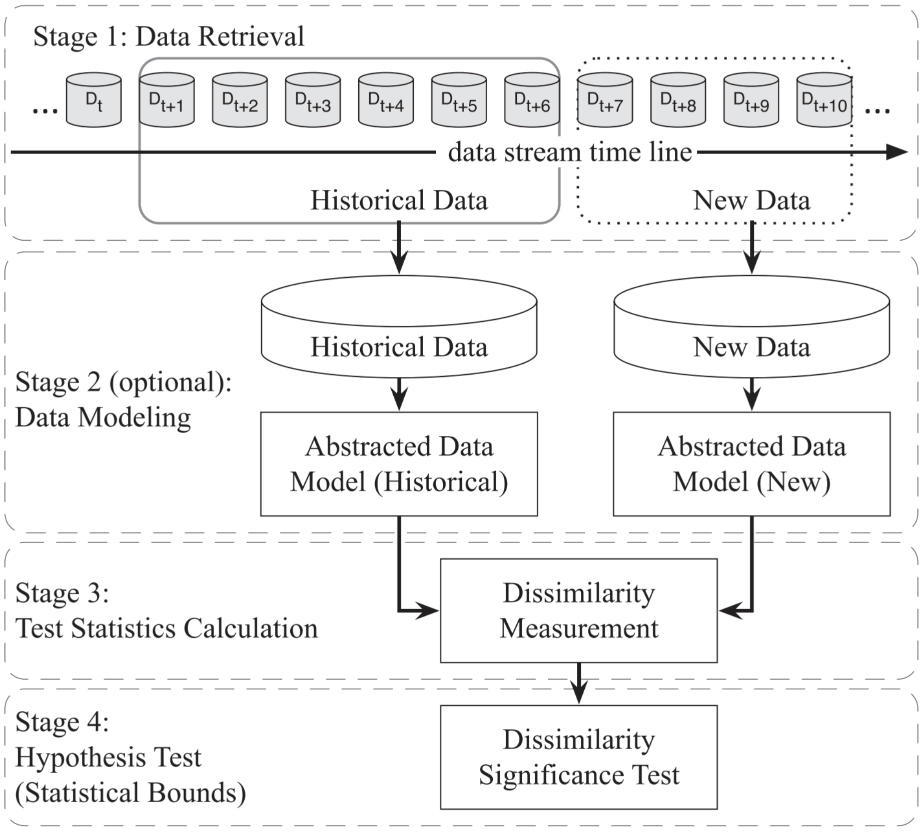

Stage 1: Data Retrieval
...
D_t
D_{t+1}
D_{t+2}
D_{t+3}
D_{t+4}
D_{t+5}
D_{t+6}
D_{t+7}
D_{t+8}
D_{t+9}
D_{t+10}
...
data stream time line
Historical Data
New Data
Stage 2 (optional): Data Modeling
Historical Data
New Data
Abstracted Data Model (Historical)
Abstracted Data Model (New)
Stage 3: Test Statistics Calculation
Dissimilarity Measurement
Stage 4: Hypothesis Test (Statistical Bounds)
Dissimilarity Significance Test

Figure 3.2: *Example overall framework for concept drift detection, from [233].*

The fast growth of research works highlighting the importance and proposing new drift detection mechanisms has recently triggered the appearance of many works surveying and benchmarking the main explicit drift detectors. The authors of [15,20,86] benchmarked the impact of different detectors such as DDM across different state-of-the-art incremental classifiers.

Gonçalves et al. in [160] and Pesaranghader and Viktor in [279], performed a comparison of the state-of-the-art concept drift detectors. In both works, the drift detection method (DDM) presented an overall good performance. A summary of the best performers in their experiment is illustrated in Table 3.1. Barros et al. in [39,107], proposed the drift detector RDDM described in Section 2.3.4.2. They benchmarked different concept drift detectors as auxiliary methods in ensembles in terms of final predictive accuracy under abrupt and gradual concept drifts. Their method (RDDM) and HDDM$_A$ were the best detectors overall depending on the type of drift (gradual or abrupt) and the base classifier used (naive Bayes or a Hoeffding tree).

Methods	**Metrics**							
	Abrupt accuracy	Gradual accuracy	Real-world accuracy	Overall accuracy	Execution time	DT	FA	MD
ADWIN2	⇓	⇑	⇑	⇑	⇑	⇓	⇑	⇓
DDM	⇑	⇑	⇓	⇑	⇑	⇑	⇑	⇑
EDDM	⇓	⇓	⇓	⇓	⇑	⇑	⇓	⇓

Table 3.1: *Summary of the comparison of classification accuracy in data streams with abrupt, gradual drifts and real-world data by Gonçalves et al. [160]. MD: mean of examples seen until drift detection; FA: false alarms; MD: miss detection rates.*

Something in common between all of these studies is that the performance of these detectors over a classifier depends on the nature of the data stream itself and the base classifier used. For this reason, and since HT and NB are the main base learners used in the previous surveys and benchmarks, in this thesis, we will use these two as base classifiers. Chikushi et al. in [86] performed a benchmark of detectors across classifiers for different datasets and concluded that every type of data needs needs its own evaluation process to decide a suitable drift detector. De Mello et al. in [108] proved that, depending on their application fields, different data streams may also need different metrics for concept drift detection.

In this thesis we aim to use different types drift handling mechanisms to deal with gradual, abrupt changes to a new or a recurring model. While these detectors can be used to deal with changes in the market behavioral structure [37,118,142,144,290,365], we must recognise the scientific evidence in this section. Thus, since none of these studies has been performed in financial data to the best of our knowledge, in Chapter 5 we will evaluate what detector and parameters to use across the main techniques from the state-of-the-art. Each drift detector may identify different patterns and make base classifier to require a different time window to recover from a concept drift. The relevant literature regarding this comparison to be performed and the different metrics involved in this recovery phase will be explained in more detail in Subsection 3.2.4.

3.2.4 Metrics to Measure Drift Detection Performance

Recently, the literature has shown that despite the good performance exhibited by many supervised drift detectors, the error rate of the learners that these detectors monitor can be based on temporal dependence [44, 384]. There can be adaptation errors in these scenarios if this temporal dependence is ignored, leading to reduced predictive performance due to suboptimal decisions regarding what base learners to use for predictions [165]. Hence, in cases of sharp changes where passive adaption is not enough, many recent approaches to handle drifts considering temporal dependence have been proposed [99, 100, 352].

Indeed, the idea of perfect change detectors lies in the ability to accurately identify each time than the data stream generative process changes. In practice, all of the studies presented in the previous subsection tend to underestimate or overestimate the number of changes in a stream, with a higher or lower degree of false or true positives. The task of concept drift detection does have associated costs, both in predictive accuracy and computational. This has motivated researchers to propose different metrics to measure the accuracy of the detection versus the ground truth and the associated costs while a model retrieves from a change.

To handle concept drift in real-world data, detectors face the problem of ignoring when an actual (ground truth) change occurs and for how long. Furthermore, there is no real baseline of what should be the predictive accuracy of a learner if the concept drift does not happen. Hence, it is not feasible to estimate if a learner has *recovered* from a drift. To do this effectively, the ground truth changes in the data stream must be known. In many domains, the most reasonable way to know this is through a controlled experiment generating synthetic datasets, that simulate ground truth changes. This gives warranties of the real switches allowing the evaluation of the change detectors. In this scenario, all changes signalled by a drift detector should be counted as *false alarms* if these occur before their true change. Apart from this, two metrics that should be considered would be the number of times a model is replaced accurately and if this occurred at the right time. The design of a concept drift detector in the relevant literature is described as a trade-off between maximising the true changes and minimising both the false alarms detected and the recovery phase of the underlying classifiers.

A drift detector adds extra complexity to an algorithm. Thus, its addition should be reflected as a faster adaptation process or increased predicted accuracy. Zliobaite in [385] analysed how depending on the scenario or the problem to solve, the cost to detect or react to a concept drift may not be worthwhile. Metrics as the duration of the recovery phase and maximum performance loss, proposed by Shaker and Hüllermeier in [312], may suit this purpose.

Shaker and Hüllermeier in [311] applied a *survival analysis* method to recognise dynamic events in data streams. Survival analysis (also known as 'time to event' analysis) represents a set of statistical tools used to know when a particular event will occur. Inspired by this work, Shaker and Hüllermeier introduced *recovery analysis* [312] one year later, which is an experimental protocol and a graphical presentation of the learner's performance in a data stream. They also proposed measures to maintain the quality and generalisation performance of the models. Their protocol aimed to estimate the inherent delay recognising changes and the recovery time; being this last when will the system recover from an event. Finally, synthetic data has the risk of being idealised and unrealistic. Hence, many approaches from the literature, such as the work by Shaker and Hüllermeier [312] try to produce semi-synthetic datasets by simulating many data streams and changes in the generative processes over time. In Chapter 5 we will propose an approach inspired by this work to generate synthetic-like data using financial data generative processes.

Figure 3.3 depicts an example of the graphical process described in Shaker and Hüllermeier [312]. Instead of using a single data stream, their proposal worked with three data streams in parallel: two "*pure streams*" and one "*mixture*" stream.

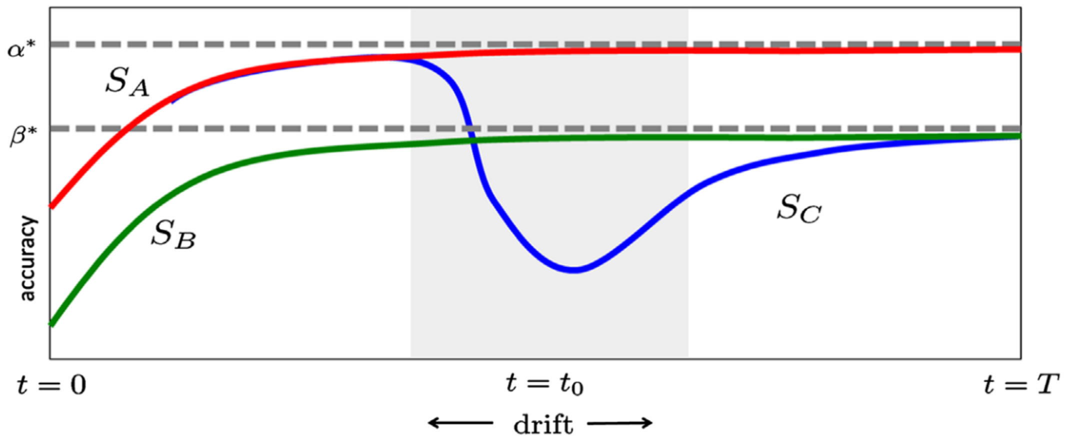


Figure 3.3: *Schematic illustration of the recovery analysis graphical process. Extracted from the work of Shaker and Hüllermeier [312].*

The three lines in Figure 3.3 represent the performance of a model these three data streams. The darker grey area represents the time window when the concept drift occurs. S$_A$, S$_B$ represent the performance of a model trained in two data streams representing the two different concepts. S$_C$ represents the performance of a model trained in a data stream that drifts from concept A to concept B. It can be seen that from the start of the drift, there is a recovery phase until $t = T$ when the performance of the model trained with the concept drifting stream converges with the performance of the model trained only with S_B. The recovery analysis protocol proposes metrics to estimate this recovery's length (t) and the maximum drop in S_C after a drift.

Shaker and Hüllermeier [312] proposed mainly two metrics to measure drift detection: i) the *duration* of the recovery phase (suboptimal performance of the algorithm) and ii) *maximum performance loss* (max. error peak). Other authors that have proposed metrics in this regard are Bifet and Zliobaite.

Bifet in [44] proposed different metrics to measure false alarms in drift detection. Some of these were *mean time between false alarms*, *missed detection rate* (accounting for the non-detected changes) and *mean time to detection* (detection delay). The metrics proposed by Bifet [44] are mostly focused on the actual performance of detectors, which is not the purpose of our research. Nonetheless, some of the metrics that will be presented in Chapter 5 used to measure the reliability of drifts in the final classification results of our experiments are based on his work. In any case, for most of the drift handling related experiments, our research will make more emphasis on the recovery analysis approach.

Zliobaite in [385] defended that a model should adapt to drifting concepts if the improvement over the error exceeds the cost of the resources required for such adaptation. There is a cost associated with the retraining and the update of a model (e.g. the cost of not predicting the market trend on time for being training a clustering algorithm at the time of a drift). Inspired by this, Zliobaite in [385] proposed RAM-hours to measure the performance of a model.

As mentioned at the end of Subsection 3.2.3, this thesis will benchmark different drift detectors in financial data. As done in many relevant approaches from the literature [16, 312], we will propose a framework to produce financial-like data to produce a controlled environment to rate concept changes accordingly. The comparison of drift detectors will be performed at two levels. First, classification accuracy will be used to select the top detectors as in related research works [15]. Then, we will propose a set of metrics to measure change detection that is inspired by the work covered in this subsection. Finally, RAM-hours will be used as a computational performance metric to report the final results of all models.

3.3 Supervised Learning under Concept Drift

3.3.1 Introduction

Chapter 2 described the theoretical requirements and limitations of online incremental machine learning algorithms for data streams. Online algorithms need to be single-pass (or one-pass), be always up-to-date and ready to predict, and need to handle changes in data streams [340]. Thus, when trying to adapt offline machine learning algorithms to work in an online setting, a common approach is to fill a buffer with the incoming data instances and train them using in a mini-batch setting to support (batch) incremental learning [144]. An example in this regard is OISVM [382], proposed as a mini-batch approach for support vector machines (SVMs) that we will cover in more detail in Chapter 4. SVMs [171] [358] have proved their performance in the literature dealing with non-linear and complex datasets and have been widely and successfully used in the financial domain for price trend prediction [275].

N.A. Syed et al. [333] proposed an incremental version of SVMs that dealt with the SVs of previous batches. However, that algorithm needed to be fully re-trained at every batch to pick which support vectors should be considered at the next batch. Cauwenberghs et al. [78] proposed another alternative for incremental learning and decremental unlearning using SVMs that has not been widely accepted in the literature [332]. Stefan [301] proposed a weighted model where the oldest SVs has a greater cost to the model, facing then changes of the feature space across time. Laskov et al. [216] proposed another online algorithm based on support vectors. However this approach suffered from a high computational cost. In Chapter 4 we will propose an incremental version of SVM using mini-batching.

In this thesis we will also use purely incremental algorithms [144, 185] such as Hoeffding trees (HT) and naive Bayes (NB) models presented in Chapter 2.

In Chapters 5 and 6 we will use them as base classifiers. However, the focus of this thesis goes beyond supervised classification, and we will use different techniques to handle different types of concept drifts such as for ensembles and meta-learning. The relevant literature in the field of data stream mining for ensemble approaches and meta-learning will be reviewed in Subsections 3.3.2 and 3.4 respectively.

3.3.2 Online Ensembles

In evolving data streams, changes may be followed by stable periods of different duration. Ensembles can store a set of weak learners trained during different periods, which makes them suitable techniques to adapt to concept drifting data streams [298]. These systems were described in Chapter 2 and introduced for adaptive learning in Section 3.2.2. In fact, offline ensembles are known for their good results predicting both cyclic, and non-stationary data such as stock prices [36, 275, 276], and in the last years, many incremental ensembles have been proposed in the data stream mining literature [207] to deal not only with stationary data and recurring drifts but as well with non-stationary data in evolving data streams [124, 156, 177, 178, 196, 205, 221].

Gomes et al. in [151] proposed a taxonomy for data stream ensemble learning derived from reviewing the most relevant approaches at that point in time (see Figure 3.4) and covering aspects like the aggregation of predictions, methods to achieve diversity, and type of model updates. In this thesis, our main approach will be a meta-learner with only one active classifier at a time. However, in preliminary work (Chapter 4), we will propose an ensemble that, following this taxonomy, will use adaptive windowing, weighted voting and a flat architecture. A full analysis of how the relevant literature falls into this category can be seen in [151].

According to Gomes et al. in a later work [154], ensembles for data streams traditionally could be divided into two groups depending on their approach to handle concept drifts:

- *Passive ensembles* (reactive) are updated continuously and assign weights to base models depending on their latest or accumulated predictive accuracy. Two examples of passive ensembles from the literature are the *streaming ensemble algorithm* (SEA) [325], *dynamic weighted majority* (DWM) [205] and the *accuracy updated ensemble* (AUE) [73].
- *Active ensembles* apply drift detection algorithms to reset weak learners. An example of these is ADWIN bagging, which combines ADWIN2 [47] and online bagging [270].

Some more recent works combining both approaches are the adaptations of random forest (RF) (introduced in Chapter 2) and random patches [232] algorithms; *adaptive* *random forest* (ARF) [152] and *streaming random subspaces* (SRP) [153, 155] respectively. These models can weight different weak learners based on their past performance and replace them when a drift detector specific to their base learners detects a change.

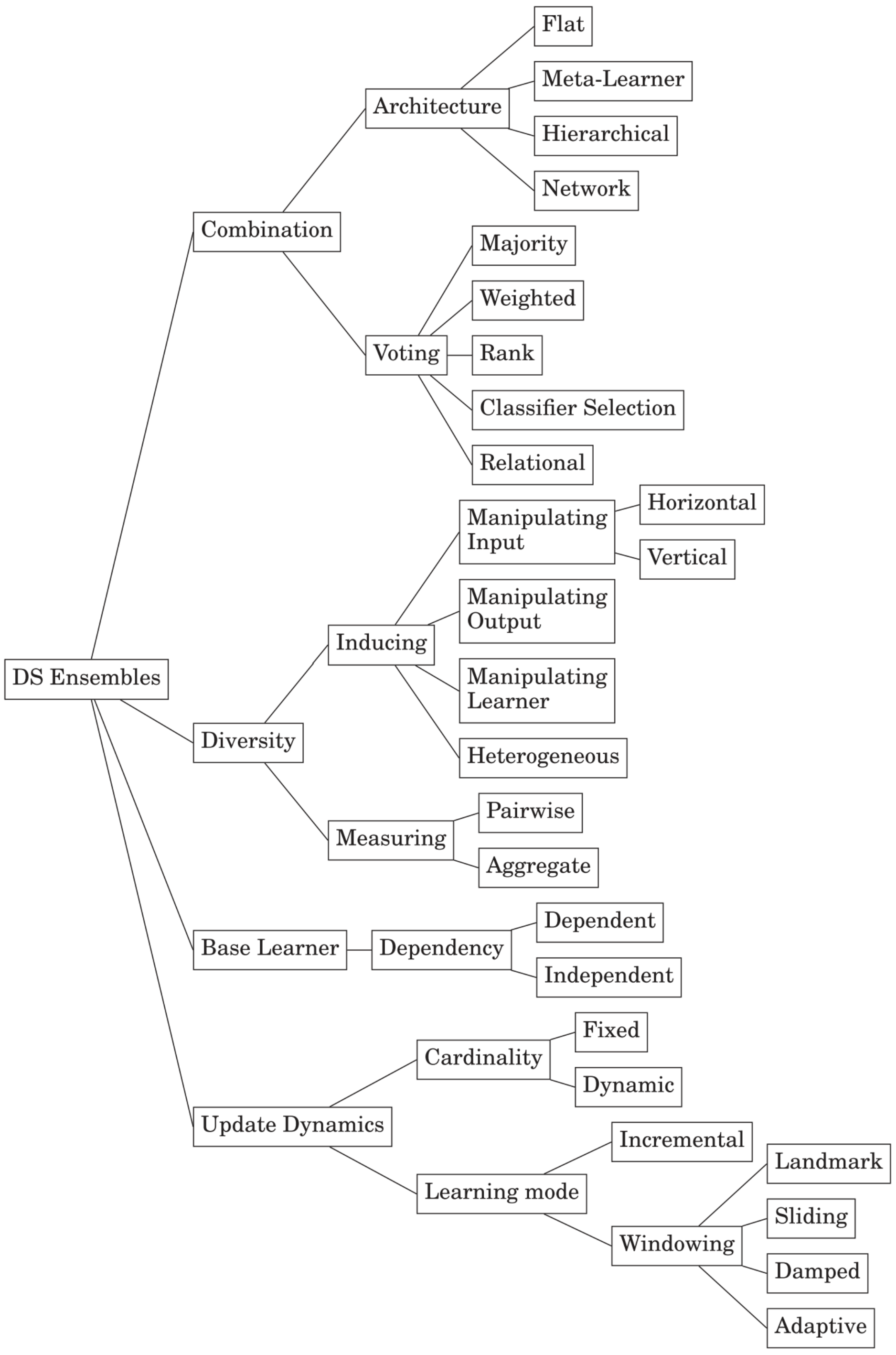


Figure 3.4: *Taxonomy of data stream ensemble classifiers. Extracted from [151].*

ARF updates its base trees continuously (HTs by default). However, if a warning is detected, it starts training a new tree in the background only with new incoming data instances. If a drift is signalled, the background tree becomes the new active tree, and the old one is forgotten. ARF trains its HTs using a resampling algorithm based on online bagging [270] and uses ADWIN2 as a drift detector, having thus some similitude with ADWIN Bagging. It uses boosting to train classifiers iteratively and increases the weight on instances that have been misclassified. Base learners (trees) are weighted using their prequential error. The main difference between the ARF and SRP algorithms is the logic applied to random subspaces in their base learners. While ARF applies them at every base tree independently (locally), SRP ensures a global subspace strategy that increases diversity across weak learners.

Another approach using boosting was proposed by Montiel et al. in [258], who adapted XGBoosting for data stream mining, namely adaptive XGB (AXGB). However, their approach was a block-based (mini-batch) ensemble and not purely incremental like ARF and SRP.

3.3.3 Block-based Ensembles

In block-based ensembles, base learners are trained with a batch of data of a fixed length. Most of these approaches create new ensemble members with new batches, setting maximum ensemble sizes and policies to update or replace the current base learners.

AUE, DWM, Learn++.NSE [124] and the recurring concept drift framework (RCD) [159] are examples of some of the first block-based ensembles for data stream mining. RCD [159] considered the detection of warning signals before drifts and incorporated the idea of background models that start training in parallel while the predictions are performed by a classifier already trained (active or foreground classifier). The addition of background base learners helps online algorithms to shorten the duration of their recovery phase and lower their maximum performance loss. This is something that has been included in many approaches thereafter like ARF [16, 152].

Block-based ensembles (see Figure 3.5) are, in general, passive approaches. Thus, as mentioned in previous sections, this type of adaption is not suitable to cope with abrupt drifts since they will adapt slowly to those changes, having out-to-date base learners and weights for the global prediction. For this purpose, RCD uses DDM to detect warnings and drifts. AXGB uses ADWIN2 (as ARF). Block-based ensembles suffer from the main dilemma of other mini-batch algorithms introduced in this chapter. Small blocks can help to react to abrupt drifts, but this has a computational cost and may damage the predictive accuracy of the ensemble in periods of stability [73]. Thus, tuning the block size is of vital importance.

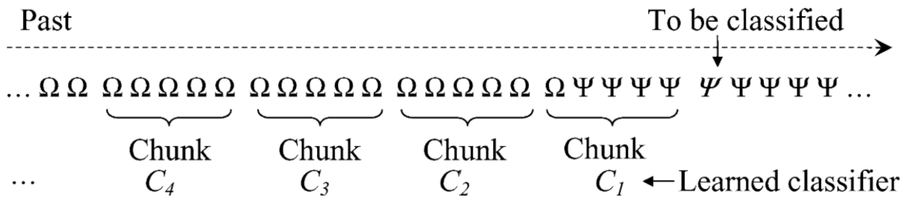


Figure 3.5: *Example training process of a block-based reactive ensemble. Extracted from the work of Yang et al. [376].*

3.3.4 Base Learners for Recurrences and Seasonalities

ARF [152] is nowadays one of the state-of-the-art methods in data stream mining, and it incorporates most of the mechanisms described in this subsection. However, a drawback of this approach is that it lacks an explicit mechanism to deal with concept recurrence or seasonalities in a data stream. Many different ensemble learners like Learn++.NSE and DWM propose a robust mechanism to deal with recurrent concepts since base learners are not updated after being inserted into the ensemble.

Approaches like ARF constantly train all active base classifiers, which may make base learners evolve and forget the previously learned concept (catastrophic forgetting) before this concept reoccurs. ARF also discard trees when a drift is detected, so these need to be trained from scratch if a concept reoccurs. For this reason, many approaches in the literature have presented the idea of a concept history to store a cold copy of previously learned base classifiers to be reused if they become relevant again in the future. The idea of a concept history adds extra challenges to identify what concept is present in the data stream at each time. Different approaches have been proposed for this purpose, like conceptual equivalence and concept similarity), described in Chapter 2. The concept history is not an item exclusive of online ensembles. It can also be used for single learners that follow a meta-learning approach to change the base learner for different concepts (see Figure 3.6). Thus, we will extend further about this topic in Subsection 3.4, dedicated to meta-learning for data streams.

3.4 Meta-learning and Detection of Recurrences

3.4.1 Introduction

As mentioned earlier in this chapter, drifts detectors collect a series of statistics to signal changes in the underlying data distribution or in the classifier performance over time. This collection process implies a delay between the start of the current drift and the time when this is detected, namely the recovery phase as seen in Subsection 3.2.4.

There are two major challenges to reduce the duration and maximum performance loss of the recovery phase: i) to anticipate when will the next drift occur and ii) devising what concept will be the next to ensure a faster adaption [370].

As mentioned in Subsection 3.3.2, many approaches try to reduce the impact of the recovery phase by incorporating a warning detector. However, signalling warnings still implies delay from the actual change point since these are based on the same data collection process only using a more sensible parametrisation (e.g. higher confidence intervals in ADWIN2). Models still need to be retrained from the warning detection point, increasing the computational cost of the training process and not being an effective solution in case of very sharp changes, when a drift will be recognised a few data instances later. In this case, the learner will have a short warning window where the background learners will not be trained with a representative sample of the current concept.

If the data stream presents stationarities or seasonalities at some point in time, a way to alleviate this problem is through model reuse. In general, machine learning frameworks doing this tend to assume that discrete concepts exist. In the last years, many research works have approached the problem of recurring concepts by reusing models trained previously [376].

Two relevant learners from the literature that incorporate the reuse of previous base learners are RCD [159], introduced in the previous subsection, and the *concept profiling framework* (CPF) [14]. These methods, independently of their number of base learners (one or many), act as a wrapper to decide at each time what is the best algorithm to make predictions and allow the use of any base model and detector, being thus meta-learners (already introduced in Chapter 2). Both RCD and CPF will be used in this thesis to compare the performance of our ensemble and meta-learner in Chapters 4 and 5 respectively. The objective of both is to improve classification accuracy, as for many other relevant methods such as *leveraging bagging* [50], and AUE [73], but this does not necessarily imply improving the detection of drifts as covered in Subsection 3.2.4.

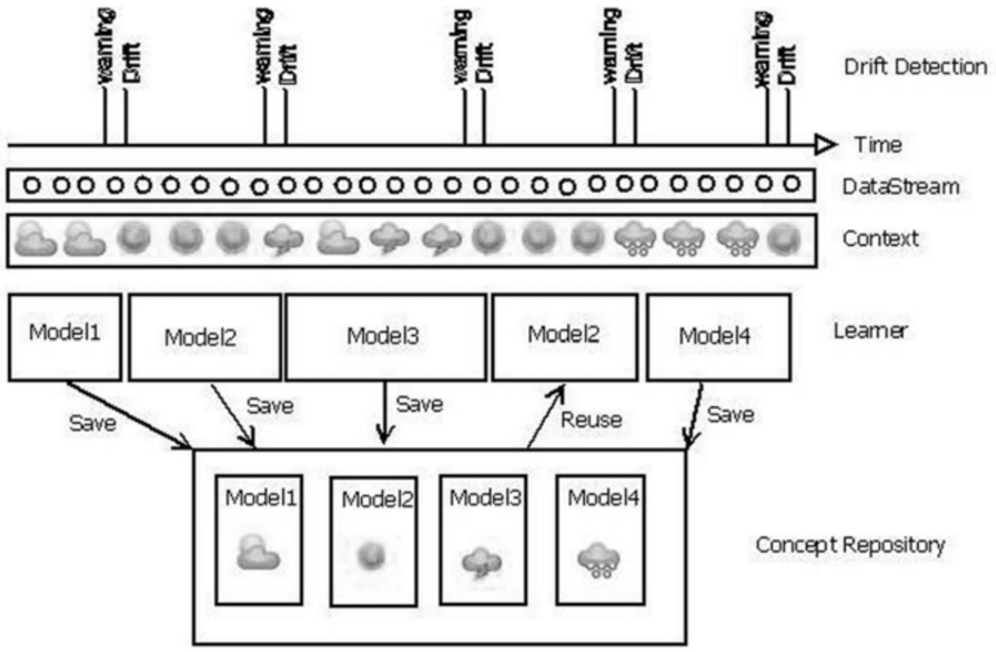


Figure 3.6: *Example framework using a concept history, extracted from [157].*

Many meta-learning approaches aim to represent concepts using non-supervised techniques to describe the current state of the stream and predict potential changes. In this regard, different research works have proposed different mechanisms to represent and measure distances among concepts. Meta-learners generally have a higher number of parameters to fine-tune since these may combine different approaches and compare learned models. For this reason, recent research works are starting to propose methods for continuous parameter tuning in non-stationary data streams to cope with this [33, 361].

To the best of our knowledge, there was a lack of literature reviews regarding meta-learning approaches for data streams at the time of writing this thesis. This fact motivated the review provided in the following subsections.

3.4.2 Drift Detection in Meta-learning

Most meta-learning frameworks from the literature evaluate models to be reused only when a change is detected. This allows these frameworks to leverage offline models, using mini-batch approaches and reusing previous models when relevant. For instance, Trajdos and Kurzynski in [343] incrementalised an offline classifier using the detector ADWIN2 to signal drifts. In any case, the majority of the new approaches from the literature work over adaptive base classifiers since these can learn gradual changes in the data stream without the help of any explicit drift detection mechanism.

Angel et al. [252] proposed a meta-learner that used hidden Markov models (HMM) to predict the sequence of change between discrete concepts. Their approach, which used fuzzy logic rules to compare classifiers for model reuse, was not able to deal efficiently with incoming data streams. Maslov et al. [244] proposed a method to use patterns acquired during previous changes and assumed a Gaussian distribution for the duration of the changes to predict the time of the next change point.

A similar approach was proposed by Chen et al. [85]. Their method predicts future changes using a probabilistic network using previous drifts. Their proposal was independent of drift detection methods and relies on volatility patterns in the data stream. ProChange, in [201] also used volatility patterns during changes and a probabilistic network to predict different types of drifts in unlabelled transactional data streams. More recently, the authors of Nacre in [370], a meta-learner with active drift detection for data streams, proposed a method called drift coordinator to anticipate change points assessing each concept.

The literature of meta-learning and ensemble learning for data streams is closely related. As mentioned earlier in this section, Gonçalves et al. [159] proposed the ensemble RCD with active drift detection and a history of previous models to handle recurring concept drifts. The evaluation of what model to retrieve is based on the data distribution of the incoming stream. For this purpose, a sample of the data received is stored in a buffer for each of the classifiers and is compared to the incoming stream in case of concept drift. Sakthithasan and Pears in [304] applied discrete Fourier transforms to decision trees to capture recurring concept drifts. The evaluation for model reuse was performed by comparing a compressed version of the learned trees.

3.4.3 Explicit Handling of Recurrences

Elwell et al. [124] dealt with recurrent concepts using a block-based reactive ensemble that did not limit the number of base learners. The authors claim that Learn++.NSE trained one concept per batch received. The algorithm employs a weighted voting mechanism using each individual classification accuracy. In the case of recurring concepts, it is expected that the weight of the base learners representing that concept will increase and hence the global prediction will take into consideration old but relevant knowledge.

Conversely, base learners would reduce their weights if their predictive error increases, not being considered for predictions when these do not match the current concept. Like other similar approaches that came after as other similar approaches that came after [8, 178], the idea of Learn++.NSE is to keep all the learned knowledge in a pool of classifiers, either active (as a part of an ensemble) or inactive (stored as a concept history), to be used in the future when they become relevant. Other meta-learning ensemble approaches of the same kind have been proposed to determine dynamically a suitable ensemble size [121, 281].

The use of a concept history became popular about a decade ago and has received different names like pool of classifiers, concept list and concept repository. Approaches like [7, 156, 157, 221, 376] proposed the explicit handling of drifts using different techniques to evaluate the relevance of historical concepts. Yang et al. introduced the idea of a concept repository and the idea of conceptual equivalence in [376] in their ensemble classifier RePro. Their method used a Markov chain to learn concept transitions. Figure 3.6 illustrates one of the first approaches targeted to model reuse in data streaming which has a flat structure. However the idea design of a concept history can follow different architectures. For instance, Sidhu and Bhatia [318] proposed a recurring dynamic-weighted majority (RDWM), which alternates two ensembles, one with active learners and another working as a pool of historical models.

Ahmadi and Kramer [6] presented the GraphPool framework, which maintained a pool of historical concepts and kept transitions between concepts using a first-order Markov chain to allow model reuse. Like Learn++.NSE, their approach is considered that every batch of data received would represent a new concept, which is not the case in every data streaming application. In the GraphPool framework, concepts that are similar among them would be merged within the history. Wu et al. in proposed the ensemble algorithm PEARL [371] as an extension of ARF that uses a probabilistic graphical model and lossy counting [241] for model reuse when a drift is detected.

A very relevant meta-learner from the literature is the *concept profiling framework* (CPF), proposed by Anderson et al. in [14]. This framework, which has only one active base classifier at a time, handles recurring concepts explicitly using a concept history and evaluating previous models to be reused when detecting a concept drift. CPF is illustrated in Figure 3.7. In CPF, new models are only inserted into the history in case of drift if these do not perform similarly to any historical model. Historical models are compared using a conceptual equivalence approach, using classification accuracy.

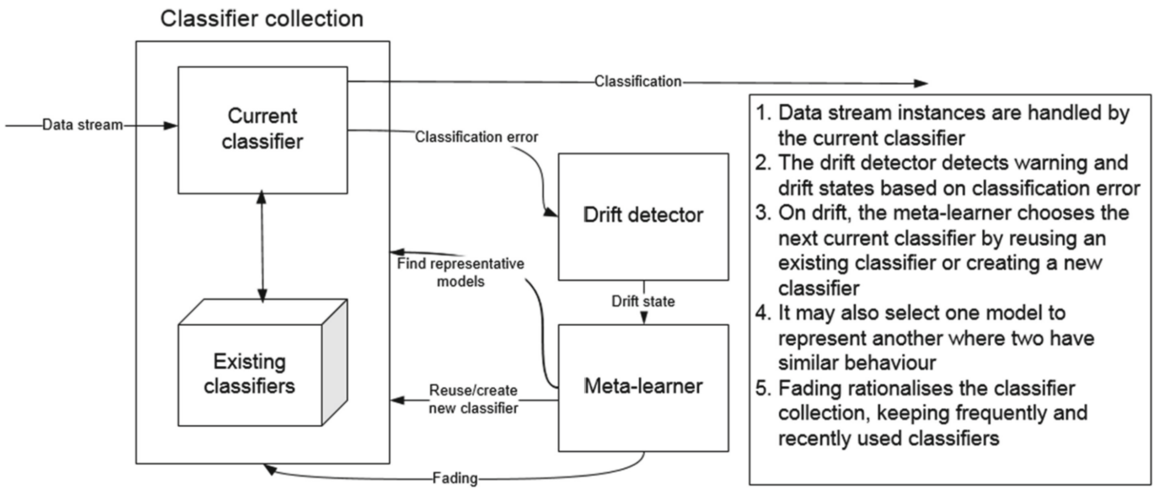

Classifier collection
Data stream
Current classifier
Classification
Classification error
Drift detector
Find representative models
Drift state
Existing classifiers
Reuse/create new classifier
Meta-learner
Fading
1. Data stream instances are handled by the current classifier
2. The drift detector detects warning and drift states based on classification error
3. On drift, the meta-learner chooses the next current classifier by reusing an existing classifier or creating a new classifier
4. It may also select one model to represent another where two have similar behaviour
5. Fading rationalises the classifier collection, keeping frequently and recently used classifiers

Figure 3.7: *The Concept Profiling Framework. Extracted from [14].*

To limit and maintain the size concept history over timeCPF, CPF prunes historical models using a mechanism called *fading*. This mechanism prunes old models depending on how frequently have these been reused, and it gives more importance to older models than recent ones. This is a design choice that may help domains with a finite number of non-evolving concepts. However, it may not suit many real-world data streams which concepts may evolve. Perhaps due to this, CPF obtained state-of-the-art results in synthetic data with clear recurring patterns but could not outperform other methods such as RCD in real-world benchmarks.

The authors of CPF suggested setting a high-similarity threshold (m) for comparing learners with the history. Lower similarity thresholds may lead CPF to have a smaller selection of more general classifiers. At the same time, CPF tries to make historical learners to differ as much as possible from each other.

A problem of CPF is that it relies on a fixed-size buffer of instances to determine what model to reuse. To avoid the high computational cost of this task and improve the scalability of CPF, Anderson et al. in [16] proposed *enhanced CPF* (ECPF). ECPF saves copies of the reused classifiers instead of only the original ones as CPF, which allows concepts to evolve over time. Furthermore, background learners start being trained when a warning is detected (as in ARF) to replace the active learner in cases where no historical models represent the new concept.

3.4.4 Concept Clusters in Meta-learning

Another representative set of online meta-learners is the one focused on an unsupervised representation of concepts and the use of different distance metrics for concept similarity [157, 221, 248, 252, 372, 383]. Several of these methods are, in fact, supervised but have a non-supervised representation of the concepts. For instance, the semi-supervised learning tree-based ensemble REDLLA was proposed by P. Li et al. [221] for recurring changes in data streaming environments with limited labelled instances. Their approach uses k-means and introduces the idea of concept similarity and *concept clusters* introduced in Chapter 2. Figure 3.8 illustrates the concept clusters from this proposal and the idea of a cluster radius to set distance thresholds in the evaluation of similarity.

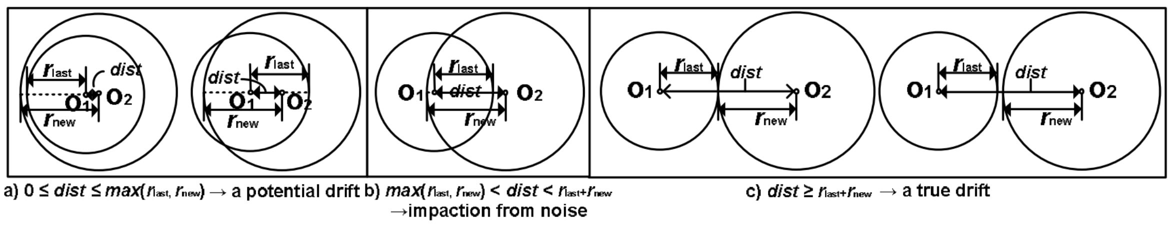

r_{last}
$dist$
O_1
O_2
r_{new}
a) $0 \leq dist \leq max(r_{last}, r_{new})$ → a potential drift
b) $max(r_{last}, r_{new}) < dist < r_{last}+r_{new}$ →impaction from noise
c) $dist \geq r_{last}+r_{new}$ → a true drift

Figure 3.8: *Cases of concept drift using the radius of a concept cluster in [221].*

The authors in [220] proposed a block-based ensemble model mixing both supervised and non-supervised techniques. Their ensemble counted with a concept history of unlimited size and was aimed to data stream classification. However, it also trained a cluster with each batch received. One of the novelties of their approach was a concept drift detection method based on the divergence of the clusters among batches. Kataris et al. presented in [198], another reactive block-based ensemble that used conceptual vectors to represent each batch to approach the idea of a concept history. They used an incremental clustering algorithm to group these conceptual vectors and generate concepts. Gomes et al. in [157] and later in [156] proposed to use two data streams in parallel. The second data stream used, counted with variables created by the user specifying context, as a proxy to have a certainty of the ground truth changes. Learners were created over time in their approach or reused from a concept repository depending on the similarity of their contextual information. Similar methods were proposed in [252] and [248] considering the idea of warning windows, later used in ARF and ECPF to train background classifiers. The approach of Gomes et al. in [157] was already illustrated in Figure 3.6.

The number of research approaches mixing unsupervised learning and meta-learning techniques to handle recurring concepts is indeed increasing. Many of these approaches are ensemble learners, which can be purely incremental or block-based. CONDOR [381] was a block-based ensemble compared to DWM by their authors, but with different weight update strategies and the addition of a meta-learning approach to reuse update previous models inside the pool of classifiers. Namitha and Santhosh in [262] presented another cluster-based method to handle recurring concepts in data streams. However, their approach, that used *clustream* [5] as base clustered, and performed unsupervised drift detection was entirely unsupervised, thus not being comparable to the rest of the approaches from this section.

Sun et al. in [331] and Chiu and Minku in [88] proposed ensembles that incorporate concept clusters to limit the ensemble maximum size using a diversity measure. The motivation behind these approaches was that a diverse pool of learners could be more likely to keep a set of representative learners over time with considerably different concepts, which should help in the case of model recurrence.

After this, Chiu and Minku in [89] proposed a similar ensemble using Euclidean distance distances for concept similarity, to handle multiple types of drifts. They intended to maximise the diversity of the ensembles having concepts that are distant among them. Their approach, namely CDCMS, only created new models in the ensemble beyond a dissimilarity threshold, and concepts were represented using the *expectation maximisation* (EM) algorithm.

As seen in this section, handling recurrence concepts is a research topic that converges both with the literature of ensemble learning approaches for data stream mining and with meta-learning methods. In the last group, non-supervised learning covers particular relevance to represent concepts and identify the current state of the ground truth. This gives advantages like bringing the ability to predict changes and facilitate the identification of previous concepts or transition sequences for model reuse. Non-supervised learning is a broader subfield that goes out of the coverage of this section. The use of these approaches as part of a meta-learner is a discussion that we continue in 3.5.

3.5 Model-Based Clustering under Concept Drift

The problem of concept drift is a data stream mining specific topic, and it involves different challenges storing, pre-processing and learning from data stream instances. In a non-supervised setting, the number of clusters , their densities, sizes or shapes can evolve due to different non-stationarities in the incoming stream. Recently, Zubaroglu and Atalay in [386] provided a comprehensive review on data stream clustering algorithms and analysed the non-supervised methods, computational complexity and predictive accuracy of these approaches.

The motivation to use non-supervised learning in this thesis can be seen in the discussion started in Subsection 3.4 about the inclusion of non-supervised methods as a support for model reuse in data stream classification.

Back in 2001, Wagstaff et al. and two years later Xing et al. in [373] proposed methods for clustering with similarity information using side-information (a *context*, as seen in Subsection 3.4). Many recent research works have approached the problem of time changing (and recurring) concept representations in a streaming setting using data stream clustering or deep clustering methods [116,262,379]. In these, micro-clusters or latent features would be used to make a synopsis of the incoming instances and reduce the computational cost of finding similarities among data distributions.

However, the problem of representing a concept (or model) using non-supervised learning started becoming popular with model-based clustering approaches. These algorithms find a model to fit best to the input data and are robust to noise [161, 246]. Section 3.4.4 introduced some research works using the model-based clustering method *expectation maximisation* (EM) [112] to improve model reuse. This algorithm fits a mixture of Gaussian distributions to the data [76]. Chiu and Minku in [89] used it in CDCMS to create concept representations and keep a diverse ensemble learner. Zheng et al. in [383] used it to minimise the intra-cluster dispersion and cluster impurity. Tsang and Chen in [347] applied the Baum–Welch algorithm, an especial case of EM, to both detect the time of a change point and predict the next state (or concept) of financial data using an HMM. Gomes et al. in [154] also hypothesised about using Baum–Welch in conjunction with HMMs for data streaming scenarios. In any case, Baum–Welch is not an online approach. Still, it has been used in the financial domain together with other specific versions of EM and Gaussian mixture models (GMM) to forecast change direction in stock prices [274, 380] and to represent market regimes [114,170,209,347]. Although incremental versions of the EM and Baum-Welch algorithms have been proposed [103, 363], one of their major disadvantages is their assumption of normally distributed data instances. This can cause many challenges in complex domains or deal with non-stationarity distributions where changes may not be foreseeable.

An alternate approach to represent different concepts in the literature is through the use of data partitioning methods. These, also unsupervised, can be compared to model-based clustering or other techniques with micro-clusters since all of these are able to summarise a data distribution into a set of locally optimal structures. For instance, Angelov in [21] proposed several ensemble algorithms that have a (rule-based) model optimised per data cloud (see Figure 3.9). They used the term *data cloud* to refer to a set of prototypes identifying a concept and created an autonomous approach to partition the data using their data clouds.

As presented in the book by Angelov [21], the machine learning literature has many approaches to interpreting the state of data distributions that do not necessarily need to be non-supervised. An example of this is instance-based learning methods such as nearest neighbour based algorithms (introduced in Chapter 2). These can be seen as a particular type of prototype-based classifiers [278] which requires the entire dataset to be in memory. A more conventional example of a prototype-based classifier could be the SVM algorithm. Support vectors can be considered prototypes reduced from the input data; only these are needed to classify. As a first approach in this thesis, we will use support vectors as a mechanism to transfer knowledge in an incremental SVM based classifier over time while allowing adaption to new changes. This will be done as part of the preliminary studies in Chapter 4. Something that differs across prototype-based classifiers impacting the computational cost of the approaches is the selection or generation mechanism to produce prototypes [21], which impacts directly on their computational cost. As covered in Chapter 2, these can be reduced (as seen in the previous paragraph) or created from the input distribution.

One example of a classifier that generates prototypes is *learning vector quantisation* (LVQ) [203]. Different incremental algorithms based on this algorithm have been proposed [374, 382]. OISVM, presented in [382], combines uses an approach based on LVQ to summarise the data input and feed it into an SVM classifier and reduce the computational cost at the training stage. This is as well a common use of prototype generation techniques like self-organising maps (SOM), growing neural gas (GNG) in the literature [113, 224, 264, 344]. PG techniques have also been used in the financial domain for data partitioning and model selection [91, 277]. Ajalmar et al. [264] provided a good overview of these, and Smith and Alahakoon [322] compared them looking at their growth rate, growth conditions, growth inhibition, and data example pruning.

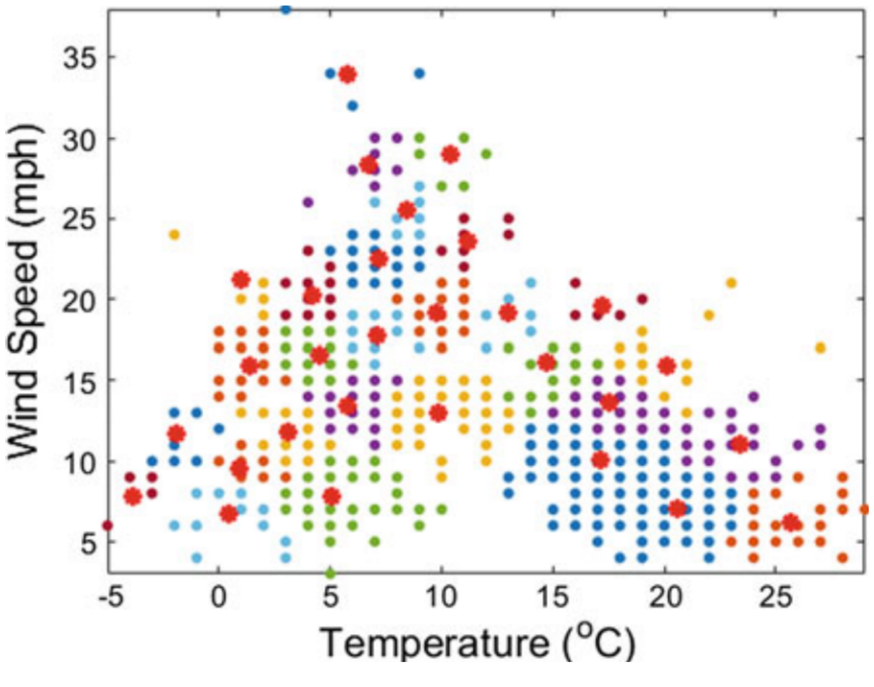


Figure 3.9: *Data clouds formed around a set of prototypes (red dots). Extracted from [21].*

The SOM algorithm, described in Chapter 2, is an offline method and thus is only able to learn static data. For this reason, this algorithm has been adapted in different research works [10, 11, 226, 289, 316] that have suggested dynamic (growing) methods for online learning. An example of these is growing self-organising maps (GSOM) [137, 362]. It grows nodes at the edges of the map when the total distance of an example exceeds a threshold, which allows it to track regions that may present dynamic behaviours when the original SOM would stabilise and lose its capacity to re-shape. GSOM is an incremental approach, but not adaptive since this does not have a forgetting factor like GNG (explained in Chapter 2).

Regarding other surveyed methods to create or select prototypes, SVMs, LVQ or instance-based algorithms (kNN) are supervised approaches and thus differ from neural gas and self-organising maps based techniques. Non-supervised applications like representing non-discrete concepts are not able to leverage these approaches. Moreover, although LVQ has also been used recently to learn in non-stationary environments [324], to the best of our knowledge, this algorithm, as SOM, has not been widely applied to data stream mining yet. The performance of an incremental version of LVQ and GNG is compared in [327]. GNG has been proven to be an effective method reducing the number of instances massively in a dataset preserving the original topology [136]. It has already been used in conjunction with a state of the art machine learning classifiers [60, 224, 267].

The main benefits of GNG against other commonly model-based unsupervised methods such as EM are its ability to handle non-Gaussian distributed clusters and its growing (online) evolving nature. GNG does not have an offline phase as EM, and it can be trained continuously, unlike SOM. Furthermore, EM has slow convergence, which can impact online scenarios at high frequencies. For this reason, GNG will be the primary technique used in this thesis to generate prototypes. In any case, one of the research objectives of this thesis is the creation of a meta-model where the different base algorithms are interchangeable; this includes the non-supervised component to support model reuse. We expect this thesis to attract future research which may perform studies on specific base algorithms and different domains.

3.6 Contributions Regarding Price Trend Classification

Computational finance and, more specifically, the problem of stock price forecasting is an application area that has attracted extensive AI research since the 1990s. Many literature reviews of deep learning [218, 268], machine learning [79, 129, 172, 179], and neuro-fuzzy methods [29] have been published in this domain since then. These look at different financial indices and benchmark different machine learning approaches and econometric methods. For instance, Ferreira et al. in [129] reviewed 2,326 financial investment papers from the Scopus research platform that were published between 1995 and 2019. The most commonly used machine learning methods for prediction in these reviews involve SVMs, ensembles and neural networks. Of these, in this thesis, we will use SVMs and different online ensembles for stock trend prediction.

Overall, the above-mentioned literature reviews confirm that machine learning techniques can be used to predict price changes, but this entirely depends on the time horizon and efficiency of the market in the period predicted. Cavalcante et al. in [79] provided another interesting review of pre-processing and clustering techniques used in the financial domain and to forecast future market movements. They pointed out the relevance of concept drifts in financial markets and suggested that the data stream mining literature is of great importance in future research due to the non-stationarity and evolution of financial markets [80].

As covered in Chapter 2, the prediction of future financial trends can be approached using fundamental or technical analysis. As an example of the first approach, the authors of [148] focused on the short term, intraday and high-frequency forecast using news data. Nonetheless, despite the controversy regarding the potential of this to produce profitable trading strategies [128, 230], as introduced at the start of this chapter, this thesis focuses on technical analysis because these have been widely used in short term trading [336].

Hence, some of the literature reviews already cited describe common technical indicators used for future stock market value and trend prediction. Many of these papers, like [29] have shown that different pre-processing steps like the frequency level of the input data can impact its predictability. While a common approach in this regard is data *normalisation*, in the literature of data stream mining data normalisation is not a usual practice since maximum and minimum values for each attribute in the data stream are unknown beforehand [48]. Some research works cited on these surveys, as Patel et al. [275], discretised features based on the human approach to investing and deriving the technical indicators using assumptions from the stock market. This last approach, though, introduces human bias in the process, being an opposite way to approach the problem of trend prediction compared to recent deep learning approaches that feed dozens of automatically generated indicators [218]. While we are aware of many potential ways to approach the problem of price trend prediction, as the approaches reviewed in this section, the focus of this thesis is instead on the usage of online incremental techniques in financial data streams.

In our work, we will focus on a common approach used in the literature proposed by Kara et al. in [195]. They proposed the selection of ten technical indicators as feature subsets of publications presented by domain experts and researchers of this field [27, 115, 183, 199, 200, 211, 377], which can be found in Table 3.2. This approach has been used in several works afterwards [179].

Indicator	Formula
A/D	$\frac{H_t - C_{t-1}}{H_t - L_t}$
CCI	$\frac{M_t - SM_t}{0.015 D_t}$
LWR	$\frac{H_n - C_t}{H_n - L_n} \times 100$
MACD	$MACD(n)_{t-1} + 2/n + 1 \times (DF_t - MACD(n)_{t-1})$
MOM	$C_t - C_{t-n}$
RSI	$100 - \frac{100}{1 + (\sum_{i=0}^{n-1} Up_{t-i}/n)/(\sum_{i=0}^{n-1} Dw_{t-i}/n)}$
SMA	$\frac{C_t + C_{t-1} + \ldots + C_{t-n+1}}{n}$
SD	$\frac{\sum_{i=0}^{n-1} K_{t-i}\%}{n}$
SK	$\frac{C_t - LL_{t-n}}{HH_{t-n} - LL_{t-n}} \times 100$
WMA	$\frac{n \times C_t + (n-1) \times C_{t-1} + \ldots + C_{t-n+1}}{n + (n-1) + \ldots + 1}$

C_t: closing price; L_t: lowest price; H_t: highest price at time t; DF: $EMA(12)_t - EMA(26)_t$; EMA: Exponential moving average; $EMA(k)_t$: $EMA(k)_{t-1} + \alpha \times (c_t - EMA(k)_{t-1})$; α: smoothing factor: $2/1 + k$; k: time period of k minute exponential moving average; LL_t and HH_t: mean lowest low and highest high in the last t minutes; $M_t : H_t + L_t + C_t/3$; $SM_t : \sum_{i=1}^{n} M_{t-i+1})/n$; $D_t : (\sum_{i=1}^{n} |M_{t-i+1} - SM_t|)/n; Up_t$: upward price change; Dw_t: downward price change at time t.

Indicators: A/D: accumulation/distribution oscillator; CCI: commodity channel index; LWR: Larry William's R%; MACD: moving average convergence divergence; MOM: momentum; RSI: relative strength index; SMA: simple moving average; SD: stochastic D%; SK: stochastic K%; WMA: weighted moving average.

Table 3.2: *Technical indicators used in the analysis. Formulas as reported in [195].*

The reader must note that Kara et al. [195] uses a mathematical notation that considers t as the last known value. For instance, they compute the simple moving average of the close price at time t, which usually represents the target feature in time series forecasting. In this thesis, we have respected the original notation from their original paper, but we refer to the time step t as the last know value instead of the future time horizon to be forecasted. Another important point about this paper is that the presented indicators were not selected as an intraday but rather for an interday forecast. However, as far as we know, there is no other standard set of technical indicators for high frequencies; hence, we will use this.

3.7 Summary

This chapter reviewed the state of the art of data stream mining and machine learning for stock trend prediction. Although an initial systematic literature review took place initially in this thesis, the review has been an amalgamation of related research and searches performed during the whole Ph.D. period (2017-2022).

The application of AI to computational finance is a research field of high research attention since the 1990s. However, online incremental algorithms for data streams have not been widely applied to financial forecasting yet. We have approached the financial domain through the problem of concept drift as a sort of structural break that can occur at any frequency level. This has been covered in Section 3.1. After this, Section 3.2 covers relevant and recent works in data stream mining.

Section 3.3 covered recent literature for supervised machine learning. The section was introduced with offline approaches used in financial forecasting like SVMs, which will be used in preliminary work to our main proposal in Chapter 4. After this, Section 3.3 focused on the data stream mining field and covered works the recent trends of ensembles. Section 3.4 covered meta-learning in data stream classification. Many meta-learning approaches in the literature use non-supervised algorithms to identify the recurrence of a concept and retrieve previous models or detect drifts. We have covered these works and other approaches based on prototypes used for data partitioning in Section 3.5. Finally, Section 3.6 reviewed the stock price trend prediction field and described the most common set of indicators used in the literature for technical analysis.

This thesis contributes to the topics of meta-learning and ensembles for supervised learning under evolving data streams. Our primary approach will be proposed in Chapter 5, and the differences against relevant works in the literature will be covered in Subsection 5.1.4. We will apply our main proposal to price trend prediction in the financial domain following the approach covered in Section 3.6, with a semi-synthetic and a real-world dataset in Chapters 5 and 6 respectively. The use of a semi-synthetic dataset will be to create a controlled environment to measure the suitability of concept drift detection approaches as also performed by the relevant literature (see Subsection 3.2.4). Before our main proposal, we will describe our introductory research work regarding the scalability of online incremental learners for data streams and their applicability to financial data. These initial studies will be presented in Chapter 4.

Chapter 4

Preliminary Studies

This chapter will detail the first two experiments that serve as the base to our main proposal, which will be described in Chapter 5.

The first experiment will be detailed in Section 4.1. This experiment will aim to improve the scalability of traditional machine learning algorithms to be used in a continuous training setup. We will propose a system that deals with concept drifts passively and that makes use of both supervised and non-supervised methods. The results demonstrate that incremental mini-batch techniques and topology extraction methods can improve the scalability of training machine learning models continuously.

The second experiment will focus on the application of online incremental machine learning algorithms to stock market price classification. In this experiment, we propose a system that deals with concept drifts both passively and actively. This is detailed in Section S4.2. As seen in Chapter 3 ensembles are known for their good results predicting during cyclic and non-stationary scenarios as stock market prices. The proposed system ensembles many base classifiers to deal with the complex behaviour of financial time series and reuses previous models when these improve the new classifiers at the time of a concept drift. The system is applied to predict the SPDR S&P 500 Exchange-Traded Fund price movement direction one minute ahead, improving the results of other algorithms from the online incremental machine learning literature.

4.1 Using GNG to Model Data Distributions in Streams

4.1.1 Introduction

Data stream mining is a growing research field covering machine learning algorithms that adapt to learn over time in continuous data streams. The rise of Data Mining approaches over streams of data gained popularity a few years back with technologies such as Apache Kafka and Spark Streaming, mentioned in Chapter 2. Since then, it has gradually been adopted in industry to deal gradually with huge amounts of data. However, while Stream Processing technologies are common, online incremental algorithms are still in a phase of early adoption in the industry, with libraries like *CremeML*, *scikit-multiflow* or *River*, also mentioned in Chapter 2. In this regard, the standard in the industry is still to use specific infrastructures and high resources in terms of memory, computing nodes and GPUs for data preparation and model retraining.

As mentioned in Chapter 1, one of the goals of this thesis is to apply novel machine learning techniques to learn and predict high-frequency data. For this, the first experiment of this chapter studies the advantages of using data stream mining algorithms from the scalability point of view. The goal is to compare the behaviour of static models and tackle issues that arise in online learning scenarios that we will face later in this thesis. To do this, in this first experiment, we focus on one of the most common traditional machine learning algorithms for stock trend prediction; support vector machines (SVM). SVMs are well known in the literature [171] [358] for their good results dealing with non-linear and complex datasets. However, these classifiers piggyback a high computational complexity [82] when dealing with large-scale data, hence these can be problematic operating at high frequencies. In the literature, the use of this method in these scenarios has required its incrementalisation [342] or to combine it with other techniques such as *prototype selection*, or *prototype generation* techniques such as *learning vector quantisation* [382] [264] [224] [344] [113], to reduce the amount of data in training stage. Ajalmar et al. [264] provided an overview on this.

In this section, we propose an adaptive approach through the incrementalisation of the SVM classifier. Our approach trains a new support vector machine for every new batch of data received. Then, it retrieves previous knowledge of the prior model (previous batch) to train each new one. We have also used this in combination with the PG technique growing neural gas to train the SVM classifier in a reduced representation of the current batch and help it to scale. The final objective of this section is to verify that online incremental machine learning methods can help improve runtime, not affecting the classification accuracy. Thus we pursue to validate the relevance of data stream mining techniques for classification tasks at high frequencies. This will allow us to use these techniques for stock trend classification at high frequencies in the remainder of this thesis.

This section is structured as follows: first, Section 4.1.1 has introduced this first experiment that we aim to perform; Subsection 4.1.2 explains the features of the proposed system for the experiment. The experimental protocol and the parameters tuned will be explained in Subsections 4.1.3 and 4.1.4 respectively, and Subsection 4.1.5 will contain the experimental results of this section. Finally, Section 4.1.6 will explain the conclusions reached and the future lines of work in this regard, as part of this thesis and new research branches from here.

4.1.2 System Features

Our approach in this section will inherit features from two previous incremental frameworks using SVMs. The SVM classifier creates one or a set of hyperplanes during training in a high-dimensional space at the largest distance to the nearest prototypes of any class [59]. This is built in terms of a relatively small number of training examples. These examples, namely support vectors, are the nearest data points of each class to this hyperplane, defining each class boundary. The two works that we will leverage in the current section are:

- GNG-SVM [224], uses of growing neural gas (GNG) prior to the classification task. Their underlying idea was to reduce the number of input examples replacing them with a set of prototypes reproducing the topology of the input examples. A GNG model was applied independently to each class. After this, topologies of all classes were combined and then the SVM classifier is trained with the the combined output of all GNG models.
- OISVM [382], also leverages from prototype generation before the classification task but creating prototypes simultaneously instead of through different models like GNG. OISVM works in a mini-batch setting and retrains new models for

each batch, reusing support vectors of the SVM classifier of the previous iteration. However, as seen in an earlier study mentioned in [328], OISVM presents scalability bottlenecks over time in certain scenarios by creating unnecessary support vectors if the boundaries overlap across models of previous and new iterations.

In this section, we will propose an algorithm, iGNGSVM, that combines both approaches and is also based on an initial study of different incremental algorithms to improve scalability on online learning tasks performed by us in [327]. iGNGSVM will use GNG as a PG method to summarise the dataset sent to the SVM classifier, as seen in GNG-SVM. It will inherit support vectors from classifiers trained at previous batches, as OISVM. This aims to be an introductory approach to classify continuous data streams of information at high frequencies in an online learning fashion.

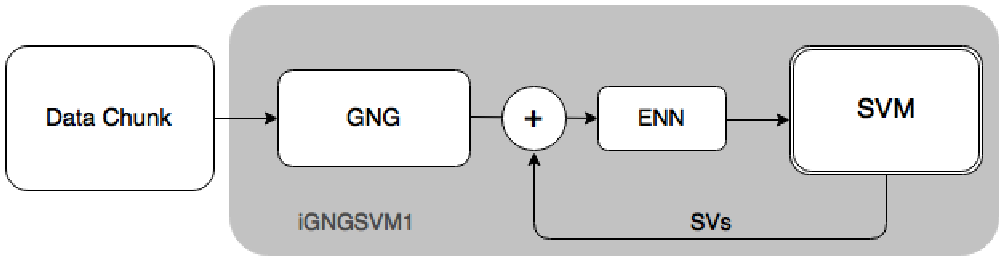
4.1.2.1 Definition of the iGNGSVM Algorithm

As summarised above, iGNGSVM performs two steps iteratively each time a new mini-batch of training examples is received through an incoming data stream. The first step involves using the GNG algorithm for prototype generation. This is done to reduce the computational cost of training the SVM classifier in a second step.

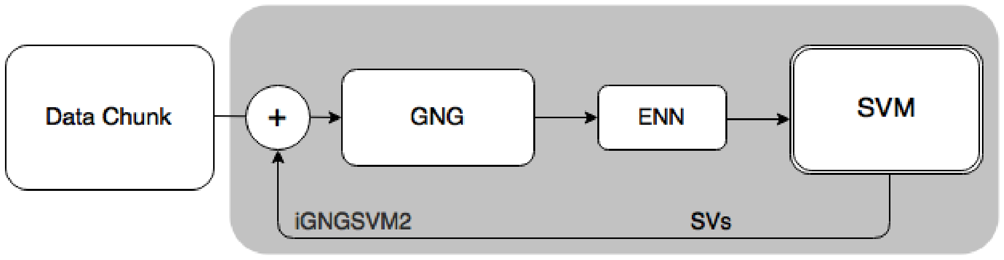
The growing neural gas algorithm [136] was introduced in Section 2.3.6.3. The number of prototypes generated in a topology by a GNG is specified in the stopping criterion (step 11) in Algorithm 2.3. The value of the stopping criterion, all together with the λ and a_{max} parameters (in charge of the generation of removal of examples, respectively), is a key factor for the computational cost of GNG in iGNGSVM. Assuming a stable concept, the larger the value of the stopping criterion, the lower the quantisation error of the new topology, since the set of prototypes generated approximate gradually to the incoming distribution. However, the larger the stopping criterion, the greater the computational cost of GNG and the larger the distribution received by the SVM classifier (and thus its cost). The value of these three parameters is crucial to produce a good summarisation of the incoming batches and not penalise the classification performance.

In this section, we focus on linear models and binary classification for the sake of simplicity. For this setting, as described in Section 2.2.3.4, the SVM classifier aims to find the hyperplane that maximises the separation distance between two classes.

The training examples supporting this hyperplane (its nearest examples) are called support vectors (SV). The behaviour of a linear SVM classifier [171] [82], was shown Algorithm 2.2. Since SVM receives the output of GNG (first step), any mention to data points or vectors in Algorithm 2.2 refers to this in the iGNGSVM framework.



(a) *iGNGSVM*$_1$



(b) *iGNGSVM*$_2$

Figure 4.1: *iGNGSVM*$_1$ *and iGNGSVM*$_2$ *training workflows.*

The technique we propose, iGNGSVM, aims to be an adaptive approach that reuses previous knowledge over time. Hence, mini-batches are not independent tasks besides the training of a new SVM classifier. Consequently, each iteration that receives a batch of data reuses the support vectors that were obtained from the SVM classifier in the previous batch. These SVs are merged with the prototypes obtained by GNG in the new batch at the training stage (see Figure 4.1a). The fact of training new classifiers aims to act as a forgetting mechanism and over time, while the inheritance of SVs across iterations seeks to evade catastrophic forgetting [296].

An issue found in predecessors of our proposal such as OISVM was model bloating due to the increase in the number of the SVs across iterations if the boundary of the class gradually evolves across batches [328]. This makes the model unscalable and unfeasible for data stream mining. To tackle this, we used the prototype selection technique *Wilson's edited nearest neighbor* (ENN) [69, 341] as an intermediate step between GNG and SVM. ENN, briefly introduced Section 2.2.3.4, is covered in

Algorithm 2.1. iGNGSVM uses ENN to remove SVs that are no longer relevant due to a gradual concept drift or prototypes surrounded by a majority of prototypes or SVs from the opposite class. Hence, cleaning data points causing the unbounded growth of SVs over time seen in the predecessors of this proposal.

Algorithm 4.1 iGNGSVM$_1$ training algorithm. Extracted from [328].

1. It receives a set of training examples S (data-chunk in Figure 4.1).
2. It divides the set of examples in one subset per class ($S_1, ..., S_n$ / n classes).
3. It obtains the topology of every isolated subset using GNG with a stopping criterion that can be a number of prototypes to create or a percentage according to every subset size.
4. It merges all the subsets, putting the topology of all the classes together ($S*$).
5. If *iteration number* > 1: It adds the SVs of the previous iteration as prototypes to $S*$ to remember the prior model.
6. It runs ENN removing prototypes from $S*$ or SVs surrounded by a majority of neighbours of the opposite class.
7. It trains a new SVM using the new data chunk topology $S*$.
8. It saves the SVs calculated for the next iteration. Then it waits for the starting condition to process the next data chunk (step 1).

The initial version of iGNGSVM (iGNGSVM$_1$) proposed is explained in Algorithm 4.1. First, data instances received from a Data Steam are stored in a buffer. Once this buffer reaches a given size (starting criterion), the instances from the buffer are used as a batch of data, and a training iteration takes place. Then, training instances are separated depending on their class, and a separate instance of GNG creates a topology of each class distribution. After this, both topologies are merged with the SVs from the previous iteration. ENN receives the resulting set for noise removal and feeds the SVM classifier of the current iteration.

In parallel to the iGNGSVM$_1$, we will present a second version of the algorithm aimed at non-stationary datasets with fast drifts. Figure 4.1 illustrates both versions of iGNGSVM. The main difference between both versions is that iGNGSVM$_1$ (Figure 4.1a) inherits the SVs as prototypes (it merges then with the output topology from GNG), and iGNGSVM$_2$ inherits the SVs as examples (and thus are fed into GNG together with the batch of data). This design makes previous knowledge have less weight in iGNGSVM$_2$ and be more suitable for settings that change continuously. Finally, both versions of iGNGSVM use ENN to filter out no longer relevant SVs. The full version of iGNGSVM$_2$ is shown in Algorithm 4.2.

Algorithm 4.2 iGNGSVM$_2$ training algorithm. Extracted from [328].

1. It receives a set of training examples S (data chunk in Figure 4.1).
2. If *iteration number* > 1: It adds the SVs of the previous iteration as examples to S.
3. It divides the set of examples in one subset per class ($S_1, ..., S_n$ / n classes).
4. It obtains the topology of every isolated subset using GNG with a stopping criterion that can be a number of prototypes to create or a percentage according to every subset size.
5. It merges all the subsets, putting the topology of all the classes together ($S*$).
6. It runs ENN removing prototypes from $S*$ that are surrounded by a majority of neighbours of the opposite class.
7. It trains a new SVM using the new data chunk topology $S*$.
8. It saves the SVs calculated for the next iteration. Then it waits for the starting condition to process the next data chunk (step 1)

4.1.2.2 Computational Complexity

In order to understand the solution proposed in this section, in this subsection, we want to clarify that the computational complexity of iGNGSVM depends mainly on its three algorithms: GNG, ENN and SVM.

- In our proposal, we have used the implementation of WEKA of LibSVM[1]. According to [82] and [1], its *Big O* notation complexity is $O(n_{features} \times n^2_{samples})$ assuming $O(n)$ for each kernel evaluation.
- For GNG and ENN, their basic implementation has a notation of $O(n^2)$ according to [249] and [189] since they iterate through the neighbourhood of each data instance received by them.

[1] LIBSVM is a Library for support vector machines: http://www.csie.ntu.edu.tw/∼cjlin/libsvm/

Table 4.1 shows the estimated complexity of the iGNGSVM$_1$ algorithm step by step in Big O notation. The main difference with iGNGSVM$_1$ is the order of the steps, but not their complexities. Loops over a set of examples in this notation have a complexity of $O(n)$. When the algorithm only iterates over a number of classes as in step 4 in Algorithm 4.1. The complexity is $O(k)$ when the iteration is only over the number of classes (see step 4 in Algorithm 4.1).

Repeat the next for every mini-batch	Complexity	Input
(1) Divide the batch of instances in one subset per class.	$O(n)$	|TS|
(2) Obtain the topology of every subset using GNG by separate.	$O(k \times n^2)$	$\sum(\|S_{class_i}\|)$
(3) Merge all the subsets in a prototypes final set.	$O(k)$	$\#classes$
(4) Run ENN to label and remove noise.	$O(n^2)$	$\|S*\|$
(5) Inherit the SVs of the previous iteration.	$O(n)$	SVs
(6) Train a new SVM using the new data chunk topology.	$O(n^3)$	$\|S*\| + SVs$
(7) Save the new SVs for the next iteration.	$O(log(n))$	SVs

$|S*| = \#classes \times stoppingCriterion$

Table 4.1: *iGNGSVM$_1$ complexity analysis. Extracted from [328].*

iGNGSVM does not improve the computational complexity of any of its three components. Instead, it reduces the cost of running the SVM classifier by reducing the number of input patterns in this. For this reason, in Table 4.1 we also show the dataset input size at every step as reference. The batch-size |TS| represents all of the data instances received in one batch.

The global computational complexity of both versions of iGNGSVM is $O(n^3)$, corresponding to the larger O in Table 4.1 due to the SVM classifier. As mentioned before, the scalability improvement is performed by training the SVM using a number of prototypes of many orders of magnitude lower than the number of instances received in a batch ($|S*| + SVs$ rather than the full batch |TS|).

4.1.2.3 Preliminary Studies and Validation

This subsection uses synthetical data of a non-stationary nature to compare differences between the two versions of iGNGSVM proposed in the previous subsection. Figure 4.2, compares iGNGSVM$_1$ and iGNGSVM$_2$ in a dataset for binary classification built by sampling two overlapping distributions of two dimensions which centres move at constant speed on both axes. Classes are represented in blue and red colours.

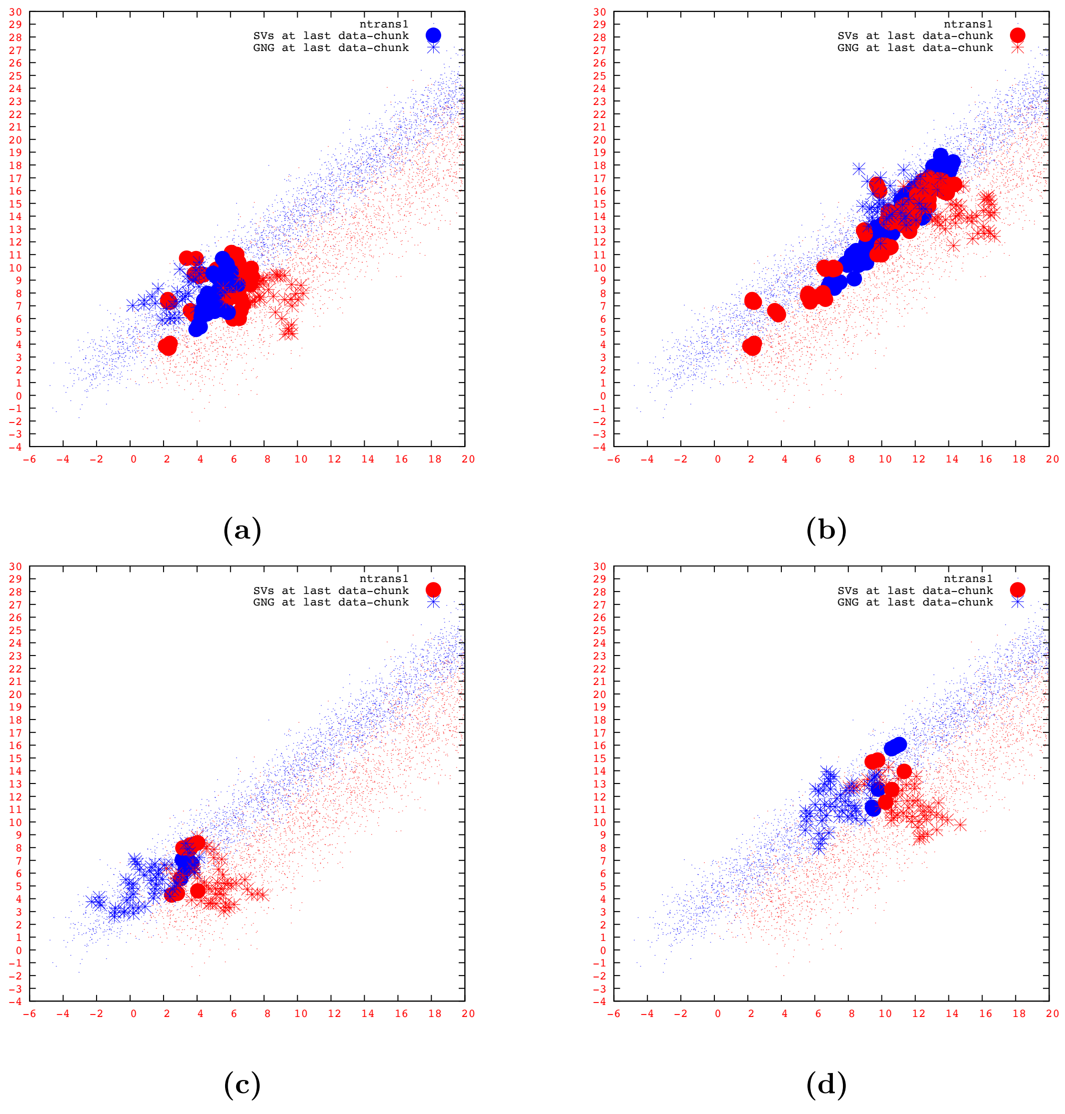


Figure 4.2: *Support vectors created by iGNGSVM$_1$ (top row, 4.2a and 4.2b) and iGNGSVM$_2$ (bottom row, 4.2c and 4.2d). Extracted from [328].*

Figure 4.2 depicts the behaviour of the different versions of iGNGSVM under an evolving data stream. Small dots in Figure 4.2 represent instances for both classes in the entire data streams. Stars and circles represent the prototypes created by GNG in the current batch and the SVs outputted by SVM in that batch respectively. Figures 4.2a and 4.2c show that iGNGSVM$_1$ is the version generating more SVs even at the initial batch (Figure 4.2a). Figures 4.2b and 4.2d show how iGNGSVM$_1$ keeps a summary of the data distribution in the form of SVs even in the final batch (4.2c).

Table 4.2 shows the classification performance and numbers of final SVs for iGNGSVM$_1$ and iGNGSVM$_2$ with different batch sizes and a different number of neighbours in ENN, or disabling ENN (N/A). Tests were run for the non-stationary dataset of Figure 4.2 (*ntransSingle*1) and an static version of it not moving the axes (*stbal*1). In the non-stationary dataset, iGNGSVM$_1$ and iGNGSVM$_2$ tend to obtain better results when ENN is not disabled. In the static dataset, ENN does not seem to be beneficial in classification performance. However, the use of ENN helps reduce the number of SVs and, therefore, improving our proposal's scalability with a cost of classification accuracy lower than 1%.

		*stbal*1				*ntransSingle*1			
|TS|	# ENN$_{ne}$	% Acc$_1$	SVs_1	% Acc$_2$	SVs_2	% Acc$_1$	SVs_1	% Acc$_2$	SVs_2
100	N/A	91.96	753	91.88	52	85.36	3,714	89.72	54
	3	91.40	268	91.44	28	84.6	745	90.28	12
	5	91.00	249	90.72	28	83.28	667	91.44	12
200	N/A	91.76	400	91.56	44	84.84	1,910	88.44	54
	3	91.40	96	90.72	20	85.04	372	89.56	10
	5	91.24	109	90.56	27	84.32	344	89.76	4
300	N/A	91.48	282	91.56	40	84.8	1,263	86.04	50
	3	91.36	101	91.40	25	85.00	214	86.6	3
	5	91.24	102	91.32	23	84.80	105	88.4	6
500	N/A	92.00	161	91.44	45	84.64	759	85.76	51
	3	91.56	40	91.12	21	85.6	38	85.76	7
	5	91.32	36	91.52	17	84.6	4	83.48	4

ENN$_{ne}$: number of neighbours (N/A if ENN is disabled); |TS|: batch size;
Acc$_1$ and Acc$_2$ refer to the classification accuracy of iGNGSVM$_1$ and iGNGSVM$_2$ respectively;
SVs_1 and SVs_2 refer to the final number of SVs for iGNGSVM$_1$ and iGNGSVM$_2$ respectively.

Table 4.2: *Results of iGNGSVM$_1$ and iGNGSVM$_2$ for several batch sizes over a stationary (stbal1) and a non-stationary (ntransSingle1) synthetic dataset. Extracted from [328].*

Concept drifts in the same batch are not handled by our approach since it is not purely incremental. However, iGNGSVM adapts to changes occurring between batches with the removal of SVs using ENN and GNG to represent the current topology. In iGNGSVM$_2$, any SV outside of the main distribution of prototypes will be likely to disappear over time depending on the parameters selected in GNG. ENN may not be self-sufficient to handle the growth of SVs over time (see Figure 4.2b). In summary, iGNGSVM$_1$ exhibits more long-term memory, being more suitable for recurring concepts; iGNGSVM_2 shows an adaptive behaviour that tends to forget faster the previous data distribution and may suit data streams under continuous change.

4.1.3 Experimental Protocol

In this subsection, we introduce the setting used for the first experiment of the thesis. We compare the accuracy rates and run times of both iGNGSVM variants against a mini-batch version of LibSVM (namely LibSVM*). We have developed LibSVM* as a baseline for our experiments. The idea is to compare the raw executing time of using SVM in batches without inheritance of SVs, prototype reduction and noise removal to iGNGSVM. The static version of LibSVM was not valid for the current experiments since the amount of data to be processed did not fit in the main memory. Thus we needed either a mini-batch or an incremental way to feed these examples.

For the first experiment, we will not use data from the financial domain. We have two large binary classification datasets from the UCI repository: HIGGS [2] and SUSY [3], due to their size. HIGGS has 11M data instances and 28 features, and SUSY has 5M data instances and 18 features. These synthetic sets were generated using Monte-Carlo simulations for classification tasks using deep learning in [35]. Since these datasets are in a format ready for machine learning tasks, we have not performed any cleaning or pre-processing steps. The server used for this experiment is a Macbook Pro 13 inch from early 2011 with an i5 processor of 2,3 GHz and 4GB of RAM. The goal of using this machine for the experiment is to show the performance of iGNGSVM with limited computational resources, which is eventually always the scenario when dealing with massive datasets or data streams of information at high frequencies. Both datasets are passed to iGNGSVM using the MOA framework[4] as a finite, ordered stream of data.

In the experiments of this section, we will perform a prequential evaluation. This means that while the instances received for training are stored in a buffer until reaching a minimum batch size, these are first used for testing and evaluating the model. These tests are performed over the last SVM classifier trained. This evaluation method avoids the accumulation of many batches of data in memory. The results are reported using the evaluation task from MOA

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

[3] https://archive.ics.uci.edu/ml/datasets/SUSY

[4] MOA is an open-source framework for data stream mining related to the WEKA project: http://moa.cms.waikato.ac.nz

moa.evaluation.WindowClassificationPerformanceEvaluator. This task reports mean average results on windows of a given number of data instances. The default window size is 1,000 instances. The final result reported in the experiments will be the mean accuracy across all windows.

4.1.4 Parameter Tuning

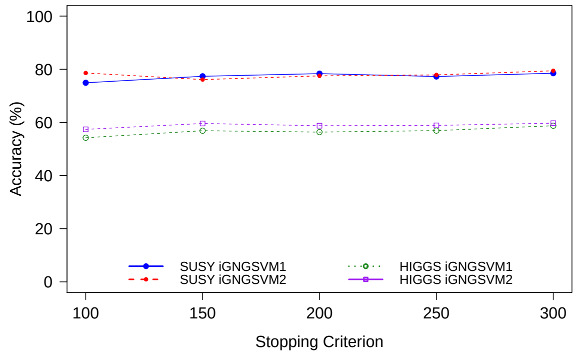
Large values for the stopping criterion in GNG or increasing the number of patterns received by SVM will clearly increase the runtime of iGNGSVM. ENN is also a crucial component in iGNGSVM as seen in Subsection 4.1.2.3. In consequence, in this subsection we will optimise the values for *batch size*, the GNG *stopping criterion* and *the number of neighbours* in ENN. All other parameters are set to their default values in related research [69] [224]. As mentioned already, for the SVM classifier, we have used the version of WEKA of LibSVM, and we have used a linear kernel with a plain vanilla setup.

- For the *stopping criterion of GNG*, the goal is to reduce the input set to the lowest possible number of prototypes to avoid scalability bottlenecks in SVM and still create an accurate topology for the current batch; thus, we will explore values in the range of [100, 500] prototypes generated.
- For the *batch size*, small sizes increase the run time since the number of iterations increases, and with this, the number of topologies and SVM classifiers is built over time. Large sizes will encounter bottlenecks in the classifier and will not fit in memory in the server used. We will explore sizes in the range [25, 2 million] data instances per batch.
- For the *number of neighbours in ENN*, following recommended values from the literature, we will explore to consider the 3, 5, and 7 closes instances as neighbours for prototype reduction.

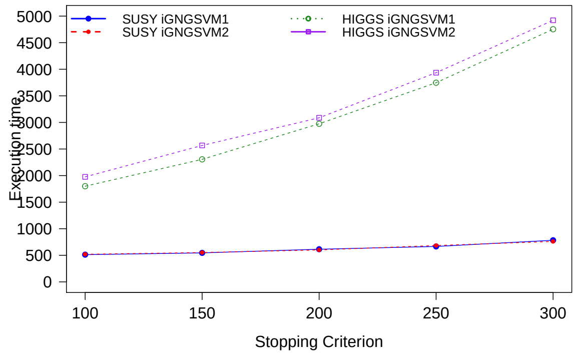
Parameter tuning was performed over the first 2M instances of each dataset. These instances were excluded from the final experiments in Subsection 4.1.5. The selected values were optimised to reduce run time, not compromising the classification performance. In Table 4.3 we show the parameters that obtained maximum accuracy rates

Dataset	Algorithm	Batch size	GNG stop.crit.	ENN	SVM cost
SUSY	iGNGSVM$_1$	400k instances	300 prototypes	3 neighbours	8
	iGNGSVM$_2$	400k instances	300 prototypes	*off*	8
HIGGS	iGNGSVM$_1$	100k instances	300 prototypes	3 neighbours	8
	iGNGSVM$_2$	100k instances	150 prototypes	3 neighbours	8

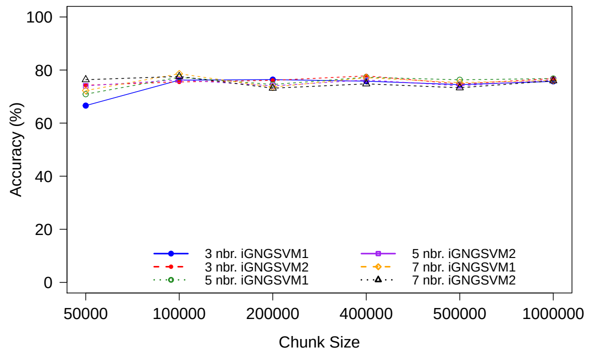
Table 4.3: *Parameters obtaining maximum accuracy rates in iGNGSVM.*



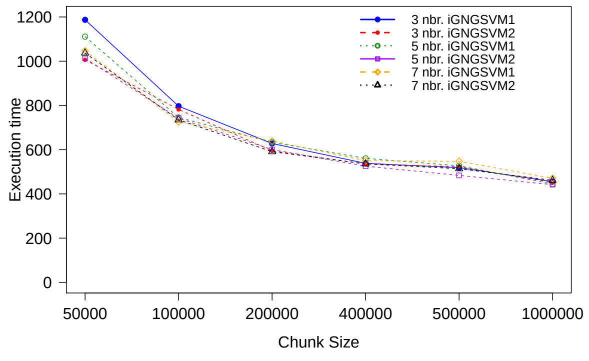
(a)



(b)



(c)



(d)

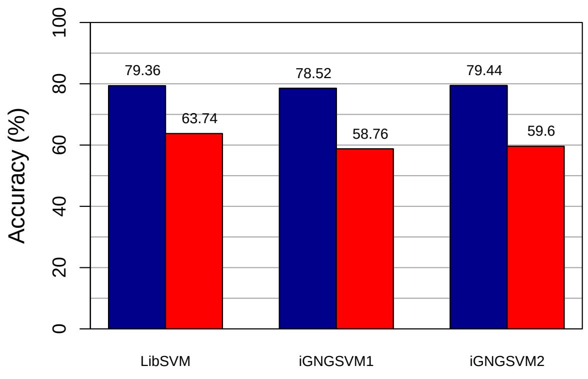
Figure 4.3: *Accuracy rates (4.3a) and run times (4.3b) for different stopping criterion in SUSY and HIGGS, and across batch sizes and number of neighbours (4.3c and 4.3d) in SUSY.*

in SUSY and HIGGS. However, the best balance of accuracy vs execution time was obtained using a similar setting but batch sizes of 1 million examples as shown in Figure 4.3. Hence, in the benchmark of the next subsection, we used the parameters from Table 4.3 with batch sizes of 1 million examples.

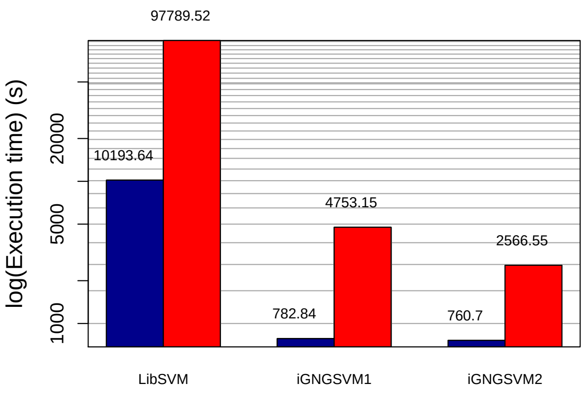
Figure 4.3 shows the performance of both versions of iGNGSVM for the different during parameter tuning. Both variants of iGNGSVM performed similarly in terms of accuracy and run time balance for both datasets with a different number of neighbours in ENN. It did not vary much for different batch sizes in terms of accuracy rates in the set used for parameter tuning. The trade-off between batch size and GNG stopping criterion was a key factor. Larger batches implied a more significant reduction of the input patterns in SVM, impacting classification performance.

4.1.5 Results

The results of the first experiment of this thesis are shown in Figure 4.4 and Table 4.4. These show the mean run time and accuracy over five executions of each algorithm in SUSY and HIGGS.



(a)



(b)

Figure 4.4: *Accuracies (4.4a) and run times (4.4b). SUSY in blue bars, HIGGS in red bars.*

In the dataset SUSY, the mini-batch version of the SVM classifier with batches of 10k examples (LibSVM*) resulted in a mean accuracy of 79.36% which was accomplished in a mean runtime of 10,200 seconds. In comparison, iGNGSVM was able to obtain comparable results in terms of classification performance in only 700-800 seconds.

In HIGGS, the largest of the two datasets, iGNGSVM loses ≈5% accuracy compared to the static SVM classifier. In any case, the reduction in terms of the runtime is visible, and for instance, iGNGSVM$_2$ has a run time speed of 37x compared to the SVM classifier.

The results achieved with iGNGSVM receive comparable classification performance to the static SVM and to the results obtained in the relevant literature [35], but for considerable lower run times. The techniques used have helped avoid scalability

	LibSVM*		iGNGSVM_1		iGNGSVM_2	
Dataset	Accuracy (%)	Time (s)	Accuracy (%)	Time (s)	Accuracy (%)	Time (s)
SUSY	79.36	10,194	78.52	783	79.44	761
HIGGS	63.74	97,790	58.76	4,753	59.60	2,567

Table 4.4: *Mean accuracy and run times obtained with LibSVM* and both versions of iGNGSVM.*

bottlenecks in large scale datasets using SVM classifiers. We have thus verified the advantages in terms of runtime reduction when using prototype-based summarisation techniques such as GNG and ENN. The inheritance of SVs also plays a role in the accuracy rates obtained. For instance, in SUSY, iGNGSVM can reduce the run time without a high cost on accuracy rate and improve the accuracy rate in iGNGSVM_2.

4.1.6 Summary & Discussion

Section 4.1 has covered the first experiment of this thesis as an introduction to online incremental learning and problems associated with the continuous learning nature of algorithms in the data stream mining domain. The experimental results have shown the power of adaptive learning and prototype generation techniques to deal with large scale data not requiring complex big data architectures. We have proposed two versions of our algorithm that differ in the mechanism used to transfer previous knowledge to new batches:

- iGNGSVM_1 transfers learning (support vectors) of the prior batch as prototypes in a new batch. Hence, it trains the new classifier using previous SVs and prototypes generated by GNG.
- iGNGSVM_2 transfers previous SVs as data instances. Thus, SVs are fed together with the instances from the new batch into GNG for the generation of prototypes.

According to our preliminary studies in Subsection 4.1.2.3 each variant suits different types of datasets.

iGNGSVM_2 transfers SVs as examples, and since GNG grows from its centre and expands, the resulting topology may not reach any SVs located at the boundaries of the data distribution. In certain scenarios, this artefact from iGNGSVM_2 can help to reduce overlap between classes and improve the separability of a dataset for data classification tasks. In other circumstances (e.g. non-stationary distributions), knowledge from previous batches can be compromised, affecting thus the performance of the algorithm in case of future recurring concept drifts. In Subsection 4.1.2.3, iGNGSVM$_2$ performed better in a non-stationary dataset. Conversely, iGNGSVM$_1$ can help in the case of recurrence in the short term since it has demonstrated to accumulate a small amount of SVs over time across batches. However, for any conclusions regarding concept recurrence in the long term, we would require more experiments that we consider out of the scope of the work aimed in this subsection.

The static version of the SVM was re-purposed as a mini-batch algorithm without the transfer of support vectors between learning iterations. This was due to the volume of data to classify and the impossibility to fit the data in memory. iGNGSVM demonstrated to be able to obtain comparable accuracies to this static version, with greater rates in individual tests (e.g. iGNGSVM$_2$ in the SUSY dataset). iGNGSVM was also able to overperform up to 37x the static classifier's run times, demonstrating its ability to execute classification tasks at scale in continuous data streams.

Both versions of iGNGSVM used ENN to remove overlapping SVs over time to avoid model bloating. However, in the experiments performed, iGNGSVM$_1$ benefited more from ENN to reduce the accumulation of SVs over time (especially in drifting data). In this scenario, the impact of increasing the number of neighbours in ENN accentuated over time as the total number of batches increased. The transfer of SVs as prototypes in iGNGSVM$_2$ acts by design as a forgetting mechanism, and individual tests in this variant performed better with ENN disabled. In any case, this behaviour depended as well on the batch size and the distributions of instances of each class received. For example, hypothetical scenarios with data skewness or different degrees of bias over time can result in different topology summaries and a changing distribution of SVs that does not need to relate to a concept drift as such.

This experiment was conducted using linear SVM and ENN methods. The study performed in iGNGSVM was not exhaustive. Most parameters used the plain vanilla configuration except the GNG stopping criterion and the number of neighbours in ENN. However, there is room to improve the results, and the reduction of SVs with ENN could be further enhanced with non-linear kernels. We believe that we have satisfied the target of this first experiment, which lies on the computational aspect. A potential consequence of the transfer of SVs designed in iGNGSVM is the loss of accuracy rate over time when receiving consequent batches. This could be the reason for the lower accuracy rates obtained with HIGGS, which had the highest number of batches. In the end, this may derive from introducing forgetting mechanisms in algorithms run over datasets not exhibiting non-stationarities as SUSY and HIGGS.

In this experiment, we have validated the use of GNG to summarise data distributions and produce knowledge that will be reused over time (SVs). In the main proposal of this thesis, in Chapter 5, we will use lessons learned from this section to summarise concepts using GNG and allow model reuse in case of recurring concept drifts. Before this, we must consider other aspects of our research. The first experiment focused on synthetic datasets, and thus we have not yet explored the behaviour of adaptive algorithms in real-world financial data. Furthermore, iGNGSVM has explored passive mechanisms for concept drift adaption. Still, it would be interesting to explore active drift handling mechanisms like the approaches explained in Chapter 2 (e.g. ADWIN2 as drift detector [46]). These aspects will be explored in the second experiment of this thesis, in Section 4.2.

4.2 Changes over Adaptive Random Forest for Recurrences

4.2.1 Introduction

In Section 4.1 we performed the first experiment with online incremental learners mainly to tackle the computational cost and scalability issues of traditional machine learning algorithms in data streaming scenarios. In this second experiment, we will propose an adaptive algorithm to handle a financial data stream. The goal is to classify price movement directions one second ahead in the SPDR S&P 500 Exchange-Traded Fund. In this regard, we will propose an ensemble-based method since, as explained in Chapter 3, ensembles are widely known for their good accuracy rates in stock trend classification. This method will be an extension of the *adaptive random forest* classifier (presented in Chapter 3) to deal actively with recurring concept drifts.

As introduced in Chapters 2 and 3, the problem of concept drift has recently shown its relevance in the financial domain as a different way to react to structural breaks and RCs. The algorithm proposed in this section will handle both gradual and abrupt regime change in the financial markets and predict the future direction of the market anytime with the best model available. We will benchmark some state-of-the-art algorithms in data stream mining and evaluate their performance in real-world financial streams. The final objective of this section is to evaluate the impact of active drift handling techniques and introduce a mechanism to handle recurring drifts explicitly that will be later used in our main proposal. This will be done by storing previous concept representations in a concept history inspired by previous research [7,156,157,221,376], but using a dynamic time window as a novelty to decide what is the most relevant model in a given point in time.

This section is structured as follows: Section 4.2.1 has introduced the second experiment of this thesis; Subsection 4.2.2 explains the features of the proposed system for the experiment. The experimental design and selection of parameters will be explained in Subsections 4.2.3 and 4.2.4 respectively. Then, Subsection 4.2.5 will present the empirical results of this section and discuss their implications. Finally, Section 4.2.6 will explain the conclusions reached and the future lines of work in this regard as part of this thesis.

4.2.2 System Features

In this section, we propose an approach that leverages work from the adaptive random forest classifier (ARF), described in Chapter 3. ARF handles gradual and concept drifts.

- *Gradual drifts* are handled passively, training decision trees incrementally.
- *Abrupt drifts* are handled actively, replacing active trees with new ones trained only with recent market data when a change detector signals a drift.

Our proposal targets the explicit handling of *recurring concept drifts* explicitly. This algorithm receives the name of Recurring Concepts Adaptive Random Forest RCARF. In our algorithm, decision trees can also be replaced with previously learned trees to retrieve seasonal and recurring behaviours previously learned. The objective of making RCARF able to adapt to gradual, abrupt and also recurring concept drifts is to prepare it to deal with the chaotic behaviour of financial data streams, which can exhibit evolving behaviours but also cycles and stationarities. To do this, we have to propose a mechanism to select the best decision tree among the old ones (in case of cyclic events) or the new active tree in case of a concept drift to a market state unseen. RCARF is described in more detail in the following subsections.

4.2.2.1 Definition of the RCARF Algorithm

Our proposal, namely RCARF, is an adaptive version of the random forest (RF) ensemble [66], introduced in Subsection 2.2.3.3. RCARF extends ARF to handle recurring concept drifts explicitly. Both ARF and RCARF use Hoeffding trees as base classifiers. Thus, we use the term tree to refer to these base classifiers in the rest of this section. In any case, it is important to note that our proposal is not base-classifier specific, and this can be replaced as a parameter in our implementation. In RCARF a tree is always in one of the following states:

- *Active:* base trees running in the ensemble used for test and training purposes. In case of drift, an active tree is moved to the concept history, defined below.

- *Background (one per active tree):* if a detector signals a warning for an active tree, a new tree starts being trained in the background (in parallel to the active tree) until the concept drift is confirmed. These trees are only used for training. Then, either background or an idle tree replaces the active. In this section, we call *warning window* to the training time of a background tree. As a feature inherited from ARF [152], the background tree inherits algorithm parameters values and subspace sizes (related to bagging).

- *Idle (concept history)*: trees that were active previously but replaced due to their low performance in a certain period (in a concept drift). These trees are not tested (by the global evaluator) nor trained unless they re-activate. In this section, when these trees are re-activated, we call them *recurring trees.*

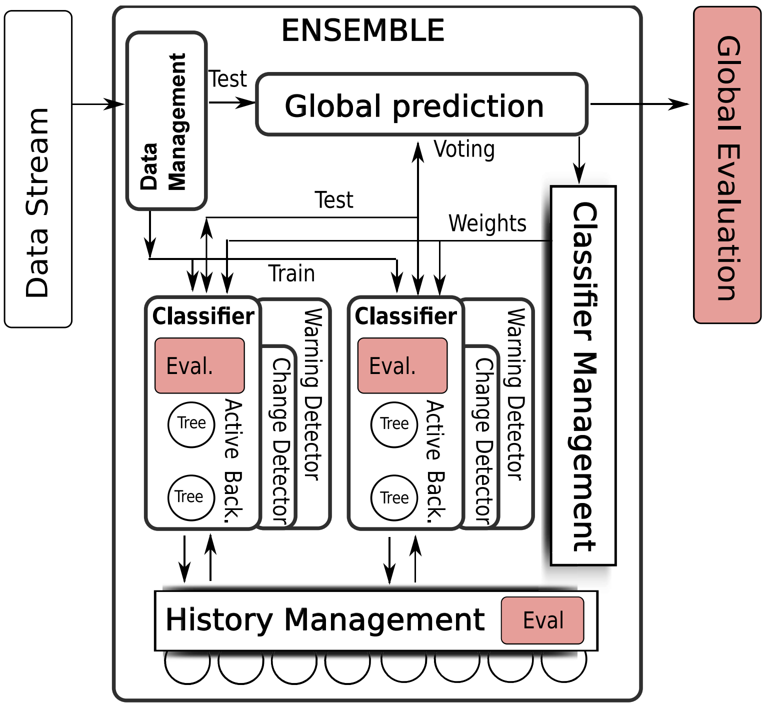


Figure 4.5: *RCARF training workflow.*

The different components of RCARF are illustrated in Figure 4.5. Data instances are received from a data stream and used for test and train. First, RCARF tests the accuracy of the top voted class using the weighted vote of all active classifiers (global evaluator). After testing, each data instance received can be used for training an active tree (and a background tree if this exists).

RCARF uses an active approach to handle concept drifts. This mechanism follows the two-stage convention in data stream mining literature described in Section 2.3.4.2. It has two (supervised) change detectors with different thresholds to signal warnings and drifts. The idea of this is to detect potential drifts as soon as possible and ensure a smooth transition to new trees. When the most sensitive change detector raises a warning signal, RCARF immediately creates and starts training a background tree for the affected classifier. If at some point later the second detector (less sensitive) signals a concept drift, the active tree will be replaced with the new background tree. Otherwise, this background tree will be discarded. When a classifier raises a warning, a temporal copy of it is made to be added to the concept history in case of concept drift. From now on, we will refer to this copy as the active tree *snapshot*. The purpose of this copy is to store the clean classifier in the concept history before it evolves (is trained) during the warning window.

We believe that the warning window represents the process of changing from one concept (state of the market) to another. Thus, the working (active) tree could add noise to the concept history since, as a feature inherited from ARF, the active tree keeps training during the warning window.

Algorithm 4.3 Random forest tree train (RFTreeTrain). Symbols: λ: fixed parameter to Poisson distribution; GP: Grace period before recalculating heuristics for split test; m: maximum features evaluated per split; t: decision tree selected; (x, y): current training instance. Adapted from [152]

```
1: function RFTreeTrain(m, t, x, y)
2:     $k \leftarrow Poisson(\lambda = 6)$
3:     if $k > 0$ then
4:         $l \leftarrow FindLeaf(t, x)$
5:         $UpdateLeafCounts(l, x, k)$
6:         if $examplesSeen(l) \geq GP$ then
7:             $AttempSplit(l)$
8:             if $DidSplit(l)$ then
9:                 $CreateChildren(l, m)$
10:            end if
11:        end if
12:    end if
13: end function
```

As mentioned at the start of this subsection, RCARF leverages some components from ARF. The first inherited part is related to the ensemble approach (bagging, weighting and voting).

The full RCARF (forest) classifier performs a prediction for each incoming data stream instance. Each instance is pushed to most base classifiers for voting (following a bagging approach). Each vote is multiplied by the base classifier's weight; a value adapted later depending on whether the related base classifier prediction matches the real class of the example. Other parts reused from ARF are the function to train new decision trees (see Algorithm 4.3) and the aforementioned active drift handling mechanism (lines 1-11, 14-15 and 22-24 in Algorithm 4.4).

Algorithm 4.4 Recurring Concepts Adaptive Random Forest (**RCARF**). Extracted from [329]. Symbols: m: maximum features evaluated per split; n: total number of trees ($n = |T|$); δ_w: warning threshold; δ_d: drift threshold; $C(\cdot)$: change detection method; S: data stream; B: set of background trees; $W(t)$: tree t weight, $P(\cdot)$: learning performance estimation function; CH: set of historical concepts; R: best transition (between B the model in CH with lowest error); TC: temporal concept saved at the start of warning window.

```
1: function RecurringConceptsAdaptiveRandomForests(m, n, $\delta_w$, $\delta_d$)
2:     $T \leftarrow CreateTrees(n)$
3:     $W \leftarrow InitWeights(n)$
4:     $B, CH \leftarrow \emptyset$
5:     while $HasNext(S)$ do
6:         $(x, y) \leftarrow next(S)$
7:         for all $t \in T$ do
8:             $\hat{y} \leftarrow predict(t, x)$
9:             $W(t) \leftarrow P(W(t), \hat{y}, y)$
10:            $RFTreeTrain(m, t, x, y)$                    ▷ Train $t$ on the current instance(x,y)
11:            if $C(\delta_w, t, x, y)$ then                 ▷ Warning detected?
12:                $lastError \leftarrow evaluate(t)$          ▷ Save overall error of t
13:                $TC \leftarrow copy(t)$        ▷ Copy current tree at the start of warning window
14:                $b \leftarrow CreateTree()$                 ▷ Init background tree
15:                $B(t) \leftarrow b$
16:            end if
17:            if $C(\delta_d, t, x, y)$ then                 ▷ Drift detected?
18:                $t \leftarrow bestTransition(t, B(t), CH)$
19:                $addToConceptHistory(TC)$                   ▷ Push current concept to CH
20:            end if
21:        end for
22:        for all $b \in B$ do                               ▷ Train each background tree
23:            $RFTreeTrain(m, b, x, y)$
24:        end for
25:    end while
26: end function
```

As a novelty, RCARF introduces a component to detect the relevance of trees trained previously and retrieve them from a collection of inactive classifiers called concept history. Hence, RCARF introduces steps required to manage historical trees and perform select the best candidate to replace active trees in case of drift (lines 12-13, and 18-19 in Algorithm 4.4). This collection of inactive trees, shared across all classifiers of the ensemble, is described further in Subsection 4.2.2.2. In order to assist the decision-making process of finding a relevant tree, RCARF has a specific component called internal evaluator (coloured in red under each classifier and concept history in Figure 4.5) that has a dynamic sliding window to consider the last data instances. The internal evaluator is explained in Subsection 4.2.2.3.

4.2.2.2 Concept History

One of the core components of RCARF and one of the novelties of our proposal is the addition of a *concept history* in an adaptive ensemble with supervised drift detectors. The concept history (CH) is a collection of trees previously trained that are available across classifiers for future use if a recurring concept drift is detected. If an active tree is inserted in the concept history, it becomes available for the whole ensemble. If a tree from the concept history is promoted to be an active tree, it is immediately removed from the concept history.

We propose RCARF under the assumption that in very sharp changes such as abrupt concept drifts, a new tree (background) may not offer a good classification accuracy in the short term if it replaces the active classifier. Hence, when a change is detected, two different actions take place. First, RCARF selects the best tree between the background and idle trees in the CH. This can be seen in line 18 of Algorithm 4.4, and in Algorithm 4.5. Second, the active tree affected is moved to the CH and becomes idle, being retrievable in a future concept drift by any other classifier of the ensemble. The idea of a concept history is appealing especially for those circumstances and also in the presence of recurring concepts. In the last scenario, the CH will store trees already trained with concepts that will reappear.

Change detectors in ARF and RCARF monitor the error statistics over time in each base tree. Depending on the thresholds set for each detector, a warning or a concept drift will be raised as the error increases. The term warning window in supervised drift detection has been previously defined in Section 2.3.4.2 as the period (and data instances processed) between a warning and a drift signal. In ARF, each active tree has its own detectors and thus its own warning window. When a change detector signals the likelihood of a concept drift (warning detection, line 11 in Algorithm 4.4) in the active classifier, a background tree is created to replace this classifier in case of concept drift. In case of confirming the drift (drift detection, line 17 in Algorithm 4.4), its warning window finishes, its background tree becomes active, and its active tree is moved to the CH. In RCARF, we propose an novel mechanism to compare the background learner to all the historical trees from the CH. At the start of the warning window, all trees from the CH start being evaluated (tested) to compete with this new background tree. This competition is performed as part of the *internal evaluator* explained in Subsection 4.2.2.3.

4.2.2.3 Internal Evaluator

RCARF has two types of evaluators visible in Figure 4.5; the ensemble (global) one, and the internal one (drawn inside classifiers and CH).

- **Ensemble evaluator** (global, line 8 in Algorithm 4.4): This evaluator handles the predicted accuracy and classification performance of RCARF.
- **Internal evaluator** (tree-specific, Algorithm 4.5): This is one of the novelties presented in this section. During the warning window, RCARF collects relevant information for the selection of the new active tree in case of concept drift. This information is collected over a dynamic sliding window containing the last examples after the warning was signalled. The internal evaluator is updated every time that a new instance is tested. Algorithm 4.6 explains the resizing mechanism of the internal evaluator sliding window in each classifier.

Algorithm 4.5 Internal Evaluator. Extracted from [329]. It computes the best transition in case of drift. Symbols: t: active tree; b: background tree; CH: concept history; c: tree from CH; $WS(CH)$: fixed window size in CH; $WS(b)$: current window size in b, $W(c)$: error statistics in c for the latest examples in $WS(CH)$; $W(b)$: error statistics in b according to $WS(b)$.

```
1: function BestTransition($t$, $b$, $CH$)
2:     for all $c \in CH$ do                                        ▷ Rank of errors of each tree in CH
3:         $addToRank(c, countErrors(W(c))/WS(CH))$
4:     end for
5:     if $minError(rank) \leq (countErrors(W(b))/WS(b))$ then
6:         $R \leftarrow extractClassifier(CH, minErrorKey(rank))$   ▷ Get and remove tree from the concept history
7:     else
8:         $R \leftarrow b$
9:     end if
10:    return R
11: end function
```

Algorithm 4.6 Resizing of the Dynamic Internal Evaluator. Extracted from [329]. Symbols: WS: evaluator window size; W: evaluator window; SI: size increments; MS: minimum size of window.

```
1: function AddEvaluationResults(value = correctlyClassifies ? 0 : 1)
2:     $removeFirstElement(W)$
3:     $add(W, value)$                                ▷ Add result [1 (error) or 0 (success)] to window
4:     $updateWindowSize()$
5:     if $(countOfErrors(W)/WS) < getErrorBeforeWarning$ then
6:         $WS = WS + SI$
7:     else if $WS > MS$ then
8:         $WS = WS - SI$
9:     end if
10: end function
```

When a concept drift is signalled (line 17 in Algorithm 4.4), the tree with the lowest error according to its internal evaluator is promoted to active. The snapshot of the active tree (taken when the warning was detected) is then moved to the CH (lines 18 and 19 in Algorithm 4.4). Figure 4.6 describes the sliding window resizing mechanism in the internal evaluator. The window size (*WS*) decreases when the mean average error of the background tree in the sliding window is lower than the error obtained by the active tree when the warning was detected. *WS* does not decrease beyond a given minimum window size (*MS*) specified as an input parameter. The window size *WS* increases when the mean average error of the background tree in the sliding window is greater than the error obtained by the active tree when the warning was detected. Otherwise, *WS* remains the same. The increments and decrements of the *WS* are performed according to an input parameter that defines size increments (*SI*).

The resizing mechanism proposed in this subsection is based on our interpretation of the error metrics obtained by the background trees during the warning window. Based on observations during the validation study of RCARF, we believe that a considerable part of the errors obtained by background trees during the warning windows is because these have not been trained with enough data instances yet to produce reliable predictions. Other reasons could be underlying data of a continuously evolving nature or a period of quick changes that led to a background classifier trained for several concepts. In these scenarios, the latest examples could be a better metric to measure the classification performance of a given decision tree; hence, *WS* decreases. Otherwise, we believe that a larger sample may be desirable since the new tree will have been tested in a more representative sample; then, then *WS* increases. For the sake of efficiency and reducing computational costs, the internal evaluator changes size dynamically during the warning window only in the active classifiers and background trees. In this section, trees from the CH have a fixed window size provided as an input parameter.

4.2.2.4 Training of the Available Trees

RCARF introduces a novel mechanism to replace the active trees. This creates a need to discuss how data instances are used to train the trees. In ARF, active and background trees are trained with data instances as soon as these become available in the data stream (lines 10 and 23 in Algorithm 4.4). However, in RCARF, there is an added constraint due to the idle trees from the concept history. Idle trees belong to a previous concept and are only trained if selected to replace the active tree.

Background and active trees are trained in parallel during the warning window. Thus, there is no data loss in the case of concept drift transitioning to a background tree since all instances received during the warning window will have been used to train the new active tree. Conversely, trees from the concept history are not trained with data instances received during the warning window if selected in case of concept change. An assumption made by us is that if an idle tree performs better for the last instances, we consider that this has already been trained in the past for a similar concept and thus re-feeding the examples of the warning window is unnecessary.

According to Alippi et al. in [7], there is an intrinsic delay from the start of a concept drift in the ground truth to the start of the warning window. This delay implies that there is at least one period where the model will be inevitably trained with more than one concept. In this section, we avoid considering this delay as part of our analysis for the sake of simplicity. For this reason, we consider that the time when a tree is better adapted to a concept is just before the start of the warning window. This motivates our reasoning in the design of RCARF. Due to this, first, RCARF uses the accuracy of a base tree before warning as a baseline for the window resizing logic of the internal evaluator; second, RCARF pushes the snapshot of the active tree before warning to the CH in case of concept drift (lines 13 and 19 in Algorithm 4.4).

4.2.3 Experimental Protocol

In this subsection, we introduce the setting used for the second experiment of the thesis. As introduced at the start of the chapter, one of the objectives of this section is to benchmark data stream mining techniques in real-world financial data. For this purpose, we have used Exchange-Traded Fund (ETF) SPY prices for the entire first quarter of 2017 at the second level from QuantQuote[5]. This ETF that will be described in more detail in Chapter 6 is one of the most heavily-traded ones, tracking the popular US index S&P 500.

The feature set used for classification tasks is a selection of 10 different technical indicators based on the work by Kara et al. [195], covered already in Chapter 3. This set of indicators is mainly used in the literature at the daily level, and most moving averages use a period of 10 days to compute most of these. We leave the default value to the last 10 prices (thus, n=10 seconds) and add extra moving averages for 5 and

[5] Data source—https://www.quantquote.com

20 seconds. Therefore, we have used a total of 14 features as described in Table 4.5. The indicators are computed using the technical analysis library TA-Lib[6] with all of the default parameters apart from the period (10).

Name of Indicators	Formulas
Simple n-second moving average (5, 10, 20)	$\frac{C_t+C_{t-1}+...+C_{t-n+1}}{n}$
Weighted n-second moving average (5, 10, 20)	$\frac{n\times C_t+(n-1)\times C_{t-1}+...+C_{t-n+1}}{n+(n-1)+...+1}$
Momentum	$C_t - C_{t-n}$
Stochastic K%	$\frac{C_t-LL_{t-n}}{HH_{t-n}-LL_{t-n}} \times 100$
Stochastic D%	$\frac{\sum_{i=0}^{n-1} K_{t-i}\%}{n}$
RSI (Relative Strength Index)	$100 - \frac{100}{1+(\sum_{i=0}^{n-1} Up_{t-i}/n)/(\sum_{i=0}^{n-1} Dw_{t-i}/n)}$
MACD (Moving average convergence divergence)	$EMA(12)_t - EMA(26)_t$
Larry William's R%	$\frac{H_n-C_t}{H_n-L_n} \times 100$
A/D (Accumulation/Distribution) Oscillator	$\frac{H_t-C_{t-1}}{H_t-L_t}$
CCI (Commodity Channel Index)	$\frac{M_t-SM_t}{0.015D_t}$

C_t is the closing price; L_t the low price; H_t the high price at time t; EMA: exponential moving average, $EMA(k)_t$: $EMA(k)_{t-1} + \alpha \times (c_t - EMA(k)_{t-1})$; α: smoothing factor: $2/1+k$; k is the time period of k second exponential moving average; LL_t and HH_t mean lowest low and highest high in the last t seconds, respectively; $M_t : H_t + L_t + C_t/3$; $SM_t : \sum_{i=1}^{n} M_{t-i+1})/n$; $D_t : (\sum_{i=1}^{n} |M_{t-i+1} - SM_t|)/n; Up_t$ means the upward price change; Dw_t means the downward price change at time t. n is the period used to compute the technical indicator in seconds.

Table 4.5: *Selected technical indicators. Formulas as reported in Kara et al. [195] applied to the 1-second level. Extracted from [329]. Exponential and simple moving averages for* 5 *and* 20 *seconds added as extra features.*

The predicted feature (class) has been computed accordingly to Kara et al. [195] for binary classification to indicate the direction of the next price change one second ahead in the EFT.

- If the close price of SPY from a given time $(t-1)$ to the second after (t) increases, then the direction is labelled as 1.
- Conversely, if the close price from $t-1$ to is equal or lower than the close price at t, then the trend direction is labelled as 0.

The experiments only consider data inside trading hours. This is because short-sellers operating in intraday markets tend to be reluctant to hold positions over non-market hours and try to close before the end of the day [84] to avoid price jumps

[6] Technical Analysis library—http://ta-lib.org/

or market behaviour changes overnight. Since some technical indicators selected rely on the 35 previous prices, the first 35 seconds each trading day are discarded after processing the technical indicators to avoid the influence of previous trends and prices before market hours.

The resulting dataset is then modelled as a finite, ordered stream of data, and evaluation and training are performed simultaneously using the *interleaved test-then-train* (prequential) evaluation [49], described in Chapter 2. The error metric used in our experiments is the accumulated classification error. This is computed as described in Equation 2-16 in Section 2.2.5.1. In any case, this accumulated error does not reflect the performance of the algorithms in particular moments. In order to show the results over time graphically, we also report the mean average error on windows over a fixed window of time as done in Section 4.1, using evaluation task from MOA *moa.evaluation.WindowClassificationPerformanceEvaluator*. The window size set in this experiment is 500 instances, which translates to windows of $\approx$ 8 minutes of length.

Again, apart from benchmarking the data stream mining state-of-the-art algorithms for stock trend classification, another goal of this section is to evaluate the utility of the novel recurring drift detection mechanism proposed in RCARF against its state-of-the-art competitors. For this, apart from ARF, we have selected the following learners the data stream mining literature: DWM [205] using Hoeffding trees as base classifiers, an RCD learner [159] of Hoeffding trees, and a Hoeffding adaptive tree (AHOEFT or HAT) [47]. When possible, all algorithms use ADaptive WINdowing (ADWIN2) [46] change detector for warning and drift detection, Hoeffding trees as the base classifier, and the same number of base trees in their configurations. The rest of the parameters used are a plain vanilla setup. Adaptive random forest is an algorithm prepared for multi-threading, training base trees in parallel. Since the number of threads can impact the results obtained by RCARF due to the shared concept history, we have decided to run the experiments in a single thread. For simplicity, we have decided to leave the impact of multi-threading as something out of the scope of our current proposal.

Given the stochastic nature of the RF-based algorithms used in this section, we will run 20 tests for each with different random seeds and report their mean results. The group of individual results per algorithm will be used for post-hoc analysis. The statistical significance of the performance differences among the algorithms will be formally studied first, testing the normality of the distribution of prediction errors using the Lilliefors test. If the null hypothesis of normality is rejected, we will rely on the Wilcoxon test [368]. Conversely, we will test for homoscedasticity using Levene's test and, depending on whether we can reject the null hypothesis or not, the process will end testing for equality of means using t-test. The significance levels that we will consider in the tests for normality and homoscedasticity are at 5%. For the rest, we will consider both 5% and 1%.

In summary, our proposal in this section aims to predict market trends in the very short term and very high frequencies (second level). Still, the generation of any trading signals used to create trading strategies is out of the scope of this thesis. All the algorithms used in the experiments use the raw values of the technical indications (without any normalisation) for binary prediction of ups or stable/downs in incoming instances every second. The ensemble approaches used in this section have a set of base classifiers which predictions are combined through voting to produce a global stock trend classification.

4.2.4 Parameter Selection and Sensitivity

This section does not perform a systematic parameter optimisation since this is only a preliminary study. Hence, the reader must be aware that the performance of the algorithms might be understated. In the experiments, we will use most of their authors' recommended parametrisations or default setups. In any case, the following parameters deserve to be mentioned.

- *Base classifier*: As mentioned in the previous subsection, we will use Hoeffding trees as the base classifier across all ensembles.
- *Change detector*: We use ADWIN2 in all of the ensembles in our experiments as the active change detection method. ADWIN2 is the default change detector in ARF, proving its good performance in [152].
- *Ensemble size*: The number of base classifiers for all ensembles in this experiment is 40. We set this value due to the results reported in [152].
- *Batch size*: ARF, RCARF and AHOEFT are purely incremental approaches. We will process data instances one by one (batch size = 1). In RCD, we will process data in batches of 600 examples, which translates to one batch each 10-minutes. For DWM, we use its default setup.

As mentioned in the previous subsection, in the experiments for ARF and RCARF, we use the ADWIN2 [47] as change detector, described in Chapter 2. ADWIN2 uses a parameter (δ) that relates to its sensitivity. Large values for δ map to smaller thresholds for changes in the statistic monitored (error rate in our case) to detect warnings or drifts. The value of parameters of this sort relies on the signal-to-noise ratio of the specific data stream and critically impact the algorithms' performance using this active drift detection approach.

In ARF and RCARF, there are two ADWIN2 detectors with different δ values to detect warnings and drifts. The value set for the warning detector (δ_w) has to be greater than the value set for the drift detector (δ_d), and a proper balance of these values is critical to ensure an appropriate transition to a new active tree after the warning window.

Parameters	ARF[M]	ARF[F]	ARF[U] & RCARF	RCD
δ_w	0.0001	0.01	0.3	0.15
δ_d	0.00001	0.001	0.15	

Table 4.6: *Sensitivity parameters for ADWIN2.*

Although further experimentation may be needed, during the validation of RCARF, we observed that the δ values set would generally impact more in the ARF ensemble. We believe that RCARF is more robust than ARF since, in case of early drifts, it can switch to an already trained (idle) tree from the concept history instead of switching to untrained background trees as ARF. Because of the sensitivity of ARF to the values set for δ, we have tested three configurations for ARF (two of them: "*moderate*" and "*fast*" (ARF[M] and ARF[F] respectively), recommended by the authors in [152]). During the validation of RCARF, we also noticed that large values for the change detector (δ_w and δ_d) are favourable in RCARF to ensure that enough changes are detected and to use the concept history. These are $\delta_w = 0.3$ and $\delta_d = 0.15$ and are also used in ARF for a direct comparison (ARF[U]). All the δ parameters used for ADWIN2 are shown in Table 4.6. RCD can only use a single ADWIN2 detector; thus, we have selected the same value used for drift detection in RCARF.

Other parameters worth a mention are the related ones to the internal evaluator. The starting size of the dynamic sliding window is 10 instances, with increments and decrements of 1 instance in the background trees and a minimum window size of 5 instances. Finally, we also tested a second configuration for RCARF that resizes the sliding window of the internal evaluator for the idle models of the concept history using the same logic as the background trees and managing independent window sizes in each of the trees. This did not result in major differences.

4.2.5 Results

4.2.5.1 Global Performance Benchmark

In this section, we summarise the results obtained in the second experiment of this thesis predicting the market trend a second ahead. Table 4.7 provides the main descriptive statistics for the accumulated error (%) for all the benchmarked algorithms over 20 runs in the full test and train set. The results are sorted in ascending order from best (lowest) to worst (highest) classification performance. RCARF obtains the most competitive results in terms of mean and median error. The results from Table 4.7 were formally tested with the protocol described in Subsection 4.2.3 and all differences among algorithms are statistically significant at 1%.

	Mean	Median	Var.	Max.	Min.
RCARF	**34.7533**	**34.7538**	0.0002	34.7791	34.7285
ARF[M]	34.8008	34.8007	0.0002	34.8362	34.7769
ARF[F]	34.8309	34.8335	0.0003	34.8591	34.7902
RCD$_{HOEF}$	35.0469	35.0469	0.0000	35.0469	35.0469
ARF[U]	35.1104	35.1114	0.0002	35.1392	35.0881
DWM	35.2364	35.2364	0.0000	35.2364	35.2364
AHOEFT	35.4661	35.4661	0.0000	35.4661	35.4661

Table 4.7: *Global comparison. Accumulated error (%) on the full dataset. Main descriptive statistics over 20 runs. Differences are significant at* 1%.

We believe that the ability to replace active trees with idle trees makes RCARF outperform ARF, which switches to barely trained base trees in the presence of quick concept changes. Since these gains in terms of error are not over the whole period, the final difference between ARF and RCARF is lower than 1% error. This is due to times of stability in the data stream passed and due to other gradual concept drifts that may be handled already by the base tree used (Hoeffding trees). In any case, the financial domain is known for its low signal to noise ratio; thus, any small gains in classification performance can create a competitive advantage for traders or investors using RCARF as an extra indicator for their decision process.

In Table 4.7, ARF[M] and ARF[F] obtain the second and third-best results, followed by RCD and AHOEFT. The fact that AHOEFT obtains the largest error was expected since this is the only not ensemble algorithm used in the experiment (it has a single tree). As covered in Chapter 3, ensembles are known in the literature for improving classification performance in different domains. ARF[M] and ARF[F] overperform ARF[U], a parametrisation that was suggested for a direct comparison with RCARF. As covered in Subsection 4.2.4, this may be because ARF is more sensitive to the drift detector used and its parametrisation than RCARF. Our proposed method can switch to already trained trees if the background tree is not the best alternative in terms of error during the warning window, and this seems to impact the prediction accuracy obtained in the second experiment.

	Mean	Median	Var.	Max.	Min.
# Drifts	3411.1500	3411.5000	3120.2395	3518	3279
Drifts per tree	85.2788	85.2875	1.9501	88	82
# CH Trees	118.2500	119.0000	46.0921	130	106

# Drifts: Number of drifts during the execution; Drifts per tree: number of total drifts during the execution divided by the ensemble size; # CH trees: number of trees in the concept history at the end of the execution.

Table 4.8: *Internal statistics for RCARF on the whole dataset over 20 runs.*

Regarding the impact of the active drift detection mechanism, Table 4.8 shows summary statistics about the number of concept drifts detected by RCARF. The mean size of the concept history at the end of the execution was 118 trees, which translates to ≈118 background drifts. ADWIN2 detected a mean of approximately *3,411* drifts (≈85 per tree). Thus *>3,000* drifts were recurring (to idle trees of the CH). The fact that most of the drifts are recurring and the good classification performance obtained by RCARF prove that the recurring drift method proposed in this section can be valuable in the financial domain. While a more profound analysis regarding the accuracy is desirable, this would require us to be aware of the ground truth regarding concept changes in this financial data stream. This would imply defining what a concept is in financial data, something that we will introduce in Chapter 5 but that we consider out of the scope of this section.

In terms of execution time, RCARF processed the whole data stream on a mean time of 35,263 seconds across the 20 tests. This is less than 10 hours to compute an entire quarter of market data at the 1-second level. Although we have not performed this experiment in the stocks market in real-time, RCARF has demonstrated its ability to classify high-frequency data streams in near real-time.

4.2.5.2 Analysis of Results over Time

Figure 4.6 shows the evolution of the error in RCARF for a short period (the first trading day of the year). The error shown is measured on windows of 500 examples, and drifts are marked with vertical lines. Red dotted lines indicate times when a background tree becomes active, while blue dashed lines indicate times when a concept history tree is re-activated (recurrent drift). Concept drifts are detected throughout the entire stream as shown in Figure 4.6.

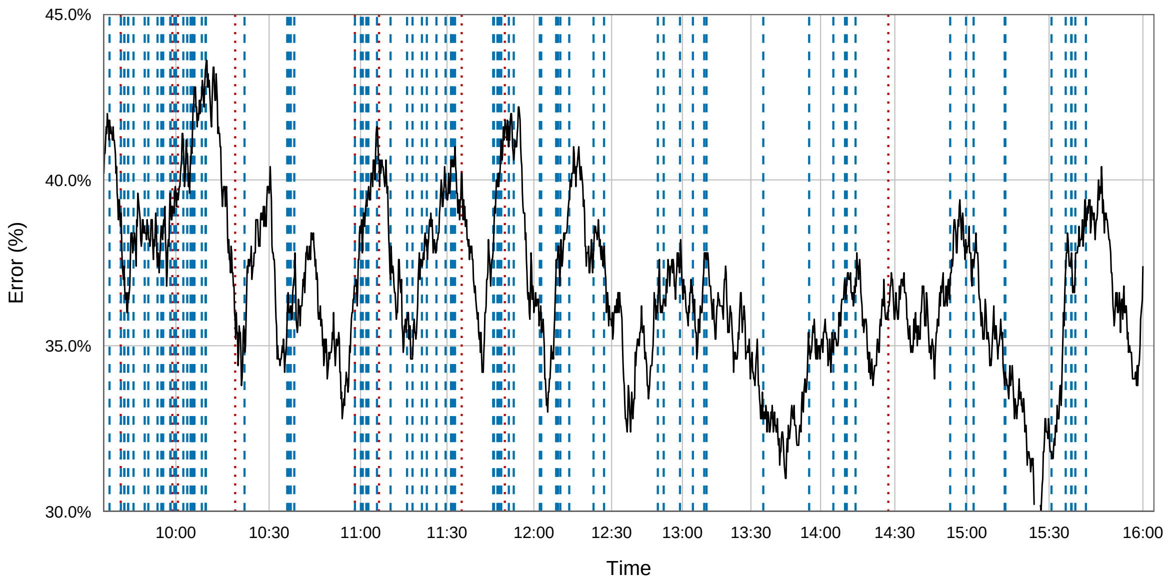


Figure 4.6: *Sample run of RCARF on a single test for the trading first day. Red-dotted and blue-dashed vertical lines mark drifts to background and recurring trees, respectively.*

The start of the sample run in Figure 4.6 shows higher errors. This is because models are not trained yet with a representative number of data instances. Consequently, ADWIN2 detects more changes at the start of the execution, in some cases with very short time frames between them. The time between drifts is are more sparse at the end of the sample execution. It can be observed that most of the transitions are to idle trees from the CH (blue dashed lines), and very few background trees are selected (red dotted lines). Again, this proves that the storage mechanism proposed in this section helps our algorithm in this real-world financial data stream.

Figure 4.7 shows how the different algorithms compare over time in terms of classification accuracy (error) during part of the test and train set. Due to the noise level at the second level, we have smoothed the plot averaging the error on windows of $1,000$ instances to make the trends visible. Still, Figure 4.7 does not seem helpful to extract conclusions of how these algorithms compare to each other. Instead, we rely on the conclusions reached in the previous subsection based on the global performance indicators, and statistical tests reported Table 4.7.

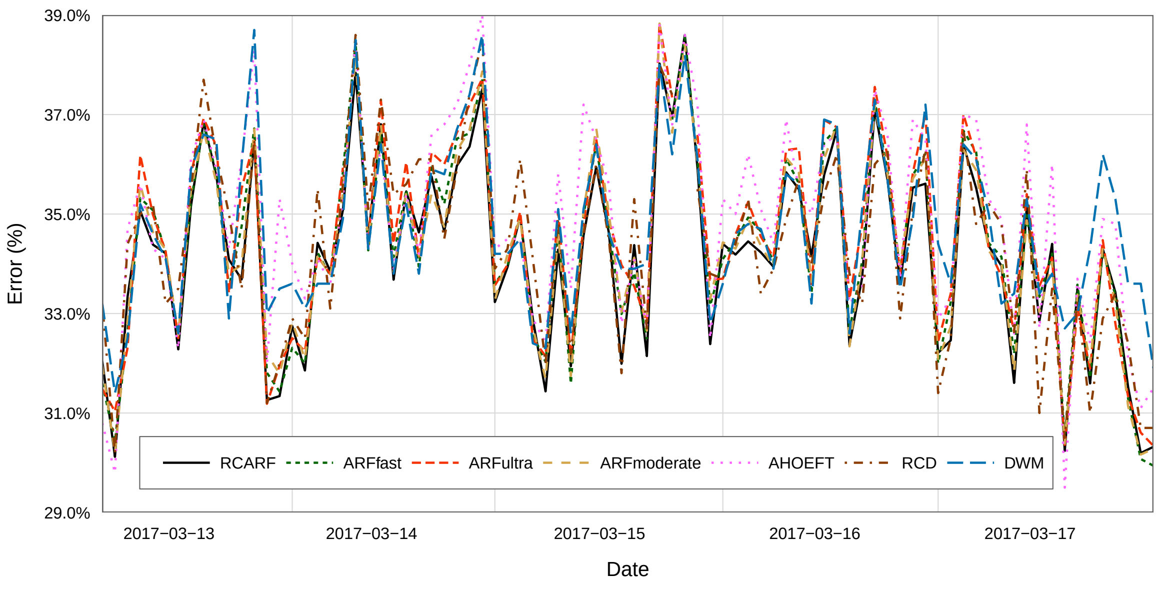


Figure 4.7: *Algorithm comparison over a sample period. For RCARF, ARF[U], ARF[F] and ARF[M] we show the average result of 20 runs.*

In any case, Figure 4.7 is consistent with the results as mentioned earlier. RCARF and ARF[M] behave similarly and often overlap. Their mean results over time tend to be below the errors seen in the rest of the algorithms. This is worth mentioning since it suggests that RCARF and ARF are superior most of the time and not under specific market conditions. RCARF proves to obtain lower errors for short periods in many circumstances. We believe that this is due to the concept history implemented in this section, which helps our method to adapt faster to concept drifts than ARF.

4.2.6 Summary and Conclusions

Section 4.2 has covered the second experiment of this thesis as an introduction to active drift handling and benchmarked state-of-the-art data stream mining approaches in real-world financial data at the 1-second level. With this in mind, we have introduced RCARF, a tree-based adaptive ensemble that handles recurring concepts.

RCARF inherits several features of adaptive random forest (ARF) and adds an element, namely concept history, to store and manage a collection of idle trees that represent previous market behaviours. For this, we have created a decision strategy that selects the best candidate between inactive and new trees to replace active classifiers when these present a concept drift. These features have been designed to optimise classification accuracy in case of abrupt drifts and provide a timely reaction to changes in near-real-time. We consider that RCARF is thus a suitable technique for high-frequency data streams.

The experiment was conducted over a whole quarter of market prices and trade volumes of the SPY with an ETF of high liquidity that tracks the S&P 500 index. Raw data was downloaded at a resolution of 1-second and was pre-processed using a common set of technical indicators from the literature for stock trend classification. The predicted feature indicated if the price increases or stays/decreases one second ahead. Once pre-processed, the dataset was modelled as a finite ordered data stream and fed for online machine learning to RCARF and a set of relevant competitors from the data stream mining literature.

RCARF offered the lowest accumulated error and was statistically significant in 20 tests with a p-value of 1% over its competitors. The main design difference between ARF and RCARF is the ability of RCARF to save old trees and retrieve them in case of recurring concepts. Thus, in this experiment, we have validated the hypothesis that a collection of historical models with previous market behaviours adds more value than simply continuous adaption. In the main proposal of this thesis, in Chapter 5, we will use the idea of reusing previous models that achieved the best results that in this chapter. This has exciting implications in the domain tackled in this thesis since it could support the notion of stationarities and cycles in the market structure. Being able to recognise these previous states can give traders an investor competitive advantage to obtain excess returns. Hence, it is something to be covered further in the rest of this thesis.

The second experiment has focused on stock trend prediction with different mechanisms to handle concept drifts. Still, as mentioned already, we do not intend to derive any trading system. Instead, we target the intersection of the data stream mining and computational finance domains from a research point of view. RCARF aims to indicate up or stable/downs in the stock price and may be helpful to traders. Recognition of this fact is an extra insight for financial experts that can lead to profits in the markets.

Future extensions of RCARF could include a refinement of the: i) internal evaluator, ii) the decision mechanism during the warning window, and iii) the management of classifiers performed in the concept history. Some of these are considered in our main proposal in Chapter 5.

Other extensions of this work beyond the scope of this thesis could include: i) high-performance optimisation of RCARF for ultra-high frequencies (tick-level resolution) like its parallelisation for distributed systems, and ii) the addition of a meta-learning layer to evaluate the likelihood of transitioning to different recurring market behaviours, as seen in many of the approaches described in Section 3.4.

Having said this, the reader must note that both sections in this chapter were part of preliminary research works that led to the proposal in Chapter 5. While iGNGSVM gave us a reliable mechanism to summarise concepts, from RCARF, we have learned the basis to build a collection of idle classifiers for model reuse. In Chapter 5 there will be further need than in this chapter to know and analyse the ground truth in order to evaluate the changes and recurrences detected further than in this section. With this in mind, we will aim to simplify our solution to propose a novel approach that allows us to analyse the accuracy of the drifts recognised. Hence, our main proposal will not use components like the SVM classifier from iGNGSVM, the bagging component or the ensemble approach from RCARF.

Chapter 5

Proposal

5.1 Meta-classifier to Deal with Drifts and Recurrences

As covered in Chapter 2, financial time series may exhibit both non-stationary and stationary behaviours. In some cases, patterns will repeat stationarily but may also evolve up to a certain degree, and non-stationary patterns may also occur within a given market state. This topic from the financial literature can be easily mapped to the literature of evolving data streams and recurring concept drifts. With this in mind and from lessons learned in Chapter 4, in this third experiment, we propose a meta-learner, namely *Growing Concept History of Recurring Classifiers* (GroCH), that replaces its active classifier in case of concept drift as illustrated by Figure 5.1. Historical classifiers can be retrieved from a list of concepts or groups called concept history. GroCH combines many meta-learners when there is a potential drift occurring. This is done to deal with concepts that come and go during gradual shifts and improve prediction accuracy during changes in the time series. We apply this algorithm to a dataset that behaves as financial data to identify different market states as groups in the concept history. Each group is identified by a topology (or set of prototypes) and has a number of old active classifiers.

This chapter is structured as follows: first, we define the features of the algorithm proposed; then, we describe the framework used to evaluate the algorithm during concept drifts, and the simulation of financial time series to know the ground truth. Finally, subsection 5.3 details the experiments performed, including the parameter optimisation process, the analysis of results, limitations and future lines of work.

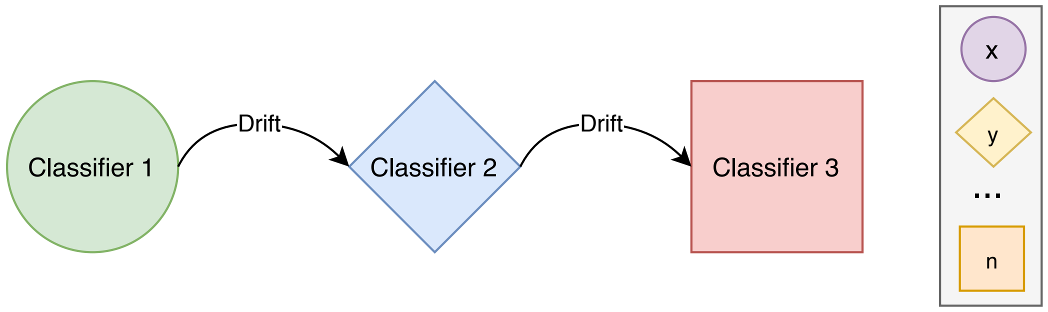


Figure 5.1: *Changes between active classifiers at concept drifts. The right hand side rectangle represents the history of previous classifiers to be reused in the future.*

5.1.1 System Features

The idea behind our proposal, a *growing concept history of recurring classifiers* (namely **GroCH**), is the development of a one-pass and adaptive framework that works for different types of drifts and stationarities or recurring behaviours. Some of the contributions of this approach are: i) new mechanism to handle recurring drifts and to evaluate using the best learner during the warning window, ii) the notion of *groups* inside the concept history, and iii) the techniques used to disable warnings and drifts. Part of our proposal is targeted at cases of abrupt drifts. In such scenarios, retrieving a *classifier snapshot* from the past has a lower computational cost than training or adapting a new classifier (see Section 4.2).

In GroCH, a classifier can be in one of three states as seen for RCARF in Section 4.2, albeit the definition of these states changes slightly from the previous work presented. These states are the following:

- *Active*: classifier currently being tested and trained. In the case of concept drift, the active classifier is moved to a group of the concept history.
- *Training in the background*: called background learner, this is initialised at the start of the warning window and trained in parallel until the detectors signal a concept drift in GroCH. Then, it can become an active classifier if this is the best performer in the last testing examples (warning window evaluator). This learner is also available for testing during the warning window; this is explained further in Subsection 5.1.3.2.
- *Idle*: The concept history is a set of groups with pools of idle classifiers that were active, replaced in previous drifts. At the time of drift, an active classifier is sent

to a group of the concept history that resembles the current concept; this is then replaced by background or an idle classifier. Throughout this work, when these classifiers are re-activated, they are called recurring classifiers.

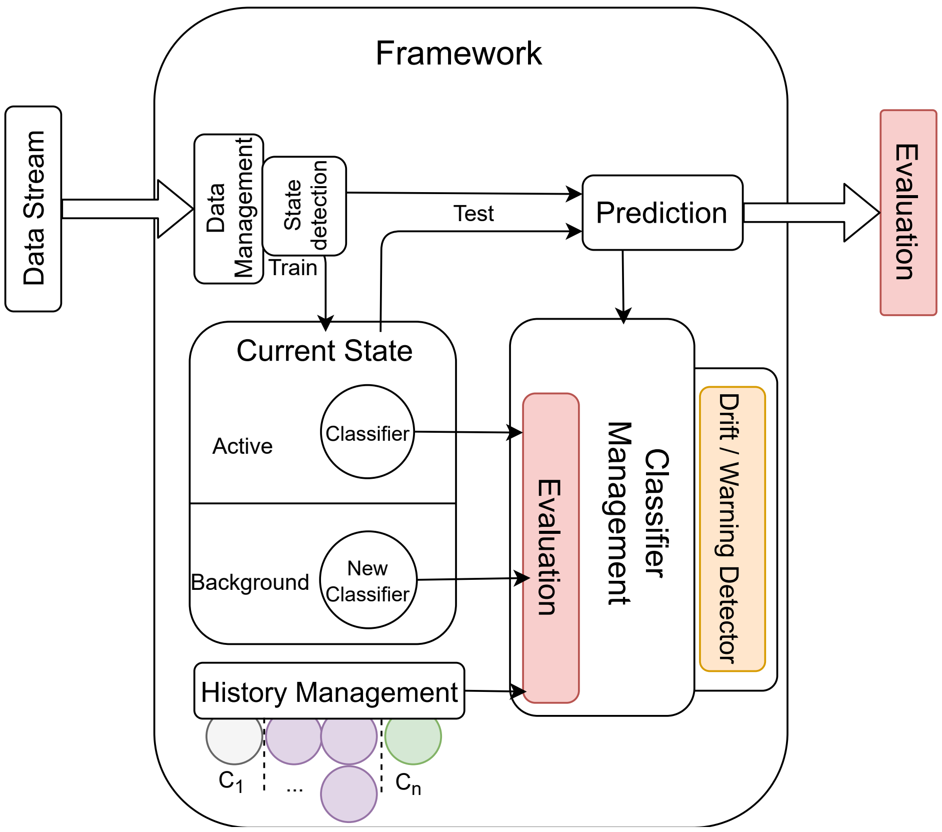


Figure 5.2: *GroCH training workflow.*

Figure 5.2 shows the flow followed in GroCH. First, incoming data examples are tested using the predictor. After this, each example is also used for training the active classifier. As in other algorithms using drift detection in the literature [14, 151, 159], when the error statistics increase over time up to a certain threshold, a warning is activated, and a background classifier is created to replace the active model in the case of drift. A change detector decides if the algorithm must be prepared for a concept drift (warning detection, Algorithm 5.1, Line 9) or if the drift has already happened (drift detection, Algorithm 5.1, Line 15).

The *warning window* is defined as the period of time that starts when GroCH raises a warning and finishes when it detects a drift. The warning window resets if its length is greater than a certain threshold (Algorithm 5.1, Line 36), but it does not reset in the case of false alarm (Algorithm 5.3). In GroCH, there is an online evaluation of all the classifiers to compare their performance on the latest examples during the warning window. Only when a drift is detected, a copy of the classifier with the lowest error according to the warning window evaluator becomes active (Algorithm 5.1, Line 28). The former active classifier is moved as an idle classifier to the closest group in the concept history (Algorithm 5.1, Line 22). If during the warning window, the active classifier performs with lower classification error than the background classifier and all the classifiers of the closest group from the concept history, and a false alarm is produced. Thus, the drift signal resets, but the GroCH remains in the warning window. As the warning window continues, in this case, the topology is not updated, and the active classifier is neither replaced nor pushed to the concept history.

In certain cases, due to the sharpness of a change, GroCH does not capture enough examples during the warning window to have a decent spatial representation of the new concept. In this chapter, we call early drifts to any changes signalled by detectors under this circumstance. A concept drift is considered to be *early* if its warning window receives less than β examples. In this scenario, GroCH does not consider historical classifiers, and only background drifts or false alarms can be produced.

5.1.2 History Management and State Detection

Algorithm 5.1 shows the overall pseudo-code for the GroCH algorithm. A warning signal (raised by a drift detector) triggers the creation and training of a new background classifier. If a drift signal confirms the drift later and the background classifier presents the lowest error, it replaces the active learner. Otherwise, this is discarded. If the active learner is replaced, this is moved to the concept history as an idle classifier.

The concept history is divided into sets called groups. Each group has a pool of inactive classifiers and a topology representation of the ground truth during the warning. Topologies are a summary of the data distribution received during a concept and are used to compute concept similarity. This is explained in Section 5.1.2.1. Groups contain a set of classifiers that were created during the execution of the algorithm and stored for future usage when an episode of concept drift impacts the performance of the active classifier. If during a drift, a group from the concept history resembles the current dynamics of the data stream, classifiers from this group are taken into consideration to be retrieved from the concept history (recurring drift).

Algorithm 5.1 GroCH algorithm. δw: warning threshold; δd: drift threshold; S: data stream; c: classifier; b: background classifier; CH: concept history; W: set of instances received during the last warning window; P: set of prototypes of the topology before warning; T: topology containing a set of prototypes or cluster-centers representing a data distribution; T_C topology representing the actual distribution; T_D: topology that represents the data distribution of the selected classifier for drift; G: a group of classifiers in CH, represented by a topology of prototypes; G_C: group of the base classifier representing the distribution before the warning window; c_{Temp}: temporal copy of the active classifier and topology saved at the start of warning window; (x, y): current training instance; μ represents the threshold for the maximum size of the warning window; β: minimum of instances seen by the active classifier for insertion in the CH; θ threshold for the minimum size of W, otherwise a drift is considered early drift; Ω: max. number of classifiers per group.

```
1: function GrowingConceptHistoryOfRecurringClassifiers($\delta$w, $\delta$d)
2:     b, $G_C$, $G_H$, CH ← 0
3:     if preTraining == True then prePopulate(CH) end if ▷ The CH can be prepopulated (see subsection 5.1.3.5)
4:     W ← -1                                                    ▷ Warning window disabled initially
5:     $T_C$ ← StartTopologyAlgorithm()                          ▷ GNG starts (only one topology)
6:     c ← CreateClassifier()
7:     while HasNext(S) do
8:         (x,y) ← next(S)
9:         if C($\delta$w, x, y) then                               ▷ Warning detected?
10:            UpdateTopologyAlgorithm($T_C$, W)     ▷ Add examples from prior warning to $T_D$ before reset W
11:            W ← 0
12:            $c_{Temp}$ ← c                                    ▷ Save a temp copy of c as snapshot
13:            b ← CreateClassifier()
14:        end if
15:        if C($\delta$d, x, y) then                               ▷ Drift detected?
16:            if size(W) $\geq \theta$ then                      ▷ Insertion and retrieval from CH
17:                P ← getPrototypes($T_C$)
18:                $G_C$ ← FindGroup(P)                          ▷ Group for storing old state
19:                if CH is empty or $G_C$ is Null then $G_C$ ← CreateNewGroup($T_C$) end if ▷ New group with $T_C$
20:                if InstancesSeen(c) $\geq \beta$ and ($G_C \neq G_H$ or Error($c_{temp}$) $\leq$ minError($G_C$)) then
21:                    if Size($G_C$) $> \Omega$ then RemoveClassifier($G_C$) end if        ▷ Usage of policy
22:                    CH($G_C$) ← $c_{Temp}$                    ▷ Push current classifier to $G_C$
23:                end if
24:                $G_H$ ← FindGroup(W)                          ▷ Group for retrieval of next state
25:            else
26:                $G_H$ ← $-1$                                  ▷ No retrievals in case of early drift
27:            end if
28:            c, $T_D$ ←SelectDrift(c, b, $G_H$)                ▷ Find best transition for next state
29:            $T_C$ ← UpdateTopologyAlgorithm($T_D$, W)    ▷ Add instances from warning to $T_D$ and replace
30:            W← -1                                             ▷ Reset of W in bkg or recurring drift
31:        end if
32:        if size(W) $\in [0, \mu]$ then                          ▷ In warning window
33:            UpdateWeights(c, b, x, y)   ▷ Update weights for the evaluation according to equation 5-1
34:            ClassifierTrain(b, x, y)                          ▷ Train the background classifier
35:            W(t) ← (x,y)                                      ▷ Add example to W
36:        else if size(W) $> \mu$ then               ▷ Disable warning if W.size > threshold $\mu$
37:            b, $c_{Temp}$ ← 0
38:            UpdateTopologyAlgorithm($T_C$, W)                 ▷ Add instances from warning to $T_D$
39:            W← -1
40:        else                                                  ▷ If not in warning window
41:            UpdateTopologyAlgorithm($T_C$, x, y)               ▷ Update topology
42:        end if
43:        ClassifierTrain(c, x, y)                 ▷ Train c on the current instance (x, y)
44:    end while
45: end function
```

A group models the relationship between a state of the data distribution (a topology) and a classifier set. For instance, in a simple scenario, a topology would relate to a unique classifier. However, in practice, a topology may map to many classifiers.

This can happen due to the lack of attributes able to reflect the hidden context in the case of concept drift. For this reason, in GroCH, each group stores a pool of many classifiers. There is still a maximum number of classifiers per group, denoted with Ω, that can be set to 1 for simple scenarios. When a group has reached Ω classifiers (line 21), a replacement policy manages what classifier should be removed. By default, the classifier retrieved the least number of times is the one replaced. In case of a tie, it replaces the oldest of the *least used* classifiers of that group (LUFO policy).

Finally, GroCH allows to pre-train the concept history. It prepopulates a set of groups and classifiers to be available at the start of the execution. This is explained further in Subsection 5.1.3.5.

5.1.2.1 Concept Similarity

A *topology* summarises the data received at a given point in time. Topologies are represented as part of the state detection module in Figure 5.2, and are a novelty in our approach. They help improve the selection mechanism in the case of concept drift. This is achieved by considering the state of the data stream apart from the performance of the active classifier, which is monitored by the change detectors.

Each group is represented by one topology, which summarises the examples received during the warning window when the group was created. Each summary or topology is computed using a space tessellation through the generation of prototypes created the first time that a classifier was added to the group.

Algorithm 5.2 Concept history grouping. CH: concept history; G: single group in CH; P: the set of prototypes or cluster-centers received; P_G: the set of prototypes describing a group G from CH; σ: maximum radius from P to P_G allowed to belong a certain group G.

```
1: function FindGroup(P)
2:     min ← FloatMaxValue()
3:     for all $G \in CH$ do
4:         distancesToGroup ← 0
5:         dist ← getMeanDistanceToNearestNeighbour(P, $P_G$)
6:         if dist < min then
7:             min ← dist
8:             group ← g
9:         end if
10:    end for
11:    if dist < $\sigma$ then
12:        return group
13:    else
14:        return null
15:    end if
16: end function
```

When a drift is signalled, GroCH *stores* the classifier that belongs to the previous concept (saved at the start of the warning window) and *retrieves* a classifier from a group representing the next concept from the CH (if it reoccurs). To perform this, distances need to be computed between the latest received examples and trained prototypes of a group topology (for retrievals and insertions, respectively).

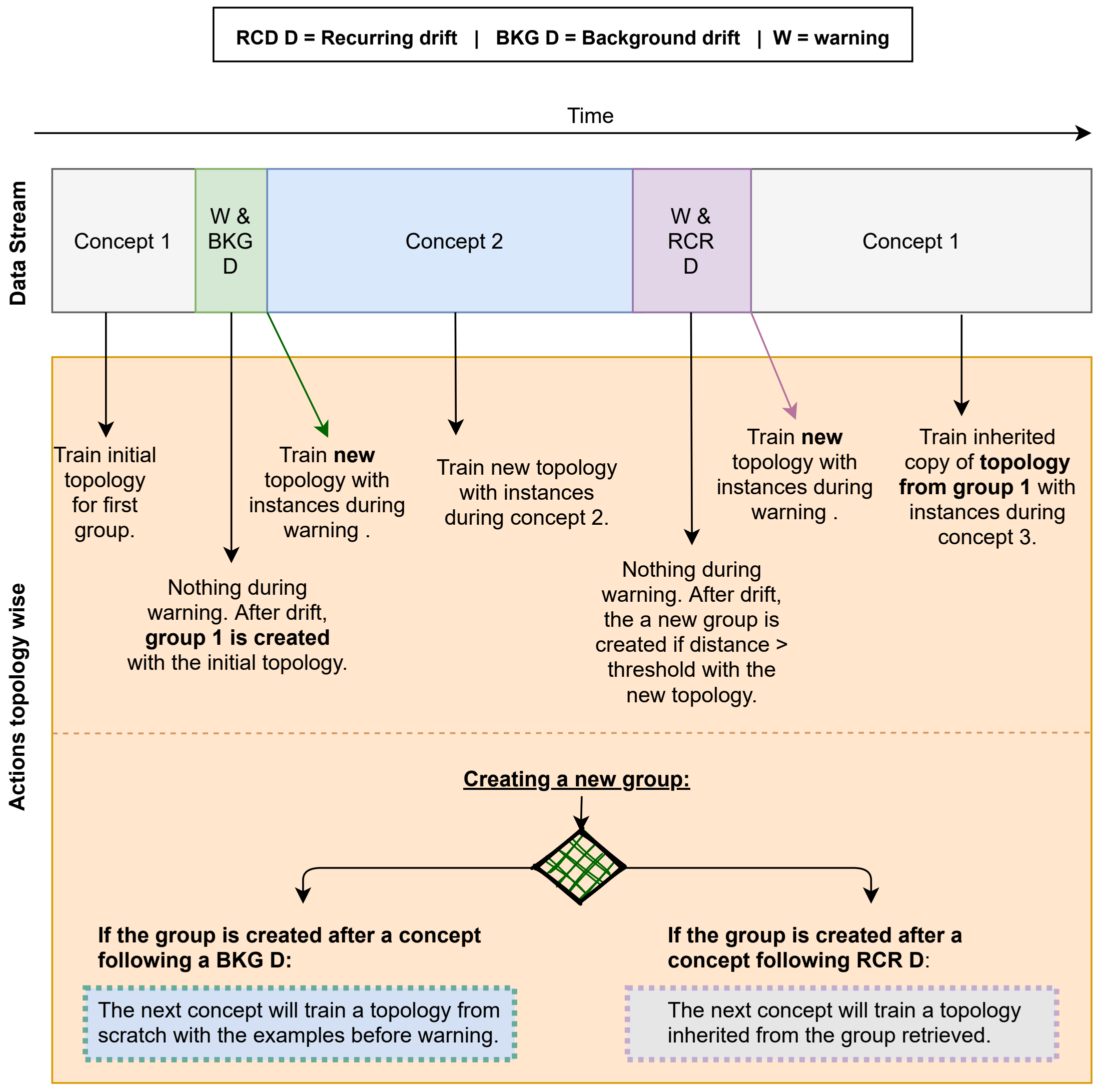

RCD D = Recurring drift | BKG D = Background drift | W = warning
Time
Data Stream
Concept 1
W & BKG D
Concept 2
W & RCR D
Concept 1
Actions topology wise
Train initial topology for first group.
Nothing during warning. After drift, group 1 is created with the initial topology.
Train new topology with instances during warning .
Train new topology with instances during concept 2.
Nothing during warning. After drift, the a new group is created if distance > threshold with the new topology.
Train new topology with instances during warning .
Train inherited copy of topology from group 1 with instances during concept 3.
Creating a new group:
If the group is created after a concept following a BKG D:
The next concept will train a topology from scratch with the examples before warning.
If the group is created after a concept following RCR D:
The next concept will train a topology inherited from the group retrieved.

Figure 5.3: *Example flow of GroCH groups' management.*

We introduce the idea of topologies to represent a set of classifiers that work well in a certain type of circumstances and support the calculation of distances for *concept similarity*. For example, each group can hold memories of how market operators reacted in the past in similar situations.

There are three primary functions concerning concept history groups:

- *Retrieval*: At the time of the drift, the mean distance between the examples received during the warning window and their (closest) prototypes are computed for all existing topologies (one per topology). If the mean distance between any of the groups is under the maximum topology radius (distance threshold σ in Algorithm 5.2), all classifiers from the nearest group are compared for the set of examples received during the warning window. Comparisons are performed by *conceptual equivalence* [376]. If two or more classifiers of a group present a similar conceptual equivalence during the warning window, the algorithm follows a FIFO (First In, First Out) or a LIFO (Last In, First Out) policy, giving priority to old or new classifiers respectively. If a classifier from the nearest group is finally selected, this is copied (not removed) from the group as a new active learner.

- *Insertion*: After the new active classifier is selected, the old active classifier is pushed to a group from the concept history (Algorithm 5.1, line 22). For this, the algorithm shall find the best group for insertion using a topology trained for the old active classifier up to the start of the warning window (from the last drift to the last warning signal). To detect the most similar group for insertion, the algorithm computes the mean distance between each group's topology and the set of prototypes of the old active learner's topology. A classifier is not inserted into a group when, during concept drift, the retrieval and insertion point to the same group and the conceptual equivalence of the old classifier is not greater than for the best classifier of the group to be inserted. The motivation for this is to reduce noise in the groups due to false positives of the drift detectors.

- *Creation*: If at the time of the insertion, there are no existing groups in the concept history, or if the mean distance between the old active learner's topology and all groups exceeds a threshold (σ), then a new group is created. The process to create a group is illustrated in Figure 5.5.

A more detailed and simplified example flows for insertions and retrievals can be seen in Figures 5.3 and 5.4 respectively. In the next page, we explain step by step each action in Figure 5.4. To simplify the example, we assume that all relevant concepts are represented in the concept history; thus, not needing to create a group.

1. First, *GroCH trains a topology* with the received examples up to the warning signal.

2. Once GroCH flags a warning, all received examples are saved in a buffer up to the time of the drift (during the *warning window*). At this time (drift) is when the algorithm evaluates insertions and retrievals.

3. The algorithm *inserts* the previous concept to a concept history group. For this, the topology trained in the previous concept (up to the warning) is compared to all existing group topologies. The ideal behaviour in the first and second drifts is to insert a group mapping to the grey and red concepts in Figure 5.4 respectively.

4. Then the algorithm *retrieves* classifiers from the nearest group to compare their performance during the warning window. The group to be retrieved is identified by computing distances to examples received during the warning window. As these examples belong to a transition period, these are considered a representation of the next concept. Assuming that the concept drifts are recurrent, the ideal behaviour in the first and second drifts is to retrieve a pre-trained classifier from a group that maps to the red and blue concepts in Figure 5.4 respectively.

5. Finally, after the concept drift, the algorithm starts classifying using the new learner retrieved. In the background, a new topology is trained, starting again step 1.

In this work, concept or *group similarity* has been computed using Euclidean distance and Mahalanobis distance distances in different experiments.

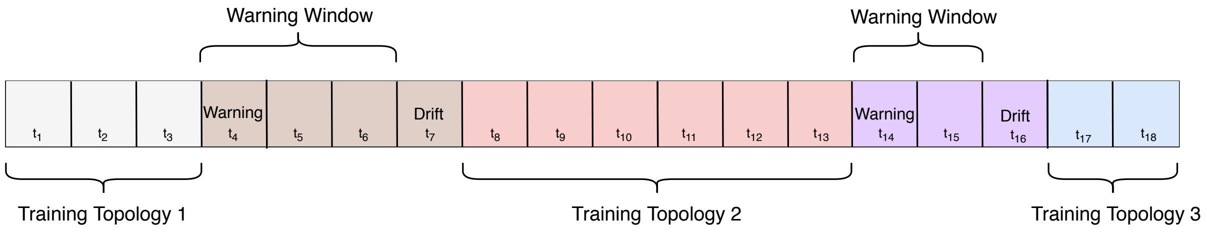
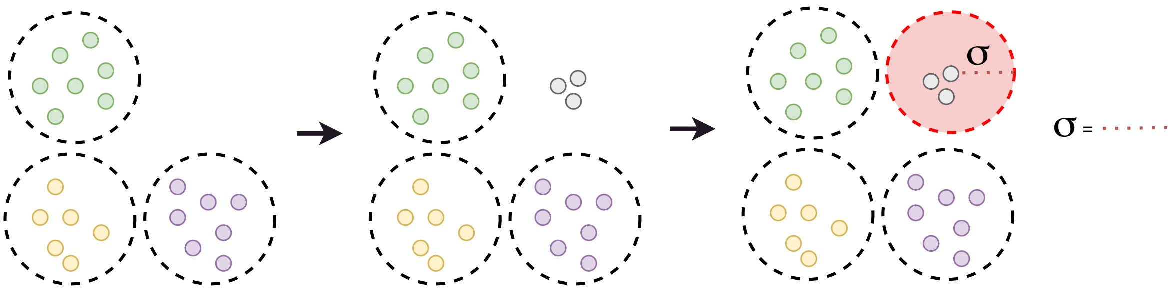
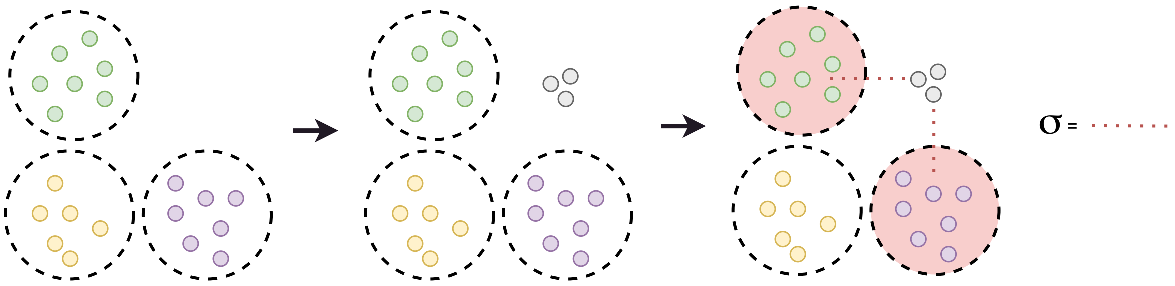


Figure 5.4: *Example of events in a data stream in our proposal. Each t in a cell represents a period of one or many time steps. Gray, red and blue coloured cells belong to separate ground truth contexts. Brown and purple coloured cells correspond to periods of transition. Drift-related events performed by the algorithm are marked as a warning or a drift; topology training and instance-buffering processes take place during the warning window.*



(a) *Creation (groups outside σ)*



(b) *Insertions and Retrievals (groups inside σ)*

Figure 5.5: *Process to create or insert into existing groups of the concept history depending on the value of the maximum topology radius (σ). In 5.5a, during retrievals, there is no chance of recurring drift, and during insertions, a new group is created. In 5.5b, the classifier is retrieved or inserted in the closest of the two (red) groups inside σ.*

5.1.2.2 Learning Topologies

The approach proposed in this chapter is a modular meta-learning framework where all base algorithms can be changed. However, in this first proposal group's topologies are computed using the growing neural gas (GNG) algorithm [136], also used in the first preliminary study.

As described in Chapter 2, GNG is a prototype generation technique that creates a growing network of neurons (which we will call prototypes from now on) interpolating the examples of the farthest distribution from the network. Hence, prototypes are created incrementally at mid-distance between the closest prototype already created in the network and the example fed with the largest accumulated squared error. The process runs until satisfying a stopping criterion that, in our case, is the drift signal. Topologies are trained as a one-pass algorithm. Each data instance received by GroCH up to the warning window is only fed to once to a topology.

Although GNG computes Euclidean distance distances by default, we have implemented Mahalanobis distance distances for experiments using these to compute similarity (as seen in the previous section). The reason to choose Mahalanobis distance distances is that Euclidean distance distances do not focus on the relationship of the different indicators of the dataset as the former does. The Mahalanobis distance distance metric considers the intersections of different features and their intrinsic correlation as part of the definition of a state or stationarity in the data stream.

Later, in the experimental section, we also mention the usage of feature subsets for the topologies. This is specific to the use of financial time series, where we have run experiments looking at smaller feature sets that are scaled only to reduce the effect of explosive price changes.

5.1.3 Classifier and Drift Management

Once a concept drift is detected, GroCH updates both the current topology and the active classifier. The classifier obtaining the lowest classification error using the examples received from the last warning is selected as the new active learner. The choice is made between i) the active classifier, ii) a classifier trained in the background since the warning signal, and iii) any classifier considered for recurrence from a concept history's group. Therefore, there are two types of drift:

- *Background drift*: the background classifier is selected as the new active learner.
- *Recurring drift*: a classifier from the nearest group of the concept history is selected as the new active learner.

There is a third case when the active classifier is selected as the best performer despite the drift signal raised by the drift detectors. This case is considered a false alarm, as it was advanced in Section 5.1.1.

5.1.3.1 Evaluation During the Warning Window

The period between a warning and a drift signal is called warning window by the relevant literature [158]. During the warning window, examples are sent to a buffer W (Line 35 in Algorithm 5.1), and the topology is not trained. Once the drift is confirmed, the active topology can either reset (in the case of background drift) or be replaced by another topology from the concept history(in the case of a recurring drift). Then, the newly reset or replaced topology is trained with the examples from the buffer W (Line 29 in Algorithm 5.1). The input parameters of the warning detector must be more sensitive than the drift detector to ensure the proper behaviour of GroCH.

GroCH evaluates the performance of the classifiers at two different levels. Following Figure 5.2, it evaluates after the prediction block and it evaluates as well in the classifier management block. The first evaluation is used for the testing task (overall performance of the algorithm over time). The second level of assessment occurs during the warning window; this is used for selecting the new active learner in case of concept drift. These two algorithm evaluation levels can be further described as follows:

- *Testing evaluation*: this is the main evaluation of the algorithm. The predicted accuracy of the algorithm is reported by this.
- *Warning window evaluation*: during the warning window, GroCH only compares results from the latest examples received. Considering that the time series is switching to a different state, this evaluator is used to optimise the decision of the best classifier if the drift is finally signalled. To enable a one-pass setting for data streams, this evaluator is updated every time that a new example is tested.

Once a concept drift is signalled, a classifier for the next concept must be selected (Algorithm 5.3). The evaluation process during the warning window processes the errors of all relevant classifiers for the full warning window. It plays a key role in this decision-making process which is as follows:

1. *Evaluation of relevant classifiers*: at the time of drift, active, background and every classifier from the closest group (if any inside a predefined threshold) of the concept history are evaluated over all the examples received during the warning window.
2. *Transition to a new classifier*: of all relevant classifiers, the one with the lowest accumulated error over the examples received during the warning window is selected as the new active learner.

5.1.3.2 Ensembling for Predictions in the Warning Window

As explained at the start of this section, GroCH creates a background classifier at the start of the warning period. Thus, during this time spawn, GroCH trains two different classifiers representing the previous and next concept in the data stream. As a novelty in our approach, GroCH becomes an ensemble classifier only during the warning window. The purpose of this is to use the best available classifier for predictions at every time. There may be cases (e.g. gradual shifts) where concepts start changing, or even end before a concept drift is signalled. We believe that GroCH can handle such scenarios using an approach where the active classifier competes with the background classifier for every prediction.

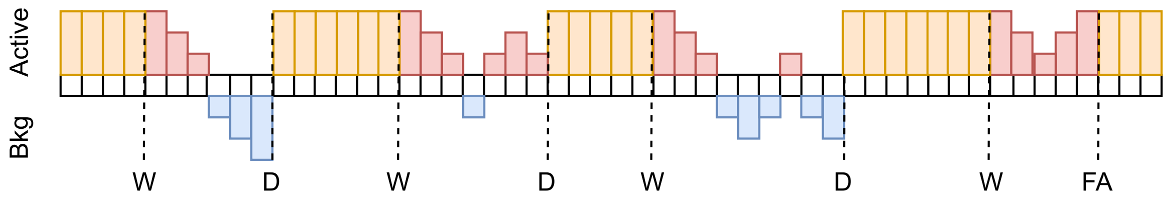


Figure 5.6: *Illustration of the weighting mechanism of GroCH in the warning window. Blue rectangles indicate predictions made by the background evaluator, and their length indicates their relative weight. W: warning; D: drift; FA; false alarm.*

To achieve this, both active and background classifiers have a weight associated. The initial weight is 1 (max) for the active classifier and 0 (min) for the background classifier. The weights of both learners evolve using a punishing factor (β) like the one seen in [205]. The main difference with [205], as well a novelty in GroCH, is that learners are also rewarded when they classify correctly to deal with the graduality of some concept drifts.

$$w_t \leftarrow (w_{t-1} * \beta * \overline{\text{correctlyClassified}} + \frac{w_{t-1}}{\beta} * \text{correctlyClassified}) \tag{5-1}$$

This evolution follows the equation 5-1 and is based on the performance of each learner during the examples received over the warning window. The update of weights can be seen in the line 33 of algorithm 5.1. Weights are reset at the start of the warning window and are only updated during this window.

For every batch of examples (each example if batch size = 1), equation 5-1 is computed over both active and background classifier. At every iteration, GroCH predicts accordingly to the learner with the greatest weight. This process is illustrated by Figure 5.6. A version of this equation for a regression setting may only imply replacing $\overline{correctlyClassified}$ and $correctlyClassified$ with error and (1−error) respectively, and to change majority voting with another aggregation technique (mean result).

5.1.3.3 Alignment of Classifiers and Topologies

The active topology (T_c in Algorithm 5.1) is not trained during the warning window as this belongs to the previous concept. However, if this warning does not materialise as a drift, the examples received during the warning window can still be relevant to this topology. This materialises in the case of having a false alarm or if another warning signal replaces the previous warning, which can occur depending on the drift detector used. In these scenarios, GroCH feeds the instances received during the warning window, saved in a buffer (W), to the topology (lines 10, 29 and 38 in Algorithm 5.1). This step ensures that classifiers and topologies have seen the same instances at the point in time of retrievals and insertions from and to the CH.

This training of the topologies with instances from W also takes place in the case of background drifts. A new topology is created then, and the examples seen during the warning window are their first training examples (line 14 in Algorithm 5.3 and line 38 in Algorithm 5.1). Albeit the active classifier is trained during the warning window, the classifier used for insertions is a snapshot of this made at the start of the warning window.

5.1.3.4 Drifts and False Alarms

The warning and drift change detectors, and the actions taken in consequence, are in lines 9-14 and 15-31 in Algorithm 5.1. Actions performed during warning that impact in the topologies are in lines 32-42, and reset of warning in lines 36-40. The steps required to manage the concept history and how to perform an informed decision as the replacement of the active classifier in case of concept drift are in lines 17-19 and 22-29.

Algorithm 5.3 Decision-making mechanism when a concept drift is detected. Warning window (internal) evaluator. It computes the best transition in the case of drift. Symbols - c: active classifier; b: background classifier; W: set of instances received during the last warning window; θ: threshold for the minimum size of W; T_C: active topology; T_G: topology of group G; G: a group of historical classifiers in concept history; idle$_C$: classifier from group G in CH; W(c): error statistics in the active classifier (c) during the warning window; W(b): error statistics in the background classifier (b) during the warning window; W(idle$_C$): error statistics of a classifier of the closest concept history group during the warning window.

```
1: function SelectDrift(c, b, G, $T_C$, $T_G$)
2:     if G is not null and size(W) $\geq \theta$ and $T_G \neq 0$ then ▷ If it is not an early drift and a relevant CH group exist
3:         for all $idle_c \in$ G do                                   ▷ Rank of errors of each classifier in G
4:             addToRank($ch, W(idle_c)$)
5:         end for
6:         if minError($rank$) $\geq W(c)$ and $W(b) \geq W(c)$ then
7:             $n \leftarrow c$                                        ▷ False alarm: same classifier and topology
8:             $t \leftarrow T_C$
9:         else if minError($rank$) $\leq W(b)$ then
10:            $n \leftarrow$ copyClassifier(CH, minErrorKey(rank))    ▷ Recurring drift: classifier and topology from G
11:            $t \leftarrow T_G$
12:        else
13:            $n \leftarrow b$                                        ▷ Bkg drift: drift to bkg classifier new topology
14:            $t \leftarrow 0$
15:        end if
16:    else                                        ▷ Select only between background and active (no drift)
17:        if $W(b) \geq W(c)$ then
18:            $n \leftarrow c$                                        ▷ False alarm: same classifier and topology
19:            $t \leftarrow T_C$
20:        else
21:            $n \leftarrow b$                                        ▷ Bkg drift: drift to bkg classifier new topology
22:            $t \leftarrow 0$
23:        end if
24:    end if
25:    return n, t                                 ▷ Return next classifier and the relevant topology
26: end function
```

In GroCH, both the active and background classifier are trained with new examples as soon as they are available (lines 43 and 34 in Algorithm 5.1). However, idle classifiers in the concept history are not retrained unless selected to become the active classifier. In the case of drift, the active classifier is replaced by either the best idle classifier or the background classifier (line 28 in Algorithm 5.1) following Algorithm 5.3. If the background classifier is selected, the training examples from the warning window would already have been used for its training. Conversely, if an idle classifier was selected, these training examples won't be fed to any learner.

In the scenario of a false alarm, the drift signal resets but the warning window continues. See Algorithm 5.3 for reference. In the case of false alarm, classifier c (the active one) and T_C (current topology) is selected. Otherwise, either the background classifier and an empty topology or the best idle classifier from the closest group and the topology of the same group are selected (see Algorithm 5.3).

GroCH considers the number of data instances received during the warning window as part of the drift decision mechanism. Many explicit drift detectors from the data stream mining literature (e.g. RDDM [38]) take this into account. Thus, this consideration could be considered excessive when using such detectors. However, this chapter proposes GroCH as a meta-learner agnostic to the drift detector used. Hence, it explicitly deals with the length of non-stable periods to improve the results when using detectors that do not consider it. GroCH allows changing these thresholds for scenarios not needing this feature.

5.1.3.5 Pre-training of the Concept History

The concept history of GroCH can be prepopulated with a set of group and classifiers. This prepopulation is made before starting the prequential evaluation of the algorithm. This, which can also be defined as a pre-training step, uses each dataset from many to populate one group from the CH with one classifier and a topology.

This pre-training step is an opportunity in many domains to prepopulate known and potential states of the time series that may reoccur. For instance, in the application area of this thesis (the financial domain), these steps can help investors with a pre-conception of what a market state is in their financial series. In this scenario, a predefined market state may not be a ground truth, albeit it can be used as a *gold standard*. In any case, we are aware of the challenge defining market states, and their definition has not been resolved or agreed upon in the financial literature.

5.1.4 Differences with the Literature

Table 5.1 covers the comparison of GroCH against against other relevant methods from the literature. Most drift detectors with state-of-the-art results [39, 160] have intervals to handle both warnings and drifts. Other algorithms from the literature like adaptive random forest [152] make use of two separate drift detectors for warning and drifts obtaining state-of-the-art results in specific types of drifts like ADWIN2 [46]. In order to make use of ADWIN2 as a drift detector in GroCH, we developed a drift detector called ADWIN-based drift detection method (ADDM). This contains two adaptive windowed confidence intervals (ADWIN2 drift detectors) to handle warnings and drift detection respectively. The code for this can be accessed in: https://github.com/cetrulin/groch-moa.

Another difference between GroCH and adaptive random forest is the error passed for the supervised drift detection. ARF uses the error after training with that instance. While the objective of this is not clear in their proposal, we assume that this could work as an approach to flag if a weak learner from ARF does not adapt appropriately to the current concept. Conversely, and according to other relevant ensembles like RCD, GroCH feeds the prediction error of the active classifier to the detector.

GroCH has a concept history with previous models to be reused in the future. RCARF, RCD, CPF and ECPF also keep historical classifiers, although the naming of this may vary. RCD, CPF, ECPF and GroCH are meta-learners that use any base classifier in their MOA implementations. They also support any drift detector as long this follows the convention created with DDM and has a warning zone.

	*CPF	RCD	ARF	RCARF	DWM	GroCH
DD uses Prediction Error	✓	✓			(no DD)	✓
DD reset after drift			✓	✓	(no DD)	✓
Trains Active during WW		✓	✓	✓	(no WW)	✓
Collection of Concepts (CH)	✓	✓		✓		✓
Conceptual Equivalence	✓			✓		✓
Ensemble			✓	✓	✓	Bkg votes during WW
Can use any base classifier	✓	✓			✓	✓

Table 5.1: *Comparison of GroCH to the relevant SOTA. Abbreviations - DD: drift detector; WW: warning window; Bkg: Background Learner; *CPF used as a cross-reference to both CPF and ECPF.*

While and RCD, ARF, RCARF, and DWM are ensemble learners with many active classifiers testing and training at a time, CPF and ECPF only have an active learner in their pool. GroCH only has a single active learner, but during the warning window (equivalent to warning zone in CPF and ECPF) the background learner is also considered for predictions (weighted).

CPF, ECPF and RCARF use conceptual equivalence; hence, they consider that a historical learner represents the current concept if it classifies the current data distribution with lower error than the active classifier. CPF and ECPF store only dissimilar classifiers in their story. Using lower levels of this threshold (m) makes them have a lower number of more generic classifiers. Conversely, a greater level makes them store a more significant number of more specific classifiers. A similar effect can also occur in GroCH balancing the concept similarity distance metric. A greater or smaller distance threshold leads to a smaller number of more general or greater number of more specific groups in the concept history respectively.

Finally, CPF, ECPF and RCD do not reset drift detectors after a drift at the implementation level. Hence, allowing warning detection straightway if the error obtained by the base learners is still unstable. Conversely, ARF, RCARF and GroCH reset the detectors. As an addition to ARF and RCARF, GroCH includes the possibility of false alarms if the active classifier is the best performer during the warning window. GroCH also has a minimum and maximum length for the warning zone that can help in domains where changes are expected to have a certain duration or sharpness. Warnings windows below and over those thresholds do not consider the history or are ignored respectively.

CPF and ECPF do not train their base learners during the warning window; ARF, RCARF and GroCH do this but save a snapshot of the classifier at the time of the warning signal to be inserted in the history in case of concept drift. If there is a drift confirmation, the snapshot saved will not include the instances seen during the warning, but the active classifiers are up to date to predict for the time being. Also, in GroCH, in cases where the base learner does not adapt well to the new concept, the background learner can be used to predict during the warning zone. DWM does not have a drift detection mechanism.

5.2 Framework to Characterise the Algorithm

In this section, we define a framework to evaluate the proposed algorithm in classification tasks and also to recognise concept drifts accurately. Another goal of this section is to evaluate if the states recognised during retrievals and insertions to the concept history are successful. In order to measure this effectively, we need to know the ground truth (GT). This is, what concept changes are there in reality and when are these produced. For this purpose, we have generated a set of synthetic time series that simulate structural breaks in financial datasets. These datasets are then pre-processed and fed to our algorithm as a data stream for price trend classification. The results are then compared to the ground truth to evaluate the performance of our contribution.

This section is structured as follows: first, we present the evaluation metrics used; then, we describe the series created, the data pre-processing and the resulting set. Finally, we summarise the classification problem, the experimental design to compare against the ground truth, the parameter search performed, and we analyse the results.

5.2.1 Evaluation Metrics

This section describes briefly the different procedures and metrics used for evaluation tasks.

5.2.1.1 Classification Error

In this chapter, the classification performance of all algorithms is evaluated following a prequential scheme. As described in Chapter 2, this scheme is aimed at scenarios where the data arrives continuously. Each sample servers two different purposes (testing and training), and these are analysed in sequential order of arrival.

The prequential error is computed by predicting with each individual instance just before training the model with it. We report the mean classification error computed over sliding windows of 1,000 instances (default value in MOA , the framework used).

5.2.1.2 Classification Performance under Switch

We use mainly two metrics to evaluate the algorithms during concept drifts.

First, we define the metric *accuracy under switch (AUS)* as the overall predicted accuracy on data instances that occur during a switch (transition between states) in the ground truth.

Then, as there might be scenarios where drifts occur to new (background) classifiers that may not produce robust predictions due to its short training window, we also report *kappa statistics under switch (KUS)*. Kappa statistic aims to subtract the bias in the actual distribution and thus is a suitable metric to evaluate the strength of the algorithms.

The reader must note that AUS and KUS are calculated as *accuracy* and *kappa statistics*, previously defined in Subsection 2.2.5. The only difference in computing such metrics is that the evaluation is only performed over the instances received during a switch in the ground truth.

Likewise, in Appendix A we also provide similar evaluation measures like *kappa temporal statistics* under switch for the results of this chapter.

5.2.1.3 Metrics for Concept Drifts and Recurrences

GroCH performs three main actions in the presence of a variation between concepts: i) detection of those changes (drifts), ii) retrieval of group from the CH representing the new state, and iii) insertion into a group of the CH.

In this subsection, we define different metrics to evaluate these three actions during variations: v1_{acc}, v2_{acc} and v3_{acc} respectively. These metrics are mainly reported as the percentage of accumulated actions performed correctly at the end of the execution of the experiments. In order to understand these performance metrics, the reader must consider the following two points:

- *A drift can occur at any data instance of the data stream.* Hence, the number of true negatives (TN) detecting drifts is a large value close to the number of examples of the data streams used in the experiments.

- *GroCH uses supervised drift detection to address changes in the data stream.* Drifts may be detected outside a ground truth change, and this is not necessarily an incorrect behaviour. There may be other changes in the data stream dynamics affecting the active learner, which may need to be replaced. We penalise concept drifts detected at the wrong time, but not insertions or retrievals. This is explained further in the following subsections.

Due to these points, and for the sake of simplicity, we mainly report percentages of drifts, retrievals and insertions that are correctly classified, provided that there are no drifts, retrievals or insertions detected in all instances.

Still, there are scenarios where depending on the sensitivity of the parameters and thresholds selected, there can be a many to one mapping between concept drift and ground truth switches (see Figure 5.7). For this purpose, in the experimental section, we also report true positives from the ground truth perspective (TP$_{GT}$). Metrics with base D express concept drift, and metrics with base GT express number of ground truth events. True and false positives, and true and false negatives for each of the metrics (v1, v2 and v3) are reported in the Appendix A.3.

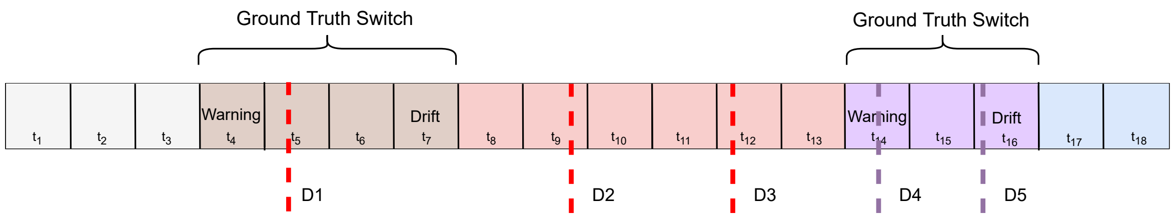


Figure 5.7: *Many to one mapping between concept drifts (vertical lines) and ground truth switches.*

We leave warnings outside of our analysis since they do not perform changes on the active learner. False alarms are considered for drifts (v1) and retrievals (v2), but these do not insert groups. Early drifts are still considered as drifts for v1 and v3, but these do not retrieve groups. Therefore, early drifts and false alarms are not considered for v2 and v3, respectively.

5.2.1.3.1 Evaluation of Drift Detection (v1) This metric rates the performance of all drifts detected by GroCH. The meaning of true and false positives, and true and false negatives in the context of this metric is:

- True positives (TP_D): drift detected when a change occurs. For this we allow a delay of d instances.
- True positives from the ground truth (TP_{GT}): switches from the ground truth detected as a drift.
- True negatives (TN_D): drifts not detected when there were no changes in the ground truth. This is expected to be close to the number of instances received and is omitted from our analysis.
- False positives (FP_D): number of drifts detected by GroCH when the ground truth does not change.
- False negatives (FN_{GT}): it does not detect a drift, but there is a change in the ground truth.

We define drifts accuracy percentage (v1_{acc}) as the rate of concept drifts detected that occur during a transition between ground truth states.

$$v1_{acc} = \frac{\text{Number of Drifts Correctly Identified}}{\text{Number of Total Drifts}} \tag{5-2}$$

where *Number of Drifts Correctly Identified* represents all drifts detected by GroCH inside the drift detection delay threshold (d) introduced in the following subsection. *Number of Total Drifts* represents all drifts signalled by GroCH.

5.2.1.3.2 Speed of Detection (v1.2) This metric reports the delay switching to a new classifier compared to the actual change in the ground truth. It only considers drifts recognised accurately (TP_D).

$$\frac{1}{n}\sum_{i=1}^{n}(t_{D_i} - t_{GT_i}) \tag{5-3}$$

where $n = |TP_D|$, D_i represents every true positive drift recognised by GroCH and GT_i represents the previous ground truth change meeting the equation:

$$d \ \geq (t_{D_i} - t_{GT_i}) \text{ for } i \ \epsilon[1, n] \tag{5-4}$$

where d represents the drift detection delay threshold and is defined in Subsection 5.2.1.4.

5.2.1.3.3 Ratio of States Recognised (v1.3) This metric rates how many states from the ground truth are covered as a group in the concept history. It also covers how many groups has the algorithm created per ground truth state. This metric is expected to have a result equal to 1 if concept history groups are prepopulated. Since this is the case in the experiments performed in the final benchmark of this chapter, we do not report this metric.

$$\text{States Ratio} = \frac{\text{Number of Ground Truth States Recognised}}{\text{Total Number of Ground Truth States}} \tag{5-5}$$

where a given ground truth state is recognised if there is any insertion to the concept history in a period that is in, or transitions from that ground truth state. Insertions are covered in Subsection 5.2.1.3.5.

5.2.1.3.4 Evaluation of Retrievals (v2) This evaluates the performance of all retrievals produced in GroCH that select groups representing the state of the ground truth at that point in time (allowing a delay of d instances). The meaning of true and false positives, and true and false negatives in the context of this metric is:

- TP_D: Once the state is represented in the CH (known), this counts how many times does GroCH retrieve a classifier from a group representing the GT state at that point in time (being considered a correct retrieval).
- TP_{GT}: switches from the ground truth covered by a correct retrieval.
- FP_D: the state has never been seen before (not known), but the algorithm retrieves a group from the CH instead of selecting the background classifier. This is expected to be equal to zero if the concept history has been prepopulated with groups representing all ground truth states.
- FN_D: $\text{FN}_1 + \text{FN}_2$
  - FN_1: the state exists (known), but the algorithm selects the background classifier instead of retrieving a group from the CH.
  - FN_2: the state exists (known), and the algorithm selects a group from the CH, but this is the wrong one.
- TN_D: the state has never been seen before (not known), and the algorithm consequently selects the background classifier.

We define the retrievals accuracy percentage ($\text{v}2_{acc}$) as the rate of retrievals to a group representing the right state. Hence, retrieving a group matching the current ground truth state.

$$v2_{acc} = \frac{\text{Number of Retrievals of the Right State}}{\text{Total Number of Retrievals}} \tag{5-6}$$

where *Number of Retrievals of the Right State* refers to all retrievals of groups mapping to their respective states, or transitioning states of the ground truth. This mapping is described at the end of the following subsection.

5.2.1.3.5 Evaluation of Insertions (v3) This metric reports the performance of all insertions performed by GroCH that select groups representing the state of the ground truth at that point in time (allowing a delay of d instances).

As defined earlier in this chapter, a new group should be created in the CH only when there is a new state. GroCH does this when the distance of the instances received between the previous drift and the last warning signal to all the CH group topologies surpasses the topology radius (threshold). Insertions are made only in cases of background or recurring drifts (not in false alarms). For this reason, this evaluation measure (v3) can help report cases where information belonging to a new state is added to a pre-existing group (mistakenly) or when a classifier is inserted into a group representing a state that mismatches the GT.

The meaning of true and false positives, and true and false negatives in the context of this metric is:

- TP_D: the state exists (known) or is created accordingly to the ground truth. Thus, the new information is added on the top of the existing or new group.
- TP_{GT}: switches from the ground truth covered by an accurate insertion.
- FP_D: the state has never been seen before (not known), but the algorithm selects a group from the CH, and the information is added there. This is expected to be equal to if the concept history has been prepopulated with groups representing all ground truth states.
- FN_D: $\text{FN}_1 + \text{FN}_2$
  - FN1: the state exists (known). However, the algorithm creates a new group in the CH.
  - FN2: the state exists (known). However, the algorithm adds the information to the wrong group.
- TN_D: the state has never been seen before (not known). Thus, the algorithm creates a new state in the CH. This is expected to be equal to if the concept history has been prepopulated with groups representing all ground truth states.

We define the insertions accuracy percentage (v3$_{acc}$) as the rate of insertions to a group representing the right state. Hence, inserting into a group matching the ground truth state prior to concept drift.

$$v3_{acc} = \frac{\text{Number of Insertions into the Right State}}{\text{Total Number of Insertions}} \tag{5-7}$$

where *Number of Insertions into the Right State* refers to all insertions into groups mapping to their respective states, or transitioned states of the GT. The mapping of ground truth states and groups is defined by the data instance when the first insertion into new groups in the concept history is produced. A group maps to the ground truth state exhibited by (or transitioned from) the data instance when the concept drift of its creation was signalled.

5.2.1.4 Speed Adapting to Changes

Due to the nature of the drift detectors used, drifts are recognised once the distribution is drifting or has drifted completely. There is an intrinsic delay from the ground truth in this detection. However, depending on the abruptcy and the difference between the initial and final concepts of the switch, this can vary. The average delay of the accurate drifts is one of the metrics reported (v1.2).

Furthermore, in our analysis, to identify a drift, retrieval or insertion as part of the ground truth, we allow a delay of d data instances, d a parameter known as the drift detection delay threshold detecting concept drifts. d affects many of the metrics reported, as seen in Subsection 5.2.1.3.2 and Subsection 5.2.1.3.1. The decision mechanism to select this is described further in the next section, along with other parameters optimised.

5.2.1.5 Memory Usage and Model Complexity

Since data streams may arrive continuously, a metric to consider in order to achieve an scalable model is the cost of the model. We use RAM-hours as our main metric to benchmark the cost of the models. Prequential running time is also reported in the Appendix A.3.

5.2.2 Research Data

In this chapter, we have used synthetic datasets to validate the behaviour of GroCH in data where the GT is known, but that still resembles a generative processes from real-world data. In order to evaluate concept drifts accordingly, to compare change detectors the ground truth changes must be known [16]. This section describes the generation of the synthetic datasets used to compare events such as retrievals, insertions and drifts to the GT. For this purpose, we define a framework to generate synthetic time series with similar properties to stock market prices.

Our proposed framework fits several auto-regressive (ARMA-GARCH) models to different stock market returns. Each model is fitted to a distinct EFT type exhibiting different behaviour. The purpose of this is to fit models to different market states; e.g. bear or bull markets, lateral movements, and periods of low vs high volatility, to simulate switches between these.

Our hypothesis is that the proposed algorithm should be able to detect these transitions as drifts or variations between market states. To the best of our knowledge, this is, although not perfect, one of the cleanest ways to model the current problem. These synthetic datasets created include recurrent changes of different sharpness. Using the work from Shaker and Hüllermeier [312] as an inspiration to model drifts, we have designed these as sigmoidal transitions of 100 and 1,000 data instances between the different states. Each state is generated through a generative process, and the simulation produces switches between them in parallel to the generation of new data points.

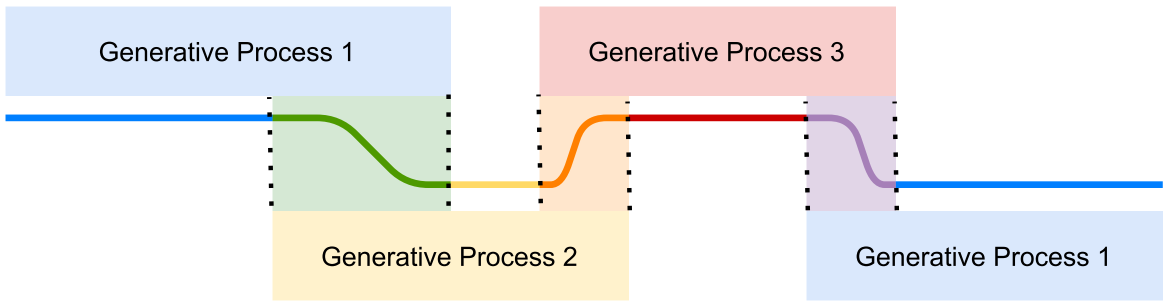


Figure 5.8: *Concept drifts illustrated as sigmoidal transitions between generative processes (green, orange and purple sections). During changes, the output simulated at each timestamp is the weighted mean of the previous and the next generative process. Both processes receive up to the last step time of the output stream (mid-line) to simulate each data instance during the transition.*

Before fitting the models, the best values for the input lags for the auto-regressive models are selected in the range of $[1 - 15]$. Once the models are fitted, the specification of each model is saved. Then, the series is generated using the prediction of the model representing a current state, or as a weighted mean of the prediction of the models representing the old and new concepts.

The simulation process, illustrated in Figure 5.8, is as follows:

1. The creation of the synthetic time series starts with the prediction of the first model. Models will only simulate using their own input data only at the start of the simulation, before the length of the synthetic series is equal or greater than the minimum input lag of the current model; thus, only if there is not enough data in the stream yet to be fed into a time series model specification.

2. From this point, the process switches to different models, with transitions of different duration, as mentioned on the previous page.

   - During a switch between generative processes, weights between the old and new models will change progressively, following a sigmoid curve for the transition. The value of the return predicted during a switch results from a weighted mean (the weight of the old model reduces progressively. The weight of the new $model = 1 - weight_{old}$). In [312], Shaker et al. proposed a similar transition from one concept to another in the domain of non-stationary data streams.
   - During the simulation process that uses ARMA-GARCH models, in a given time point, the model or models that belong to the current state, or the two current states (in case of concept drift) respectively, are used to generate the following time steps of market price returns.

3. Prices are finally reconstructed from the returns, and a set of technical indicators from the relevant literature is then computed.

The sequence of changes is created by a transition map that is summarised in the Appendix B.2. Each of the drift types (gradual or abrupt, depending or their sharpness) is generated with all of the four concepts illustrated in Figure 5.10.

The order and variation between generative processes and the duration of the changes were chosen iteratively by studying the behaviour of the base classifiers that we use later in this chapter to the synthetic sets produced. The length of stable periods between drifts has been set to a value that allows base classifiers to learn from the data stream. At the end of the execution, the generated time series of returns is reconstructed to prices using the last close price of the dataset used to fit the first model. As will be explained in the following subsection, the simulated series is used to compute technical indicators. Then, the resulting dataset is streamed using MOA to measure the classification performance during different types of concept drift (abrupt or gradual). This process is repeated for each data stream produced for our experiments. The end-to-end flow from fitting the four models to producing final data with indicators can be seen in Figure 5.9.

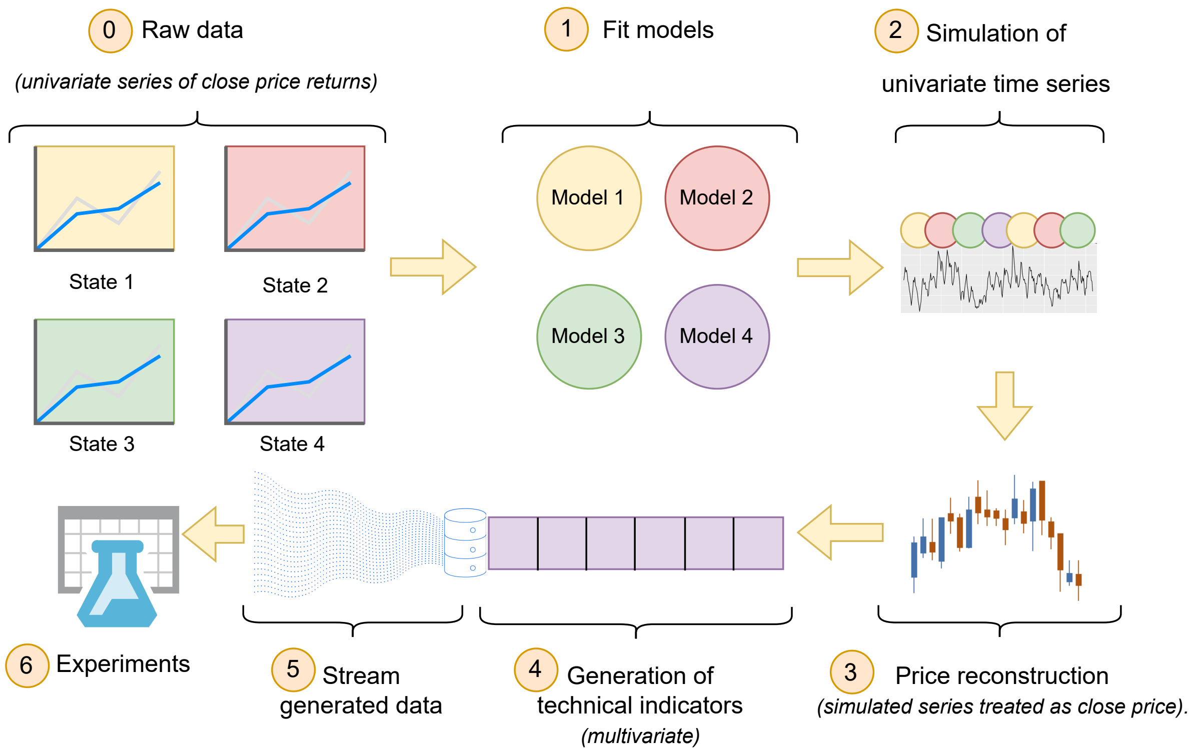

0 Raw data
(univariate series of close price returns)
State 1
State 2
State 3
State 4
1 Fit models
Model 1
Model 2
Model 3
Model 4
2 Simulation of univariate time series
3 Price reconstruction
(simulated series treated as close price).
4 Generation of technical indicators
(multivariate)
5 Stream generated data
6 Experiments

Figure 5.9: *End-to-end process from raw data to simulation and experiments.*

The entire simulation and data preparation process (steps 0 to 4 in Figure 5.9) is developed in Python 3.7. The software used to fit the ARMA-GARCH models is the *rugarch* R library, called from Python using the wrapping library *RPY*. The code of the simulation process is open source and can be found in the following URL: https://github.com/cetrulin/regime-switching-series-generator.

5.2.2.1 Assumptions and Concept Representation

In this chapter, we use research data that is based on financial price returns. Our aim for every concept in the synthetic set is to represent a financial market state.

The experiments in this section rely on the assumption that financial intraday market states can be modelled by fitting ARMA-GARCH models over the close price returns series at different frequencies. In this process, we assume that a market switching process should also be reflected as a change in the correlation between different technical indicators.

From now on, in this chapter, we talk about shifts of market states and concept drifts interchangeably. Concept drifts are changes in the generative process of the time series of price returns that also impacts the indicators generated as attributes and their mapping to ups and downs (target feature) in the datasets used for the experiments.

5.2.2.2 Generation of Semi-synthetic Series with Structural Breaks

As explained at the start of Subsection 5.2.2, we propose a simulation process that fits ARMA-GARCH models for each market state to be modelled in the generated series. One model is fitted to each raw of the series of price returns where the market behaves differently. Then we use these models to generate a series of prices.

We propose a mechanism to switch between models during the generation of the returns series. The objective of this is to simulate structural breaks and transitions between different market states.

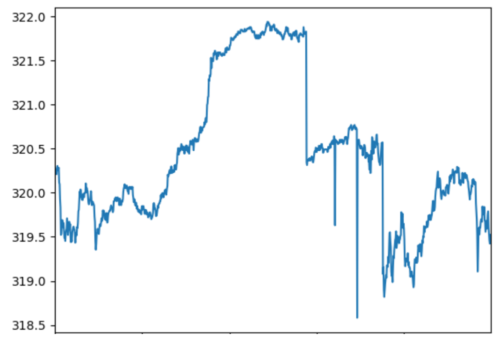
The structure of this subsection is as follows. First, we present the datasets used as a base of different market states (different *contexts*). Then we cover challenges associated with the spatial representation and separability between the states. Finally, we describe the simulation process in more depth and the variations chosen between concepts.

5.2.2.2.1 Ground Truth Market States We selected the first 1,000 data points (close prices) at the 5-minutes level resolution from January 2020 for a representative ETF from each of the following types: i) *equities*, ii) *securities (fixed-income preferred)*, iii) *real-state* and iv) *international bonds*. Each of the ETFs price return series was used to represent a ground truth raw state for generating the synthetic sets. The ground truth states fed to the generator are the next ones:

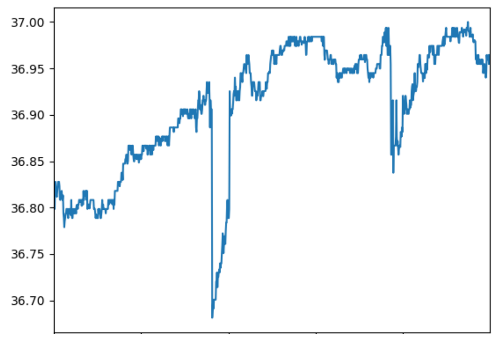
1. **State 1**: Equities - *SPDR S&P 500* ETF (SPY). Lateral movement. Price uptrend followed by the start of a downtrend. See the close prices for that period in Figure 5.10a.
2. **State 2**: *Securities/Fixed-income preferred - iShares Preferred and Income Securities* ETF (PFF). Uptrend. See the close prices for that period in Figure 5.10b.
3. **State 3**: *Real-estate - Vanguard Real Estate* ETF (VNQ). Volatile lateral movement. See the close prices for that period in Figure 5.10c.
4. **State 4**: *International bonds - SPDR Barclays International Treasury Bond* ETF (BWX). Lateral movement with a slow uptrend. See the close prices for that period in Figure 5.10d.

5.2.2.2.2 Challenges in the Generation Our goal with each of the models from the previous paragraphs is to model the generative process that defines the price returns in a market state. However, even if we had fitted models, there would be different challenges to address. The first challenge is the fact that models are fitted for a specific sample that will very likely differ from the inputs passed during the generative process, as the output of model “A” will be passed as an input series into model “B” from a concept drift as seen in Figure 5.8. “B” or any of the other models have not been trained for some of the trends, volatility or sharpness of movements exhibited by any of the other models. Thus, the generated stream that is passed as an input at every time step impacts the model, and the new outputs can exhibit behaviours never seen in the raw samples used to fit such models.

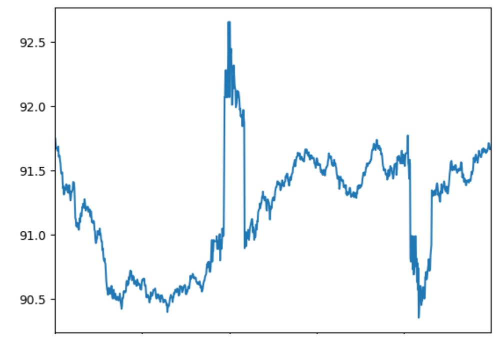
One behaviour that we had to bring down was the explosion (up and down) in values of the time series once the prices were reconstructed. Part of this was due to the size of the sets generated (1.5 million time points). We would have to handle either very low prices or values exhibited in financial time series as the effect of inflation or deflation over dozens or hundreds of years in certain symbols. This could also break the correlations between the technical indicators. Hence, having an impact on GroCH at the experimentation level once generated the test and train sets. To mitigate this, the selection of ETF was for series with mean price return close to zero in the first trading week of 2020. We looked for slightly negative or positive mean returns, thus softly down or up in prices, with more or less volatility, so these likely lateral trends define different states.



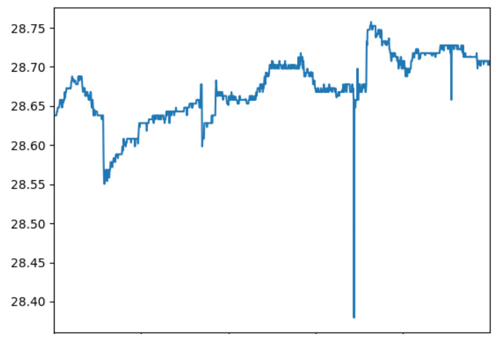
(a) *S1: Equities (SPY)*



(b) *S2 - Fixed-income preferred (PFF)*



(c) *S3 - Real-state (VNQ)*



(d) *S4 - International bonds (BWX)*

Figure 5.10: *Close price from time series used to fit the four ARMA-GARCH models.*

5.2.2.2.3 Fitting Process Each dataset described in the previous section has been converted to returns and used to fit a different model. For each model, the fitting process has been performed using the best autoregressive (p) and moving average (q) orders for both the ARMA and GARCH components and both lagged values for the GARCH component. The resulting ARMA-GARCH models have been saved from being used during the simulation process.

5.2.2.2.4 Model Optimisation The best autoregressive (p) and moving average (q) were chosen by selecting the orders with the lowest Akaike information criterion (AIC).

$$AIC = 2k - 2ln(L) \tag{5-8}$$

In equation 5-8, k denotes the number of parameters and L denotes the maximised value of the likelihood function. For model comparison, the model with the lowest AIC score is preferred [74]. The best values found through a grid search for p and q for ARMA and GARCH per ETFs are listed in Figure 5.2. This grid search covered the range of 1-25 for ARMA p and q, and 1-5 for GARCH p and q.

ETF	p_{ARMA}	q_{ARMA}	p_{GARCH}	q_{GARCH}
SPY	9	25	4	4
PFF	12	1	1	5
VNQ	9	5	1	4
BWX	21	2	1	1

Table 5.2: *Best orders obtained for MA and lags of the four ARMA-GARCH models fitted.*

5.2.2.2.5 Simulation Process The synthetic dataset was generated through a simulation of the models pre-trained during the fitting process explained in the previous subsection. This process uses a transition set that provides information about:

- *Row (time step)*: when does each switch occur.
- *Length (duration)*: how long is every switch. For this, we have used two different configurations: abrupt and gradual switches of 10 and 100-time steps, respectively.
- *New model (transition)*: what is the next model at each switch.

The forecast horizon in the simulation process is only of 1-time step during the switching process. This is explained further in the next subsection. Once the switching is performed, the forecast horizon is up to the next transition in the transition map.

5.2.2.2.6 Transition Map Proposed For the experiments of this chapter, we have simulated data streams with changes in their generative processes (concept drifts) every 5,000 data instances. We have defined both gradual and abrupt shifts, randomly allocated to each transition, but with an even proportion.

There is a total of four generative processes that are shared across streams (fitted as seen in Table 5.2). Each stream has a total of 1.5 million data instances. Hence, 75 switches per ground truth state; a total of 300. Only 1 million instances have been used for machine learning tasks. 500,000 instances have been used to test and train the models (100 concept drifts). These transitions are randomised in the transition map, and different combinations of shifts between states are evenly distributed. The transition map shared across data streams as well, can be seen in Appendix B.

5.2.2.2.7 Switching Process During the transition between the two states, two different forecasts are taken into account. The one belonging to the old model and the one that belongs to the new one. Each time step during the switching process will be a weighted average between the forecasts of both models. The weights will update in each timestep, decrementing the weight of the old model and incrementing the weight of the new ones. Changes in weights follow a sigmoidal curve as suggested in the relevant literature for recovery analysis [312] introduced in Section 3.2.4.

Regarding the duration of the switching process, this has been programmed with two possible lengths: 100 time steps for an abrupt switch and 1,000 time steps for a gradual switch. Each variation from one market state to another is either gradual or abrupt.

5.2.2.3 Feature Set Selection

Once the time series of price returns are simulated, a price for those returns is reconstructed. This reconstruction is the cumulative sum of the simulated returns from the initial close price at the start of the simulation. This initial close price is the last price seen in state 1, at the end of the raw dataset from Figure 5.10. Then, we proceed with the generation of technical indicators. The generated time series is considered at the 1-minute level. Due to the fact that we only produce price returns and not volumes with the generator, we only use indicators based on OHLC prices.

Open, *min* and *max* prices for a time step are extrapolated from the current and previous close price reconstructed.

The feature set selected has a set of technical indicators common in the stock price trend classification literature [179]. These are 15 technical indicators that overlap across most papers for price trend prediction in the financial domain [3, 27, 115, 179, 183, 200, 208, 261, 275, 377]. The full list can be seen below:

- Commodity Channel Index (CCI).
- Larry William's R (WILLR).
- Momentum (MOM).
- Moving average convergence divergence (MACD).
- Relative Strength Index (RSI).
- Simple n-minute moving average (SMA) over 5 and 10 lags.
- Stochastic D% (SD).
- Stochastic K% (SK).
- Weighted n-minute moving average (WMA) over a 10 lag.
- Exponential n-minute moving average (EMA) over a 10 lag.
- Average directional n-minute moving average (ADW) over a 10 lag.
- Triangular n-minute moving average (TRIMA) over a 10 lag.
- Rate of change (ROC) over a 10 minutes lag.
- Bollinger upper and lower bands.
- Aroon Up/Down.

The selected set is partially based on a popular set first used by Kara et al. [195], extended with additional short moving averages of 5 minutes. The initial set of indicators, also used in Section 4.2, was shown in Table 3.2. Other sets were explored but not chosen as described in Appendix A.2.4.

The reader must bear in mind the nature of the tendency to use technical indicators in financial markets. Moving averages tend used in this context for the definition of trading rules and identify buy and sell signals. The addition of indicators that use moving averages of longer and shorter lengths may reveal changes in the markets. Momentum indicators help to measure the strength of a trend by looking at price differences on short time scales. Neely et al. [263] provide some examples of trading rules that can be produced with some of these indicators.

5.2.2.4 Resulting Synthetic Set

The 30 semi-synthetic series generated can be seen in Appendix B. All of them follow a similar transition map. Thus, GT switches occur at the same time points in all sets. After the computation of technical indicators, each data stream has ≈1 million data instances. The first 500k instances are left as a *development set*, for parameter optimisation and for the training of the Mahalanobis distance matrix. The rest of the instances are left for the test and train evaluation and, thus, for the analysis of results.

As a recap from this subsection, to create the final datasets, synthetic series created were first converted to a series of prices. Then, the series of prices were then transformed to *open-high-low-close* (OHLC) data. This was converted as follows:

- *Close Price:* generated values in Figures B.1, B.2, B.3 and B.4 at Appendix B;
- *Open Price:* close price of the previous time step;
- *High Price:* same as close price;
- *Low Price:* same as close price.

Then, the set of indicators explained in Section 5.2.2.3 are computed over the synthetic OHLC data. The label is created accordingly to section 5.2.2.3, for ups or stable/downs of the close price. The class distribution for the 500k examples analysed varies depending on the synthetic set but is always in the range of 45-55% for the label *0* (downtrend) and the residual part for label *1* (uptrend).

	Mean	Std.	Min.	25%	50%	75%	Max.
RSI_{10}	50.620	13.179	1.801	41.957	49.474	57.940	99.999
WILLR_{10}	-47.700	35.520	-100.000	-79.286	-48.068	-12.532	0.000
MACD	-0.000	0.070	-2.718	-0.016	-0.002	0.013	1.953
CCI_{10}	0.794	101.982	-333.333	-78.932	0.878	86.799	333.333
MOM_{10}	-0.001	0.172	-7.050	-0.034	-0.001	0.032	7.494
SK	53.453	29.211	0.000	29.119	53.172	78.550	100.000
SD	53.453	26.516	0.000	32.360	52.857	74.788	100.000
SMA_5	260.808	21.003	220.642	242.941	260.208	274.216	333.426
SMA_{10}	260.809	21.003	220.677	242.940	260.208	274.217	333.404
WMA_{10}	260.808	21.003	220.657	242.942	260.209	274.215	333.428
EMA_{10}	260.809	21.003	220.668	242.940	260.208	274.215	333.419
TRIMA_{10}	260.809	21.003	220.684	242.940	260.208	274.217	333.404
ADX_{10}	32.167	17.341	2.630	19.003	28.266	41.481	100.000
$\text{Bollinger}_{upperband}$	260.937	21.030	221.003	243.096	260.310	274.336	333.716
$\text{Bollinger}_{lowerband}$	260.680	20.978	220.295	242.811	260.113	274.083	333.117
ROC_{10}	-0.000	0.065	-2.601	-0.013	-0.000	0.012	2.855
Aroon_{DOWN}	51.430	37.307	0.000	10.000	50.000	90.000	100.000
Aroon_{UP}	53.729	37.684	0.000	20.000	50.000	90.000	100.000

Table 5.3: *Descriptive statistics of the first 1 million instances of the first synthetic stream created. Set of indicators covered in Subsection 5.2.2.3.*

Table 5.3 shows descriptive statistics about the feature set generated for analysis. There is a clear distinction between scaled values (*RSI, WILLR, CCI, SK, SD, ADX, ROX, AROON*) and non-scaled indicators (*MACD, MOM, SMA, WMA, EMA, TRIMA, Bollinger Bands*), which may exhibit explosive behaviours depending on the market states simulated.

5.2.3 Classification Problem

The experiments of this section perform binary price trend classification in a synthetic dataset that resembles the properties of different financial time series.

As described in Section 5.2.2.2.1, four datasets representing ETFs of different types have been used. The resulting dataset has been pre-processed to predict ups or stable/downs movements. The approach followed to parse the price to binary labels and the usage set of technical indicators as a feature set can be seen in different approaches from the literature [179, 195, 275]. This is further explained in Subsection 5.2.2.3.

5.2.4 Clustering Problem

The book written by Tsan and Chen [347] proposes a framework to characterise market states and tag directional changes (DC) based on differences in time scales and price returns. Their framework provides an original approach to visualising these states by creating two indicators based on the returns and the difference in time between price changes.

This thesis does not aim to define a model to represent market states. Rather than this, we only aim to approach changes between them as a concept drift. We believe that if a market state can be reflected in the liquidity, price returns and, or volatility of a stock symbol, the technical indicators from section 5.2.2.3 should also be able to reflect this. Thus, we have the indicators presented in [347] to represent the differences between the ETF data used to generate the semi-synthetic series: *SPY*, *PFF*, *VNQ* and *BWX*. The different plots in this subsection depict the challenge of differentiating between the different stock symbols fed for the generation of the synthetic series.

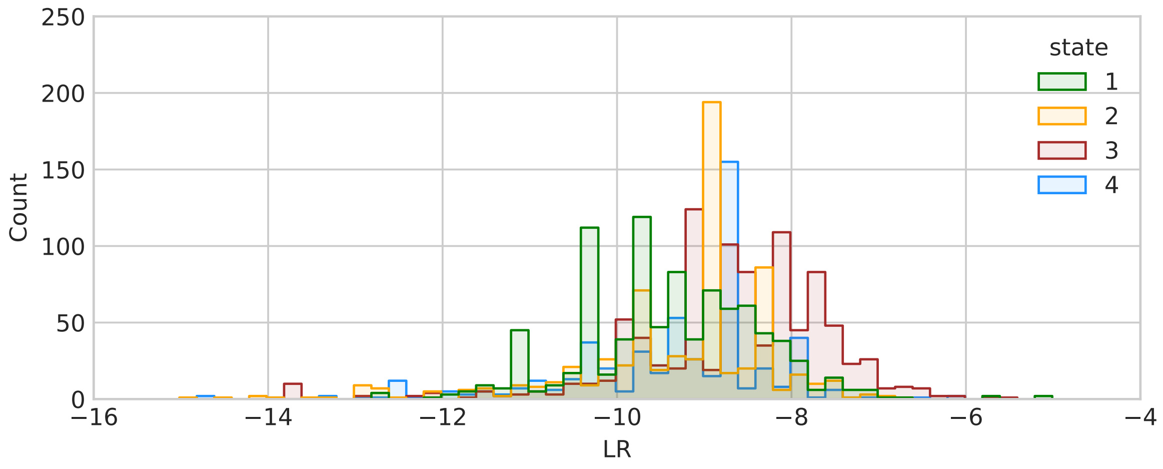
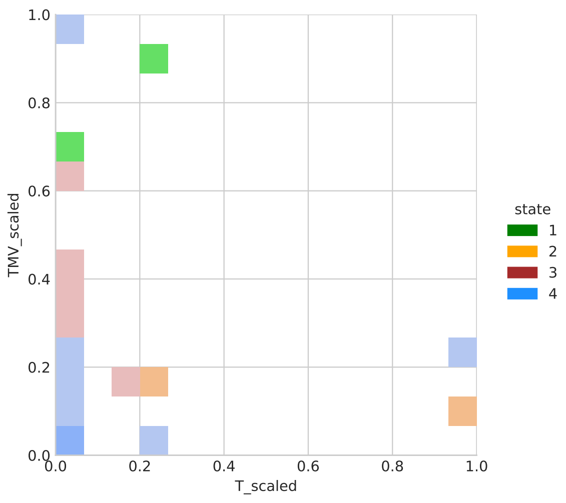


Figure 5.11: *Histogram of the logarithmic R indicator from [347] applied to the four states used in the datasets used for the generation of the synthetic sets.*

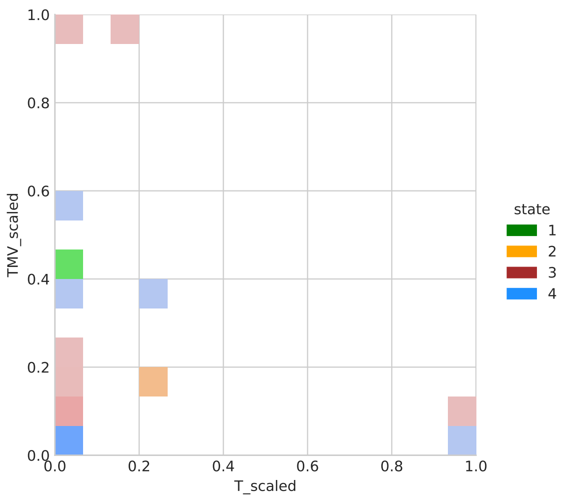
The histogram in Figure 5.11 shows how state 1 displays the lowest returns as opposed to state 3. States 2 and 4 are more centred in the histogram. It can be observed that state 4 is closer to a normal distribution, although state 2 is skewed towards three specific bins in the centre of the distribution. The set of indicators for this representation is:

- *Indicators to express price returns*: R (return), LR (logarithmic return) and TMV (total price movement).
- *Indicator to express time between each price return*: T (time).

Figures 5.12a and 5.12b show a bidimensional plot based on a price return and time components as showcased in [347] for our four states. Figure 5.12a shows that, for the raw sets, states 1, 2 and 3 do not present as many challenges to be separated from each other. However, state 4 can overlap with states 2 and 3.



(a) *Raw states*



(b) *Synthetic states*

Figure 5.12: *Representation based on the work from [347] of the fours states used for the generation of the synthetic sets used in these experiments. Raw and synthetic data in Figures 5.12a and 5.12b respectively.*

Figure 5.12b shows a similar bi-dimensional histogram over a sample of 1,000 points of those states generated synthetically. It can be seen how the separability of the different states in the vector space of the synthetic sets presents more challenges. Most states move in the histogram, and state 3 expands to the top of the histogram with greater price returns. These changes are due to the simulation process that, besides being trained for the states depicted in Figure 5.12a, it has a stochastic component. This random component will make different samples of the states evolve and change over time.

Bear in mind that the analysis of this subsection does not consider the fact that drifts and shifts producing variations between states could make the output data transform under the same generative process. Due to the graduality of transitions between states, the switching process will input data generated by the transitioning model into the transitioned model and vice versa. This fact, for instance, generated explosive price movements for different raw datasets during the selection of the four ETFs finally used.

In this chapter, we assume that a market state is represented by the same generative process (or concept [144]) and, in the case of state evolution, GroCH can create a new group for this.

This analysis shows that, even in a scenario where the data behaves identically than in the raw datasets used to train the generative processes in Figure 5.12a, there would be challenges to reach rates of retrievals and insertions greater to 75% in GroCH. Hence, we assume that one of the states may not be separable and not consider potential information loss from the subset of technical indicators used in the datasets used for testing and training.

Figure 5.13 shows how *k-means* for k=4 cannot recognise the four states from Figure 5.12b. This analysis proves the lack of signal to switch between market states accurately under all concept variations later in this chapter.

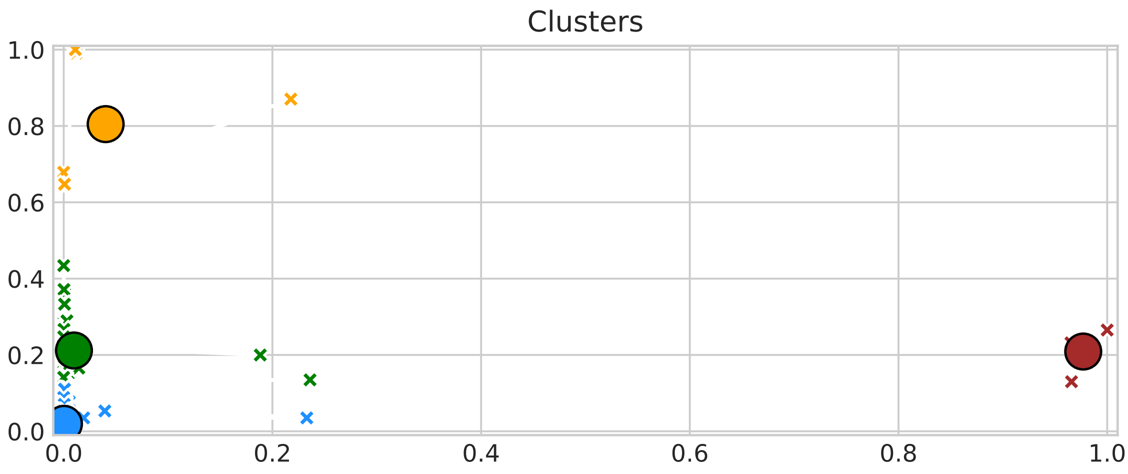


Figure 5.13: *K-means cluster centers for k=4 over the same data than in Figure 5.12b. Colouring of centroids does not correspond to ground truth states in Figure 5.12.*

5.3 Experimental Section

In this section, we present the experiments performed for a thorough evaluation of the algorithm. First, we describe the experimental plan, and then we briefly mention the parameter search performed to achieve the results that we show in the last subsection.

In order to avoid confusion and to have a clear terminology from here onwards in this section, a change in the ground truth is called a (regime) *switch*, while the events recognised by GroCH are called *drifts*.

5.3.1 Experimental Design

As explained throughout this chapter, we have designed a framework to measure the overall performance of our algorithm. Apart from classification accuracy during switches, we evaluate the identification of regime switches and the accuracy of concept history management operations such as insertions, retrievals and creation of groups.

GroCH and its main competitors in this section produce deterministic results. Thus, as explained in Section 5.2.2.4, to produce a benchmark with post-hoc tests, we simulated a set of 30 synthetic time series of 1 million data instances. Then we created a feature set of technical indicators with a binary target for price trend classification. The resulting datasets will be used in the experiments, and their results will populate the statistical tests. These synthetic sets will be streamed to all algorithms using *moa.stream.ArffFileStream*. Then, the obtained results and detected drifts will be compared to the actual regime changes designed in the simulation process (transition map) to validate, evaluate and benchmark our proposal. Part of this comparison to the ground truth involved the evaluation of the accuracies on the insertions ($v3_{acc}$) and retrievals ($v2_{acc}$) to or from (respectively) the right concept history group.

Concept history groups are aimed to represent ground truth states. However, GT states do not necessarily have a one-to-one mapping to the number of groups detected. In order to evaluate the performance of GroCH managing the CH, the equivalence or mapping between each group and each state in the transition map must be known. We have done this by creating market state and group identifiers. Even assuming a sequential assignment of identification numbers, identifier values may differ between market states and groups. Groups are identified depending on their order of creation, and any misdetection or failure detecting a drift may create a cascade effect in GroCH. In order to compute this equivalence, we map groups and classifiers depending on the position (instance) of the data stream when they were produced and then compared to the expected state at the GT (transition map described at Subsection 5.2.2.2.6).

In order to evaluate *classification accuracy* and *kappa statistics*, we have used the *window classification performance evaluator* from MOA with its default configuration, which is a window width of 1,000 data instances. The main metric used to report classification accuracy results is AUS, already described in the previous section. After this, we run post-hoc tests over the averaged results to check for statistical significance and conclude the experiments. First, we perform a normality test. If the distribution is normal, we use Welch's t-test to test for statistical significance. If the normality test is rejected, we apply the non-parametric Mann-Whitney Wilcoxon test.

5.3.2 Data Splitting

Before the parameter optimisation process, GroCH may need a previous sample of data to compute a Mahalanobis distance matrix of covariances (for concept or group similarity). Furthermore, GroCH has an optional training phase before its prequential evaluation that involves the prepopulation of the concept history. For this purpose, we allocate data instances from the data streams in the following way:

- *The initial 25% of the data stream* is used for the computation of covariances or any prepopulation (optional) at pre-training stage. To do this, the data instances used to prepopulate the CH are also used for the Mahalanobis distance matrix.
- *The next 25% of the stream* is used for the parameter tuning of all algorithms.
- *The other 50% of the data stream* is used to test and train models with their best performing parameters.

Regarding the first point above, the Mahalanobis distance matrix is created using the concatenation of the set of many datasets defined for pre-training and used to scale distances between groups. Each of these datasets is also used individually to prepopulate the concept history with one group per dataset, and one topology and one classifier per group. After this, all of these sets are concatenated to create the Mahalanobis distance matrix.

The splitting of the streams is done to support first the parameter tuning of all algorithms and then prequential evaluation. This is depicted by Figure 5.14.

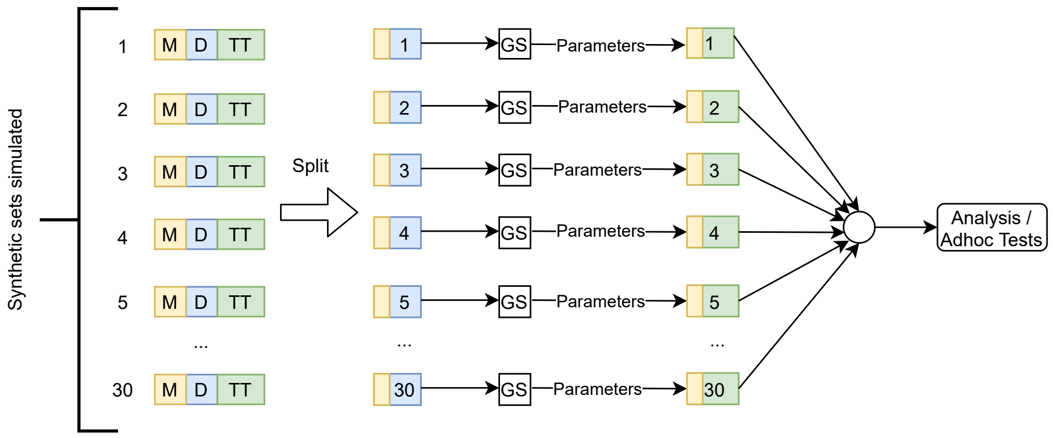


Figure 5.14: *Experimental design workflow covering from the simulation of the synthetic sets to the final results.*

5.3.3 Parameter Selection

GroCH, as the other algorithms evaluated, relies on a set of parameters that should be predefined. Due to the number of experiments needed and the large number of parameters in GroCH, the optimisation of these, is conducted in two stages:

- *Stage I* involves a larger number of parameters to be optimised with fine granular detail. Part of this phase is the selection of drift detectors and a common range of parameters for them.
- *Stage II* is focused on the exploration of a shorter range of parameters for the entire set of 30 data streams. This stage starts from the results of Stage I to narrow the search space.

This search is performed to maximise the mean accuracy during the test and training set. Albeit the metric used to evaluate the algorithm over the test and train set is mean AUS, we perform an exhaustive *grid search* using *mean accuracy* over the whole set to align with the optimisation performed for real-world datasets in the next chapter. The reader might want to consider other parameter optimisation techniques such as *Bayesian optimisation* or *hyperband* if the computational cost becomes an issue [219]. This is not the case during this thesis, as will be mentioned in Appendix C.2.

Coming back to Figure 5.14, since the optimisation process in Stage II is repeated for each of the 30 data streams, each algorithm may be trained for different parameters at each data stream in the final experiments. Stages I and II are described in Subsections 5.3.3.1 and 5.3.3.2 respectively. Findings during the pre-study performed as part of Stage I, that drive the parameter search performed for the final experiments (Stage II), will be shown in Subsection 5.3.4.

5.3.3.1 Stage I: Pre-study of Parameters for One Stream

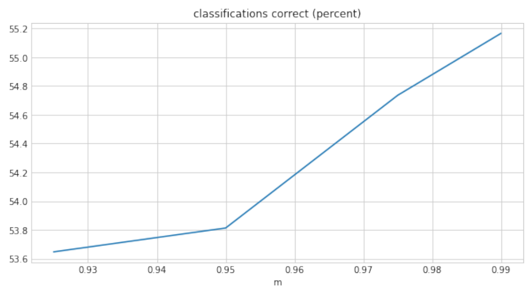
The first stage of the parameter exploration consists of a pre-study of the best performing ranges to understand the key parameters to be optimised. It involved ∼4,000 experiments per base classifier. In this subsection, we describe some of the parameters explored and analyse their impact on the first synthetic set. This data stream, with an overall binary class distribution of ∼55-45% for labels 0 and 1 respectively, is further described in the Appendix B.

Tables 5.4 and 5.5 show the parameter space explored in GroCH and its main competitors. While for GroCH a larger set of parameters need to be explored to understand the effects of their variation, for CPF and ECPF we have explored the author's recommended parameters and ranges [14, 16]. Table 5.4 shows the initial search performed in GroCH. Further details about the behaviour and description of each parameter are given in Appendix A.2.

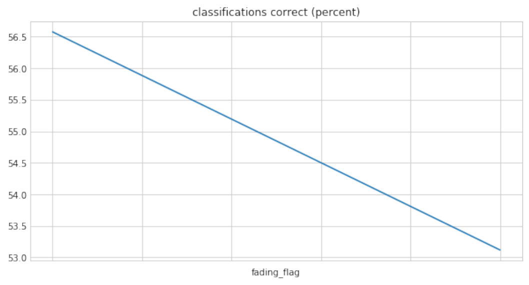
Parameter	Range explored in Stage I	Selection for Stage II
Drift Detector	DDM, EDDM, HDDM, RDDM, ADDM	EDDM and RDDM
Ω: Max. Classifiers per group	1, 3, 5, 7, 10 and 12	10
Topology radius (distance threshold)	0.5 to 3.5	1-2.5
Removal Policy from CH groups ($>\Omega$)	LUFO or FIFO	LUFO (remove less used)
Selection Policy for CE draw	LIFO and FIFO	FIFO (select oldest)
Punishing factor during WW (β)	0.5 (not optimised)	0.5
Concept similarity Metric	Euclidean, Mahalanobis	Mahalanobis
Topology Feature Subsets	6 or 9 indicators.	6
GNG lambda	5-200	5
GNG max ages	100-500	200
Min. instances seen for insertion	100-300	200
Min. instances in WW	10-100 (every 5)	15

Table 5.4: *Ranges explored during the parameter exploration in GroCH. The entire exploration process was performed for both HT and NB as the base classifier. WW: warning window; CE: conceptual equivalence.*

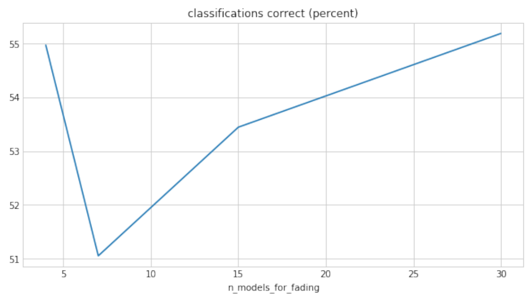
To illustrate the impact of using different values of some of the main parameters explored, we present plots over the simulated stream using HT as the base classifier in Figure 5.15. These plots are computed using the mean of all experiments with different experiments in this pre-study.



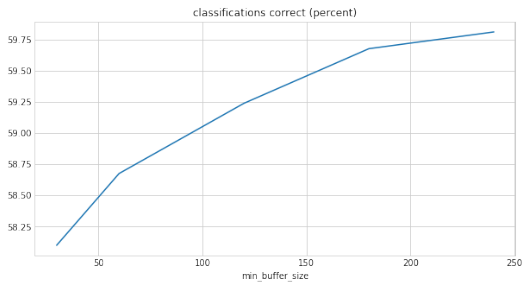
(a) ***ECPF*** *m (similarity threshold)*



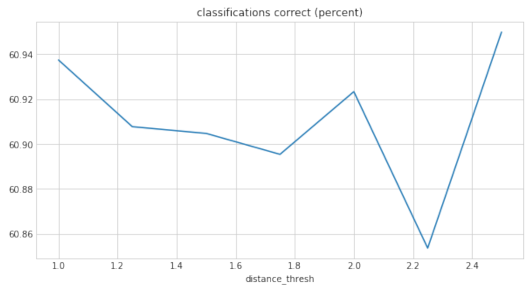
(b) ***ECPF*** *fading*



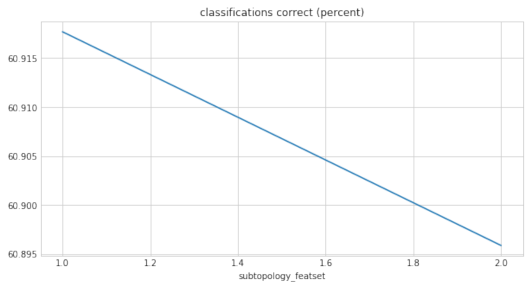
(c) ***ECPF*** *f (max concepts)*



(d) ***CPF*** *min buffer size*



(e) ***GroCH*** *topology radius*



(f) ***GroCH*** *subtopology*

Figure 5.15: *Example behaviour of parameters in ECPF and GroCH. ECPF parameters are also present in CPF behaving with the similar trends. Subtopology 1 represents the shorter feature set and 2 represents the larger.*

The impact of these parameters for naive Bayes follows similar trends. In the dataset used in this stage, the higher values of m in ECPF and CPF and greater buffer size in CPF performed better in terms of classification accuracy. In this regard, fading in ECPF and CPF did not perform well, probably due to the recurring behaviour of the stream. See the explanation of fading in Subsection 3.4.3.

Regarding GroCH's topologies, *2.5* was the radius obtaining the best result when prepopulating the concept history. Otherwise, a radius of *1* (using the shortest subtopology) and of *1.5* (using the largest subtopology) obtained the best results. The best subtopology was the one with the shortest set of features.

Two different feature subsets were chosen for the topologies as a result of the data exploration process. We selected technical indicators that, by design, are scaled. From the set of indicators used in our approach, this set is: *RSI, WILLR, CCI, SK, SD, ADX, ROC, Aroon*$_{DOWN}$ *and Aroon*$_{UP}$. In some synthetic sets with events such as prices skyrocketing or dropping over time, we observed that certain parameters could have values tending to the maximum or minimum of their scales (to 100 or 0). Thus, we also created a second set by discarding the RSI and STOCH (SK and SD) indicators. Therefore, the two subtopologies used in Stage I were:

- *Subtopology 1 (6 indicators)*: WILLR, CCI, ADX, ROC, Aroon$_{DOWN}$ and Aroon$_{UP}$.
- *Subtopology 2 (9 indicators)*: RSI, WILLR, CCI, SK, SD, ADX, ROC, Aroon$_{DOWN}$ and Aroon$_{UP}$.

The shortest subtopology (6 indicators) in GroCH performed generally better in Stage I in terms of classification, retrievals and insertions accuracy. Other parameters explored in GroCH impacting the topologies were the max. age of GNG (300) and GNG's λ (λ=5 for subtopology 1 and λ=15 for subtopology 2). The value of λ was selected by also evaluating the mean quantisation error of GNG across concepts in this stream. In Appendix A.2.2, we summarise different variants considered for the training scheme of the topologies. Multi-pass learning was also considered to reduce the quantisation errors but this was discarded as it increases the computational cost of training a topology in GroCH.

While the default value for λ in GNG is λ=200, the one-pass training forced us to decrease this value to a minimum to feed more data instances between concepts and reduce the quantisation error. GNG computational cost increases as its network of prototypes increases [131]. Hence, we were only able to go down to λ=5. Figure 5.16 illustrates the decrease of quantisation error with greater number of prototypes in GNG. This inspired the selection of lower values of λ to create more prototypes between drifts and achieve a better representation of the ground truth states.

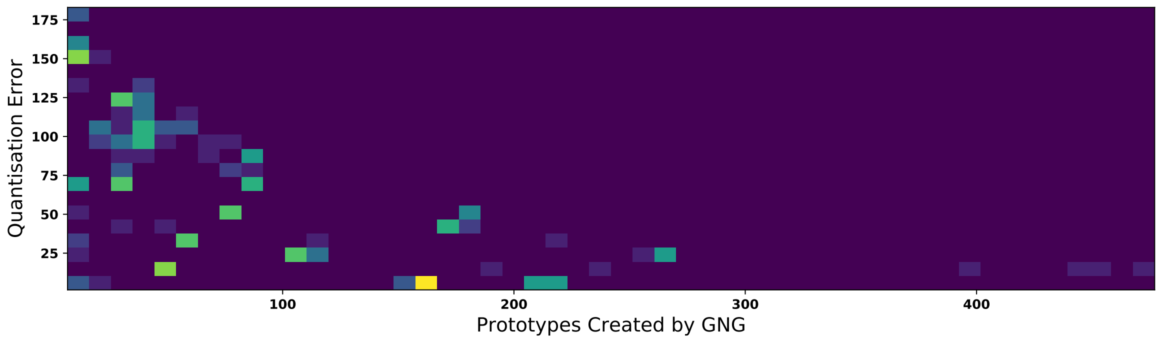


Figure 5.16: *Example of quantisation error in GNG per prototype created using λ = 100.*

Algo.	Parameter	Range explored in Stage I	Selection for Stage 2
CPF and ECPF	m	0.85, 0.9, 0.925, 0.95, 0.975, 0.99	[0.925, 0.99]
CPF and ECPF	fade Models	Yes/No	No
CPF and ECPF	Detectors	DDM, EDDM, HDDM, RDDM, ADDM	EDDM and RDDM
CPF and ECPF	f	1-15	4
CPF	Min. buffer size	30, 60, 120, 180, 240	[60, 180]

Table 5.5: *Ranges explored during the parameter exploration in CPF and ECPF based on [14, 16]. The entire exploration process was performed for both HT and NB as base classifiers.*

In GroCH we also optimised the minimum size of the warning window (15), minimum instances for insertion (200) and the maximum of classifiers per group (10). Other parameters like removal or selection policies concept similarity distance metrics, the punishing factor for weighting active versus background classifiers, were only explored briefly but left finally at their default values in GroCH (listed in Appendix A).

Finally, a key parameter optimised in all algorithms was their drift detection algorithm (EDDM and RDDM). The evaluation of the drift detectors was a major step in the exploration phase since a nested parameter exploration per drift detector had to be run. In our study, we used the best performing detectors from the literature (see Chapter 3). The reader must note that these techniques for supervised drift detection rely heavily on the base classifier used. Thus, the exploration had to be repeated for each base classifier. ADDM, proposed at the start of this chapter, was one of the drift detectors used. This is based on ADWIN2, one of the best performers in drift detectors in the SOTA (Chapter 3). However, for the experiments of this subsection, we were not able to find a parametrisation that could compete with RDDM and EDDM. Thus, despite its proposal in this section, this was no longer used in Stage II.

The reader must note that the goal of this work is not to perform an exhaustive analysis on the suitability of different drift detectors to a specific dataset. Rather, we aim to obtain a parametrisation that captures drifts at the right level of granularity. Thus, in general, drifts should map to ground truth switching events in the simulation of the time series.

As this requirement was already met for HDDM$_A$ and DDM using their default parameters for their implementation in MOA, for the sake of simplicity, this was the parametrisation selected and not optimised. Conversely, to avoid extra exploration, HDDM$_W$ was discarded since HDDM$_A$ reacted more accurately to drifts in its vanilla configuration and thus was chosen as the version of HDDM used in Stage I.

The parameter search performed for the rest of the drift detectors used each of the algorithms benchmarked, and every base classifier can be seen in Table 5.6.

Detector	Warning Range	Drift Range
ADDM	(1.0E-1, 1.0E-4)	(1.0E-2, 1.0E-5)
RDDM	1.03, 1.23, 1.288, 1.43, 1.63, 1.688	1.8, 2.0, 2.2, 2.258

Table 5.6: *Parameter range explored for warnings and drifts in the drift detectors. Warning detection parameters in RDDM are in regard to drift detection parameters are always values -0.670, -0.970 on top of the drift detection intervals. This equivalence is based on the plan vanilla parameters for RDDM. EDDM does not have any input parameters.*

A core piece of this first stage was the analysis of the accuracy detecting drifts, and recognising retrievals and insertions to the right ground truth state. The detection of concept drifts is a key aspect in GroCH, since this can impact the accuracies retrieving (v2$_{acc}$) or inserting (v3$_{acc}$) to the right group in the concept history, and thus, has a cascade effect in the classification accuracy. This is covered further in Section 5.3.4.

5.3.3.2 Stage II: Optimisation for Multiple Streams

This final stage of the parameter optimisation was applied over the 30 data streams. Algorithms were optimised for each synthetic set independently before the test and train split of the stream. As mentioned earlier in this section, this second stage started from the results from Stage I to shorten the search space due to the number of tests to be run.

The parameters used for GroCH in Stage II can be seen in the right-hand side column of Table 5.4. The main optimisation performed in GroCH was for the radius distance threshold (topology radius) for concept similarity which was performed for the values $\{1, 1.25, 1.5, 1.75, 2, 2.25, 2.5\}$. This is the range of values that we would generally recommend. The other important piece optimised for each stream was the drift detector. As base classifiers, HT and incremental NB were used. Hoeffding trees are one of the most common techniques used in the state of the art of data stream classification [240]. The other base classifier used was the incremental version (but not adaptive) of naive Bayes.

For CPF and ECPF, we performed a similar exploration than in Stage I (see Table 5.5) with the exception of fading, disabled as per results of Stage I and avoiding limiting the identification of recurrences as explained in [14], and f=4, since there are four states in the ground truth. We aim to allow four concurrent models.

Regarding drift detectors, RDDM and EDDM were the main methods used since these obtained the best overall results recognising regime switches. The RDDM drift detection confidences explored were: $\{1.8, 2.0, 2.2, 2.258\}$. For warning, values were lower by 0.670 and 0.970, in the range (1, 2). The difference in value between drifts and warnings were extrapolated from the default values of RDDM in MOA.

5.3.4 Behaviour of GroCH During the Pre-study

5.3.4.1 Drifts and Recurrences in GroCH

During the first stage of the parameter optimisation, we realised the need for an in-depth analysis of the market states. First, we found that across experiments, there could be a tendency to recognise the same group for retrieval and insertion at the same concept drift. By design in GroCH, this could be the expected behaviour in the following two cases:

1. *If the warning window starts way before the switch.* Thus, when the concept drift is signalled, the state recognised should be still the same one prior to the warning.
2. *If a concept drift is detected in a moment when the state has not changed.* This point is especially relevant if we evaluate retrievals and insertions outside the drift detection delay threshold (d), showing the importance of this parameter.

In a scenario where a concept drift is detected, but there is not an actual switch in the ground truth (false positive), the hypothetical right behaviour of GroCH would be to detect a false alarm and to continue with the same active classifier. However, GroCH, is unaware of the GT; hence, to realise the presence of a false alarm, it needs to retrieve a group to check for recurrence and find the classifier with the lowest error. In any case, there are cases where a recurring drift is recognised in case of a false positive in drift detection. This can happen, for instance, if the current state is already represented by a group of the ground truth (recurrence). This behaviour can still help GroCH improve classification accuracy, and thus, it is not undesired.

In case of a false positive in the drift detector, GroCH should retrieve the group that represents the active classifier (current ground truth state). Consequently, if a retrieval or insertion does not correspond to a regime change, the group recognised should match the current ground truth state. The drift detection delay threshold (d) is used to evaluate if a concept drift detected by GroCH corresponds to an actual regime switch. Hence, the selection of this value is of critical importance. To do so, one must take into consideration the mean length of warning windows using GroCH in this data stream and the expected maximum duration of the changes. While the first parameter can be easily obtained by running GroCH over any dataset, the second parameter would need a subject expert of the dataset utilised, unless there was actual knowledge of the ground truth (as in this chapter). In our analysis, d = 1,500 data instances. This value was obtained by exploring the range of [1,000, 2,000] instances, with being 1,000 the maximum length of regime change.

We noticed experiments in the pre-study where ground truth states and shifts leading to them may take several occurrences to be actually recognised by drift detectors. These cases were outputting low hit rates in retrievals and insertions in regard to that ground truth state.

A dilemma that arises here is whether to penalise misretrievals if a state is not represented yet by a group in the concept history as it would be done by default. As part of our analysis, we decided only to penalise misretrievals if there is already a group from the concept history that already represents that ground truth state. The rest of the retrievals and all insertions are taken into account for the statistics.

Suppose retrievals or insertions occur outside of the threshold of d instances after a ground truth change. In that case, these will be counted as accurate events as long as they retrieve or insert the group that maps to the current state of the ground truth. Conversely, retrievals and insertions that occur inside the drift detection delay threshold are considered successful if the closest group by concept similarity belongs to the destination and the starting ground truth state in the current shift, respectively.

As mentioned in the Appendix A.2, three key parameters of GroCH impacting the identification of recurring states are i) λ in GNG, ii) the topology radius and iii) the drift detectors with their respective parameters. Growing neural gas generates (through interpolation) a spatial summary with a number of prototypes λ times lower than the number of data instances received. Changing the feature set used in the topology also has a cascade effect in some of these parameters. The sub-topology with more features needed a greater topology radius to successful (1.5 instead of 1) and greater GNG λ. λ=15 obtained better results for the subtopology with the largest number of features, while λ=5 was the best overall. We believe that this is because adding extra features to the subtopology over the generated streams added noise, and greater λ can help clean part of this noise by summarising its spatial representation; especially if this noise is at the outer boundaries of the topologies, as explained in Chapter 2.

5.3.4.2 Impact of Parameters in GroCH

Results obtained in the pre-study helped to understand the behaviour of the different algorithms and different base classifiers compared in the simulated stream. As explained in previous subsections, GroCH, CPF and ECPF use supervised drift detection algorithms and thus, changing the base classifier produces a cascade effect in drift detection and classifier management actions such as replacing or training a new classifier. Furthermore, each base classifier used has a different degree of adaptability. While the incremental version of naive Bayes does not have any forgetting mechanism, Hoeffding trees reset node statistics as part of the split, performed over time. Therefore, this section explains results for each of the base classifiers used, leading to the final parameter set explored with the 30 streams for the experiments.

For naive Bayes as a base classifier, RDDM was the best performing drift detector. Over HT as a base classifier, EDDM has been the drift detector ensuring the greatest classification accuracies. An issue that arose here, though, is that in those experiments, EDDM barely detects any drifts, and most of them are signalled at the start of the data stream.

Covered in Chapter 2, HT (or VFDT) is an algorithm designed for stationary data, but that has achieved good results in concept drifting scenarios. This can be explained by the splitting algorithm of the leaf nodes, which resets the statistics of those and works, thus, as a kind of forgetting mechanism. According to these results, we believe that the synthetic stream used in Stage I is easily interpreted by HT and thus, this adapts well to the switches defined of 100 and 1,000 data instances of length. Furthermore, the four recurrences that this stream has (simulating market states) exhibit stationary patterns (in terms of price returns) despite their different price scales over time. It could also be the case that the switches from the ground truth are not abrupt enough for HT to fail in the short term. Regarding EDDM, this is a drift detector that, as explained in Chapter 2, is targeted to gradual drifts. It measures the distance of two misclassifications in terms of classification accuracy. The results with HT show that once the classifier has been training with enough data instances, its classification accuracy stabilises and thus, EDDM barely detected drifts after this point.

Different combinations of drifts detectors and base classifiers react differently, adapting and reacting to concept drifts. The impact of changing predefined parameters will vary significantly across them as the supervised drift detector, and replacement of classifiers relies on the classification accuracy obtained by the base models. Misdetection of concept drifts impacts in the accuracies retrieving (v2_{acc}) or inserting (v3_{acc}) to the concept history. The selection of the drift detector has a critical impact on the performance of all of these algorithms. On the one hand, a parametrisation very sensitive to drift will tend to replace classifiers at any minor drop in classification accuracy, not giving them time to be trained with a representative amount of data and become robust learners over time. On the other hand, a parametrisation set for very abrupt drifts can help in different critical scenarios but may not be able to detect relevant regime changes at higher frequencies which may be more gradual in nature.

The last fact that we observed during the pre-study is that the prepopulation of the concept history helps GroCH to identify all ground truth states, and thus, to improve classification accuracies during the development set and final test and train set used in this stage. Conversely, experiments not applying this prepopulation only were able to detect up to three out of four ground truth states. Experiments not pre-training only detected three out of four ground truth states. Thus, we believe that pre-training helps to tackle the limitation presented at the start of the section regarding the spatial representation and separability between ground truth states in the datasets used. Consequently, all the experiments in Stage II prepopulate the concept history.

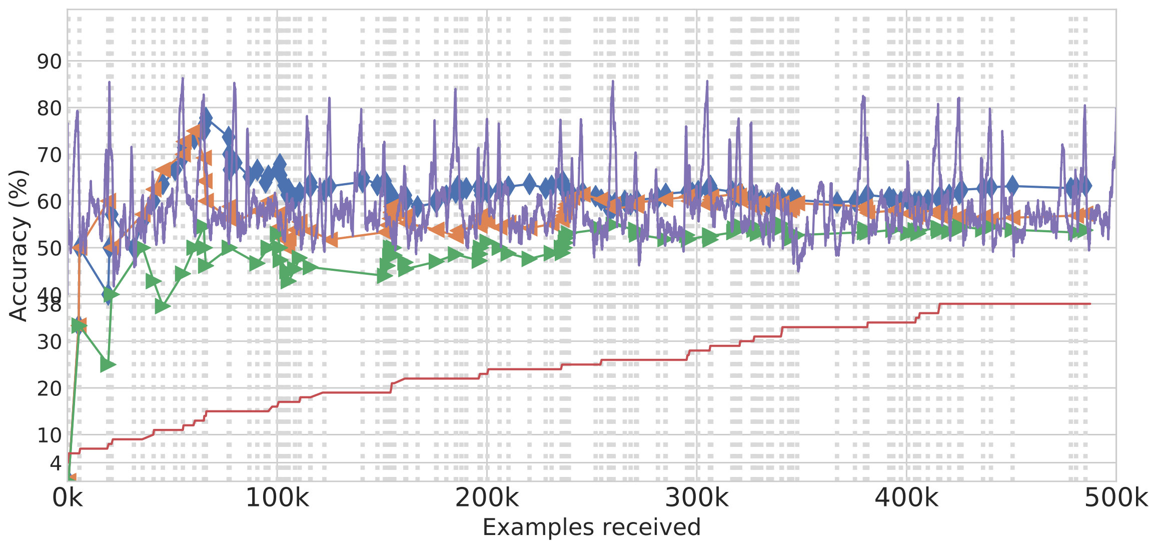
5.3.5 Results

In this section, we present the results of this chapter. First, we analyse the performance of GroCH detecting drifts and recognising the ground truth states in the simulated streams. Second, we benchmark GroCH with CPF and ECPF, its main competitors in the literature. Finally, we discuss the conclusions and limitations of our work.

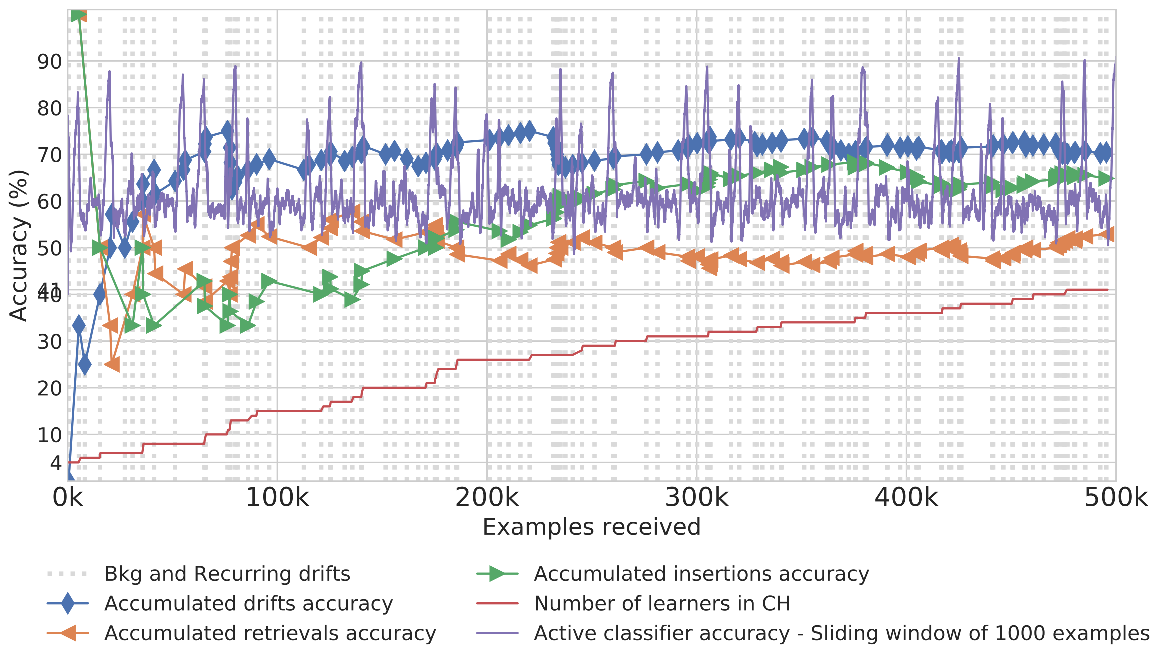
5.3.5.1 Performance Retrieving and Inserting Groups

Figures 5.17a and 5.17b show the accumulated accuracies detecting drifts, and retrieving or inserting the right ground truth state using naive Bayes and Hoeffding trees, respectively, as base learner in one of the synthetic streams simulated. Markers indicate the time of a drift (blue), retrievals (orange) and insertions (green). It can be seen how results may vary depending on the base learner. For instance, in this specific stream, when using HT, GroCH obtains greater accuracies detecting structural breaks up to d instances later; it also obtains better accuracies in HT in insertion to groups representing the right state each time.

The red line in Figures 5.17 shows the increase over time of the pool of learners stored in the concept history up to the maximum number of classifiers per group selected in Stage 2. Something that is not visible in this example but could also occur is that groups of a certain age could become obsolete if the correlation between the feature subset used for the topologies changes over time. In the semi-synthetic streams generated in this chapter, this can happen, for instance, if the generated price in the artificial series rises up or down much in time.



(a) *NB*

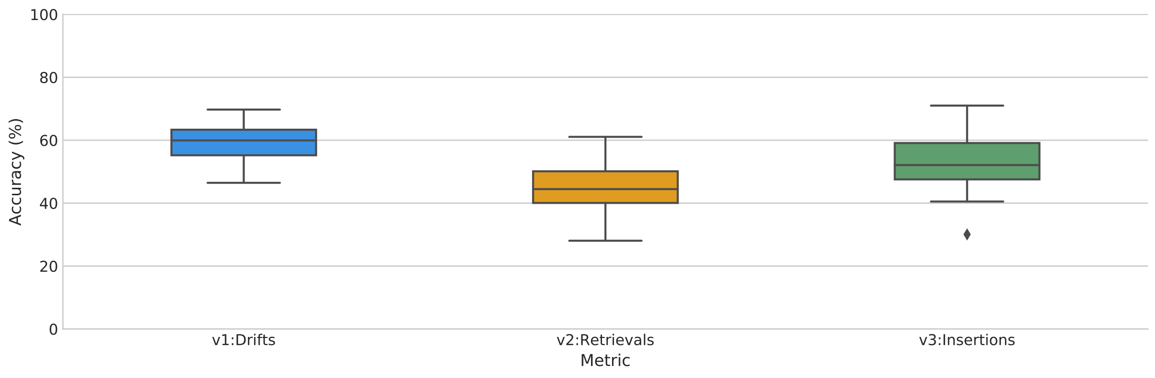


(b) *HT*

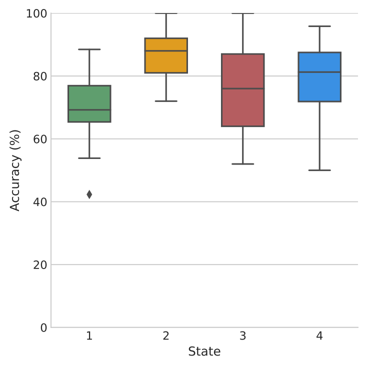
Figure 5.17: *Drift and group detection metrics in the first stream simulated.*

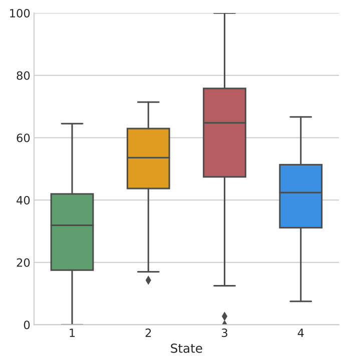
Another example is when prices vary much from the start, duplicating or beyond. In such scenarios, GroCH would create a new group. An alternative to this representation using prices is to use price returns, but this is currently out of scope as we focus on technical indicators.

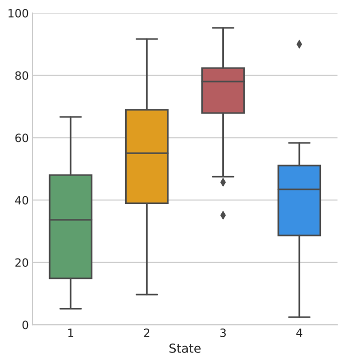
Figures 5.18 and 5.20 provide a quantitative point of view regarding the performance of GroCH dealing with changes in the 30 data streams simulated. These figures show the accuracies inserting ($v2_{acc}$) or retrieving ($v3_{acc}$) from a group representing the right state, and the percentage of drifts detected within d instances from GT changes. Subfigures 5.18a and 5.20a provide a summary of this, while the rest of the sub-figures show the performance per ground truth state.



(a) *Overall metrics*







(b) *Switches* **(c)** *Retrievals* **(d)** *Insertions*

Figure 5.18: *Analysis of drifts in GroCH using naive Bayes as base classifier.*

Subfigures 5.18a and 5.20a show mean metrics across experiments in the range of ≈50%-60% the accuracy for drifts (v1), retrievals (v2) and insertions (v3). An analysis of this in a quantitative point of view is a challenge since v2 and v3 rely not only on v1 but also on the time factor. We did not see significant differences in GroCH reacting to gradual versus sharper drifts. The size of the warning window in GroCH can come not only due to the speed of the drifts but also due to the delay detecting these. There can be scenarios where a learner keeps classifying the previous state well; thus, the supervised drift detector does not trigger even a warning signal. This scenario is acceptable from the point of view of GroCH, which tries to keep the best classifier during times of change, but can create problems in cases where this delay is converted into a sharp drift all of a sudden after the GT transition has ended. The mean delay detecting concept drifts (v1) inside d is 647 and 711 instances for NB and HT respectively. Figure 5.19 shows the distribution per base learner of these delays in all drifts that occurred in the 30 streams. Medians are similar across learners, but there is a slight greater delay on gradual drifts on Hoeffding trees. These can be due to the adaption ability of this learner.

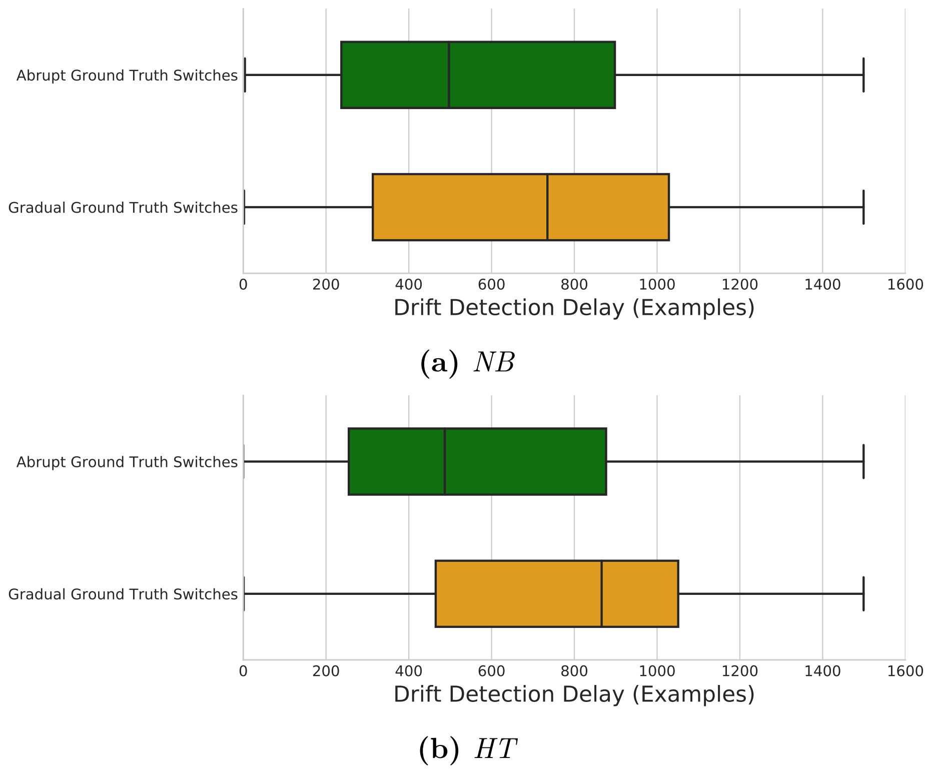
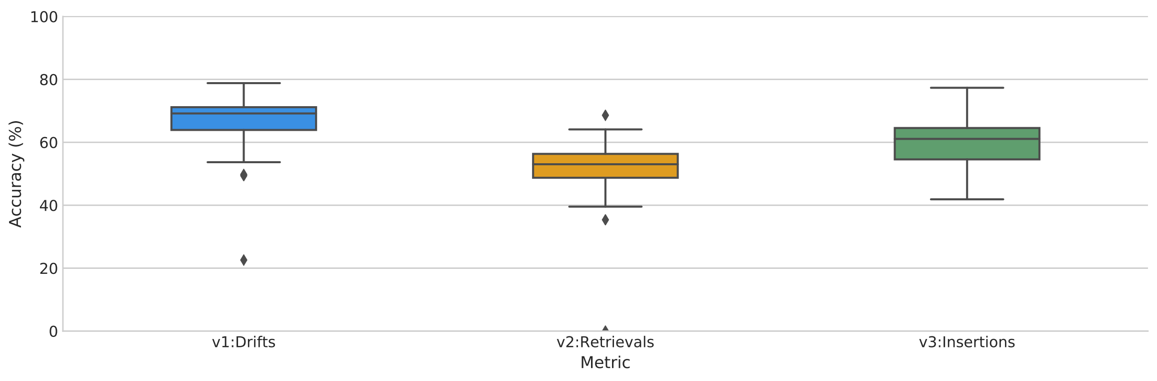
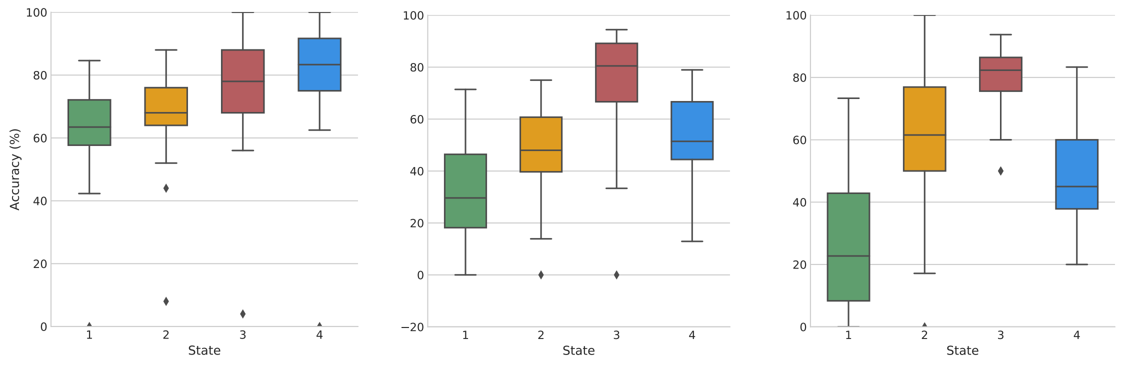


Figure 5.19: *Distribution of drift detection delay per drift abruptcy using d = 1500.*

Subfigures 5.18b and 5.20b have a different interpretation than v1 in (a). They measure for each GT state what percentage of the switches leading to each state has been successfully detected by the drift detectors. This metric (as well as v1) is independent of the result of the drift, which can be a false alarm, a background drift or a recurring drift that retrieves and inserts to the right group or not. Subfigures 5.18b and 5.20b measure detection of transitions from the ground truth perspective (TP$_{GT}$). Thus, this does not penalise the overdetection of drifts, as done by the measure v1$_{acc}$ in 5.18a and 5.20a. It can be seen how GroCH has detected ≈70-80% of the GT transitions in the 30 data streams. In Figures 5.18a and 5.18b we also see that drift detectors miss more transitions to state 1 consistently using both base learners.



(a) *Overall metrics*



(b) *Switches* **(c)** *Retrievals* **(d)** *Insertions*

Figure 5.20: *Analysis of drifts in GroCH using Hoeffding trees as base classifier.*

Subfigures c and d in 5.18 and 5.20 show retrievals and insertions per market state. Total numbers of insertions and retrievals are reported in Appendix A.3. The mean number of groups created on the top of the four prepopulated groups in NB across the 30 streams is 0.73 extra groups, in a range [0, 3]. In HT, the mean is 0.53, in a range [0, 2].

- *Insertions*: Figures 5.18d and 5.20d represent the accuracy of the insertions performed in GroCH split by the GT state at the time of the insertion. This has been recognised in overall with a ≈50-60% accuracy, better when using Hoeffding trees.
- *Retrievals*: Figures 5.18c and 5.20c represent the accuracy of the retrievals performed in GroCH split by the GT state at the time of the retrieval. This has been recognised in overall with a ≈50-60% accuracy when using Hoeffding trees, but with lower accuracies in naive Bayes.

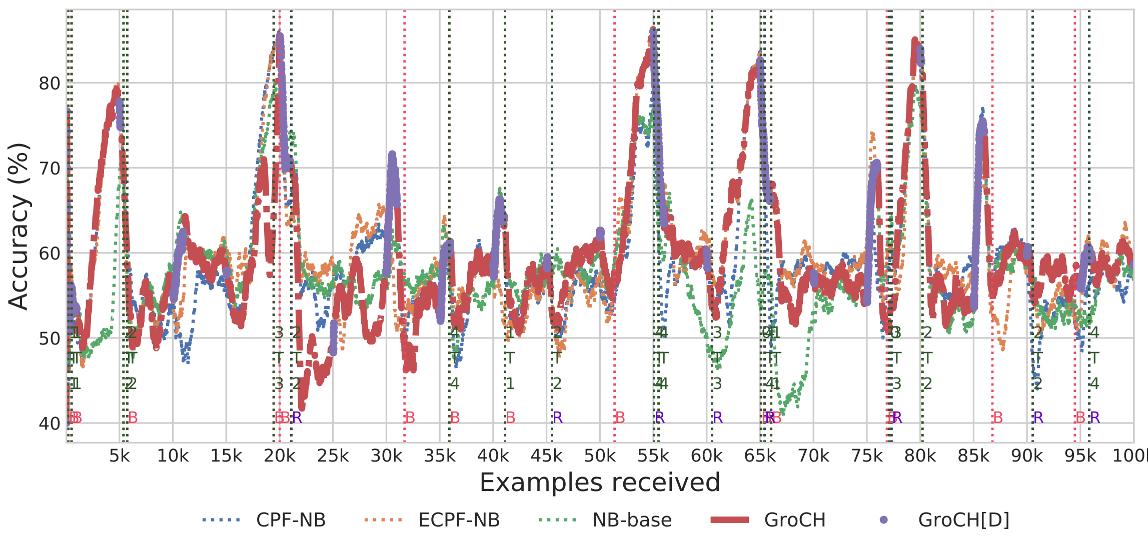
In general, state 1 is the worst-performing state in drifts, retrievals and insertions across all experiments with both base classifiers. This is followed by state 4 (second-worst). Plots shown in Section 5.2.4 justify this behaviour. State 4 is not shown as an easily separable state in Section 5.2.4, and state 1 is wrapped inside state 4 in the synthetic sets (Figure 5.12b in Section 5.2.4). The full set of metrics in this study, both overall and by state (including confusion matrices) can be seen in Figures A.10 and A.11 at Appendix A.3.

In the results obtained, GroCH appears to recognise groups and drifts better using Hoeffding trees. However, these tests are not entirely comparable since the parameter optimisation was performed for each of the 30 streams and base classifiers. Hence, Figures 5.18 and 5.20 use different parameters and components across the different executions for the 30 data streams (e.g. different drift detectors and intervals). Different parameters when using naive Bayes as base classifier may lead to similar or better results, but the selection made in this section was for the best performing configuration in each of the 30 development sets (see Section 5.3.2).

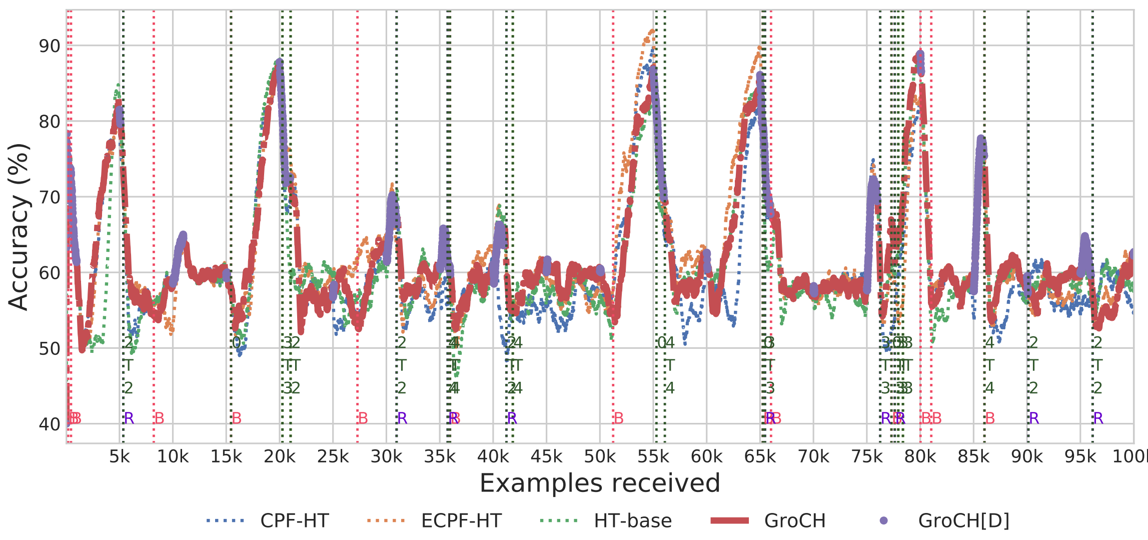
5.3.5.2 Benchmark

In this section, we compare the results obtained in GroCH and other relevant algorithms using 30 simulated data streams with their best parametrisation for each one (see section 5.3.3.2). But first, the reasoning behind the comparison to be performed must be explained. Subfigures 5.21a and 5.21b show an example run for the first 100k instances of the same data stream shown in the previous section (Figure 5.17). They provide a good overview of the performance of each algorithm over time. In Figure 5.21, background drifts detected with GroCH are marked in red. Recurring drifts are marked in purple, and retrievals (signalling the number of the group and state retrieved) from the concept history are marked in green. Ticks in the x-axis, every 5,000 instances, represent the start of a change in the ground truth. The accuracy of GroCH during ground truth switches is represented by GroCH[D].

All metrics under regime switches used in this section correspond to the prediction results in the data instances marked by a purple discontinuous line (same instances marked by GroCH[D]) in Figure 5.17. Metrics under switch are computed only during periods where the ground truth is shifting from one state to another, not being specific to any algorithm or drift detection signals. Hence, these are relevant metrics to rate all algorithms during shifts.



(a) *NB*



(b) *HT*

Figure 5.21: *Sample performance of GroCH, CPF, ECPF, and their base learner, along with drift-related actions in GroCH for the first 100k instances of the first stream simulated. Symbols: B: background drift; R: recurring drift; T: transition.*

In Figure 5.21 we see a different behaviour across meta-learners when we change the base classifier. This is one of the reasons why we have decided to carry on our study with two different learners, one that is not able to forget (incremental naive Bayes), and another one that can forget as part of its nature (Hoeffding trees).

For instance, in Figure 5.21 GroCH performs worst in the case of concept drift at early stages using NB (see the drop during the switch in instance 20k), but it does not see any significant relative drops to the other algorithms after that point. Figure 5.21 shows how not every GT switch triggers a background or recurring drift. As part of the behaviour designed, GroCH only changes the active learner if: i) there is a drift signal due to a decrease in its accuracy over time and ii) the active learner is not the best performer during the warning period. Figure 5.21 also shows how drifts occur at distinct times when using different base learners. The base naive Bayes by itself and CPF tend to be the worst performers in terms of classification accuracy in the first 100k instances for this stream.

Moving on to a quantitative analysis of results, Table 5.7 shows the mean AUS over the 500k instances reserved for test and train in the 30 data streams (over 100 concept drifts). GroCH obtains the best overall results during switch using HT as the base classifier with a statistical significance of 0.01 to the rest. Using NB as the base classifier, ECPF is the algorithm obtaining the best results with a p-value of 0.01.

Base	Algorithm	Mean	Std. Dev.	Min.	25%	Median (50%)	75%	Max.
HT	CPF	65.33	0.97	62.29	64.98	65.62	65.88	66.76
	ECPF	66.46	0.66	63.31	66.37	66.60	66.78	67.10
	GroCH	**66.83**	0.27	66.36	66.64	66.76	67.09	67.32
	Base	**66.79**	0.40	65.70	66.59	66.74	67.10	67.53
NB	CPF	60.67	0.59	59.19	60.35	60.69	61.09	61.77
	ECPF	**63.78**	0.62	62.38	63.29	63.92	64.22	65.01
	GroCH	63.24	0.50	61.77	63.05	63.26	63.52	64.12
	Base	61.17	0.45	59.99	60.88	61.07	61.56	61.93

Table 5.7: *Summary of accuracy under switch results across the 30 synthetic sets for 500k examples. Best mean results statistically significant per base learner marked in bold.*

Figure 5.22 shows that, using HT as base classifier, GroCH obtains higher accuracies than CPF and ECPF during structural breaks. However, HT tends to adapt very well to the majority of the data streams simulated, showing that in these data streams with changes of 100 and 1,000 data instances, an active drift detection mechanism may not be necessary. HT is even able to deal with periods of change and compete with GroCH, which is the best algorithm during switch in terms of accuracy.

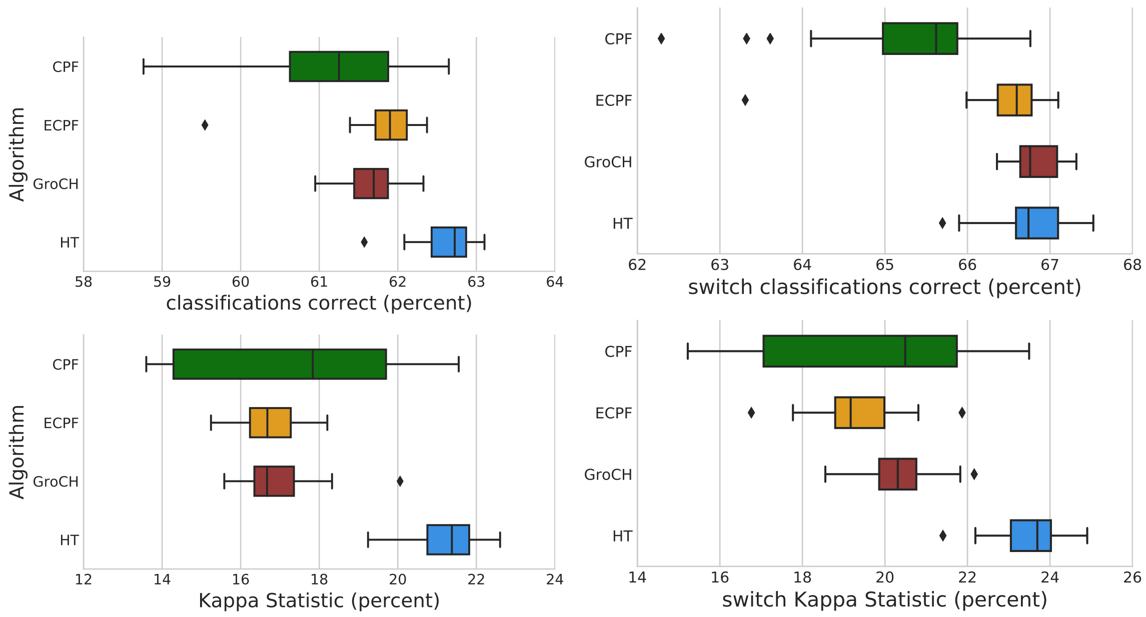


Figure 5.22: *Distribution of classification accuracy and kappa statistic overall (left) and during switch (right) across the 30 data streams with HT as base learner for 500k instances.*

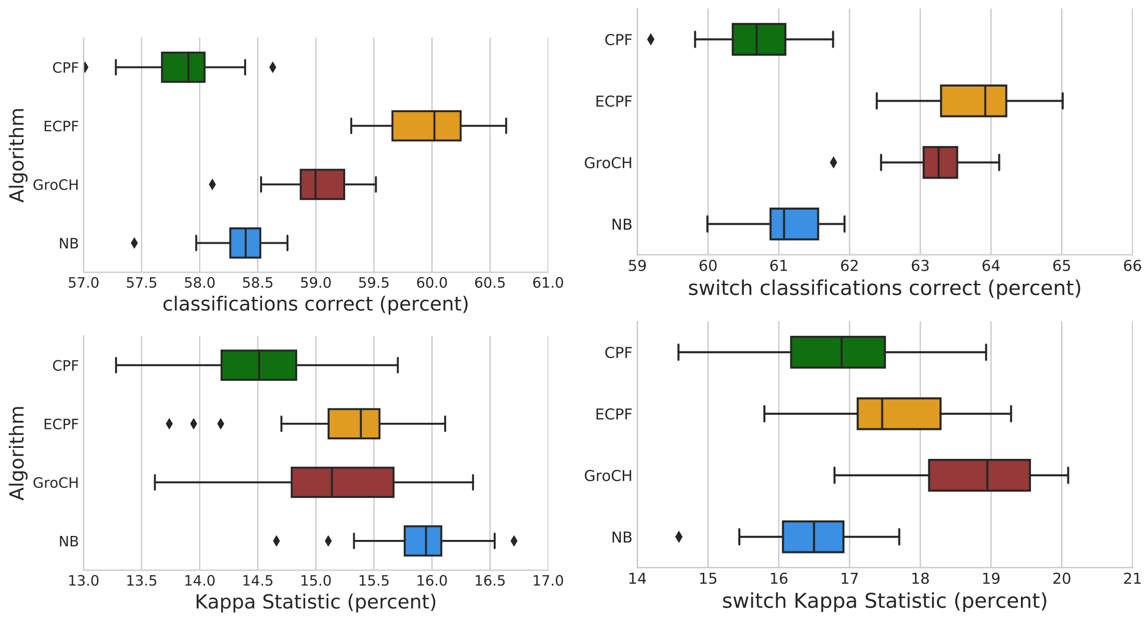


Figure 5.23: *Distribution of classification accuracy and kappa statistic overall (left) and during switch (right) across the 30 data streams with NB as base learner for 500k instances.*

Figure 5.23 shows that, using NB as base learner, ECPH is the best algorithm overall and during ground truth changes in terms of classification accuracy. However, looking at other metrics such at the kappa statistic, the base classifier NB appears to obtain the fairest predictions in this problem for all the data instances. However, the main goal of our research is to classify accurately during these times of change. Here, we find that GroCH is the strongest algorithm in terms of kappa statistic during regime switches, being statistically significant at 0.01 with Welch's Test.

Base	Algorithm	Mean	Std. Dev.	Min.	25%	Median (50%)	75%	Max.
HT	CPF	19.52	2.53	15.22	17.06	20.49	21.74	23.49
	ECPF	19.35	1.02	16.76	18.79	19.17	19.99	21.87
	GroCH	20.38	0.81	18.56	19.86	20.31	20.76	22.16
	Base	**23.52**	0.84	21.40	23.05	23.69	24.02	24.90
NB	CPF	16.93	1.02	14.58	16.17	16.89	17.50	18.93
	ECPF	17.61	0.83	15.80	17.11	17.46	18.29	19.28
	GroCH	**18.79**	0.86	16.79	18.12	18.95	19.55	20.09
	Base	16.49	0.63	14.59	16.06	16.50	16.91	17.70

Table 5.8: *Mean results in terms of KUS for the 30 for 500k instances in the 30 streams. Best mean results per base learner statistically significant marked in bold.*

Kappa statistic is a good measure to find the best classification algorithm in data streams tasks as it considers the randomness of each class. The reader must note that the simulated sets are based on different ground truths; class balance is expected to change over time. Table 5.8 summarises the kappa statistic obtained by all algorithms in the same experiments as in Table 5.7.

When looking at this metric on the experiments using HT, GroCH also has better results than ECPF during switches (KUS). CPF obtains competitive results with both, but it shows a high deviation across the experiments and does not look like a reliable meta-learner. Hoeffding trees obtain the best results during switches in terms of kappa statistics; hence these could be considered the best overall algorithm. The reader must note that parameter optimisation was performed for classification accuracy. Different experiments would need to be run to verify whether the other algorithms behave more suitably for this metric with other parameters during the development set.

As algorithms evolve over time, the best parameters for them should also change. This is, though, something that has been left as a future piece of work, and it is a research trend in machine learning for data streams. We have seen that HT was the best classifier, although GroCH obtained competitive AUS. GroCH was the best performer using NB in terms of KUS, but NB itself (without a meta-learner) performed better outside changes (during moments of stability). We have experienced through Stages I and II, though, that these meta-learners are algorithms susceptible to the parameters selected. Thus, experiments with a shorter data stream could be beneficial.

Table 5.9 shows the results at the middle of the test and train data streams. For 250k instances, GroCH using HT is the best algorithm during changes, and this is statistically significant at 0.01. The second best algorithm predicting is ECPF using HT as its base classifier, but this is not statistically significant (Mann–Whitney p-val 0.2) compared to running the base classifier (HT) alone. However, HT is still the strongest algorithm in terms of kappa statistic (see Figure 5.24).

Base	Algorithm	Mean	Std. Dev.	Min.	25%	Median (50%)	75%	Max.
HT	CPF	64.94	1.01	62.72	64.27	65.07	65.75	66.36
	ECPF	66.57	0.68	63.76	66.38	66.65	66.95	67.49
	GroCH	**66.93**	0.37	66.13	66.70	66.99	67.17	67.55
	Base	66.55	0.45	65.39	66.29	66.59	66.85	67.31
NB	CPF	60.72	0.89	59.01	60.12	60.80	61.35	62.23
	ECPF	**63.54**	0.74	62.03	63.03	63.64	64.07	64.81
	GroCH	**63.26**	0.65	61.63	63.03	63.29	63.74	64.21
	Base	60.93	0.63	59.79	60.50	60.93	61.37	62.66

Table 5.9: *Summary of accuracy under switch in 30 synthetic sets for 250k examples. First 50% instances from Table 5.7. Best mean results statistically significant marked in bold.*

Regarding algorithms using naive Bayes as their base classifier, both ECPF and GroCH are the best performers, statistically significant with a p-value of 0.01 to the rest. However, GroCH is significantly stronger during GT changes (see Figure 5.25). From these results, we see that, in the case of stationarities, both GroCH and EPCH are more beneficial in the short term. We would expect different base classifiers and their base learners to be converging in the metrics reported in the long term. Nevertheless, we saw that in the overall results for 500k instances, naive Bayes was the strongest algorithm (kappa statistics) besides obtaining one of the lowest classification accuracies (see Figure 5.23). A reason for this could be that an incremental naive Bayes is trained with a set that represents well the four different ground truth states, but the meta-learners keep replacing their classifiers every time that there is a minor inconvenience.

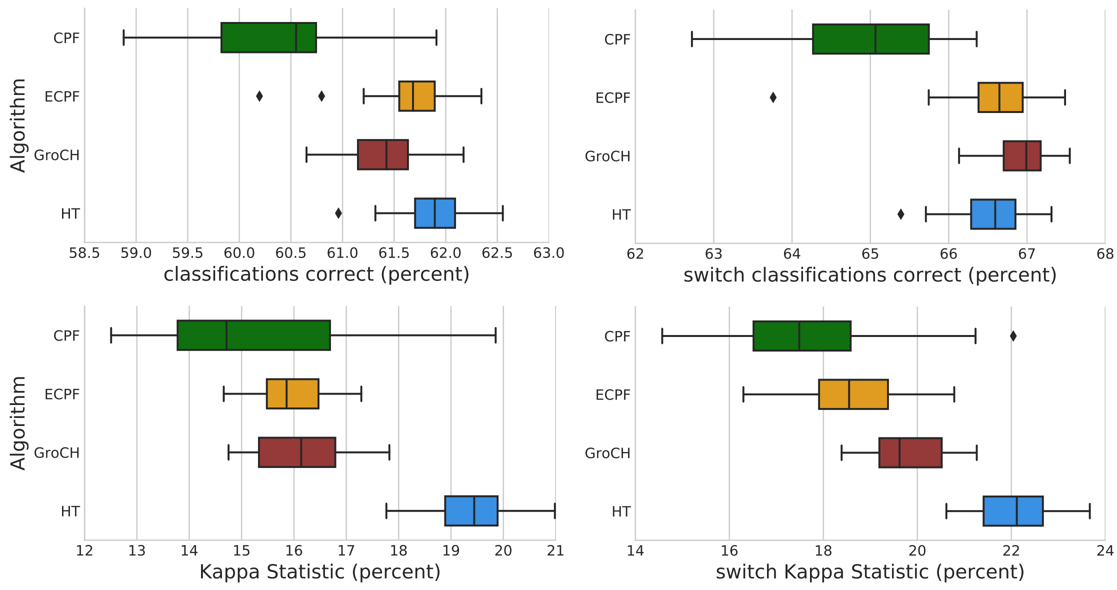


Figure 5.24: *Distribution of classification accuracy and kappa statistic overall (left) and during switch (right) across the 30 data streams with HT as base learner for 250k instances.*

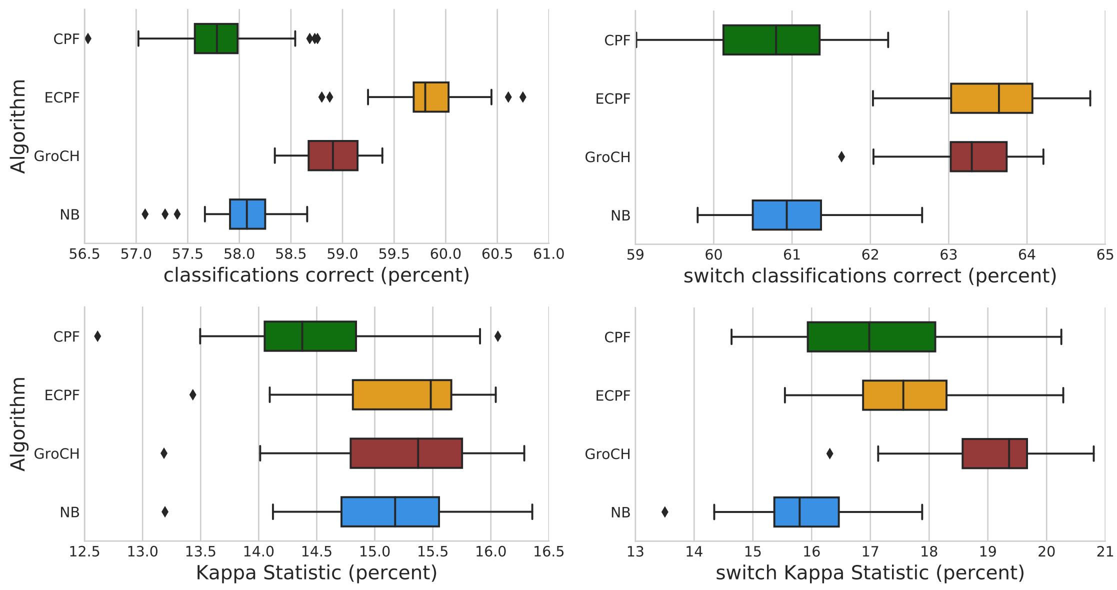
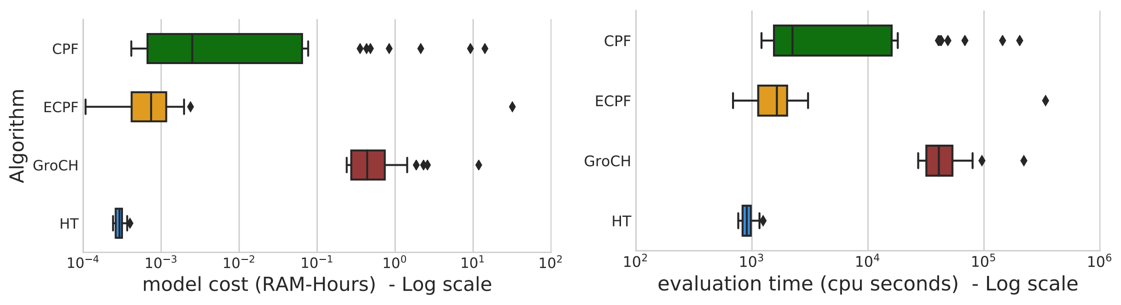


Figure 5.25: *Distribution of classification accuracy and kappa statistic overall (left) and during switch (right) across the 30 data streams with NB as base learner for 250k instances.*

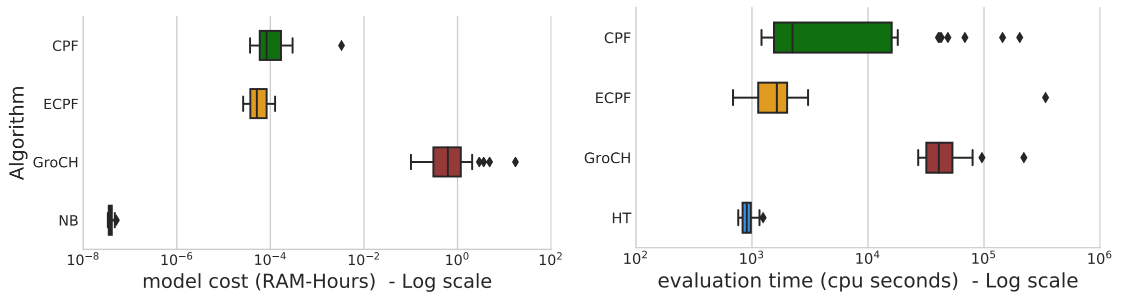
In any case, our current research aims to classify better during changes, and for this purpose, if we consider both metrics GroCH is the best meta-learner using NB. GroCH also performs better than its base classifier when using NB.

Base	Algorithm	Mean	Std. Dev.	Min.	25%	Median (50%)	75%	Max.
HT	CPF	9.26E-01	3.03E+00	4.14E-04	6.68E-04	2.50E-03	6.41E-02	1.42E+01
	ECPF	1.06E+00	5.81E+00	1.07E-04	4.18E-04	7.43E-04	1.16E-03	3.18E+01
	GroCH	1.03E+00	2.12E+00	2.39E-01	2.74E-01	4.38E-01	7.37E-01	1.18E+01
	Base	2.95E-04	3.68E-05	2.42E-04	2.61E-04	2.92E-04	3.15E-04	3.99E-04
NB	CPF	2.21E-04	5.90E-04	3.68E-05	5.93E-05	8.32E-05	1.69E-04	3.32E-03
	ECPF	5.96E-05	2.76E-05	2.63E-05	3.73E-05	5.14E-05	8.33E-05	1.26E-04
	GroCH	1.52E+00	3.17E+00	1.01E-01	3.05E-01	6.21E-01	1.18E+00	1.73E+01
	Base	3.92E-08	5.11E-09	3.37E-08	3.52E-08	3.80E-08	4.09E-08	5.13E-08

Table 5.10: *Model cost per algorithm and base classifier (RAM-hours).*



(a) *Hoeffding trees as base learner*



(b) *Naive Bayes as base learner*

Figure 5.26: *Distribution of runtime performance metrics, RAM-hours (left) and CPU seconds (right), across the 30 data streams in the different algorithms for 500k instances.*

Table 5.10 shows the cost of each model expressed in RAM-hours. It can be seen how GroCH is the algorithm with the greatest computational overhead for both base learners. ECPF is the meta-learner with the lowest mean computational cost. However, in Figure 5.26 it can be seen how, considering outliers, ECPF is the algorithm with the greatest maximum computational cost in terms of CPU seconds also the algorithm with the greatest maximum in RAM-hours using NB. Thus, in certain scenarios, ECPF is not a computationally cost-effective solution.

In summary, results in this section show that ECPF and GroCH as promising alternatives for periods of change when the base classifier did not have any forgetting mechanism (NB). Between these two, GroCH showed a better strength classifying (kappa statistic). GroCH seems to outperform all of the competitors in the short term (250k data instances). However, in the long term, base learners and competitors can improve their behaviour to the stationary behaviour of the series due to the recurrences. In the results using Hoeffding trees, while GroCH is still the best performer (greatest mean and lower std. in AUS), there is no statistical significance compared to its base classifier. Regarding kappa statistics, we see that in experiments using NB, meta-learning increases the strength of the predictions during changes.

Finally, while GroCH can be seen as a computationally expensive algorithm, this should not impact scenarios where the model needs to operate at minute level (or 30-second level) frequencies. In any case, the improvement of the computational cost in GroCH, which should first focus on the calculation of topologies, is left as future work. We extend on this in Section 5.5. As of now, the best cost-effective solution for the overall stream relies on the base learners themselves. But it depends on the context and application areas whether it is actually worth applying a meta-learner or not. For instance, any improvement during changes in the data distribution can become a competitive advantage in the financial domain.

The improvement of the predictions in GroCH relies on future work to find i) more accurate drift detectors to predict structural breaks, ii) feature sets data with more signal, iii) fewer clustering limitations (as covered in Section 5.2.4), and iv) more accurate distance metrics to differentiate states.

5.4 Limitations

One of the limitations of our approach, GroCH, is its high computational cost. However, this is not an issue impacting only our approach. The experimental results have demonstrated how other meta-learners exhibit higher computational costs in some scenarios. In the case of GroCH, we have observed that one of the main computational bottlenecks in our experiments was training topologies using GNG over time. This impacted significantly when many consecutive changes in the GT were not signalled by the drift detectors, causing the topology to train with extra thousands of examples.

GNG creates new prototypes by interpolation in the spatial areas with the most significant error. This causes a computational burden as the network of prototypes grows. In our scheme, we have trained topologies using a one-pass setting. To achieve a good spatial representation of the topologies and minimise the quantisation error, we had to use low values for λ (5); this is by default in the order of hundreds (200). Thus, while the cost in our experiments was a current limitation of our work, using a different algorithm to train topologies could improve this issue. This is out of scope from this thesis, as various extra experiments would need to be carried out, and we would probably see an impact of this change in other parameters.

Another significant limitation in our approach is that the supervised drift detection mechanisms do not detect every potential shift in the generative process. Thus, groups may not be representing the ground truth states accurately. Therefore, transitioning between states is not an optimal task. The pre-training of the concept history helps in this regard, but it is not exempt from potential noise that could be input in the history, causing the creation of new groups later. A mechanism to prune unused groups over time and unsupervised drift detection would help GroCH converge to a better solution. Both solutions are mentioned as future lines of work.

Finally, a limitation of the dataset selected is the hard separability between different market states (as seen in Subsection 5.2.4). This makes GroCH have difficulties detecting all different concepts. These issues could also be due to a lack of signal in the technical indicators selected (Subsection 5.2.2.3) used for classification and (the subset of them used for) clustering tasks. This can impact as well in the metrics for concept similarity used. A major issue with the Mahalanobis distance distance is that this needs to compute the inverse of the correlation matrix for the calculations. This cannot be computed if the variables are highly correlated [359]. This has not impacted the model in any of our experiments. Still, there could be constraints to compute the matrix in specific periods if the correlation across features drifts much over time.

5.5 Future Lines of Work and Changes in GroCH

Stages I and II comprehended the initial data exploration, pre-study and parameter optimisation in different phases. During these phases, we experienced that by reducing the topology radius, more groups would tend to be created, and GroCH would not recognise relevant concepts in the concept history that often.

We saw that in this scenario, the classification and retrievals accuracy (retrieving the right ground truth state) would tend to be overperform over time other experiments with greater values for the topology radius. However, this makes the accuracy of the insertions drop significantly. We believe that this could be due to the set of technical indicators selected; these may not represent well the current context in the ground truth in all scenarios. An automated feature selection process could be a future line of work.

The detection of drifts and the recognition of previous market states could be other flags that may be also valuable for traders. These, together with the evolution of volume over time and the big picture of things in the market (daily and weekly figures), can help indicate the return of an event, a market season, or a bear, bull, or very volatile market. These can translate to strong price pumps and dumps and be eventually more valuable than a simple up or down signal. Whether this works or not for real-world scenarios to identify regime or market state changes, it is a matter of market efficiency and adaptability to the market dynamics. During our experiments, this was demonstrated to be related to the selection of the feature set for classification and clustering tasks. Different indicators may exhibit different levels of signal or noise in different periods.

This domain is also subject to frequent changes over time. These changes impact the class distribution, the feature to target probability distribution (real changes) and the correlation and mutual information across the different features selected (virtual and feature drifts). Pursuing a profitable model would need to include trading strategies, fundamental indicators and sentiment from news and social media to tackle the lack of information and the high degree of noise in financial markets.

Guided by our initial hypotheses and data analysis at early stages, we came up with the logic for Algorithm 5.1. There was, in any case, a long design process iterating through different variants of this algorithm during Stage I of the study. In this section, we cover most of the versions that we explored. We believe these could lead to future research for this or related algorithms. In the following three pages, we list point by point these different variants of GroCH. We left them for future research as they would need further experimentation and refinement.

- *Insertions in case of early drift*: the main version of GroCH proposed in this thesis does not insert classifiers in the concept history in the case of early drifts (this is when the size of the warning window is smaller than a certain threshold). During the pre-study covered in this section, we ran some experiments inserting classifiers when early drifts were signalled. Still, we did not see any major improvement, and we left this feature disabled to be consistent with the retrievals.

- *Insertions in case of false alarms*: this was briefly explored during the design of the algorithm but considered out of scope in our current work.

- *Training of active classifiers during the warning window*: other algorithms from the literature do not train their active classifiers during warning (e.g. CPF), and thus, predictions during this time window are based on the training state of the classifier before warning. While this is something that may or may not work in different data streams, it is something that could be implemented as a variant in GroCH.

- *Training of retrieved learners with examples of the previous warning window*: GroCH assumes that recurring learners have been already trained on relevant or similar datasets and does not train them with the data instances received during the last warning window. We performed a small set of experiments during the pre-study in this matter, and we did not see significant improvements in any of the metrics over time. It may be worth exploring this, though, when applying GroCH to different data streams.

- *Retrieval of many groups within the maximum topology radius*: a small experiment we performed in GroCH was to use the cluster radius as a threshold to bring all groups inside. Then, all groups retrieved would compete, and we would select the one with the greatest conceptual equivalence (having a classifier with a lower error in the warning window). We observed that short term, this resulted in an improvement in retrieval and insertions accuracy. However, after 250k data instances, we obtained better results with the initial approach (get only the closest group). We left this potential change in the algorithm as a future line of work. Further experimentation may be needed to determine the root cause of the results that we saw in the pre-study.

- *Continuous training of group's topologies*: currently in GroCH, topologies are not updated over time, and if this changes much, eventually a new group will be created with a new topology, probably representing the evolution of a state that has an old topology in place. A future line of work here is to allow topology updates. One way to approach this is to update or replace the topology at the time of the insertion. We evaluated this during the pre-study and found that due to misinsertions, some noise would be introduced at some point, impacting the performance of the predictive model. Proper experimentation with this needs to be carried out to update topologies continuously over time to deal with the evolving nature of concepts in the concept history and not compromise the number of groups.

- *Continuous training of group's learners instead of classifier insertions*: GroCH saves a classifier in a group of the concept history at the time of a concept drift. A variant that we considered in this aspect was to keep a single classifier per group in the concept history. Then, at a concept drift, data instances would be fed (for training) to the classifier of the closest group instead of pushing another or replacing this classifier. In initial tests, this version showed promising results but presented many challenges in terms of computational costs as data instances would need to be kept in memory up to the concept drift. Since this approach required more work for high-performance and memory optimisation, it was left as future work.

- *Merging of classifiers in a group, pruning, and control of the number of groups*: since groups may become obsolete, a mechanism to remove groups that are no longer used over time could help increasing runtime performance in GroCH. This mechanism, similar to the method used by GroCH to remove the unused classifiers with the LUFO policy, could remove groups that have not been used in a given number of concept drifts or the group that has been used less recently if a maximum number of groups, which we would have to define previously, is reached. Another manner to address this would be adding mechanisms a fading in CPF [14], to constrain the size of the collection of groups.

- *Dynamic selection of parameters*: the number of parameters used in GroCH is quite large. An approach that handles different values, depending on the performance obtained over time, may be another future line of work. For instance, various experiments were run on this behalf during the design of RCARF [329] to set different values in their internal evaluator.

- *Unsupervised drift detection*: GroCH would greatly benefit from an approach to recognise drifts from changes in the topology instead of using supervised drift detection techniques. This is a significant change in the design of the algorithm, but it could be considered a future piece of work.

- *Dynamic selection of feature subsets for topologies*: the feature subset for the topologies in the experiments of this section was primarily aimed to discard technical indicators that are not in a given scale or can vary hugely over time, damaging the idea of ground truth states and recurrences in GroCH. In any case, future work with feature engineering at the topology level, or the use of a different attribute space for the topologies (e.g. usage of other indicators such as ARIMA forecasts or price returns instead of indicators based on close price) is of great interest to improve the results in both synthetic and real datasets.

- *Changes in the topologies*: we have conducted experiments in GroCH using GNG as the PG algorithm to build topologies. As covered in Chapter 2, different algorithms could be used for this purpose. In the case of GNG:

  - In this chapter, GroCH used the drift signal as a stopping criterion. It trained the algorithm using a one-pass training schema. We explored the usage of multi-pass training using a quantisation error percentage for early stopping, but this created performance bottlenecks in the training of GNG, and we had to leave it as future work.
  - During the experimental design, we also considered not to use topologies in GroCH and calculate distances over all the examples received for a group. However, we found that using prototypes generated with GNG, lower in number than data instances, helps to reduce the computational costs when computing distances.
  - Other distance metrics beyond Euclidean distance and Mahalanobis distance distances could also be used for concept similarity.
  - Another variant explored at the early stages in the design of the algorithm was the computation of a Mahalanobis matrix per topology or concept and then comparing the changes in the matrix correlations as a concept similarity measure. This was discarded due to the computational cost of computing such matrices, but it may be worth to be revisited in the future. In our proposal, GroCH used a single matrix to scale distances between groups.

5.6 Summary

In this chapter, we have proposed a meta-algorithm to handle different types of concept drifts in data streams. The main goal of our algorithm is to improve classification accuracy during times of change in time series of various natures.

Section 5.1 presented our approach, a *growing concept history of recurring classifiers* (GroCH) and compared it to the relevant literature. GroCH has a history where classifiers are grouped using a non-supervised approach. Our work brings together many different ideas from the state of the art of data stream classification. GroCH tracks change actively using a supervised drift detector. Classifiers can be retrieved from this history when the drift detector recognises a concept drift. If no similar models are stored from the past, a new classifier is created. If the base classifier adapts well to the non-stationary nature of the data stream, the detector should not raise any concept drifts. Thus, GroCH also allows passive adaption to concept drift. In the case of stationarities, as soon as a concept drift is detected, GroCH checks for classifiers from a similar state of the stream (closest groups from the concept history) that obtain a lower error than the currently active and background learners.

To evaluate GroCH and the classifier management operations performed, we proposed a framework that allows us to know the ground truth regarding changes in the states of the stream. This was done in Section 5.2. First, we selected a set of time series with different generative processes and fitted different ARMA-GARCH models to them. Each model represented the generative model of one market state in our framework. We simulated transitions between them of varying levels of sharpness. From the resulting time series, we created a feature set using a set of technical indicators common in the state of the art. This dataset was then used as an artificial data stream in MOA. To benchmark GroCH to state of the art, we use different metrics as kappa statistics and classification accuracy during times of change in the ground truth (KUS and AUS respectively).

Section 5.3 covered the experiments of this chapter. GroCH obtained competitive results with state of the art during changes and provided more robust predictions than its main competitors. As a drawback, GroCH exhibited a high computational cost. GroCH may not be suitable for online scenarios at the one second and higher frequencies if the current implementation of GNG [136] is used to train topologies. Section 5.4 explained this and other limitations of our approach.

In any case, in this chapter, we have introduced GroCH as a meta-learner that is agnostic to the base learner, non-supervised techniques, distances to identify groups, and to the drift detectors used. Our objective was to propose a scheme to manage classifiers in various concept drifting scenarios that can inspire future research in data stream learning. Section 5.5 pointed out different variations of our approach that may lead to future related work.

To wrap up, in this chapter we have presented and benchmarked GroCH in semi-synthetic data stream generated from ETF price returns. We have used a common approach in the literature to produce technical indicators and predict price ups or stable/down movements in the next time step. In the synthetic sets produced, GroCH outperformed its main competitors in terms of kappa statistics during regime changes. The next step in this thesis is to use and analyse the performance of GroCH in a real-world data stream. This will be done in Chapter 6.

Chapter 6

Application to Real Financial Data

6.1 Introduction

In this thesis, we have approached financial time series as a data stream mining scenario. Section 2.1 described them as a complex dynamic system in which behaviour can drastically change depending on business cycles or drastic events. This is also widely covered by the academic literature [54, 259, 347]. The experiments performed in Chapters 4 and 5 have demonstrated the suitability of different algorithms to deal with massive data streams and high-frequency data.

After these experiments, a benchmark with real-world data for the main proposal of this thesis that was presented in Chapter 5 is still missing. This chapter will perform such a benchmark as the fourth and last experiment of the thesis. Hence, we will apply the meta-algorithm proposed in Chapter 5 to improve stock trend classification in real-world data from the financial domain. Our aim is not to produce a trading strategy. Instead, our approach will predict ups and stable/down movements (not their strength) in the market price to act as a trading signal (or indicator) for decision-making processes for traders or investors, or to be fed to a trading strategy afterwards. This chapter is structured as follows: Section 6.2 covers the experimental design, introduces the classification problem targeted, describes the datasets used and their pre-processing steps, and explains the experimental protocol; Section 6.3 will show the results and provide an analysis of all the experiments. Finally, Section 6.4 will summarise the chapter and conclusions reached with real financial data streams.

6.2 Experimental Design

6.2.1 Classification Problem

A stock trend can be defined by the general direction of the market prices. In Chapter 5 we proposed a meta-algorithm for data stream classification applied to financial-like synthetic data. The data preparation in Chapter 5 used technical indicators as a feature set to train different active classifiers over time. This was based on the financial literature [179], where, also backed by research focused on business cycles changes and market inefficiencies [273], technical analysis is used to identify trendlines and patterns in events to predict uptrends or downtrends. Technical analysis was described in more detail in Section 2.1.

6.2.2 Financial Data

In this experiment, we have chosen SPDR S&P 500 Trust ETF (SPY) prices as a real-world financial data stream. The SPY ETF tracks a well-diversified market-cap-weighted index of U.S. large-and midcap securities known as the Standard & Poor's 500 Index (S&P 500). The S&P 500 is typically used as one of the main benchmarks to measure the stability and financial health of the U.S. economy. The SPY, which gives almost all of its funds into common stocks included in the S&P 500 Index, is one of the best-recognized and oldest US-listed ETFs, with large liquidity and trading volume.

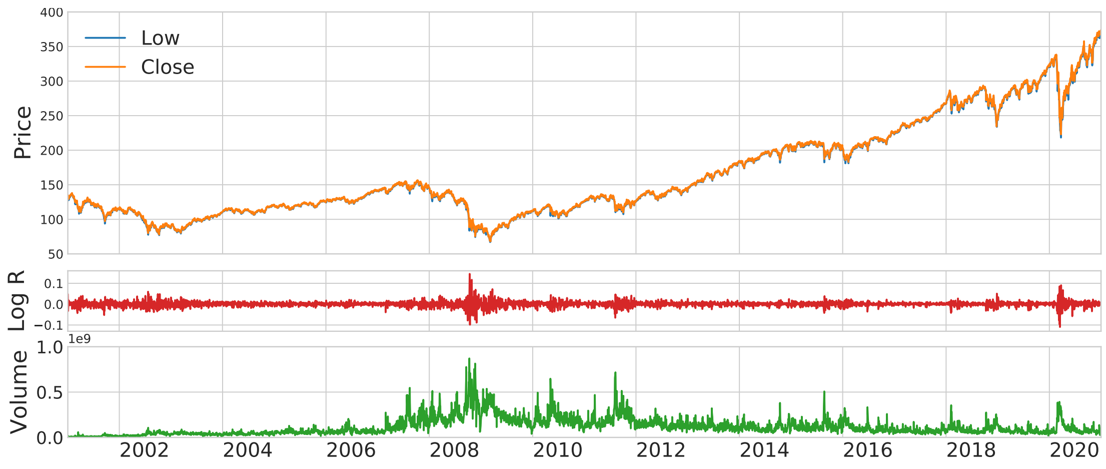


Figure 6.1: *SPDR S&P 500 Trust prices, log returns and volume from 2001-01 to 2020-12.*

Figure 6.1 illustrates close and low prices of the SPY over time between January 2001 and December 2020. This time range, which has periods of stability, crashes, and inflation, will be used across all our experiments at different frequencies at the intraday level. In Figure 6.1, periods of instability such as the Great Recession in 2008 and the COVID-19 pandemic in 2020 correspond to more significant trading volumes (in green) and volatilities (price returns in red colour).

In this chapter, we will split the fourth experiment of the thesis into two sub-experiments to compare the algorithm and its main competitors from Chapter 5 in different circumstances. These two sub-experiments will be named Experiment I and II along this section for the sake of simplicity. Furthermore, we will run experiments for different sampling frequencies at the minute and second levels in Experiment I. Since the number of instances explodes at the second level, we will shorten the length of the individual tests at the second level to reduce computation. Thus, we divide Experiment I into sections A and B depending on the frequency of the data as shown in Table 6.1.

	Period	Frequency	Period per test
Experiment I.A	2001-2020	1h, [30, 15, 10, 5, 1] min	1 year
Experiment I.B	2016-2020	[30, 15, 10, 5, 1] s	1 quarter
Experiment II	2011-2020	10 min	10 years

Table 6.1: *Summary of SPY periods and frequencies used at each experiment.*

The mean volume of data that will be used in the tests of Experiment I for each frequency is illustrated in Figure 6.2. Experiment I.A and I.B are coloured in blue and orange, respectively, in this plot. Regarding the volume of data in Experiment II, there is a single test period of 82,989 instances.

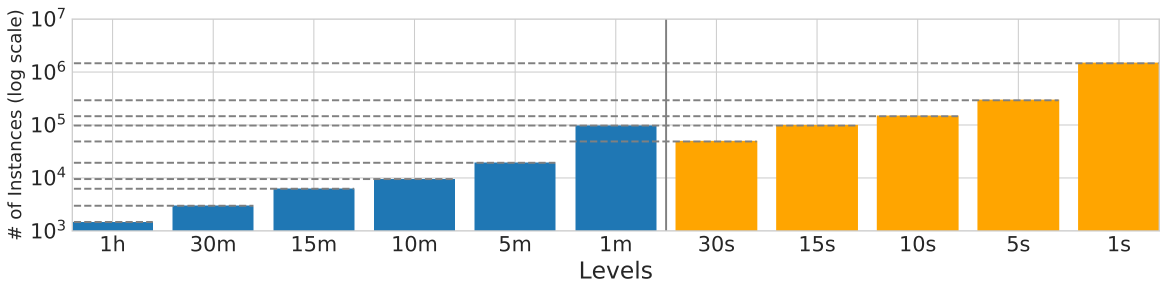


Figure 6.2: *Mean number of instances per period for different frequencies in Experiment I.*

The approach that will be followed to parse the price to binary labels and the set of technical indicators that will be used as a feature set are common approaches in the relevant literature [179, 195, 275]. All this will match the data preparation performed in Chapter 5 and explained in Subsection 5.2.2.3.

The class balance distribution, even for the minute level, varies depending on the frequency. We will use the mean class balance distribution across all seeds as a baseline for classification accuracy in our experiments. The baselines for all frequencies that will be tested in this chapter are shown in Table 6.2.

	Exp I.A						Exp I.B					Exp. II
Class	1 h	30 m	15 m	10 m	5 m	1 m	30 s	15 s	10 s	5 s	1 s	10 m
0	47.98	49.11	49.73	50.32	51.40	54.55	52.81	54.19	55.21	57.42	68.48	49.57
1	52.02	50.89	50.27	49.68	48.60	45.45	47.18	45.81	44.79	42.58	31.52	50.43

Table 6.2: *Classification baselines. Mean class balance (%) across all ind. tests per experiment.*

6.2.3 Experimental Protocol

As introduced in the previous subsection and in Table 6.1, we will cover a total of three different periods in our experiments.

- *Experiment I.A*: this experiment will train models with yearly datasets between 2001 and 2020. Models will be trained independently for a total of 20 annual periods. The objective is to show the statistical significance and benchmark the different models from Chapter 5 at different intraday frequencies.
- *Experiment I.B*: this experiment will also train models independently for 20 different periods. However, this time the data will be at the second level to compare the performance of the same algorithms than in Experiment I.A at greater frequencies.
- *Experiment II*: the second sub-experiment of this chapter will compare the performance of the same algorithms than in Experiment I over a long period of market data (10 years). The objective of this will be to see how the different algorithms can perform when running continuously for many years at high frequencies without retraining.

In all the sub-experiments, the classification error is reported as in Chapter 5 (see Section 5.2.1.1). The following two subsections will explain the protocol used to optimise models and run each sub-experiment in more detail.

6.2.3.1 Protocol in Experiment I

GroCH and its main competitors are meta-learners that produce deterministic results for the selection of base learners and detectors used. To perform post-hoc tests over the experimental outcome, check for statistical significance and reach conclusions, Experiment I will split the entire period from Table 6.1 into 20 different subperiods. Models will be tested and trained using a prequential evaluation for each of these periods.

Each of those 20 subperiods will be considered individual tests for the post-hoc analysis. Thus there will be a total of 20 tests per group representing the results of an algorithm at a frequency level. This is illustrated in Figure 6.3, where each algorithm, except from HT or NB, represents a combination between a meta-learner and a base classifier. Over each group of results, we will perform a normality test. If the distribution is normal, we will use Welch's t-test to test for statistical significance. Conversely, we will apply the non-parametric Mann-Whitney Wilcoxon test if the normality test is rejected.

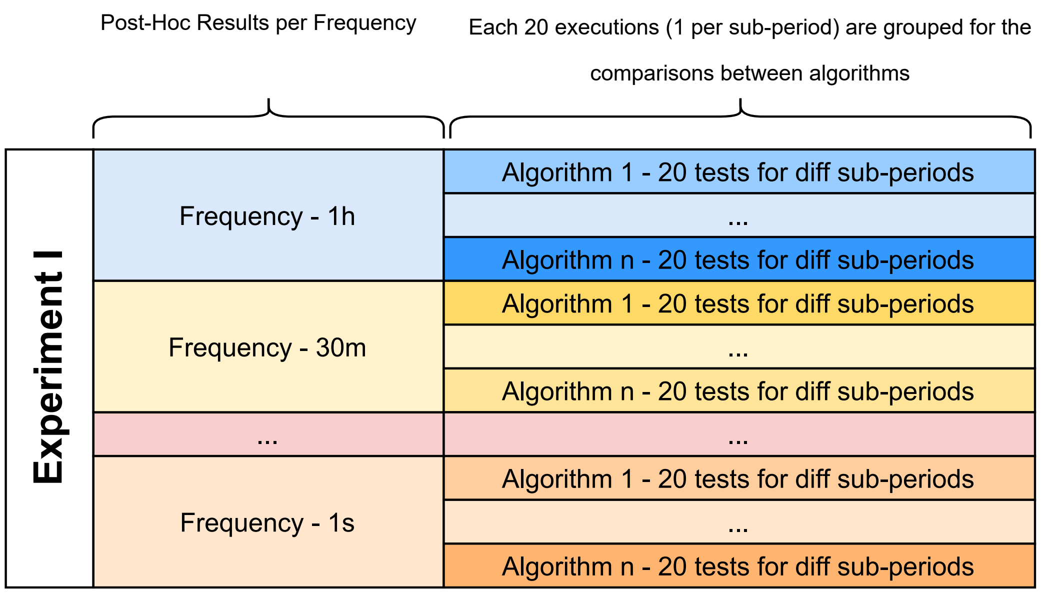

Post-Hoc Results per Frequency
Each 20 executions (1 per sub-period) are grouped for the comparisons between algorithms
Experiment I
Frequency - 1h
Algorithm 1 - 20 tests for diff sub-periods
...
Algorithm n - 20 tests for diff sub-periods
Frequency - 30m
Algorithm 1 - 20 tests for diff sub-periods
...
Algorithm n - 20 tests for diff sub-periods
...
...
Frequency - 1s
Algorithm 1 - 20 tests for diff sub-periods
...
Algorithm n - 20 tests for diff sub-periods

Figure 6.3: *Methodology with real data in Experiment I.*

Before each test is run, we will perform the parameter tuning of GroCH and its competitors independently for each of the 20 periods. This process will be further repeated for each of the frequencies of data that will be tested. Thus, the optimisation will be performed for each algorithm, base classifier, level of data (11 different frequencies) and period (20 periods per frequency) at the intraday level. This will be a total of 220 datasets used for parameter tuning over three meta-learners using two base classifiers. The parameters to be optimised will be selected according to the study performed in Chapter 5. Thus, the exploration will coincide with Stage II, in Section 5.3.3.2.

Figure 6.4 illustrates the allocation of data for testing, training and parameter tuning purposes. In this regard, we will select parameter values for each individual test. Hence, each algorithm and subperiod will use different parameters. With this in mind, we will select parameters for each test using the previous period of data (following temporal order). For instance, for a test in algorithm *1* using a period p for training, we will select parameter values for algorithm *1* before training using period $p-1$. We will call *development set* (devset), for each individual test, the dataset used for tuning the parameters of an algorithm.

Likewise, data from the time before each devset will be used to pre-train the concept history and compute the Mahalanobis distance matrix of GroCH. This will be named *Mahalanobis set* or pre-training set for the rest of the chapter. As in Chapter 5, we will pre-train the concept history of GroCH with different price trends. Due to the number of datasets used for the tests in this chapter (220), the selection process of these trends used to train states in GroCH is automated to select an uptrend, a lateral movement and a downtrend using mean price returns as described below:

- *State 1*: the lateral movement has been selected as the set of timesteps with the highest standard deviation of when the mean returns are close to zero (lower than 1×10^{14}, which was obtained through trial and error for the periods used).
- *State 2*: the downtrend has been selected as the set of timesteps with the minimum mean returns, filtering down the periods of close to zero return to avoid issues with lack of liquidity at the second level.
- *State 3*: the uptrend has been selected as the set of timesteps with the maximum mean returns.

Each of these trends will be selected programmatically from the period comprehending the Mahalanobis set and used to train a historical classifier in a group of the concept history. This will create one group per trend at the pre-training stage. Finally, the sets of data selected to pre-train the groups will be concatenated and used in conjunction to compute the Mahalanobis distance matrix used in GroCH.

The difference between the Mahalanobis (pre-training), development and test (and train) sets is illustrated in 6.4.

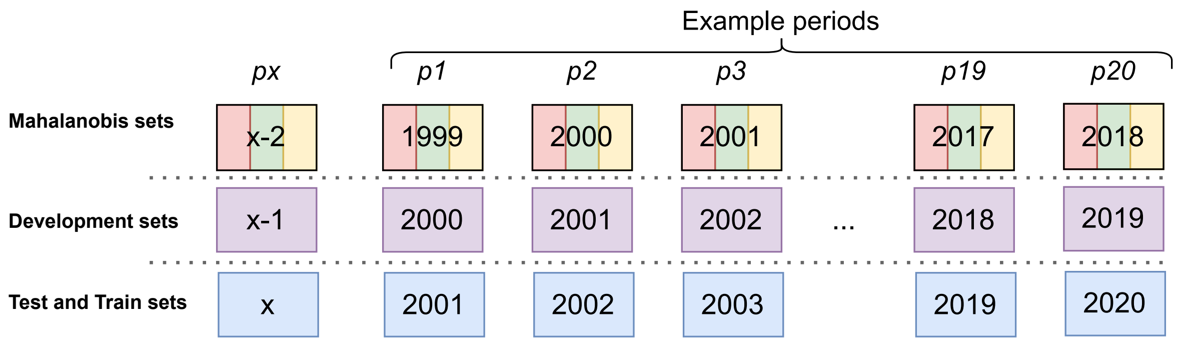


Figure 6.4: *Example data split across 20 periods in Experiment I.A.*

The reader must note that with the pre-training of GroCH using this automated approach in the financial domain, we risk adding noise to the models of the concept history and also to affect the parameter tuning process. In this regard, we have made our best to include states as separable as possible in each SPY test. Still, as explained in the previous chapter, a deeper analysis would be required in order to optimise the performance of GroCH.

As previously introduced, our only objective in this chapter is to explore the applicability of GroCH to these types of real-world streams and see if we obtain comparable results to our main competitors from Chapter 5.

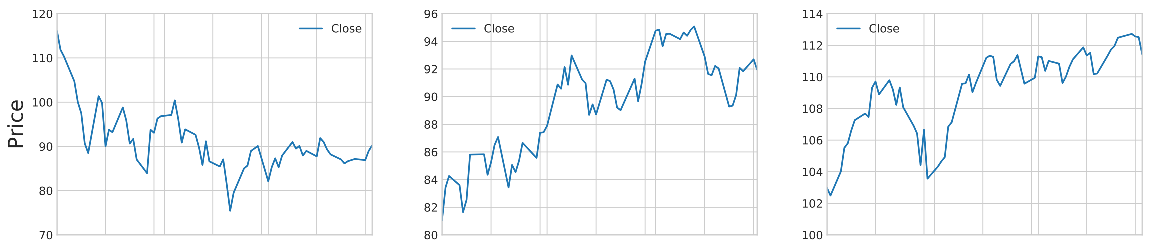
6.2.3.2 Protocol in Experiment II

In Experiment II, we will evaluate the performance of the best base classifier and meta-learners over a period of 10 years of data. The goal is to benchmark the best approaches in a continuous data stream mining scenario.

Experiment II will be focused on the price trend data frequency and the base classifier obtaining the best classification performance across meta-learners in Experiment I. The test and train period from Experiment II will cover the period 2011-Q1 to 2020-Q2. The full-year 2010 will be used used as a development set for parameter tuning following the approach from Experiment I. Regarding the set for the pre-training of the concept history and the computation of the Mahalanobis distance matrix, the selection of the states will be as follows:

- *State 1 (S1)*: bear trend made by the entire Q4 of 2008.
- *State 2 (S2)*: bull trend made by the whole Q2 of 2009.
- *State 3 (S3)*: overall uptrend made by the entire Q4 of 2009 that ends with a lateral trend.

Market prices belonging to the periods used to pre-train the concept history are illustrated in Figure 6.5.



(a) *SPY Q4 2008 (S1)* **(b)** *SPY Q2 2009 (S2)* **(c)** *SPY Q4 2009 (S3)*

Figure 6.5: *Periods of SPY data used for the Mahalanobis set in Experiment II.*

As in Experiment I, each state will be used to train a classifier and a concept history group. For the computation of the Mahalanobis matrix, we will concatenate these three. The objective of this pre-training is still to give GroCH a reference of separable price change trends. One of the differences with Experiment I is that, in the former experiment, we are automating the process of selecting the pre-training states due to the number of tests that will be performed.

We expect the manual selection for Experiment II to improve the accurate detection of regime switches and, with this, the classification results of GroCH.

6.3 Experimental Results

6.3.1 Experiment I

In Experiment I, we will benchmark the performance of the meta-learners of Chapter 5 and their base classifiers. The main difference with the previous chapter is that since the ground truth regarding concept drifts is not known for the real datasets, we can only benchmark the algorithms during the whole period, and not only during changes.

A summary of mean results can be seen in Figures 6.3 and 6.4. It can be appreciated how the classification accuracy obtained is greater for the higher frequencies, albeit this is due to the class balance that was shown in Table 6.2. In Figure 6.4 it can be appreciated how models do not seem to learn in the datasets with the lowest number of examples in Experiment I.A (frequencies 1h, 30min and 15min). Only frequencies below the 15-minutes level have positive kappa statistic percentages.

Algo	Base	1h	30m	15m	10m	5m	1m	30s	15s	10s	5s	1s
CPF	HT	50.287	50.245	50.626	50.726	51.559	54.147	52.004	53.553	54.642	57.022	68.207
	NB	49.698	49.779	50.215	50.615	51.622	52.957	51.328	52.343	52.638	54.197	63.075
ECPF	HT	50.015	50.743	50.750	50.964	51.661	54.171	52.100	53.560	54.659	57.054	68.218
	NB	49.959	49.457	50.377	50.937	51.514	53.420	51.789	52.709	53.759	55.920	66.532
GroCH	HT	50.700	50.708	50.374	50.851	51.574	54.160	52.112	53.402	54.491	56.866	68.149
	NB	49.527	49.404	50.156	50.565	51.585	53.309	51.644	52.787	53.682	55.446	65.696
HT	Base	51.078	51.054	50.797	51.223	51.813	54.321	52.258	53.669	54.700	57.018	68.237
NB	Base	49.632	49.481	50.559	51.131	51.991	53.282	51.876	52.768	53.431	55.188	64.124

Table 6.3: *Mean accuracies across frequencies in Experiment I.*

Algo	Base	1h	30m	15m	10m	5m	1m	30s	15s	10s	5s	1s
CPF	HT	-2.082	-1.349	-0.169	0.555	1.457	1.500	0.473	0.491	0.584	0.998	0.431
	NB	-0.911	-0.365	0.496	1.314	2.915	2.670	1.220	1.265	1.174	1.438	1.892
ECPF	HT	-2.615	-0.969	-0.101	0.782	1.642	1.248	0.338	0.302	0.298	0.516	0.197
	NB	-0.555	-0.517	0.676	1.721	2.820	1.647	1.290	1.015	0.878	0.749	1.427
GroCH	HT	-1.681	-1.124	-0.780	0.531	1.332	1.517	0.322	0.139	0.348	0.714	0.338
	NB	-0.843	-1.089	-0.003	0.940	2.591	2.780	1.341	1.150	1.160	1.422	2.010
HT	Base	-1.803	-1.141	-0.193	1.466	2.074	1.766	0.721	0.752	0.875	1.425	0.655
NB	Base	-0.446	-1.014	0.776	2.145	3.455	3.132	1.713	1.360	1.353	1.512	2.273

Table 6.4: *Mean kappa error across frequencies in Experiment I.*

In the following subsections, we will analyse the results in more detail for Experiments I.A and Experiment I.B, as this will help us to conclude about the results of Experiment I.

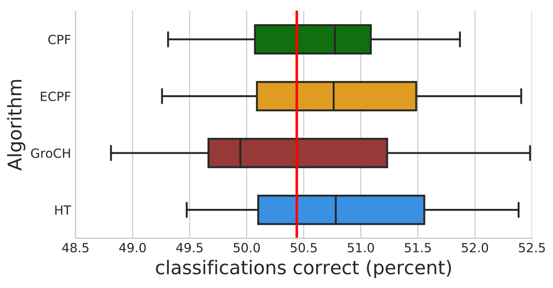
6.3.1.1 Experiment I.A: Minute-level Yearly Models

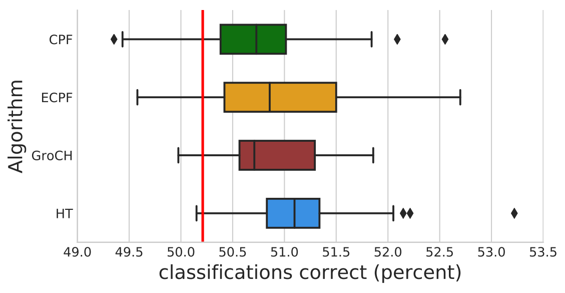
As shown in Figures 6.3 and 6.4, the meta-learners benchmarked only learn for higher frequencies than the 15 minutes level. Thus, we discard lower frequencies from this analysis. Table 6.5 summarises all mean results across tests for all algorithms and frequencies.

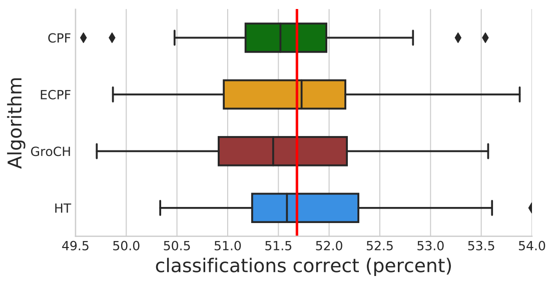
Freq.	Algo	Base	Acc. (%)	Kappa (%)	K.Temp. (%)	Time (CPU s)	Cost
15 min	CPF	HT	50.625635	-0.168635	4.140715	4.440432	7.031443e-07
		NB	50.215269	0.496475	3.306056	3.354200	5.464557e-07
	ECPF	HT	50.750495	-0.100835	4.392347	4.543093	7.590412e-07
		NB	50.376513	0.676401	3.610212	2.909704	4.457551e-07
	GroCH	HT	50.373846	-0.780477	3.643696	45.363687	3.307277e-05
		NB	50.156231	-0.002787	3.207645	39.261750	2.927758e-05
	HT	Base	50.796646	-0.193421	4.467256	0.560665	2.239306e-09
	NB	Base	50.559023	0.775780	3.970774	0.314121	6.520575e-10
10 min	CPF	HT	50.726358	0.555074	4.592256	10.686744	8.549173e-06
		NB	50.614641	1.314386	4.378276	10.572358	1.017589e-05
	ECPF	HT	50.963872	0.782200	5.046427	6.330838	1.061184e-06
		NB	50.937006	1.721050	4.996811	4.353567	8.008626e-07
	GroCH	HT	50.851440	0.531301	4.848333	76.482718	8.471903e-05
		NB	50.564950	0.940169	4.276489	65.581534	6.656402e-05
	HT	Base	51.222738	1.465544	5.546203	0.931045	5.335082e-09
	NB	Base	51.130890	2.145092	5.368556	0.446600	9.270582e-10
5 min	CPF	HT	51.559246	1.456773	6.679428	34.144369	4.637861e-05
		NB	51.621863	2.914828	6.795963	33.177237	7.189500e-05
	ECPF	HT	51.660859	1.642485	6.872221	34.122541	4.754290e-05
		NB	51.513824	2.820053	6.572342	27.664668	6.786252e-05
	GroCH	HT	51.573544	1.331863	6.713741	284.276269	5.771703e-04
		NB	51.584565	2.591281	6.724578	267.658174	5.956631e-04
	HT	Base	51.812925	2.074320	7.162360	2.402851	2.784169e-08
	NB	Base	51.990867	3.455130	7.503564	0.852650	1.769944e-09
1 min	CPF	HT	54.147265	1.500145	9.905884	1678.826702	2.686176e-02
		NB	52.957081	2.669583	7.525732	541.812300	4.615015e-03
	ECPF	HT	54.170954	1.248487	9.952015	120.959643	7.366512e-05
		NB	53.420338	1.647105	8.476755	816.013173	9.702578e-03
	GroCH	HT	54.160451	1.516583	9.929975	9016.449602	9.500103e-02
		NB	53.308749	2.780393	8.230000	5477.057671	5.428380e-02
	HT	Base	54.320535	1.766477	10.241612	36.444984	2.189349e-06
	NB	Base	53.281945	3.131916	8.159272	3.103569	6.442434e-09

Table 6.5: *Mean evaluation results of all models at the [15, 10, 5, 1] minute levels. Acc: accuracy; K.Temp: kappa temporal statistic; Cost expressed in RAM-hours.*

Figures 6.6 and 6.7 show a boxplot of the classification accuracies obtained for each frequency using Hoeffding trees and naive Bayes as base classifier respectively. The base classifier also appears in the benchmark, and the baseline for classification accuracy appears in Figure 6.6 to allow a direct comparison.


(a) *15 min*


(b) *10 min*


(c) *5 min*

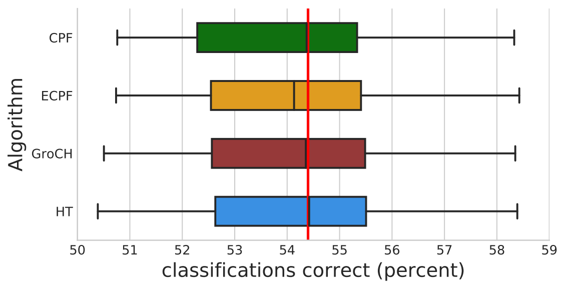
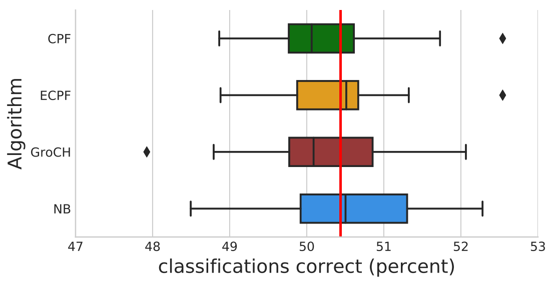
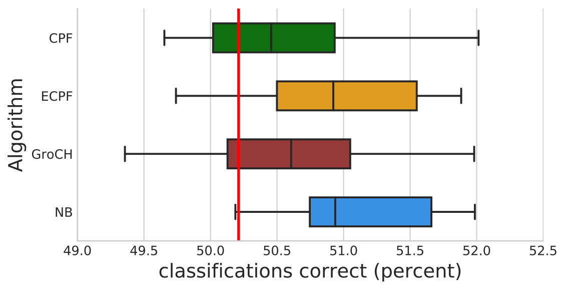
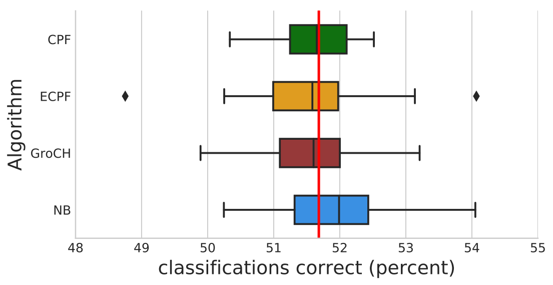

(d) *1 min*

Figure 6.6: *Distribution of classification accuracy using HT as base classifier across the 20 periods at the [15, 10, 5, 1] min level.*


(a) *15 min*


(b) *10 min*


(c) *5 min*

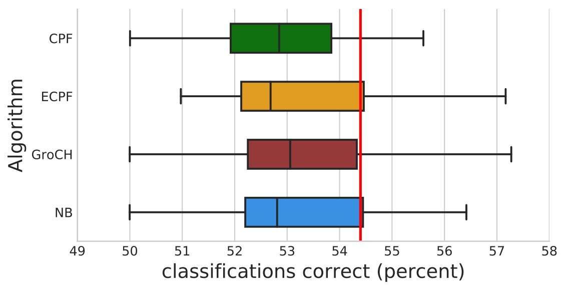
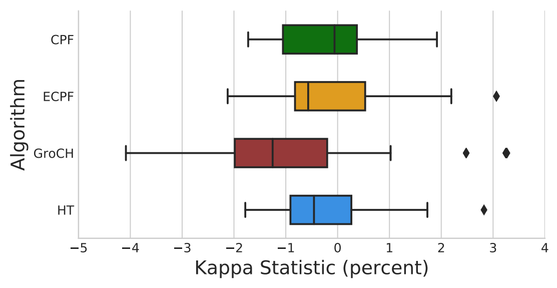
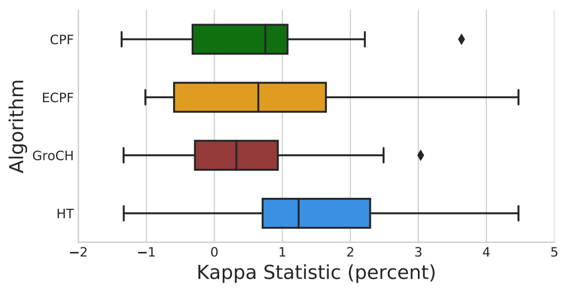

(d) *1 min*

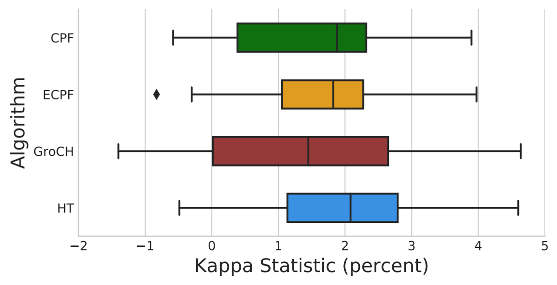
Figure 6.7: *Distribution of classification accuracy using NB as base classifier across the 20 periods at the [15, 10, 5, 1] min level.*



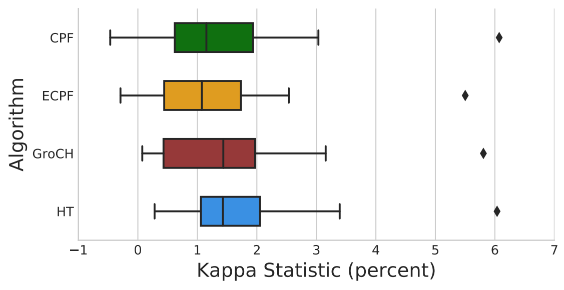
(a) *15 min*



(b) *10 min*

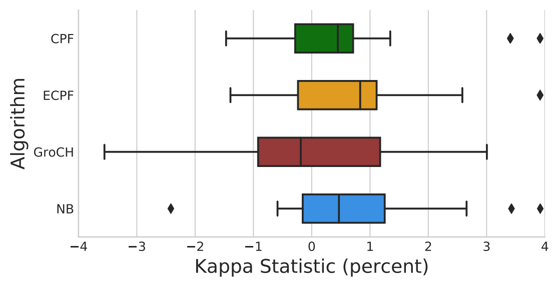


(c) *5 min*

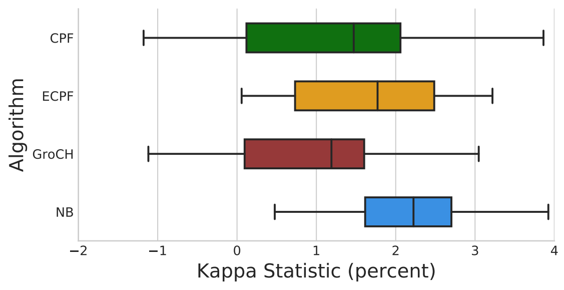


(d) *1 min*

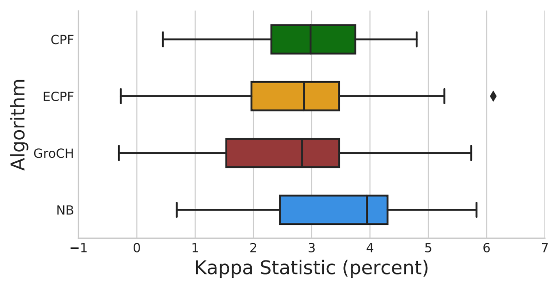
Figure 6.8: *Distribution of kappa statistic using HT as base classifier across the 20 periods at the [15, 10, 5, 1] min level.*



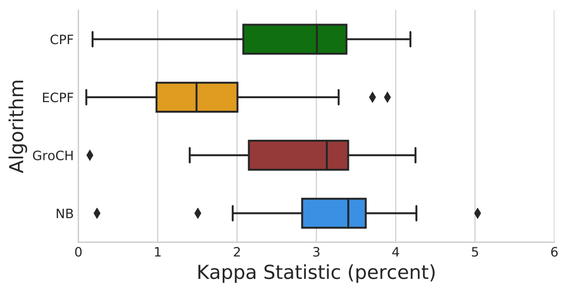
(a) *15 min*



(b) *10 min*



(c) *5 min*



(d) *1 min*

Figure 6.9: *Distribution of kappa statistic using NB as base classifier across the 20 periods at the [15, 10, 5, 1] min level.*

Figures 6.8 and 6.9 show a boxplot similar to the previous ones with Hoeffding trees and naive Bayes as base classifiers, respectively as well, but using the kappa statistic as a performance metric. In all of these figures, it can be seen that GroCH does not improve the other meta-learners from the state of the art. In any case, in general, meta-learners underperform their base classifiers and baseline for classification. It appears that the selection of 20 subperiods may not be appropriate for these data stream mining algorithms.

Across all tests of Experiment I.A, none of the algorithms offers a statistically significant better performance than GroCH at a p-value of 0.01. Thus, we can conclude that GroCH offers comparable results in these datasets.

6.3.1.2 Experiment I.B: Second-level Quarterly Models

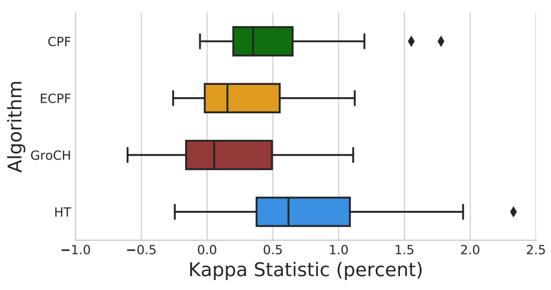
In Experiment I.B we benchmark the algorithms at higher frequencies up to the 1-second level. As mentioned in Section 6.2 due to the amount of data and the number of tests run, each subperiod represents a quarter of the SPY. Table 6.6 shows mean performance results of the meta-learners compared and base classifiers across all tests for the different frequencies used. Thus, for {30, 15, 10, 5, 1} seconds.

Figures 6.10 and 6.11 offer the distribution of kappa statistics across tests using Hoeffding trees and naive Bayes as the base classifiers, respectively for the four higher frequencies. In the first place, these figures suggest that the usage of naive Bayes as the base classifier seems to benefit drift detectors, which is discussed further in the next subsection.

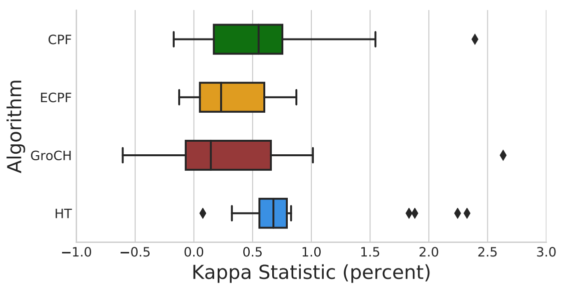
When using naive Bayes, it appears that GroCH obtains the best performance across tests for the three highest frequencies; see {10, 5, 1} second level frequencies in Figure 6.11. However, GroCH is not statistically significant at a p-val of 0.05 when compared to its base classifier. Hence, we cannot conclude that GroCH is the algorithm with the greatest classification accuracy at second level frequencies. Again, as for the minute level, the subperiods selected at the second level do not seem to suit the purpose of the chosen meta-learners. Different reasons for this are discussed in Subsection 6.3.1.3, and Experiment II is selected accordingly.

Freq.	Algo	Base	Acc. (%)	Kappa (%)	K.Temp. (%)	Time (CPU s)	Cost
30 s	CPF	HT	52.003609	0.473161	5.200072	109.546227	2.490056e-04
		NB	51.327781	1.219921	3.846578	210.976599	1.100825e-03
	ECPF	HT	52.100362	0.338144	5.393849	49.025945	1.253487e-05
		NB	51.789261	1.290411	4.762889	23.225465	4.400021e-06
	GroCH	HT	52.111786	0.321968	5.418116	1492.244260	7.454636e-03
		NB	51.644088	1.340549	4.468484	1317.496278	6.360009e-03
	HT	Base	52.257807	0.720936	5.696948	10.299842	3.033626e-07
	NB	Base	51.876226	1.712654	4.924695	1.834570	3.808227e-09
15 s	CPF	HT	53.553330	0.491457	8.388418	334.889022	2.418849e-03
		NB	52.342541	1.265194	5.971442	998.479729	9.035679e-03
	ECPF	HT	53.559716	0.302304	8.409514	193.064178	6.492341e-04
		NB	52.709386	1.014607	6.708591	159.512693	8.130591e-04
	GroCH	HT	53.401620	0.138540	8.098338	3999.240984	3.571885e-02
		NB	52.786762	1.150000	6.851343	6214.750751	6.342857e-02
	HT	Base	53.669268	0.752361	8.613913	34.794436	2.111591e-06
	NB	Base	52.767708	1.360118	6.798289	3.137084	6.512004e-09
10 s	CPF	HT	54.642359	0.583919	10.268508	911.971597	8.698563e-03
		NB	52.637610	1.173558	6.168360	2201.440803	3.878805e-02
	ECPF	HT	54.659169	0.298263	10.305259	144.308335	3.025303e-05
		NB	53.759271	0.878207	8.494832	93.909170	1.835333e-05
	GroCH	HT	54.490741	0.348318	9.973504	8799.605140	1.338077e-01
		NB	53.681975	1.160491	8.317790	11812.214214	1.929398e-01
	HT	Base	54.700306	0.874751	10.378035	77.748629	7.117689e-06
	NB	Base	53.431109	1.352624	7.812912	4.606199	9.561615e-09
5 s	CPF	HT	57.021831	0.998195	15.067405	18393.775704	9.693022e-01
		NB	54.196955	1.437704	9.231002	3494.468176	6.056469e-02
	ECPF	HT	57.054195	0.515769	15.143417	414.355138	1.294060e-04
		NB	55.919672	0.749211	12.859058	943.735555	6.960924e-03
	GroCH	HT	56.866468	0.714473	14.765123	29851.930866	8.720825e-01
		NB	55.446024	1.422432	11.867166	51279.313167	1.715619e+00
	HT	Base	57.017599	1.425214	15.051285	297.441687	5.383996e-05
	NB	Base	55.188306	1.512116	11.311316	9.142903	1.897897e-08
1 s	CPF	HT	68.206638	0.431090	29.069109	10560.589654	1.925935e-02
		NB	63.075230	1.892301	16.899395	30531.168885	2.770322e-01
	ECPF	HT	68.218475	0.197363	29.092652	3493.713902	3.199121e-03
		NB	66.532303	1.426525	25.283804	3200.600923	1.452818e-03
	GroCH	HT	68.149391	0.338118	28.954258	65410.637579	4.274959e-01
		NB	65.695672	2.010251	23.264354	274999.193882	1.307663e+01
	HT	Base	68.236915	0.655152	29.123149	9503.935768	8.961293e-03
	NB	Base	64.123848	2.272633	19.387134	118.975285	2.469706e-07

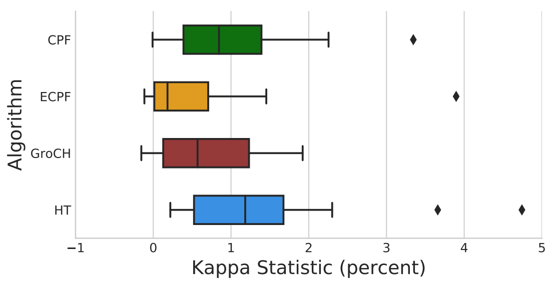
Table 6.6: *Mean evaluation results of all models at different second level frequencies. Acc: accuracy; K.Temp: kappa temporal statistic; Cost expressed in RAM-hours.*



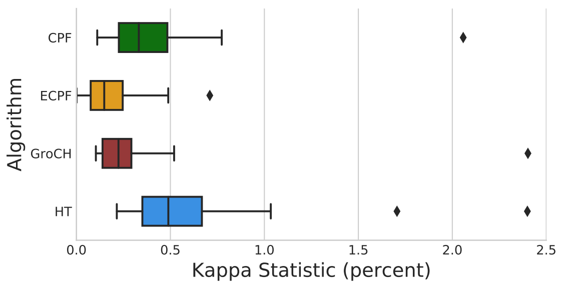
(a) *15 s*



(b) *10 s*

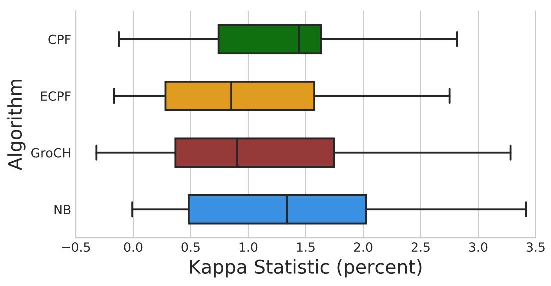


(c) *5 s*

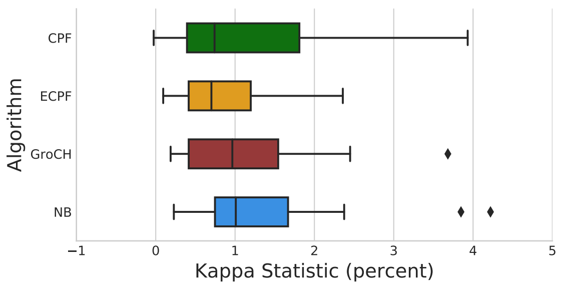


(d) *1 s*

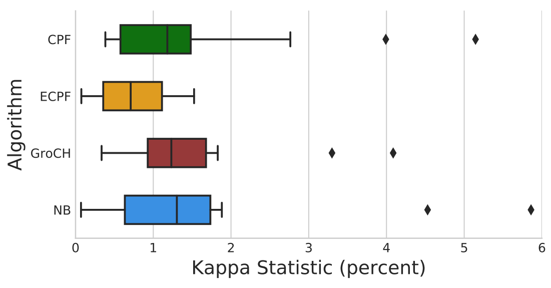
Figure 6.10: *Distribution of kappa statistic using HT as base classifier across the 20 periods at the [15, 10, 5, 1] seconds level.*



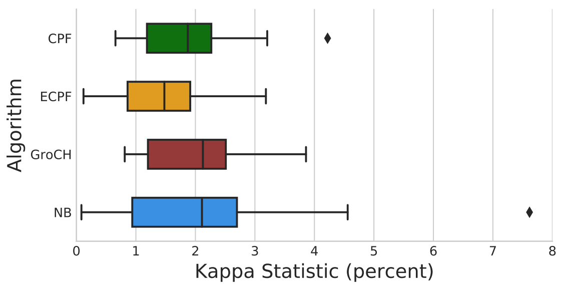
(a) *15 s*



(b) *10 s*



(c) *5 s*



(d) *1 s*

Figure 6.11: *Distribution of kappa statistic using NB as base classifier across the 20 periods at the [15, 10, 5, 1] seconds level.*

6.3.1.3 Analysis of Experiment I

Experiments I.A and I.B have shown how, in general, meta-learners obtain comparable results across all frequencies for the same base classifier for the yearly and quarterly subperiods, respectively. GroCH does not improve the accuracy performance of any of the models with statistical significance on any of the frequencies. It can be seen how for different frequencies, either ECPF or GroCH can be the best meta-learner in terms of classification performance, but not in all cases they do better than the base classifier. For instance, using naive Bayes as the base classifier at the 15-seconds level, GroCH obtains the best classification accuracy over all the meta-learners using NB and overcomes its base classifier. However, GroCH does not improve the kappa statistic obtained by its base classifier (NB) for this frequency (15s).

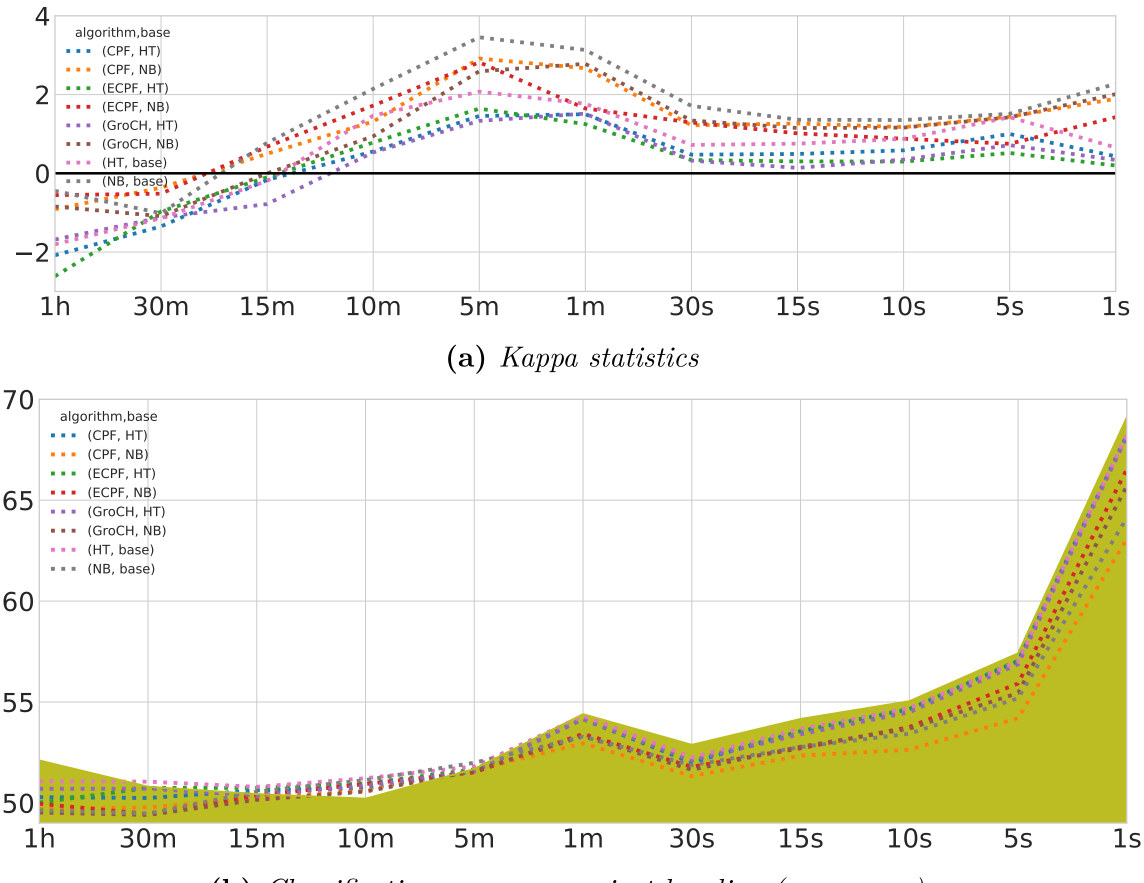
It appears that second level frequencies seem to benefit GroCH when compared to other meta-learners. It is not clear if this is due to the price trend dynamics at such levels of granularity or due to having more data. It can also be due to the different subperiods selected, representing quarters from Q1 of 2016. In general, it can be seen how all learners obtain greater kappa temporal statistics for the higher frequencies. This expresses a lower error when not considering temporal dependency. This can be caused by the higher degree of noise at the higher frequencies.

At this point, we have not mentioned yet how do different models compare in terms of cost. These behave has already been seen in Chapter 5. GroCH can be the meta-learner with the highest computational cost. Although in some tests, CPF obtains comparable or even highest costs. ECPF tends to be the meta-learner performing with the lowest cost in most circumstances.

Chapter 5 focused on the learning performance during structural breaks. However, in this chapter, this is not feasible since, in real-world financial data, we lack ground truth knowledge regarding actual shifts.

There is a challenge to draw a line on what algorithm is best for times of change. The benchmark of GroCH for error metrics that comprehends the whole data stream (with moments of stability) does not seem to show the benefits that we saw in the previous section in Experiment I. An open question addressed in Experiment II is how these classifiers would handle longer data periods.

Something that catches our attention is the fact that in some scenarios, GroCH and ECPF obtain a greater accuracy using naive Bayes rather than Hoeffding trees as a base classifier. A hypothesis here is that HT could adapt to some of the gradual drifts, which could be more frequent than in the synthetic set from the previous chapter. This adaption could create a snowfall effect on the drift detectors that may not be able to trigger drifts from the variations of errors obtained by the base classifier.



(a) *Kappa statistics*

(b) *Classification accuracy against baseline (green area)*

Figure 6.12: *Kappa statistics and classification accuracy across frequencies in Experiment I.A and I.B.*

Parametrisations that are more sensitive to concept drift for the detector could help use HT as a base learner. This has not been something visible from the parameter optimisation done beforehand. As seen in Figure 6.12, the 10-minutes level frequency appears to be the frequency where the algorithms learn more, and this could mean that the data has a higher signal-to-noise ratio.

This section has tested the performance of GroCH in short periods that may need or not of adaptive models. The experimental results show that our novel approach GroCH obtains comparable results to other meta-learners from the state of the art. However, base classifiers also obtain comparable results with a lower computational cost. This could mean either that this dataset is not suitable for this technique, or any improvement of classification accuracy during moments of change, as analysed in Chapter 5, is jeopardised by times of stability in the model. We are afraid that this analysis cannot be performed at this stage in this chapter since we are not aware of the real ground truth regarding structural breaks in these data streams.

As a final approximation of what an infinite data stream could be in the financial domain, in Experiment II, we will compare all algorithms for a period of 10 years. Led by the results of Experiment I, this will be done at the 10-minutes level and using naive Bayes as the base classifier.

6.3.2 Experiment II: 10 Years of Intraday Data

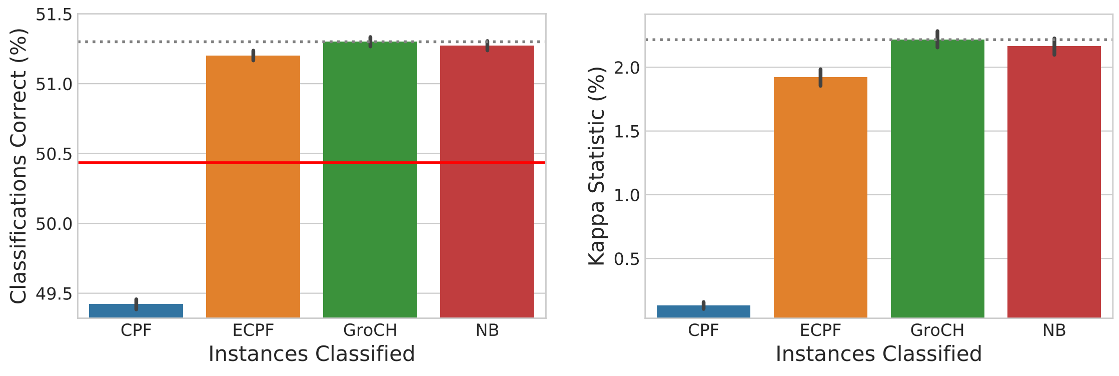
Figure 6.12 showed that in Experiment I, models tend to beat their baseline (class distribution) between the 15 and the 5-minutes level. Models also report the highest mean kappa statistic between the 10 and 1-minute level frequencies. This could be due because the signal-to-noise ratio of this intraday frequency being better for the current periods [138]. 10-minutes seems to be the only frequency where all learners overcome their baselines. Hence, in Experiment II, this is the level of data that we will use. We will also focus on naive Bayes as the base classifier since the kappa statistics obtained with NB in Experiment I are higher across meta-learners.

Algo	Acc. (%)	Kappa (%)	K.Temp. (%)	Kappa M (%)	Time (CPU s)	Cost
GroCH	51.3001	2.2162	5.2374	-0.1698	7868.3	0.082269140
NB	51.2734	2.1645	5.1862	-0.2244	3.9	0.000000008
ECPF	51.2005	1.9240	5.0551	-0.3734	54.6	0.000011598
CPF	49.4211	0.1310	1.6103	-4.0522	11299.1	0.217060600

Table 6.7: *Mean results in Experiment II. Acc: accuracy; K.Temp: kappa temporal statistic; Cost expressed in RAM-hours.*

Table 6.7 summarises the mean results of Experiment II. Algorithms are sorted in descending order by classification performance. As for Experiment I, the results show the overall accuracy of GroCH, because the ground truth for this set is unknown.

We cannot perform a significance analysis as in Experiment I, but these learners give deterministic results. Thus we believe that GroCH could offer a better classification performance than the other algorithms used in this chapter for this long period, but we cannot confirm this statistically with a population of experiments.



(a) *Classification accuracy* **(b)** *Kappa statistics*

Figure 6.13: *Classification accuracy and kappa statistics across classifiers in Experiment II.*

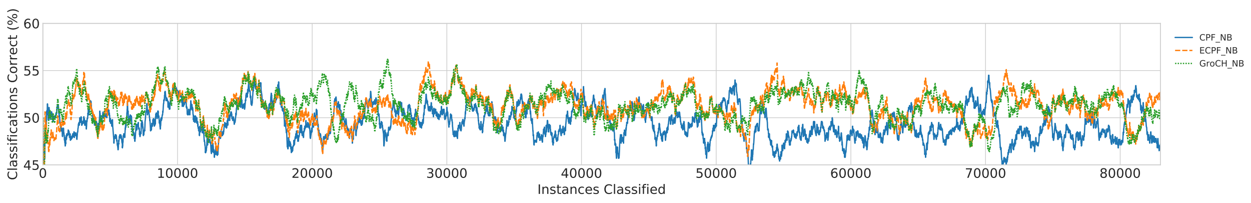


Figure 6.14: *Classification accuracy of meta-learners over time in Experiment II. GroCH, ECPF and CPF in green, orange and blue, respectively.*

All meta-classifiers obtain greater accuracy than the base learner (NB), and GroCH seems to be the algorithm obtaining the greatest classification performance for accuracy and kappa statistics. This can be seen more closely in Figure 6.13, which shows both metrics for each of the meta-learners and naive Bayes (base classifier) against the classification baseline (red line). Figure 6.14 shows the performance of the three meta-learners during the experiment and, while GroCH seems to be the best overall performer in terms of classification accuracy, there are times where ECPF overperforms this.

Regarding the model's cost, as shown in Table 6.7, CPF is the model with the highest computational cost, followed by GroCH. ECPF proofs to have a lower cost in terms of memory and a lower running time in Experiment II.

As for other experiments, naive Bayes obtains comparable results with lower computational costs. We believe that this is due to the high degree of noise in this domain.

GroCH is a novel approach and offers promising results since in some scenarios it compares or improves other meta-learners from the state of the art. Different approaches to clean the data or improve the detection of changes in the financial domain still need to be approached as future lines of work.

Even if GroCH and the other meta-learners improved the accuracy of NB during moments of change with these real datasets from the financial domain, it would not be measurable due to the lack of knowledge about ground truth changes. The longest the periods or greatest the number of examples, the most likely the obtained accuracy metrics could end converging with the results obtained by the non-adaptive algorithm naive Bayes due to the potential presence of recurrences in this domain and the potential prevalence of stability periods over time. However, all this depends on the definition of concept drift in the financial field, which could be the focus of a future research piece by itself.

6.4 Summary

In this chapter, we wanted to benchmark the meta-algorithm proposed in Chapter 5 for stock trend classification in real-world financial data. This was introduced in Section 6.1. Section 6.2 explained the classification problem, the research data and the experimental protocol. As part of this protocol, we decided to split the fourth experiment of this thesis into two sub-experiments: one experiment where different frequency levels are addressed to find the frequency level with the best signal, and a second experiment running the algorithms over a long period of time.

Section 6.3 analysed the experimental results of this chapter. Results obtained in Experiment I concluded that these meta-learners are not suited for the current selection of yearly periods at the minute levels or quarterly periods at the second level. Then, in Experiment II, GroCH obtained the best classification performance for a single long period.

Clustering financial data streams is a subject that presents numerous challenges due to symbol liquidity, large amounts of data, the presence of unbalanced data at different points in time, concept drifts, and a high degree of noise [54, 347].

Identifying good quality clusters to leverage from GroCH in a real-world dataset highly depends on the level of signal shown by different technical indicators used as feature sets and their evolution over time. There may be periods in different symbols with a clear trend that a machine learning model, either supervised or non-supervised, could learn. A major challenge is learning continuously and adapting to all these changes as data stream mining algorithms.

GroCH demonstrates some drawbacks as it relies on the choice of the pre-training sets and, in general, it has a higher computational cost than ECPF. We feel that this experiment has not been the right test-bed for GroCH since we are not aware of the ground truth regarding changes in the generative process of the data streams used. While Chapter 5 demonstrates that GroCH can beat its competitors during changes in the ground truth, we believe that this novel approach is still a work in progress due to the high degree of noise in these streams.

Different methods may need to be used to clean the streams. For instance, a model to recognise a higher degree of noise could be a valuable future line of work since different actions could be triggered based on this. Also, GroCH is a valuable technique for other types of data streams exhibiting recurring patterns, and its application to other domains is future research. In any case, different approaches that we consider out of the scope intended for this thesis will need to be revisited to reduce the cost of the model over time as covered in Chapter 5.

This page is intentionally left blank.

Chapter 7

Conclusion and Future Work

In the last years, many experts have tried to attack the problem of stock market price prediction using different AI methods [179]. In this thesis, we wanted to focus on predictions under structural changes in the financial domain. The most recent literature [347], introduced from Chapter 1, suggests the use of conventional machine learning and statistical approaches to analyse structural breaks and RCs. As covered in Chapters 2 and 3, financial time series are a type of data stream. Albeit the data stream mining literature has not been widely covered for forecasting in this field. In this thesis, we have introduced techniques from the literature of data stream mining that can be used for stocks market classification in the financial domain. We have proposed to use these techniques to deal with the intrinsic changes of this type of non-stationary streams and approached some extra challenges in these, such as scalability bottlenecks in continuous machine learning scenarios.

In this regard, Chapter 4 performed the first two experiments, which served as preliminary work to our main proposal. The first experiment, in Section 4.1, focused on improving scalability in data stream mining scenarios. We compared the accuracy and runtime performance of incremental against batch machine learning techniques. An adaptive learning framework named iGNGSVM using prototype generation techniques was proposed. Two different versions of this algorithm were presented to suit different speeds of change in non-stationary data. The results obtained reported a significant computational performance improvement in iGNGSVM against static algorithms for comparable classification accuracies. The second experiment of Chapter 4 was focused on a real-world financial application for adaptive learning algorithms. Section 4.2 proposed RCARF, a version of the random forest classifier to handle recurring concepts explicitly. Ensembles like RF are known in the literature for their excellent results predicting during cyclic and non-stationary scenarios as stock market prices (see Chapter 3). RCARF overperformed most of the data stream mining literature algorithms classifying price movement direction in the SPDR S&P 500 Trust ETF (SPY), proving that the recurring concepts mechanism introduced is helpful in this domain.

Since the use of data stream mining approaches is not common in computational finance, the lack of resources, available benchmarks and datasets where the ground truth regarding drifts were available made us design our framework to generate synthetic data in Chapter 5. In Chapter 5, a meta-algorithm algorithm called GroCH was proposed to handle different types of concept drifts in data streams. The main goal of this algorithm was to improve classification accuracy during times of change in time series of various natures, such as ETF price trends changing over time. Chapter 5 accomplished the third experiment of this thesis that consisted of applying the algorithm mentioned above to predict changes in market states and assigning different adaptive classifiers to predict ups and stable/down movements during each market state. In the synthetic sets produced, GroCH outperformed its main competitors in terms of kappa statistics during RCs. An in-depth analysis of the misdetection of drifts was performed at the market state level. Part of this error was attributed to the lack of signal on the market states selected.

In Chapter 6, we validated GroCH for stock trend classification using real-world financial data. For this purpose, we used intraday data from the SPY at different frequencies. GroCH obtained comparable results to its main competitors in the state of the art of data stream mining and overperformed them at an individual test for a continuous data stream of 10 years of market prices at the 10-minutes level.

The main focus of this thesis has been stock trend prediction with adaptation to concept drift. As mentioned above, our research focused on data stream mining algorithms and was not intended to derive any trading system. Implementing such a system may need reframing the classification problem to consider extra variables to discriminate the direction of price changes and their magnitude. The different algorithms proposed in this thesis have been designed to predict short-term market trends to a certain point. How to profitably exploit market regularities is yet to be determined, and we consider it out of the scope in this thesis. While it is clear that the results are compatible with arguments against the EMH, we cannot claim that we can beat consistently buy and hold and, therefore, we cannot reject it.

As a result of the experiments in this thesis, our main proposal, GroCH, has proven to improve classification accuracy under times of change and obtain comparable results to other algorithms from the literature in terms of overall predictive performance. We believe that GroCH opens good research prospects for financial forecasting during times of market instability and structural breaks. We consider to have validated initial working hypotheses and achieved the main research goals of this thesis. These will be summarised in Sections 7.1 and 7.2 respectively. After these, Section 7.3 discusses future work in the research line opened in this thesis.

7.1 Validation of Working Hypotheses

This research work was carried out under a set of working hypotheses described in Section 1.3. The following ones have been validated during our experiments.

> *(1) Traditional static machine learning techniques do not scale and thus are not suitable for high frequency and non-stationary data that needs models to be up to date with the latest trends.*

In Chapters 2 and 3 we covered the principles and related research claiming that static machine learning models need adaption mechanisms to scale for continuous learning scenarios. In the first experiment of this thesis, in Section 4.1, we have demonstrated how an adaptive framework such as iGNGSVM can help to scale a computationally costly static machine learning algorithm. On the one hand, the usage of adaptive mechanisms does not guarantee that the model will be stable over time. This depends on the sharpness of the changes of the underlying data stream. It also depends on the forgetting or reuse mechanisms developed in the model. On the other hand, improving the scalability of a model does not imply that this will perform at any frequency level; higher frequencies will need more rapid responses and model updates. In any case, for these reasons, the online incremental machine learning paradigm appears as a good defacto choice to build continual learning models.

> *(2) Adaptive and incremental machine learning algorithms allow learning in near-real-time. Thus, predictions can be made with a model trained with the most recent data, avoiding extra computational costs or bottlenecks.*

We validated this hypothesis in the second experiment by proposing a system that dealt with concept drifts both actively and passively in Section 4.2. The proposed system ensembled many base classifiers to deal with the complex behaviour of financial time series, being still able to perform in near real-time. We also proved that the mechanism to detect recurrences is helpful for classifying price movement direction at minute level resolutions in the SPY Exchange-Traded Fund.

> *(3) It is possible to measure and typify market states (regimes) in intraday financial data. It is also possible to model these states and simulate a scenario where the ground truth is known.*

We validated this hypothesis in the third experiment, in this thesis's main proposal. GroCH was suggested as a meta-learner with a history of previous classifiers that are grouped in a non-supervised fashion. The underlying idea was to create groups representing market states. To validate this, we proposed a framework in Chapter 5 where different ARMA-GARCH processes were fit to different market behaviours from different ETFs. This was based on underlying research claiming that market regimes can be differentiated from each other by the volatility of the price returns [347]. The results obtained by GroCH in Chapter 5 proved that it was possible to cluster market states and recognise them in a simulated setting.

> *(4) The use of adaptive techniques will improve prediction accuracy, especially during concept drifts or changes in the underlying high-frequency data.*

This hypothesis was validated in the third experiment, with a semi-synthetic dataset that simulated the priority of high-frequency data from four ETFs. We benchmarked GroCH to the state of the art using different metrics such as kappa statistics and classification accuracy during times of change in the ground truth. The was empirical evidence about GroCH being able to improve classification accuracy during concept drifts and obtain comparable results to other algorithms from the literature in terms of overall predictive performance. GroCH was also tested with real-world high-frequency data in the fourth experiment of this thesis. The use of adaptive approaches obtained higher overall accuracies (during times of stability) with statistic significance compared to other non-adaptive incremental methods (e.g. incremental naive Bayes) in a wide battery of tests.

7.2 Validation of Objectives

After the completion of this thesis, we consider achieved the following set of initial research goals:

> *(1) Development of new machine learning algorithms to improve the state of the art regarding forecasting of price trend in financial markets. This development should be driven by the combination of the study of the state of the art and the experimental work performed in this thesis.*

As mentioned at the start of this section, in this thesis, we have created two different algorithms that have been applied to the aforementioned domain. RCARF and GroCH were proposed in Section 4.2 and Chapter 5 respectively. Both algorithms obtained competitive results with state-of-the-art data stream mining approaches and overperformed many of their competitors in individual tests. Thus, we consider this research goal accomplished.

> *(2) Modification of techniques from the relevant literature to predict structural change in high-frequency data. Different techniques based on online incremental machine learning for data streams are reviewed for this purpose.*

In Chapters 4 and 5, we proposed RCARF and GroCH as online incremental learning algorithms with explicit concept drift handling mechanisms. As a novelty in our research, these drift detection methods were applied to predict structural change in the financial domain at high frequencies in synthetic and real-world data.

> *(2.1) Detection of RCs through concept drift detection techniques.*

As mentioned above, this objective was achieved in both Chapters 4 and 5.

> *(2.2) Once these structural breaks are detected, our next objective is to find the structural patterns in the market state dynamics to detect recurrences. Our research's primer objective is generating specialised models for scenarios, such as more volatile markets, or for up or downtrends may be more effective.*

This was achieved in our main proposal, GroCH (see Chapter 5). After the detection of structural breaks caused by RCs, market states are summarised using a non-supervised approach and receive the name of groups. Groups are stored by GroCH for future reference at the concept history.

> *(2.3) Our final goal in this regard is to reuse effectively previously trained models when a recurring market state is detected.*

This was also achieved in GroCH in Chapter 5. When a concept drift is recognised, the closest market state is recognised as a recurrence, and a classifier is retrieved from its group at the concept history.

> *(3) Adaption of algorithms and creation of methodologies for the identification of market states, which involves modelling and simulation of these high-frequency time series to adapt and react to structural changes.*

The algorithms mentioned above, RCARF and GroCH, were proposed for this purpose. This was achieved by presenting an approach to model and simulate high-frequency time series with structural changes and recurrences in Chapter 5 using ARMA-GARCH models and sigmoidal transitions between market states inspired by the work by Shaker and Hüllermeier [312].

> *(4) Application of the techniques proposed to the financial domain in the intraday market and prediction using high-frequency price series.*

This objective was also achieved. During this thesis, we applied different approaches from the state of the art in computational finance. For instance, the methods used to represent the datasets used for machine learning tasks, the time series techniques used to simulate synthetic series of price returns, or even techniques to visualise the differences between market states in Section 5.2.4.

7.3 Future Work

Data stream mining is an emerging research field, and its application to computational finance and, more specifically, the detection of RCs leaving behind the time series literature is a novel approach. For this reason, below, we present future research lines in different directions.

First of all, data stream mining algorithms must be scalable to deal with continual learning tasks avoiding performance bottlenecks. In this regard, in Chapter 4 we have shown how online incremental learning methods can help to reduce computational costs. While the results obtained are promising, future work parallelising training tasks in approaches such as iGNGSVM in Section 4.1 can bring significant gains in terms of execution time when learning in near real-time. Section 4.1.6 covered some of the most relevant future lines of work for iGNGSVM; the first approach presented. Scaling the algorithms presented in this thesis further will allow us to use them at higher frequencies beyond the second level.

A disadvantage of the techniques presented in this thesis is their level of complexity, reflected in a large number of input parameters or the need for pre-training (e.g. previous market states in GroCH). Hence, the use of adaptive thresholds and parameters for the different methods proposed could be another future line of work. These would help: first, optimise the balance between classification performance and computational cost in methods such as GNG, used in iGNGSVM and in GroCH to summarise market states; second, adapt the distance thresholds used to identify what is a relevant market state to be retrieved. A different approach in this regard could be a change detection based approach to swap parameter values, thresholds or other inner components when a concept drift is detected.

Another area that we have not tackled in this thesis is the like-hood of class unbalance at different points in time since these models aim to run for an infinite data stream. The usage of auto-adaptive filtering as part of our approaches to perform sampling when handling unbalanced datasets might be worthwhile.

In this thesis, we have applied different algorithms from the state of the art in data stream mining and adapted our own methods to predict price trend direction at high frequencies. RCARF and GroCH were proposed with this in mind in Chapters 4 and 5 respectively. These techniques were able to compete with other algorithms from the state of the art of data stream mining and to improve their classification performance in individual scenarios like during changes in the ground truth, and obtaining more robust results (less deviation across tests or lower kappa errors than their competitors). Specific future lines of work were proposed for these approaches in Sections 4.2.6 and 6.4 respectively. In any case, none of the algorithms proposed are domain-specific. A future line of work could be their application to online learning scenarios in dynamic systems exhibiting non-stationary behaviours or continuous evolution of the data stream with recurrences over time such as IoT sensor-based systems or cybersecurity.

Another line of work is the simulation of synthetic datasets exhibiting RCs in the financial domain. We were not aware of any other approach in the literature aiming the same during this thesis. Thus, we have presented a novel approach inspired by related research from the data stream mining and computational finance literature. While we are satisfied with the outcomes of our framework to simulate RCs, this can be improved in future work in various ways. For instance, other models from the literature of RCs such as regime-switching autoregressive models.

We believe that the application of data stream mining techniques is still a work in progress. There is a high degree of noise in these streams, and different denoising methods may need to be explored. As covered in the previous chapters, clustering financial data streams is a subject that presents numerous challenges. Improving the unsupervised learning of market states depends on the signal to noise ratio displayed by the different technical indicators selected and their evolution over time. While this thesis used a common approach in the literature to produce technical indicators and predict price direction in the next time-step, other methods could be used in this regard. For example, techniques such as *PCA*, *factor analysis*, or *autoencoders* could be used to summarise an initial larger set of technical indicators.

Another major challenge in this regard is the continuous learning setting. Apart from challenges adapting incremental online models, a must is to provide feature sets reflecting signal and changes along concept drifts. Different approaches may also need to be studied for the detection of drifts in the financial domain. A significant piece of research would be to propose a formal definition of concept drift in the financial field, allowing us a degree of confidence about the ground truth to rate the detection of changes and recurrences in real-world data. But this should be solely the focus of a future research piece. Approaches using price returns could perform better in the presence of recurrences at stocks that experiment with significant price scale changes over time.

We are aware that a large amount of work is pending in this new research line and that new challenges will arise as this develops. However, we believe this initial work will contribute to the study of data stream mining methods and their application to price trend prediction and detection of structural breaks in the financial domain.

Appendix A

Our Proposal in Detail

This appendix describes the parameters of our proposal, GroCH, and an explanation of the work performed during parameter tuning and experiments.

A.1 Parameters of the Developed Version

In this section, the complete list of parameters of our main proposal, namely GroCH. GroCH has many base learners, detectors and different components with thresholds that can be specified in advance as described in Chapter 5. Some of them, as the four parameters listed below, are common parameters of many meta-learners in MOA and thus, we have kept the same names and their command line abbreviations (between brackets) listed in other algorithms.

1. *Base learner* (-l): classifier to train. Vanilla setup of the base classifier by default.
2. *Evaluator* (-f): classification performance evaluation method in each base classifier for voting. BasicClassificationPerformanceEvaluator by default; this is an evaluator that takes into account the mean error since the initialisation of the current algorithm.
3. *Drift detection method* (-x): change detector for drifts and its parameters. By default, ADWINChangeDetector -a 1.0E-5 (as in ARF[F] in Section 4.2).
4. *Warning detection method* (-p): change detector for warnings (start training bkg. learner). By default ADWINChangeDetector -a 1.0E-4 (as in ARF[F]).

Parameter	Abbr.	Description
saveClassifiersOnFalseAlarm	m	Should the algorithm save the classifiers in the concept history in the case of false alarms?
windowResizePolicy	W	-1 (default) to consider all examples during the warning window (window always growing).
eventsLogFile	e	File path to export events as warnings and drifts.
disableEventsLogFile	g	Export event logs. If disabled, then events such as warnings or drifts detected are not logged (default: enabled).
eventsLogFileLevel	h	0 only logs drifts, 1 (default) logs drifts and warnings, 2 logs every data example.
warningWindowSizeThreshold	i	Threshold for warning window size that disables a warning (default: 500).
minInsertionThreshold	I	An active classifier should have been trained with at least this number of examples to be inserted in the CH (default: 0).
maxGroupSize	M	Max number of classifiers allowed in a group of the history (default: 1).
periodBetweenDrifts	O	Maximum duration of a concept. (default: inf).
insertionPeriod	N	Max size of the environments in CH (default: inf).
alwaysInsertClassifiers	A	Should the classifiers be inserted in a group even if the retrieval is from the same group and the retrieved performs better?
groupReplacementPolicy	P	Policy to replace a classifier from the concept history group when this reaches its maximum size (LUFO or FIFO) (default: FIFO).
priorityOfRecurringClassifiers	G	Policy to replace a classifier from the concept history group when this reaches its maximum size (New or Old) (default: Old).
topologyRadius	t	Max distance allowed between topologies to be considered part of the same group (default: 1).
distanceMetric	s	Distance metric for concept similarity. This can be Mahalanobis or Euclidean (default: Euclidean).
numberOfMatrices	v	Should we have a (Mahalanobis) matrix per concept or a single matrix (single) to scale distances?
minTopologySizeForDrift	a	Minimum number of prototypes created before allowing a drift (default: 1).
minWSizeForDrift	w	Minimum number of instances in warning window W before allowing a drift (default: same as minimum window size).
updateGroupTopologies	j	Should the topologies of groups be updated when inserting a new classifier into them? If disabled (default), these will not be updated.
topologyAttributeSubset	k	Subset of features selected to represent group concepts in the CH. as a comma-separated list (default: -1).
disableTopologyLearner	q	Should the learner for topology summaries be disabled?
trainOnlineFlag	H	Should the topology be trained on the go or only when a drift is detected?
topologyLearner	c	Prototype generation algorithm. (default: GNG.class -l 50 -m 200 -a 0.5 -d 0.995 -e 0.2 -n 0.006 -c inf -b).
multiPassTopologyTraining	y	Should the topology training be one-pass, or to have training epochs until a stopping criterion is met? (instances only feed once)
multiCluster	Y	If selected, all groups with a topology inside the distance threshold are considered for concept similarity in a retrieval.
insertExamplesOnly	Q	If selected, groups will have a single classifier only and will send their training instances to it.
initialiseConceptHistory	n	The concept history is initialised with pre-defined sets.
arrfsToinitialiseCH	o	ARFF files with an identical feature set to main dataset (pre-training set)
trainAfterDriftEvaluation	B	If enabled, it evaluates drifts as a predictive performance decrease (examples will not be fed to learner before drift detection).
testWithBKGOnWarn	K	If enabled, background learners will be used for testing rather than active ones during warnings (suitable for very sharp drifts).
insertOnEarlyDrift	J	If enabled, the active classifier will be inserted in case of early drift.
weightedTestsOnWarn	L	If enabled (default), both active and background learners will be tested (using a weighting mechanism).
beta	C	Punishing factor for misclassification at each classifier competing during the warning window (default: 0.5).
trainActiveOnWarn	T	If enabled, the active classifier is trained during warning too (this was enabled in the initial version of GroCH)
forceEarlyDrifts	F	If enabled, it forces background drifts at early (sharpest) drifts (type 1 only: when the warning window is too small).

Table A.1: *Parameters in the implementation and design of GroCH.*

Besides these, GroCH has several parameters that serve different functions and reproduce different experiments that we will mention later in this appendix. These parameters are listed in Table A.1, which also shows relevant default parameter values. In Chapters 5 and 6, and in this appendix, any parameter not mentioned explicitly has used its default values.

A.2 Parameter Tuning in GroCH

In this section, a discussion about parameter tuning in GroCH. This parameter tuning process refers to Stages I and II of the analysis performed in Chapter 5, and during the validation of GroCH at the development stage. The optimisation performed in different algorithms in Chapter 5 was conducted over the ranges specified in Table A.2.

Algorithm	Parameters to optimise:
GroCH	Lambda and maximum age GNG (-l): [1-300] and [50-500] Topology radius: [0.25-6] Drift detector: All applicable (see Stage I in Chapter 5) Group maximum size: [1-15] Base classifier: NB, HT
CPF	Base classifier (same as GroCH) Drift detector: All applicable (see Stage I in Chapter 5) Fading: yes/no Pool size (f): 4, 15 Similarity (m): 0.85,0.9,0.925,0.95,0.975, 0.99 Minimum buffer size: 30,60, 120, 180, 240
ECPF	Same as CPF except from min buffer size (not existent in ECPF).
HT	Default in MOA.
HAT	Default in MOA.
RCD	Base classifier (s): same as GroCH Significance value (s) - similarity threshold as a p-value: [0.01, 0.05] Max. instances to represent each distribution (b): [100 - 500] Frequency of tests (t): [b, 500] instances. N. Nearest neighbours (k): 1, 3, 5, 7 Drift detection method (d): All applicable
DWM	Base classifier (s): Same as GroCH. Period (p): 1, 25, 50, 75, 100 Drift confidence (b): 0.25, 0.5, 0.75 Punishing factor (theta): 0.005, 0.01, 0.025, 0.05
ARF Single	ADWIN confidences as in Section 4.2 No bagging of any kind, and only one base classifier

Table A.2: *Ranges of values used in parameter tuning during this thesis.*

Besides the algorithms and ranges shown in Table A.2, during the validation of GroCH, many other parameters were considered, and algorithms that had not been considered for the experimental section of this thesis, such as RCD and DWM, were also benchmarked. Regarding drift detection methods, all drift detectors listed in Table 3.1 in Chapter 3 were evaluated during the development of different pieces of the algorithm. For instance, HDDM$_A$ was the version of HDDM obtaining better results detecting drifts when compared to the ground truth changes of the semi-synthetic dataset used in Stage I. Hence, this was the version of HDDM used in the rest of the thesis.

In order to rerun the experiments or parameter tuning performed in this thesis, a set of scripts can be found in the following GitHub repository: https://github.com/cetrulin/groch-moa/tree/main/experiments. The file *README.md* in the root of the repository outlines the detailed setup process and usage of the algorithms.

A.2.1 Parameters for Concept Similarity

Three key parameters of GroCH that impacted the detection of recurring concepts during our studies were: a) lambda λ (frequency of generation of prototypes); b) the topology radius (distance threshold); and c) the drift detectors with its respective sub-parameters.

Figure 5.16 in Chapter 5 illustrated the decrease of quantisation error with greater number of prototypes in GNG. This inspired the selection of lower values of λ to create more prototypes between drifts and achieve a better representation of the ground truth states. There is an increase in the computational cost of GroCH when decreasing this value, as this generates a prototype interpolating the region with the greatest error residual every λ iteration, which has a complexity of O(n^2). We finally selected values of λ on the range of $[5, 30]$. While a $\lambda = 1$ offered the best concept representation, especially during short warning windows, the computational cost of this was unbearable for a data streaming scenario.

The topology radius value depended on different parameters, like the above mentioned λ, or the maximum age of the connections in GNG. Common values explored during parameter tuning for the maximum age of the connections were listed in Table A.2. Another of these parameters is a flag to prepopulate or not the concept history (*initialiseConceptHistory* in Table A.1). During the validation of the algorithm, we noted that the creation of groups with the ground truth states helps during parameter tuning to find the right topology radius to optimise the separation between ground truth states. Hence, the best parameters set was found, and the detection of drifts was more accurate when pre-training GroCH. The most accurate range of values for the topology radius in our studies was $[1, 2]$.

We noticed that, if prepopulating the concept history, we could assign a topology radius closer to 1 and obtain greater accuracies detecting drifts. Conversely (not pre-training), this would penalise the creation of new groups and not all of the ground truth states would be ever created as groups otherwise. In any case, the result of this last part would need a more profound study for each application domain and dataset used. In our case, it was known that one of the states was not completely separable from the other three, as shown in Section 5.2.4.

A last critical parameter impacting the topology radius was the feature set (or subset) used for the unsupervised representation of concepts. As covered in Section 5.3.3.1, two different subsets of technical indicators were selected. The best results in the subset with the broadest number of indicators (*9*) were achieved using a topology radius in the range $[1.5, 2.5]$. The best results in the shortest subset (*6* indicators) were achieved using a topology radius between $[1, 1.75]$. We believe that the root cause of this is that few indicators in the subset with the broadest number of indicators had extreme values closer to 0 or 100. Thus, topologies could be farther away than compared to the shortest subset.

A.2.2 Design of Concept Representations

During the design of GroCH, three different methods to measure the similarity of groups were compared; Euclidean distance distances, a single Mahalanobis distance matrix to scale distances between a group and the data points received during the warning, and an approach to produce correlation matrices during the warning window and compare these matrices to a correlation matrix produced for each group. This last approach has a higher computational cost due to Mahalanobis distance matrices since a new matrix would need to be computed for every comparison. The approach with the single Mahalanobis distance matrix obtained lower errors classifying drifts than the approach with Euclidean distance , especially in cases where the stock price was very variable in the data stream.

Another approach evaluated during the design of GroCH was the training methodology of concept history groups. In this regard, we initially considered many variants for GroCH that are supported as parameters in the implementation listed in Table A.1:

- *Update of group topologies*: one of these was a method that we designed to replace the set of prototypes in a group at the time of insertion. In some scenarios, this jeopardised prepopulated CH groups when not recognising a drift or a recurrence properly. In any case, this could have been an artefact of the ground truth states used in our semi-synthetic sets (as covered in Section 5.2.4) and may deserve a more in-depth analysis in future work.
- *Multi-pass training of GNG*: another was a multi-pass instead of a single-pass training for GNG. Multiple passes (epochs) could help GNG to have a more accurate concept representation during a warning window. However, this design added extra challenges like selecting the stopping criterion of GNG. As covered in Subsection 5.1.2.2, we used the detection of a drift to stop in the single-pass approach. When exploring the multi-pass approach, we used two different stopping criteria.

  1. Number of prototypes: halt training when generating a number of prototypes equal to a percentage (threshold) of the examples received during the warning window
  2. Quantisation error: stop training when the quantisation error stops decreasing across iterations (beyond a threshold).

  The multi-pass approach was discarded as it increased the computational cost of GNG, not being able to be trained online like with the single-pass approach.

A.2.3 Definition of the Default Parameter Values

To test and validate the parameter space covered in this subsection, we have run different experiments with the synthetic set presented in Section 5.2.2.2.

Here the reader must note that each individual set of data representing a market state in our experiments has been extracted from their first appearance in the semi-synthetic time-series simulated.

As stated in Subsection 5.2.4, the use of raw prices would cause issues in the algorithm. One of these reasons could be the different scale of the close prices between the raw and reconstructed synthetic-series. The appearance of states 2, 3 and 4 come following a recovery phase due to their ground truth changes. We removed instances from that recovery phase to allow a clean generative process for each state. We produced a feature set of technical indicators as explained in Chapter 5.

At this point, we performed an analysis on the correlation between metrics such as global classification accuracy and percentage of true positives in drifts detected, groups accurately recognised in retrievals and insertions and the final number of groups created in the concept history. Figure A.1 shows a positive correlation between the true positives detecting drifts and the global classification of GroCH. While this correlation is not as strong for retrievals and insertions, we believe that it could be due to cascade effects in the generation process of the concept history groups. For instance, a retrieval of the right group may not improve the accuracy of GroCH if the classifier retrieved was inserted incorrectly before into that group.

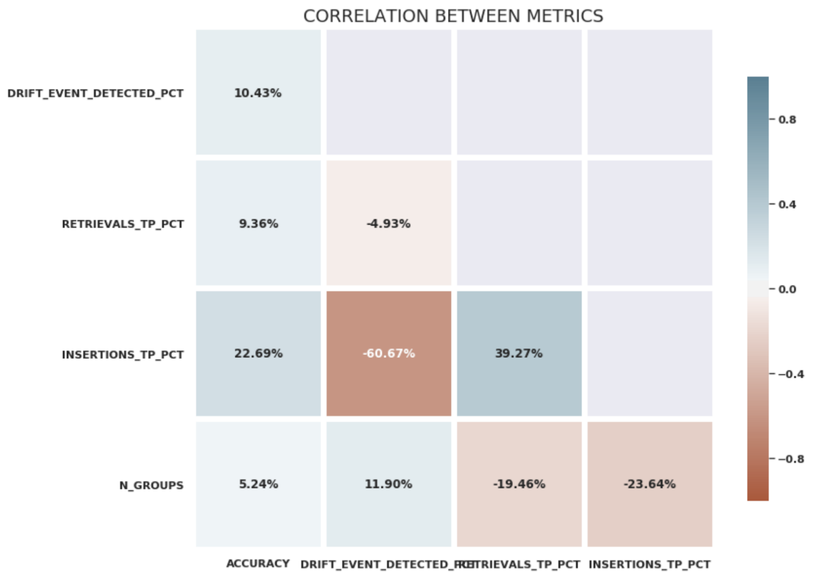


Figure A.1: *Correlation of metrics across experiments in GroCH during Stage I.*

Another analysis on the recognition of drift-related events was performed over the development set during Stages I and II of the experiments, as presented in Section 5.3.5. Different parameter values were selected to maximise two metrics: a) the true positives detecting drift and b) the classification accuracy during periods of change.

Different drift detectors from the literature were used for this purpose. However, EDDM, HDDM_A and RDDM were the methods detecting most of the ground truth changes with a reasonable delay from their start point (*1,500* data instances), followed by very sensitive parametrisations of ADDM (e.g. greater confidence values than ADWIN in ARF[F] in Chapter 4.2). To set a value for maximum allowed delay (d) in GroCH, different values were explored $[500, 2, 500]$ instances.

Another detector used in the experiments was EDDM, a drift detector targeted to gradual drifts (as explained in Chapter 2) that measures the distance of two misclassifications in terms of classification accuracy. However, during our experiments, we experienced that once the base classifier of any meta-learner had been trained with enough data instances, its classification accuracy stabilised. Thus, EDDM barely detected drifts after this point. This was the case especially using HT as base classifier during the final experiments of Chapter 5.

The selection of values in Stage I of the experiments was shown in Table 5.4. We observed that the choice of policies for the management of groups using the *age* of a classifier (number of instances received since its creation) helped to achieve a good trade-off between classifiers keeping previous relevant knowledge and novelty. The primary metric to remove a classifier from the concept history is its classification accuracy. However, we observed that the majority of the warning windows tend to last less than 100 data instances. Classifiers inside the same group are more likely to forecast similarly under a concise sample of data and, therefore, there was a need for a second method to prioritise classifiers in case of a tie.

The length of a warning window depends on the duration of a change and the delay of the detector. The latter is based on the robustness of the classifier and the sharpness of the change (how different both generative processes).

A.2.4 Other Experiments

Other experiments that may be worth a mention were: a) the use of a more comprehensive set of technical indicators; b) the use of different drift detectors and internal window sizes for the internal evaluator in the preliminary study in Section 4.2.

The first approach was explored while reviewing the literature for a standard set of technical indicators. In this regard, we used data for 2017 in the SPY at the one-minute resolution. We looked at the mutual information between different indicators to extract a set of uncorrelated features from the complete list of indicators in TA-Lib. Although promising, this approach did not improve the predictive accuracy of the approach later used for data pre-processing, based on the work from Kara et al. [195].

The second approach was evaluated to improve the selection of recurring classifiers in RCARF. An initial evaluation was performed over a variant of adaptive random forest using different confidence intervals in ADWIN. We called this ARF$_{Heterogeneous}$, visible in Figure A.2. The first drift detected is represented with a grey dotted vertical line. Figure A.2 shows the deviation of the error over time in ARF in these variants (lines black and red) using default parameters (orange and blue lines). This figure reports the error averaging time windows instances as seen previously in this thesis. The red line represents a version of ARF where the detectors of all base classifiers have different confidence intervals. In contrast, the black line represents a version of ARF where there are two different settings for ADWIN across base classifiers.

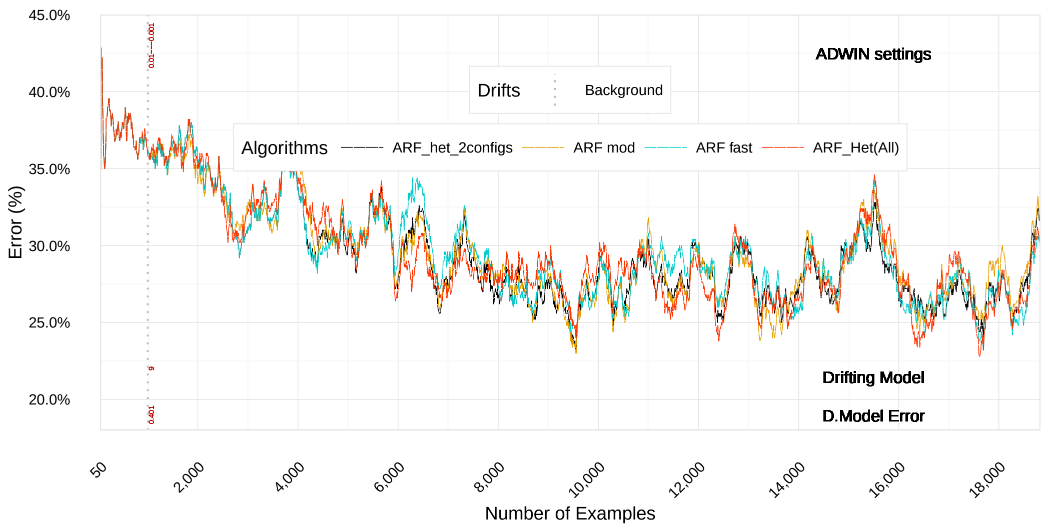


Figure A.2: *Evaluation of several versions of ARF varying drift detection settings in the internal evaluator.*

The new variants of ARF reduced the classification error marginally. Regarding different window sizes in RCARF across base classifiers, the improvements in classification accuracy were not significant. In any case, this future line was finally discarded since the dynamic internal evaluator was finally replaced in GroCH to simplify the solution. The usage of all the data instances during the warning window in GroCH did not negatively impact the detection of recurrences during the initial validation phase and helped streamline the logic of insertions and retrievals.

A.3 Detailed Results for GroCH

In this section, we provide the full tables of results from the experiments in Chapter 5. The detailed experimentation process can be found in Section 5.3.

A.3.1 Drift and Group Recognition

Summary results regarding the recognition of concept drifts at the right time in the ground truth and retrieving and inserting the right state in the concept history. To reduce the headers of the tables and fit these on a page, we have abbreviated them. A summary of all the abbreviations is present in Table A.3.

Metric	Abbreviation
Number of ground truth transitions	GT_T
Number of ground truth transitions correctly recognised	GT_C
Number of ground truth transitions incorrectly recognised	GT_I
Number of ground truth transitions recognised with insertions or retrievals	GT_R
Recognition accuracy of a drift-related event	Acc.
Ground truth state	State
Number of retrievals detected	R_T
Number of retrievals to the right state	R_C
Number of retrievals to an incorrect state	R_I
Number of insertions detected	I_T
Number of insertions to the right state	I_C
Number of insertions to an incorrect state	I_I

Table A.3: *Full form of headers from Tables A.4 to A.7.*

As mentioned in Chapter 5, due to the prepopulation of groups the value of some performance measures for drift-related events is equal to zero. These metrics have not been reported in Tables A.10 and A.11. Tables A.4 to A.7 summarise the results recognising concept changes across the 30 synthetic streams.

To state	GT_C	GT_I	GT_T	Acc.
1	541	239	780	69.359
2	656	94	750	87.467
3	562	188	750	74.933
4	565	155	720	78.472

Table A.4: *Transitions recognised by GroCH using NB (mean of all streams).*

Retrieve State	R_C	R_I	R_T	GT_R	GT_T	Acc.
1	281	662	943	223	780	29.799
2	595	599	1194	417	750	49.832
3	524	364	888	333	750	59.009
4	390	578	968	290	720	40.289

Table A.5: *Retrievals by GroCH using NB (mean of all streams).*

Insert State	I_C	I_I	I_T	GT_R	GT_T	Acc.
1	216	464	680	188	750	31.765
2	317	296	613	258	750	51.713
3	827	303	1130	566	780	73.186
4	345	539	884	324	720	39.027

Table A.6: *Insertions by GroCH using NB (mean of all streams).*

To state	GT_C	GT_I	GT_T	Acc.
1	487	293	780	62.436
2	506	244	750	67.467
3	575	175	750	76.667
4	585	135	720	81.250

Table A.7: *Transitions recognised by GroCH using HT (mean of all streams).*

Retrieve State	R_C	R_I	R_T	GT_R	GT_T	Acc.
1	240	494	734	203	780	32.698
2	373	414	787	303	750	47.395
3	570	196	766	390	750	74.413
4	436	429	865	362	720	50.405

Table A.8: *Retrievals by GroCH using HT (mean of all streams).*

Insert State	I_C	I_I	I_T	GT_R	GT_T	Acc.
1	138	321	459	130	750	30.065
2	268	202	470	254	750	57.021
3	811	220	1031	583	780	78.661
4	327	371	698	317	720	46.848

Table A.9: *Insertions by GroCH using HT (mean of all streams).*

	Summary					v1					v2				v3			
S	a_{v1}	$v1_{md}$	$v1_{dd}$	a_{v2}	a_{v3}	T	TP_{GT}	TP_D	FP_D	FN_{GT}	T	TP_{GT}	TN_D	FN_D	T	TP_{GT}	TP_D	FN_D
1	63.27	665	410	54.17	47.62	147	60	93	54	40	72	32	39	34	63	27	30	33
2	60.27	686	417	55.63	52.42	292	82	176	116	18	142	56	79	63	124	49	65	59
3	67.46	594	383	44.33	51.85	169	62	114	55	38	97	34	43	57	81	37	42	39
4	53.45	686	418	45.3	62.24	275	73	147	128	27	117	38	53	72	98	44	61	37
5	55.04	654	416	36	43.24	258	73	142	116	27	125	32	45	89	111	43	48	63
6	46.46	660	435	50.9	44.71	607	90	282	325	10	222	59	113	116	170	59	76	94
7	60.26	617	433	42.35	71.01	156	62	94	62	38	85	32	36	52	69	42	49	20
8	69.74	651	410	42.98	47.52	195	82	136	59	18	114	38	49	66	101	42	48	53
9	55.73	568	413	37.01	62.3	323	88	180	143	12	154	45	57	100	122	58	76	46
10	62.04	592	399	45.32	55.37	274	80	170	104	20	139	49	63	85	121	51	67	54
11	55.76	675	432	61.07	51.54	321	86	179	142	14	149	60	91	59	130	55	67	63
12	56.76	627	418	40.85	51.88	333	86	189	144	14	164	48	67	99	133	50	69	64
13	63.27	650	408	39.05	45.35	196	76	124	72	24	105	33	41	64	86	37	39	47
14	64.9	683	406	44.6	50	83	83	159	86	17	139	49	62	84	112	46	56	56
15	49.37	637	425	28.06	40.52	316	85	156	160	15	139	26	39	111	116	35	47	69
16	47.5	653	429	43.9	52.69	560	91	266	294	9	205	63	90	128	167	64	88	79
17	61.22	638	412	50.94	60.47	196	73	120	76	27	106	44	54	53	86	43	52	34
18	62.17	664	419	36.94	41.94	230	74	143	87	26	111	34	41	71	93	33	39	54
19	66.84	671	433	46.73	63.54	187	72	125	62	28	107	44	50	62	96	50	61	35
20	47.24	652	447	54.39	46.33	616	88	291	325	12	228	62	124	106	177	57	82	95
21	63.93	643	400	41.75	59.3	183	70	117	66	30	103	36	43	65	86	42	51	35
22	66.88	700	406	48.19	51.32	154	66	103	51	34	83	34	40	45	76	36	39	37
23	65.27	686	433	46.88	52.33	167	63	109	58	37	96	37	45	62	86	37	45	41
24	59.6	625	432	33.67	58.57	198	69	118	80	31	98	25	33	76	70	34	41	29
25	56.42	600	393	44.12	62.5	296	83	167	129	17	136	49	60	79	120	58	75	45
26	56.6	655	400	39.8	59.49	212	69	120	92	31	98	29	39	63	79	34	47	32
27	47.79	587	419	30	30.07	542	88	259	283	12	200	26	60	144	153	28	46	107
28	58.01	684	420	50.77	54.13	281	83	163	118	17	130	48	66	69	109	44	59	50
29	46.49	674	419	53.64	48.6	598	93	278	320	7	220	64	118	103	179	61	87	92
30	63.37	638	423	45.87	56.99	202	74	128	74	26	109	37	50	59	93	40	53	40

Table A.10: *Drift detection metrics for the 30 synthetic streams in GroCH using NB. See metrics in Subsection 5.2.1.3. S: Stream; v1$_{dd}$: v1 delay std, v1$_{md}$: v1 delay mean; a: accuracy for v_n; v_T: total number of events recognised (TP_D+FP_D).*

	Summary					v1					v2				v3			
S	a_{v1}	$v1_{md}$	$v1_{dd}$	a_{v2}	a_{v3}	T	TP_{GT}	TP_D	FP_D	FN_{GT}	T	TP_{GT}	TN_D	FN_D	T	TP_{GT}	TP_D	FN_D
1	70.42	706	421	47.56	60	142	66	100	42	34	82	33	39	46	75	35	45	30
2	76.16	738	394	48.96	58.14	151	74	115	36	26	96	44	47	56	86	41	50	36
3	49.39	688	426	50.26	46.58	488	89	241	247	11	189	58	95	99	146	54	68	78
4	65.61	733	394	56.32	77.33	157	65	103	54	35	87	44	49	44	75	49	58	17
5	69.18	732	453	57.65	59.15	146	62	101	45	38	85	41	49	36	71	38	42	29
6	53.68	671	407	48.66	42.86	462	90	248	214	10	187	56	91	105	147	50	63	84
7	71.88	697	392	40.16	52.43	192	82	138	54	18	122	41	49	74	103	43	54	49
8	62.06	674	413	35.37	41.88	282	89	175	107	11	147	39	52	95	117	39	49	68
9	77.7	712	404	54.64	67.42	148	73	115	33	27	97	46	53	47	89	48	60	29
10	71.28	684	423	56.03	61.46	195	79	139	56	21	116	48	65	53	96	47	59	37
11	69.28	695	421	56.32	64	153	70	106	47	30	87	38	49	40	75	42	48	27
12	22.58	934	314	0	50	62	3	14	48	97	11	0	0	14	2	1	1	1
13	66.84	639	396	41.96	60.78	196	84	131	65	16	112	39	47	65	102	47	62	40
14	70.37	745	413	64.1	65.71	135	61	95	40	39	78	38	50	28	70	40	46	24
15	66.98	720	410	55	53.92	215	79	144	71	21	120	45	66	55	102	45	55	47
16	62.07	701	432	51.77	56.64	261	82	162	99	18	141	51	73	72	113	54	64	49
17	63.36	706	414	57.75	63.93	131	62	83	48	38	71	38	41	36	61	36	39	22
18	49.78	666	436	49.73	45.59	448	86	223	225	14	185	52	92	94	136	48	62	74
19	72.99	698	379	58.04	56.7	174	77	127	47	23	112	52	65	47	97	46	55	42
20	71.05	744	394	54.35	64.63	152	74	108	44	26	92	42	50	51	82	47	53	29
21	69.17	727	407	68.6	68.49	133	67	92	41	33	86	50	59	27	73	43	50	23
22	62.08	685	414	39.55	61.39	269	89	167	102	11	134	40	53	85	101	48	62	39
23	78.81	709	401	52.44	63.51	118	63	93	25	37	82	36	43	41	74	38	47	27
24	67.87	695	422	52.8	54.63	221	77	150	71	23	125	50	66	64	108	47	59	49
25	69.93	716	385	50.6	64	143	68	100	43	32	83	39	42	44	75	45	48	27
26	71.2	722	361	61.25	68.06	125	63	89	36	37	80	39	49	33	72	45	49	23
27	70.34	715	391	46.43	54.55	145	70	102	43	30	84	35	39	45	77	38	42	35
28	67.48	708	374	60	68.29	163	80	110	53	20	90	43	54	38	82	45	56	26
29	67.46	694	432	53.25	64.29	126	62	85	41	38	77	38	41	37	70	39	45	25
30	72.41	696	389	54.26	65.43	145	67	105	40	33	94	43	51	52	81	46	53	28

Table A.11: *Drift detection metrics for the 30 synthetic streams in GroCH using HT. See metrics in Subsection 5.2.1.3. S: Stream; v1$_{dd}$: v1 delay std, v1$_{md}$: v1 delay mean; a: accuracy for v_n; v_T: total number of events recognised (TP_D+FP_D).*

A.3.2 Benchmark

In this subsection, a summary of results corresponding to the 500,000 data stream instances from the benchmark of our main proposal (Subsection 5.3.5.2).

Base	Algorithm	Mean	Std. Dev.	Min.	25%	Median (50%)	75%	Max.
HT	CPF	61.1349	1.004547	58.76333	60.62685	61.25149	61.8775	62.64982
	ECPF	61.85033	0.49993	59.54511	61.71605	61.90026	62.11478	62.3718
	GroCH	61.67791	0.327411	60.9491	61.44508	61.69359	61.87466	62.32655
	Base	62.63565	0.337457	61.57368	62.43086	62.72465	62.86994	63.10358
NB	CPF	57.85928	0.32818	57.0121	57.67509	57.90297	58.0419	58.629
	ECPF	59.9815	0.392469	59.30492	59.65941	60.02097	60.24684	60.63879
	GroCH	59.03512	0.307967	58.10907	58.86976	58.99713	59.24424	59.51724
	Base	58.36426	0.266608	57.43728	58.26331	58.39579	58.52195	58.75565

Table A.12: *Summary of classifications correct (percent) in 30 synthetic streams of 500k instances.*

Base	Algorithm	Mean	Std. Dev.	Min.	25%	Median (50%)	75%	Max.
HT	CPF	17.29972	2.813694	13.59709	14.28973	17.83613	19.70067	21.55121
	ECPF	16.6986	0.724736	15.2446	16.23714	16.67991	17.27597	18.20634
	GroCH	16.91344	0.930013	15.5831	16.34927	16.67228	17.35622	20.05844
	Base	21.29493	0.757599	19.24287	20.75327	21.37709	21.81796	22.60449
NB	CPF	14.4864	0.515804	13.27975	14.18814	14.51176	14.83104	15.70606
	ECPF	15.24563	0.54277	13.73823	15.10961	15.38777	15.54797	16.1134
	GroCH	15.18826	0.614319	13.61423	14.79227	15.13843	15.66901	16.35334
	Base	15.89019	0.424558	14.66105	15.76512	15.9485	16.08041	16.70657

Table A.13: *Summary of kappa statistic (percent) in 30 synthetic streams of 500k instances.*

Base	Algorithm	Mean	Std. Dev.	Min.	25%	Median (50%)	75%	Max.
HT	CPF	26.48001	2.18253	21.44015	25.42281	26.77329	28.12576	29.67345
	ECPF	28.11835	1.081054	23.18845	27.83209	28.31853	28.70408	29.32968
	GroCH	27.75184	0.616322	26.50944	27.3788	27.72746	28.02408	28.92478
	Base	29.88821	0.746867	27.63401	29.55821	30.00158	30.51334	30.93085
NB	CPF	19.93903	0.694363	18.3298	19.58103	19.95889	20.35005	21.63672
	ECPF	24.16178	0.758738	22.82757	23.63076	24.22071	24.7872	25.68581
	GroCH	22.45009	0.579084	20.83175	22.14982	22.38784	22.902	23.45185
	Base	20.98263	0.610954	18.89239	20.6708	20.98397	21.35233	21.95983

Table A.14: *Summary of kappa temporal statistic (percent) in 30 synthetic streams of 500k instances.*

Base	Algorithm	Mean	Std. Dev.	Min.	25%	Median (50%)	75%	Max.
HT	CPF	65.32533	0.97268	62.29016	64.97591	65.62045	65.87695	66.76241
	ECPF	66.46007	0.66097	63.3076	66.36693	66.59745	66.77737	67.10042
	GroCH	66.83105	0.267869	66.3576	66.63998	66.75936	67.08623	67.32013
	Base	66.79092	0.402847	65.69726	66.5883	66.73923	67.09791	67.52553
NB	CPF	60.66812	0.58723	59.18958	60.34835	60.68651	61.09239	61.76687
	ECPF	63.77527	0.615368	62.3843	63.29322	63.91858	64.21544	65.01337
	GroCH	63.24164	0.500201	61.77339	63.04694	63.25861	63.5215	64.11524
	Base	61.16945	0.452656	59.99057	60.88282	61.07377	61.55579	61.92891

Table A.15: *Summary of classifications correct under switch (percent) in 30 synthetic streams of 500k instances.*

Base	Algorithm	Mean	Std. Dev.	Min.	25%	Median (50%)	75%	Max.
HT	CPF	26.60035	2.27166	19.66886	25.64289	27.29232	27.74393	29.81834
	ECPF	29.3733	1.511392	22.03342	29.16041	29.65348	30.08372	30.80671
	GroCH	30.16795	0.649967	29.16677	29.7281	30.05301	30.61406	31.46461
	Base	30.05798	1.029353	27.48536	29.48472	29.90909	30.78117	32.19102
NB	CPF	16.31081	1.351973	13.00246	15.69095	16.2862	17.30899	19.11456
	ECPF	23.15143	1.435907	19.93479	21.79007	23.44614	24.2933	25.93638
	GroCH	22.08055	1.085793	19.16965	21.68654	22.17428	22.54105	23.96935
	Base	17.23127	1.055965	14.5344	16.52097	17.13206	17.91567	19.38494

Table A.16: *Summary of kappa temporal statistic under switch (percent) in 30 synthetic streams of 500k instances.*

Base	Algorithm	Mean	Std. Dev.	Min.	25%	Median (50%)	75%	Max.
HT	CPF	22032.3	45640.32	1208.338	1547.941	2235.265	16007.06	203157.5
	ECPF	12848.87	61677.4	686.7325	1131.8	1638.861	2008.94	339396
	GroCH	50175.88	36305.98	27087.66	31836.39	40797.17	53424.89	220578.3
	Base	930.2354	118.4019	761.7896	830.3386	902.3249	977.6034	1247.464
NB	CPF	662.361	626.0211	139.4424	395.558	472.3134	682.6097	3734.519
	ECPF	426.6354	143.4488	237.7504	306.7374	403.3177	496.2689	841.9483
	GroCH	54582.99	45210.96	16805.97	30763.15	43949.7	56869.68	262528.7
	Base	18.89028	2.460772	16.21461	16.96542	18.30425	19.72032	24.73088

Table A.17: *Summary of evaluation time (CPU seconds) in 30 synthetic streams of 500k instances.*

This page is intentionally left blank.

Appendix B

Description of the Datasets Used

In this appendix, we explain in more detail the semi-synthetic datasets generated as part of the experiments in Chapter 5. We explain the ground truth transitions and illustrate each of the artificial datasets created in Stage I and II in Sections B.1 and B.2.

B.1 Summary of Ground Truth Changes in the Synthetic Sets

The generation of the semi-synthetic datasets used in Chapter 5 was explained in Section 5.2.2. All datasets have been created using the same process. Four different ARMA-GARCH models were fit to one time series of price returns each. We generated a dataset of price returns, later reconstructed to prices and technical indicators, by changing the actual generative process over time.

In order to create a controlled testing scenario for GroCH, transitions between generative processes were manually selected. This map of transitions, shared across all synthetic sets, is summarised in Table B.1. It details the transitions between the generative processes (1-4) during the simulation. Changes can have different duration ($\{100, 1,000\}$ instances) and occur every $5,000$ instances.

The transition map from Table B.1 was initially created for 6 million instances, and the order of the changes between generative processes and their duration was randomised. Table B.1 shows the distribution of changes for the first 1.5 million instances since we have used only these as part of our experiments. The first 500k instances were used for pre-training, computation of Mahalanobis distance matrices and parameter tuning, and the last 1 million instances for the prequential evaluation.

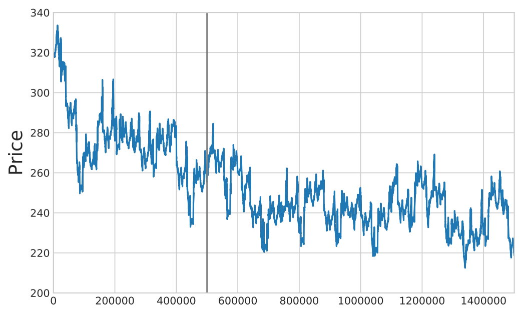
From	To	Duration	Starting in instance # (k represents thousands)
1	2	100	5k, 245k, 485k, 725k, 965k, 1205k, 1445k, 905k, 185k, 425k, 665k, 1145k, 1385k, 605k, 125k, 365k, 845k, 1085k, 1325k, 65k, 305k, 545k, 1025k, 785k, 1265k
	3	100	1355k, 755k, 155k, 1475k, 875k, 275k, 995k, 395k, 1115k, 515k, 35k, 1235k, 635k
		1k	1055k, 455k, 1175k, 575k, 1295k, 695k, 1415k, 815k, 215k, 95k, 935k, 335k
	4	100	1245k, 45k, 405k, 285k, 165k, 525k, 645k, 765k, 885k, 1005k, 1125k, 1365k, 1485k
		1k	105k, 585k, 465k, 225k, 345k, 705k, 825k, 945k, 1065k, 1185k, 1305k, 1425k
2	1	100	1290k, 1410k, 90k, 330k, 210k, 570k, 450k, 690k, 810k, 930k, 1050k, 1170k
		1k	1470k, 30k, 510k, 150k, 390k, 270k, 630k, 750k, 870k, 990k, 1110k, 1230k, 1350k
	3	100	70k, 190k, 550k, 430k, 310k, 670k, 790k, 910k, 1030k, 1150k, 1270k, 1390k
		1k	1210k, 1330k, 10k, 490k, 370k, 250k, 130k, 610k, 730k, 850k, 970k, 1090k, 1450k
	4	1k	1435k, 235k, 415k, 1135k, 655k, 175k, 895k, 1375k, 115k, 835k, 355k, 1075k, 595k, 1195k, 1315k, 475k, 535k, 775k, 55k, 295k, 1015k, 1255k, 1495k, 955k, 715k
3	1	100	100k, 460k, 340k, 220k, 580k, 700k, 820k, 940k, 1060k, 1180k, 1300k, 1420k
		1k	40k, 160k, 520k, 400k, 280k, 640k, 760k, 880k, 1000k, 1120k, 1360k, 1480k, 1240k
	2	100	25k, 265k, 145k, 385k, 505k, 625k, 745k, 865k, 985k, 1105k, 1225k, 1345k, 1465k
		1k	1285k, 85k, 565k, 445k, 325k, 205k, 685k, 805k, 925k, 1045k, 1165k, 1405k
	4	100	1095k, 495k, 1215k, 615k, 1335k, 735k, 135k, 1455k, 855k, 15k, 255k, 975k, 375k
		1k	1395k, 795k, 195k, 915k, 315k, 1035k, 435k, 555k, 1155k, 75k, 1275k, 675k
4	1	100	120k, 480k, 360k, 240k, 600k, 720k, 840k, 960k, 1080k, 1200k, 1320k, 1440k
		1k	60k, 180k, 540k, 300k, 420k, 660k, 780k, 900k, 1020k, 1140k, 1260k, 1380k, 1500k
	2	100	1430k, 110k, 350k, 470k, 230k, 590k, 710k, 830k, 950k, 1070k, 1190k, 1310k
		1k	1250k, 1370k, 50k, 170k, 530k, 410k, 290k, 650k, 770k, 890k, 1010k, 1130k, 1490k
	3	100	80k, 320k, 200k, 560k, 440k, 680k, 800k, 920k, 1040k, 1160k, 1280k, 1400k
		1k	20k, 140k, 500k, 380k, 260k, 620k, 740k, 860k, 980k, 1100k, 1220k, 1340k, 1460k

Table B.1: *Ground truth changes in synthetic sets for 1.5 million examples (300 changes).*

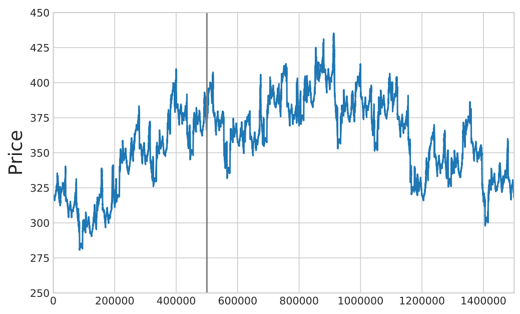
B.2 Synthetic Sets Generated

In order to create a feature set of technical indicators, the series of price returns generated were first reconstructed. The initial price for the reconstruction was chosen using the last price value of the raw series used to fit the first generative process in Section 5.2.2.

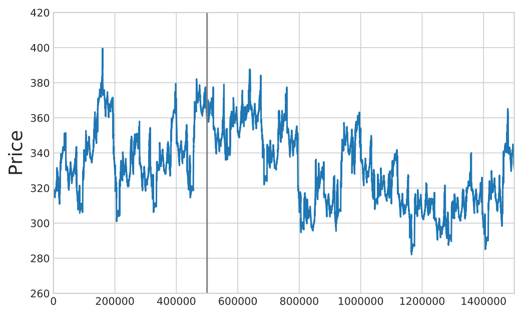
A total of 30 series of prices were reconstructed to be pre-processed and used in Stage II of the study in Chapter 5. All of these are in Figures B.1 to B.4. For Stage I, only the first series, present in Subfigure B.1a, was used. In all figures of this section, the first 500k examples represent instances used for pre-training, computation of Mahalanobis distance matrices and parameter tuning. The last 1 million instances (after a black vertical line) were for test and train in the experiments of Chapter 5.



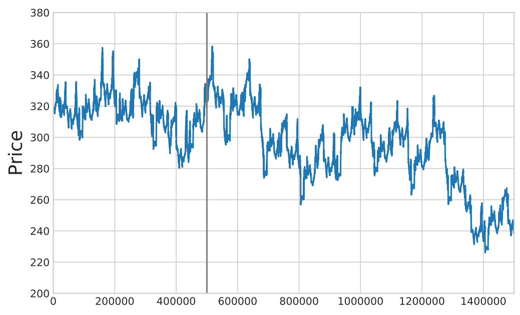
(a)



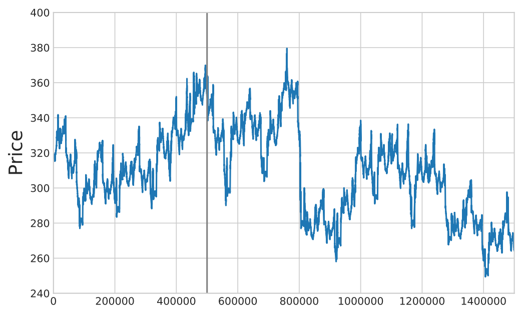
(b)



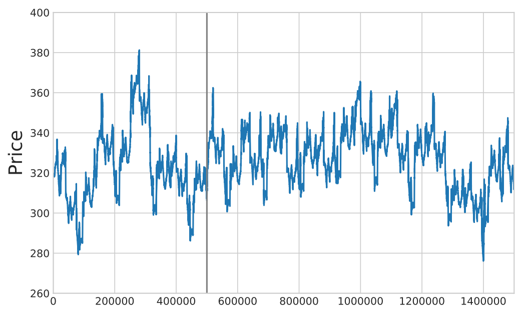
(c)



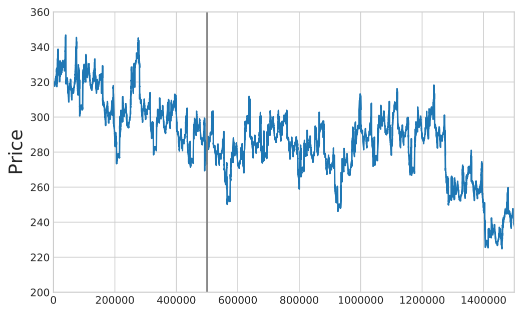
(d)



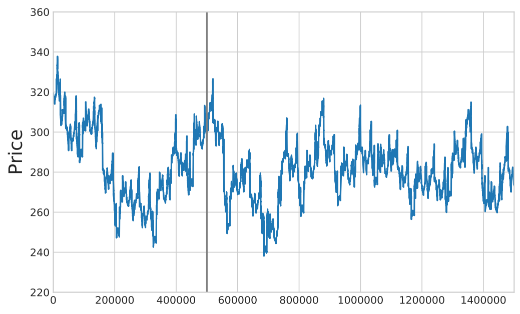
(e)



(f)

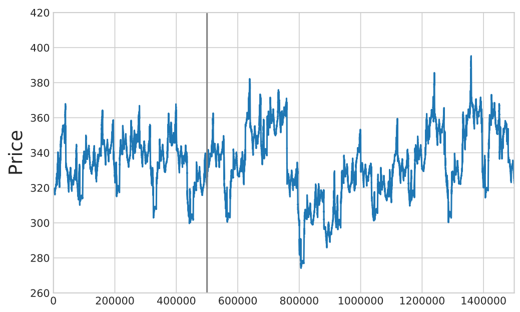


(g)

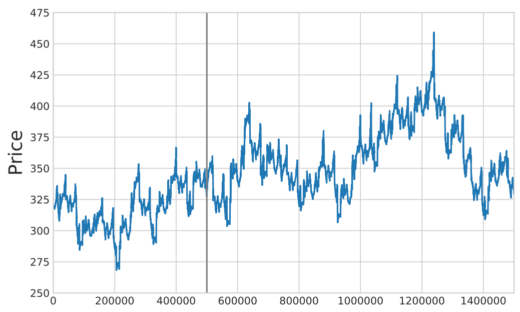


(h)

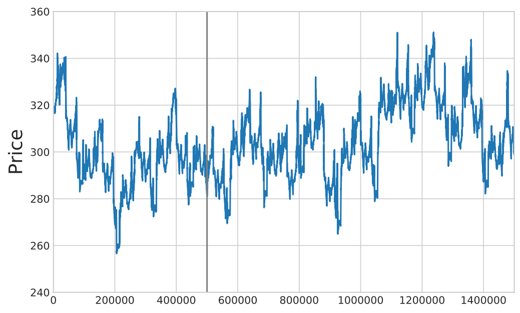
Figure B.1: *Reconstructed close price from synthetic sets 1-8 generated in Chapter 5.*



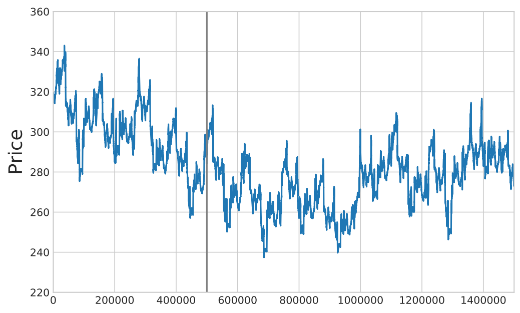
(a)



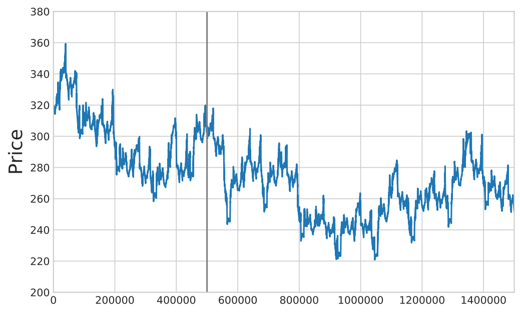
(b)



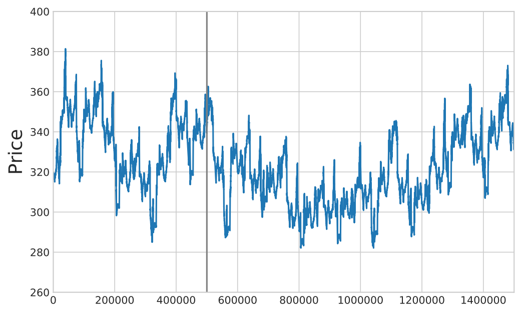
(c)



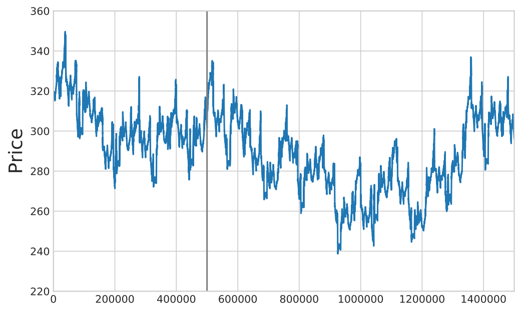
(d)



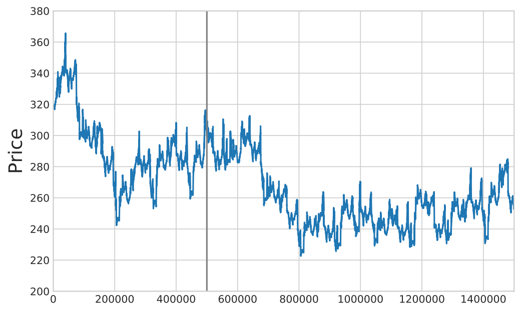
(e)



(f)

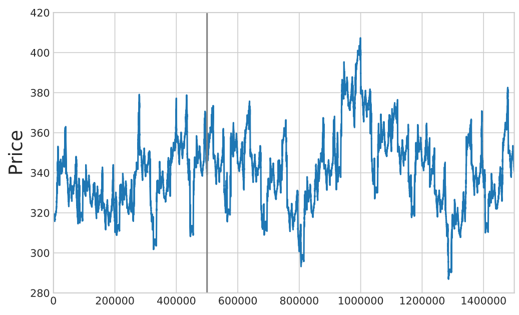


(g)

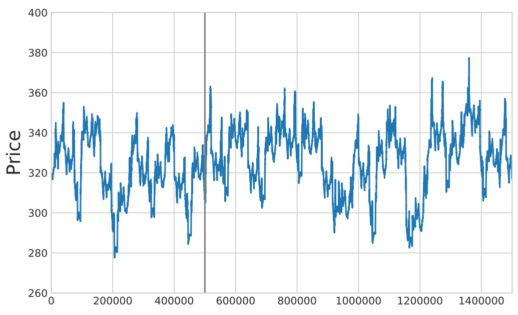


(h)

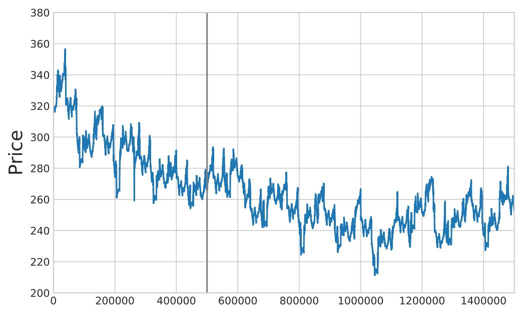
Figure B.2: *Reconstructed close price from synthetic sets 9-16 generated in Chapter 5.*



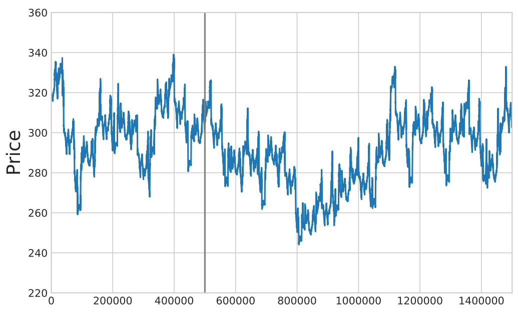
(a)



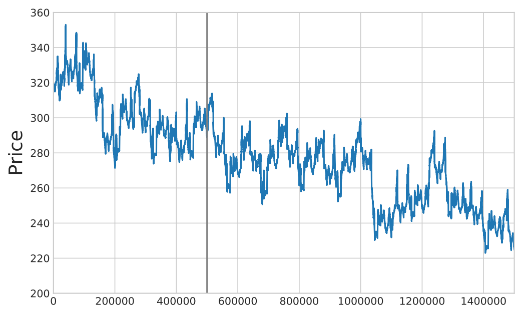
(b)



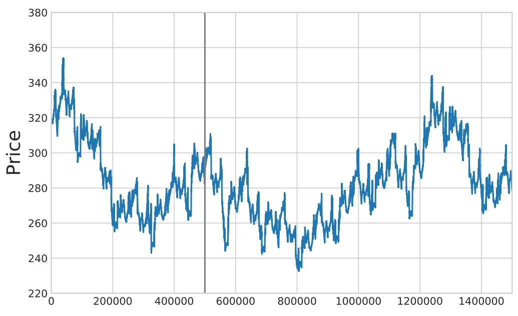
(c)



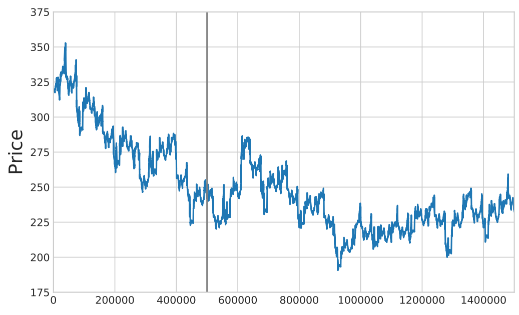
(d)



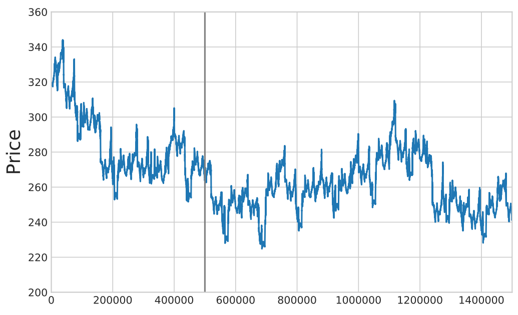
(e)



(f)

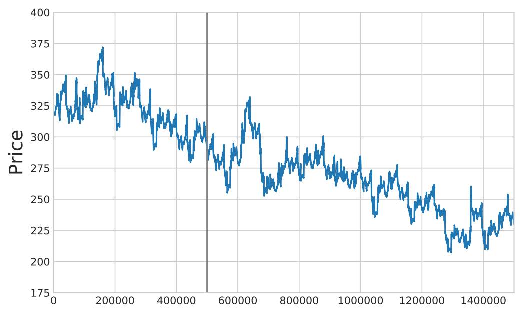


(g)

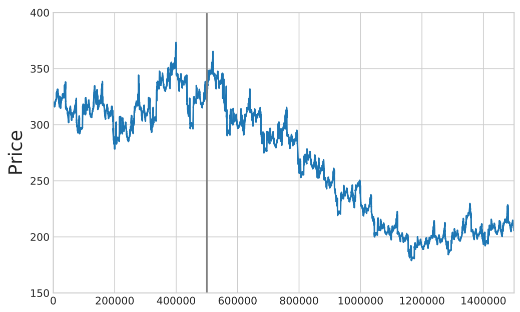


(h)

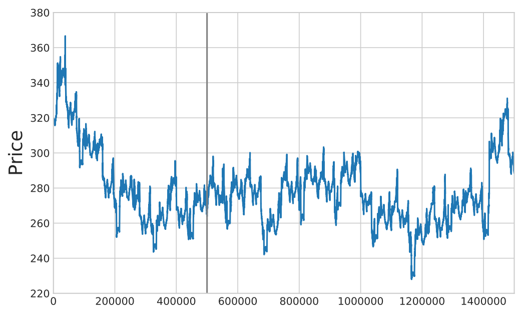
Figure B.3: *Reconstructed close price from synthetic sets 17-24 generated in Chapter 5.*



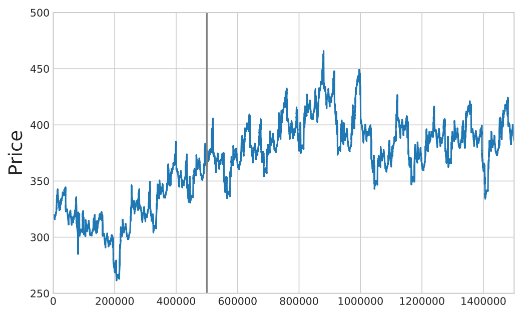
(a)



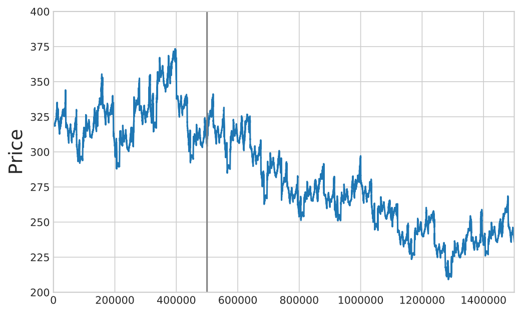
(b)



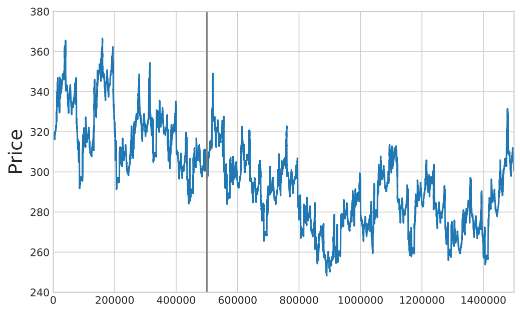
(c)



(d)



(e)



(f)

Figure B.4: *Reconstructed close price from synthetic sets 25-30 generated in Chapter 5.*

Appendix C

Evaluation Environments

In this appendix, the environments used to train and evaluate the machine learning methods proposed along with this thesis.

C.1 Hardware in Preliminary Studies

During the initial stage of this thesis, experiments were run in a *Macbook Pro 2011 13-inch Early 2011*. This laptop had a 2nd generation 2.3GHz dual-core *Intel Core i5* CPU and 4GBs of RAM. This machine was used for the first time in the preliminary studies in Section 4.1.

Since the second part of the preliminary studies needed more computational power, the experimental setup was migrated to a virtual machine in *Microsoft Azure*. Thus, in Section 4.2, the experiments were run in a *D3 v2/DS3 v2 instance*, with ≈14 GBs of RAM and four vCPUs.

C.2 Hardware Configuration in the Main Proposal

There were two upgrades to the servers used for our main proposal.

First, some of the experiments mentioned in Appendix A, the development and validation phase of the algorithm (GroCH), were performed in a *Dell XPS 7590* with 32 GB of RAM, a six-core *Intel Core i7 9750H* CPU and an NMVe SSD drive.

Second, Stages I and II of the experiments in Chapter 5, and all experiments in Chapter 6, were run in the *Sonic* HPC cluster[1] owned by University College Dublin. This is a SLURM cluster with 53 *Intel Xeon* and *Intel E5* nodes (1,468 cores) with 128 GBs to 1.5TB of RAM per node.

Having access to this cluster allowed us to conduct an exhaustive parameter optimisation (*grid search*) using different computing nodes in parallel for each combination of parameters.

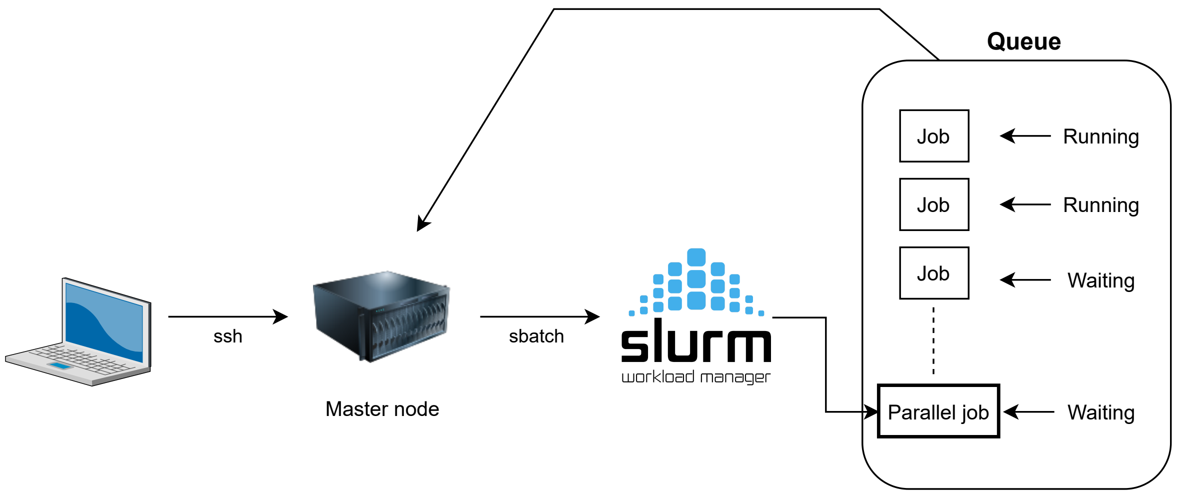


Figure C.1: *Example workflow of a job in SLURM.*

SLURM stands for *Simple Linux Utility for Resource Management*. As illustrated in Figure C.1, jobs are triggered from a head node (*sbatch* command) and distributed to the slaves (queue). In our experiments, machine learning tasks ran sequentially, and parallel threads were run when using different parameters, algorithms, or datasets.

[1] https://www.ucd.ie/itservices/ourservices/researchit/researchcomputing/sonichpc/

Glossary

Accuracy Under Switch

Evaluation measure used in this thesis that computes the classification accuracy only over instances received during a ground truth switch. 143, 263

Adaptive Learning

Subset of incremental learning methods also designed with forgetting mechanisms to avoid overfitting over time. 7, 48, 49, 56, 61–63, 71, 77, 85, 90, 93, 103, 105–107, 111, 123, 126

Adaptive market hypothesis

Hypothesis maintaining that the efficiency of a market evolves as market participants adapt to an environment that changes continuously. In this regard, initially adaptive heuristics explaining the market may become static in certain market situations, being financial markets predictable in specific periods [228]. 58, 263

Artificial Intelligence

Subfield of computer science about developing methods that would typically require human intelligence. 1, 2, 5, 39

Autoencoder

Type of unsupervised neural network. 226

AutoML

Subfield of AI about automated selection and tuning of models. 47

Bagging

Ensemble learning subsampling method to reduce variance in a dataset and increase diversity across base learners. 28, 29, 49, 108, 109, 124, 255

Bear Trend

Downward trend in financial markets. A bear market is a period of a market where the price tends to decrease. 14, 150, 191, 204

Bull Trend

Upward trend in financial markets. A bull market is a period of a market where the price tends to increase. 14, 150, 191, 204

Catastrophic Forgetting

Problem in AI where an algorithm might tend to forget knowledge learned previously and still relevant when learning new information. 61, 74, 93

Centroid

Center of a cluster or area with the greatest density (depending on the algorithm) in non-supervised learning. 24, 25, 50, 51, 163

Cluster

A group of instances computed by a non-supervised learning algorithm. It represents a segment of the data distribution in a certain period. 23–25, 39, 40, 47, 50, 51, 80, 82, 83, 85, 163, 192, 217, 252

Computational Finance

This is an application area of computer science which deals with problems of the financial domain. 1, 3, 4, 58, 88, 124, 220, 224, 226

Concept

A state in the regime dynamics of the data stream for which a machine learning model is stable. See concept drift. 2, 5, 6, 9, 39, 41, 45–47, 50, 54, 60–62, 64, 68, 69, 74–83, 85, 88, 92, 98, 104–106, 111, 113, 114, 119, 120, 123–125, 127, 128, 131, 136–138, 141, 142, 144, 151, 153, 163, 165, 169, 190, 193, 194, 219, 220, 228, 230–232, 236, 252, 260

Concept cluster

Cluster representing a concept in a data stream. 47, 80, 81

Concept Drift

Change in the probability distribution or mapping between the independent and dependent features in a data stream. xiii, xxv, xxvi, 2–7, 9, 11, 16, 36, 39, 40, 42–46, 48–50, 56, 57, 59–71, 77, 79, 80, 82, 86, 88, 89, 94, 98, 104–109, 111, 112, 114, 119, 120, 122–128, 130, 132, 133, 135–139, 142–145, 149–154, 157, 161, 171–173, 175, 179, 183, 193, 195, 205, 213, 216, 217, 220, 222–226, 236, 252, 260

Concept History

Also known as concept list in the literature. A pool of previous learners which is used for model reuse in case of recurring concept drifts. In GroCH, a concept history is made of a set of groups. 46, 74, 78–80, 106–114, 116, 118–130, 132–136, 138–142, 146–149, 164, 165, 169, 171, 173, 175, 176, 181, 190, 192, 193, 195, 202–204, 224, 230–234, 236

Concept History Group

In GroCH, a group defines a concept in the pool of historical models. Each group is represented by a topology and contains many idle classifiers. xxvii, 125–136, 139–141, 144–149, 163–165, 167, 171–174, 176, 177, 179, 181, 190, 192, 193, 195, 196, 203, 224, 231–234, 236

Concept Similarity

This is an unsupervised metric to measure distances between concepts in the data stream mining literature. 46, 74, 80, 81, 128, 131, 141, 170, 172, 174, 190, 194

Conceptual Equivalence

Supervised metric to measure the similarity between two models by comparing their classification performance during a period in a data stream. 46, 47, 74, 78, 79, 132, 141, 167, 192

Continuous Learning

A subfield of machine learning that includes techniques and algorithms to learn continuously from a data stream. 4, 5, 11, 35, 52, 56, 57, 61, 63, 103, 221, 226

Data Maturity

The ability of an organisation to extract insights and make decisions using its data. 4

Data Stream

Continuous data flow of information, in the form of ordered data instances or chunks (sets of instances or batch) arriving at different speeds or time intervals. v, ix, 1–3, 5–11, 23, 26, 30–34, 36–44, 46, 47, 49, 50, 53, 55, 56, 60–64, 66–71, 74–83, 86, 88, 90, 92, 97, 99, 104, 106–110, 113, 118–123, 125, 128–130, 133, 135–137, 142, 144, 149, 151, 152, 157, 159, 165–167, 171–173, 175, 178, 179, 181, 183–188, 192, 195–198, 212, 214, 217, 219–221, 223, 225, 226, 230, 231, 240, 252

Data Stream Mining

Also known as data stream analysis or data stream learning. It is a subfield of machine learning that investigates learning models for continuous and infinite data streams of information. 37, 38, 41, 46, 47, 50, 55–57, 60–63, 70, 71, 73, 74, 81, 82, 85, 86, 88, 90, 91, 93, 99, 103, 106, 109, 114, 116, 123, 124, 140, 197, 203, 209, 217, 219, 220, 223, 225, 226, 259

Deep Learning

Subfield of machine learning based on deep artificial neural networks (conventionally of more than three layers). 4, 46, 55, 61, 85, 99

Deterministic Algorithm

Algorithm that produces the same output under all circumstances for a given input. 164

Efficient market hypothesis

Hypothesis in finance maintaining that all stocks trade at their fair value because all available information about the market is already incorporated into its prices. 13, 264

Ensemble

Machine learning method that uses a diverse set of weak learners and aggregates their output to produce stronger predictions. 4, 27–29, 48, 49, 62, 63, 65, 70, 71, 73–75, 77–83, 86, 88, 89, 106, 107, 109–112, 117, 118, 120, 123, 124, 137, 141, 219, 222

Euclidean distance

Straight distance between two points or instances in a multivariate space. 24–26, 29, 47, 81, 133, 135, 194, 231

Example

This is a row of data containing a set of features received by the algorithm. Also called instance in this thesis. 22, 29, 30, 49, 84, 91–96, 99, 101, 102, 104, 114, 117, 121, 126–128, 130–139, 144, 192, 194, 205, 216, 232, 244

Explainable AI

Subfield of AI that makes model predictions interpretable and understandable. 5

F1-score

Classification performance measure that expresses the mean between precision and recall. 31

Factor analysis

Unsupervised machine learning algorithm commonly used for dimensionality reduction. 226

Feature bagging

Random (feature) subspace technique based on the bagging method and used in the *random forest* algorithm. 29

Gold Standard

Term used in statistics to refer to a result that is accepted as the best available option under reasonable circumstances. This is commonly used in many subfields of AI to refer to target or dependent features obtained by automated labelling processes and without certainty of being a ground truth. 140

Grid Search

Exhaustive parameter search over specified values or ranges. 156, 166, 250

Ground Truth

Term used in AI to refer to the actual real values of a variable (usually the target feature). xxvi, 6, 9, 10, 16, 31, 35, 39, 40, 50, 67, 81, 82, 114, 120, 124, 125, 128, 133, 140, 142–150, 154, 157, 163, 164, 169, 171–176, 178, 179, 181, 185, 186, 190, 191, 194, 195, 205, 212, 214, 216, 217, 220, 222, 225, 226, 230–234, 236

Holdout

Evaluation scheme where train, validation and test splits are independent samples. 34, 35

Incremental Learning

Subset of machine learning methods designed to learn incrementally over time. 2, 9, 37, 62, 70, 103, 223, 225, 251, 257

Instance

Data example to be used in pre-training, parameter tuning, training or evaluation tasks. 5, 22–24, 26, 28–31, 33–35, 37–39, 41–45, 47–50, 52, 53, 62, 64, 70, 73, 75, 80, 82, 83, 85, 94–97, 99, 100, 103, 108–114, 116–118, 121, 122, 134, 138, 139, 141–145, 147–150, 157, 159, 164, 165, 169, 170, 173–176, 178, 179, 181, 183, 185, 186, 189, 192, 193, 199, 233–235, 240, 243, 244

Kappa statistics

Classification performance measure that expresses agreement between observations. Initially introduced by Cohen [93], it considered the randomness of each class for rating the strength of a classifier. xxviii, 31–33, 143, 165, 185, 186, 189, 195, 196, 209, 213–215, 220, 222

Kappa Statistics Under Switch

Evaluation measure used in this thesis that computes the kappa statistic only over instances received during a ground truth switch. 143, 266

Lag

In the time series field, a feature value in a certain point in time receives the name of lag. 18

Latent Space Representation

Representation of a data point in a lower dimensional space with all the relevant information. 60

Machine Learning

Subfield of AI based on the study of computer algorithms able to learn automatically using data. 1–9, 11, 22–24, 29–31, 33–38, 44, 55–57, 59–62, 70, 75, 83, 85, 86, 88–90, 99, 106, 157, 185, 217, 219, 221, 223, 224, 249, 250

Mahalanobis distance

This method can be seen as an extension of the Euclidean distance, where features are weights based on a covariance matrix that accounts for the variance of each feature and the correlation among features. 24, 26, 133, 135, 159, 165, 190, 194, 202–204, 231, 243, 244

Majority voting

Aggregation technique in classification tasks where the global prediction of an ensemble is the most voted class by all the weak learners (or the one with most weight if using *weighted majority voting*). 28, 138

Mann Whitney Wilcoxon Test

Non-parametric test to be applied to independent tests. It tests the equality of means in two independent samples. Used in post-hoc tests to check for statistical significance and validate the results. 117, 165, 201

Meta-learning

Machine learning subfield including techniques that can decide when to train, when and what model to replace, when to forget a learner and when to create one. by using the evaluation performance metrics of active and historical models. 47, 74, 76–78, 81, 88, 124, 134, 189

Mini-batch Learning

Subset of incremental learning methods, adaptive or not, that receive sets of instances (batches) of data at a time. A purely incremental algorithm would receive instances one by one. 63, 70, 73, 74, 77, 89, 91–93, 99, 102, 104

Mutual information

In statistics, known as mutual dependence, or non-linear correlation, between two variables. 234

Neural Network

Machine learning algorithm designed to mimic the human brain. This is traditionally used for prediction with non-linear data. xi, 4, 29, 50, 61, 64, 86, 251, 254, 259

Normalisation

In machine learning and statistics, this term usually refers to scaling the input data to an algorithm before training or evaluation tasks. 86, 117

One-pass Learning

Fashion to perform learning tasks seeing the data once rather than having multiple passes (or epochs). This is usual in online learning settings and is also known as *single-pass* learning. 38, 48, 70, 126, 134, 136, 169, 190, 194

Online Incremental Machine Learning

Subfield of machine learning aimed to deal with infinite data streams. These are single-pass and always up to date. 7, 8, 11, 13, 17, 32, 33, 36, 55, 57, 60, 70, 89, 91, 221, 223, 258

Online Machine Learning

See online incremental machine learning methods. v, 1, 2, 8, 11, 27, 33, 36, 47, 55, 56, 123

Out of Sample

In statistics, term used to refer to data unseen by the model. It can be used to refer to a test set or future time points in a time series. 34

Precision

This classification performance measure represents the percentage of positive classifications being correct. 31, 255

Prequential

Interleaved test-then-train evaluation, a convention in the field of data stream mining. 31, 35, 38, 73, 99, 116, 140, 143, 165, 166, 201, 243

Principal Components Analysis

Principal components analysis, known as PCA, is a dimensionality reduction algorithm that aggregates features into a smaller set of attributes. 226

Prototypes

Set of instances generated or selected by an algorithm. Term used to refer to the output of a prototype reduction algorithm. 31, 50, 52–54, 83–85, 88, 91–94, 96–98, 100, 103, 104, 125, 130, 169, 174, 190, 194, 230, 232

Quantisation Error

In this thesis, it refers to the difference (error) between the initial ground truth distribution and a summarised topology. 92, 169, 230, 232

RAM-Hours

Computational performance evaluation measure reporting megabytes of RAM used per hour. xxx, 31, 33, 69, 149, 188, 206, 210, 214

Recall

This classification performance measure represents the percentage of true positives classified correctly. 31, 255

Recovery Analysis

Recovery analysis [312] is another research topic that has arisen from the survival analysis research topic. While the former represents how long will it be until an event occurs, recovery analysis refers to, after that event, when will the system be able to recover. 68, 69, 157

Recurring Neural Network

Type of neural network with a design that allows learning sequentially, keeping a memory in temporal data, but also in other domains where the context is crucial. This technique is not part of the state of the art in data stream mining and had not been used in conjunction to drift detection techniques in the literature to the best of our knowledge at the time of writing this thesis. 267

Regime Change

In this thesis, this term refers to a change in the behaviour of a time series over time due to external changes. 1, 5, 12–16, 57–59, 106, 175, 196

Regime Switch

Used interchangeably with *regimechange* to express a change in the time-series dynamics. In the relevant literature, these are usually attributed to changes in geopolitical variables, which are then reflected but not explicitly captured by a financial time series. 164, 172, 173, 181, 185, 204

Reinforcement Learning

Subfield of machine learning based on reward functions and policies. 55, 61

Skewness

In statistics. it refers to the level of asymmetry in the data distribution. 5, 34, 104

Stability-plasticity Dilemma

Balance between the retention of previously learned knowledge (stability) while adapting to new concepts (plasticity). 5, 39, 60

Structural Break

Sudden change of the behaviour of a time series in a given point in time. In this thesis, we relate this to the problem of concept drift. 3, 7, 13, 16, 39, 41, 42, 56–58, 60, 88, 106, 142, 153, 176, 183, 189, 212, 214, 219, 221, 223, 224, 226

Stylised Facts

Term used in economics to refer to consistent empirical findings accepted as truth. 61

Subtopology

In this thesis, we call subtopology a topology that uses a feature subset of all the attributes used to evaluate and train a model. Different feature subsets used in this regard will constitute different subtopologies. 168, 169, 174

Survival Analysis

Statistical method that studies the time left for an event to happen. 68, 259

Time Series

Sequence of numerical observations gathered sequentially in time. Traditionally used for forecasting and simulation. 3, 5, 7, 9, 16, 18–20, 22, 36–38, 56, 59, 61–63, 87, 89, 125, 135, 136, 140, 142, 150–154, 157, 160, 164, 171, 195, 197, 219, 220, 222, 224, 243

Topology

Term used in the research paper that introduced growing neural gas [136] to refer to the unsupervised relationships between instances in the data set, which should be preserved when performing a summary of the data generating prototypes. In this thesis, we use this term to refer to the set of prototypes or instances representing a concept. 29, 51, 53, 54, 85, 89, 91, 92, 94, 98, 100, 103, 104, 125, 128–136, 138–140, 148, 165, 169, 172, 174, 189–194

Transfer learning

Subfield of machine learning that reuses knowledge previously learned by a model. Typically, pre-trained models would be retrieved to transfer previous learning and then fine tuned to be adapted to a new problem. In this thesis, we mainly relate this term to the task of model reuse in meta-learning. 46, 61, 268

Warning window

Period of time between the detection of a warning and a drift. xxvi, 45, 75, 81, 108–114, 118, 120, 124, 126–139, 141, 142, 170, 172, 173, 178, 192, 232

This page is intentionally left blank.

Acronyms

ADWIN2

Drift detector based on the ADaptive WINdowing algorithm. 45, 49, 64

AHOEFT

Adaptive Hoeffding tree (same as HAT). 116

AI

Artificial intelligence. vi, 1–4, 11, 12, 22, 55, 59, 85, 88, 219, 251, 252

AMH

Adaptive market hypothesis. 58

ARF

Adaptive random forest. 72–74, 79–81, 107, 108, 111, 116, 118–120, 122, 123, 141, 142

ARMA

Autoregressive–moving-average model. 19, 20, 155, 156

AUS

Accuracy under switch performance measure. 143, 165, 166, 183, 185, 189, 195

BWX

International bonds - SPDR Barclays International Treasury Bond ETF. 154, 161

CH

Concept history. 111–114, 120, 121, 131, 138, 140, 144, 147, 148, 164, 165, 228, 232

CPF

Concept profiling framework. 75, 79, 80, 141, 142, 167, 168, 172, 174, 176, 182, 183, 185, 192, 193, 212, 215

DC

Directional change. 161

DDM

Drift detection method. 45, 65, 73, 171

ECPF

Enhanced concept profiling framework. 80, 81, 141, 142, 167, 168, 172, 174, 176, 183, 185, 186, 188, 189, 212, 213, 215, 217

EDDM

Early drift detection method. 45

EMH

Efficient market hypothesis. 13, 14, 16, 22, 58, 220

ENN

Wilson's edited nearest neighbours 29, 30, 93–96, 98, 100, 102–104

ETF

Exchange traded fund. 114, 123, 154, 155, 160, 161, 163, 196, 198, 220, 222

FIFO

First in first out policy. 132

FN

False negative in a classification task. 31, 145, 147, 148

FP

False positive in a classification task. 31, 145, 147, 148

GARCH

Generalized autoregressive conditional heteroskedasticity model. 18–21, 155, 156

GNG

Growing neural gas. 31, 52–54, 85, 91–95, 97, 98, 100, 102–105, 134, 135, 169, 170, 174, 189, 190, 194, 225, 230, 232, 265

GroCH

Growing concept history of recurring classifiers. 10, 125–128, 130, 131, 133–142, 144–148, 150, 155, 163–167, 169–174, 176–183, 185–190, 192–196, 201–204, 209, 212–217, 220–225, 227–233, 235, 243, 249

GT

Ground truth. 142, 147–150, 159, 164, 165, 173, 178, 179, 183, 186, 189

HAT

Hoeffding adaptive tree. 48, 49, 64, 116, 263

HDDM$_A$

Drift detection method based on Hoeffding's bounds using simple moving averages. 46, 65, 171, 230, 234

HFT

High frequency trading. 1–3, 6

HPC

High-performance cluster. 250

HT

Hoeffding tree. 48, 49, 66, 70, 73, 167, 168, 170, 172, 175, 176, 179, 183, 185, 186, 201, 213, 229, 234

iGNGSVM

Online incremental GNG-SVM. 9, 92–105, 124, 219, 221, 225

IoT

Internet of things. 226

KUS

Kappa statistics under switch performance measure. 143, 185, 186, 195

LIFO

Last in first out policy. 132

LUFO

Least used last out policy. 130, 193

ML

Machine learning. 57, 58

MOA

Massive online analysis framework for data stream mining. 27, 32, 55, 99, 116, 141, 143, 152, 165, 171, 172, 195, 227, 229

NB

Naive Bayes. 26–28, 66, 70, 167, 170, 172, 179, 180, 183, 185–187, 189, 201, 212, 214–216, 229

OHLC

Open-high-low-close financial prices. 157, 159

PFF

Securities/Fixed-income preferred - iShares Preferred and Income Securities ETF. 154, 161

PG

Prototype generation. 30, 31, 84, 91, 92, 194

PS

Prototype reduction. 30, 31

RC

Regime change. 1–3, 7, 9, 14–16, 59, 106, 219, 220, 223, 224, 226

RCARF

Recurring concepts adaptive random forest. xxix, 9, 107, 109–114, 116–124, 126, 141, 142, 220, 223–225

RCD

Recurring concept drifts framework. 73, 75, 77, 79, 116, 118, 120, 141, 142

RDDM

Reactive drift detection method. 45, 65

RF

Random forest. 29, 71, 116, 219

RNN

Recurring neural network. 61

ROC

Receiver operating characteristic. 32

S&P 500

Standard and Poor's 500. xiii, 59, 64, 89, 106, 114, 123, 154, 198, 220

SOTA

State of the art. 141, 170

SPDR

Standard & poor's depository receipts. xiii, 89, 106, 154, 198, 220

SPY

Equities - SPDR S&P 500 ETF. 114, 115, 154, 161, 198, 199, 203, 209, 220, 222

SVM

Support vector machines. 30, 70, 83, 85, 86, 88, 90–96, 99, 100, 102–104, 124, 265

TA-Lib

Technical analysis library: https://www.ta-lib.org/. 115, 235

TL

Transfer learning. 46, 61

TN

True negative in a classification task. 31, 144, 145, 147, 148

TP

True positive in a classification task. 31, 144–148, 179

UCD

University College Dublin. vi

VFDT

Very fast decision tree algorithm. Original version of the Hoeffding tree. 48, 175

VNQ

Real-estate - Vanguard Real Estate ETF. 154, 161

Bibliography

[1] Abdiansah, A., and Wardoyo, R. Article: Time complexity analysis of support vector machines (svm) in libsvm. *International Journal of Computer Applications 128*, 3 (Oct. 2015), 28–34. Published by Foundation of Computer Science (FCS), NY, USA.

[2] Abu-Mostafa, Y. S., and Atiya, A. F. Introduction to financial forecasting. *Applied Intelligence 6*, 3 (7 1996), 205–213.

[3] Adcock, R., and Gradojevic, N. Non-fundamental, non-parametric bitcoin forecasting. *Physica A: Statistical Mechanics and its Applications 531* (2019), 121727.

[4] Adebiyi, A. A., Ayo, C. K., Adebiyi, M. O., and Otokiti, S. O. Stock price prediction using neural network with hybridized market indicators. *Journal of Emerging Trends in Computing and Information Sciences 3*, 1 (2012), 1–9.

[5] Aggarwal, C. C., Philip, S. Y., Han, J., and Wang, J. A framework for clustering evolving data streams. In *Proceedings 2003 VLDB conference* (2003), Elsevier, pp. 81–92.

[6] Ahmadi, Z., and Kramer, S. Modeling recurring concepts in data streams: a graph-based framework. *Knowledge and Information Systems 55*, 1 (2018), 15–44.

[7] Alippi, C., Boracchi, G., and Roveri, M. Just-in-time classifiers for recurrent concepts. *IEEE Transactions on Neural Networks and Learning Systems 24*, 4 (4 2013), 620–634.

[8] Almeida, P. R., Oliveira, L. S., Britto Jr, A. S., and Sabourin, R. Adapting dynamic classifier selection for concept drift. *Expert Systems with Applications 104* (2018), 67–85.

[9] Alonso-Monsalve, S., Suárez-Cetrulo, A. L., Cervantes, A., and Quintana, D. Convolution on neural networks for high-frequency trend prediction of cryptocurrency exchange rates using technical indicators. *Expert Systems with Applications* (1 2020).

[10] Amarasiri, R., Alahakoon, D., Smith, K., and Premaratne, M. Hdgsomr: a high dimensional growing self-organizing map using randomness for efficient web and text mining. In *The 2005 IEEE/WIC/ACM International Conference on Web Intelligence (WI'05)* (2005), IEEE, pp. 215–221.

[11] Amarasiri, R., Alahakoon, D., and Smith, K. A. Hdgsom: a modified growing self-organizing map for high dimensional data clustering. In *Fourth International Conference on Hybrid Intelligent Systems (HIS'04)* (2004), IEEE, pp. 216–221.

[12] Andersen, T. G., and Bollerslev, T. Answering the skeptics: Yes, standard volatility models do provide accurate forecasts. *International economic review* (1998), 885–905.

[13] Andersen, T. G., Davis, R. A., Kreiss, J.-P., and Mikosch, T. V. *Handbook of financial time series.* Springer, 2009.

[14] Anderson, R., Koh, Y. S., and Dobbie, G. Cpf: Concept profiling framework for recurring drifts in data streams. In *Australasian Joint Conference on Artificial Intelligence* (2016), Springer, pp. 203–214.

[15] Anderson, R., Koh, Y. S., and Dobbie, G. Lift-per-drift: An evaluation metric for classification frameworks with concept drift detection. In *Lecture Notes in Computer Science* (2018), vol. 11320 LNAI, pp. 630–642.

[16] Anderson, R., Koh, Y. S., Dobbie, G., and Bifet, A. Recurring concept meta-learning for evolving data streams. *Expert Systems with Applications 138* (2019), 112832.

[17] Andreou, E., and Ghysels, E. Structural breaks in financial time series. *Handbook of financial time series* (2009), 839–870.

[18] Ang, A., and Bekaert, G. How regimes affect asset allocation. *Financial Analysts Journal 60*, 2 (2004), 86–99.

[19] Ang, A., and Timmermann, A. Regime changes and financial markets. *Annu. Rev. Financ. Econ. 4*, 1 (2012), 313–337.

[20] Angelopoulos, A., Giannopoulos, A. E., Kapsalis, N. C., Spantideas, S. T., Sarakis, L., Voliotis, S., and Trakadas, P. Impact of classifiers to drift detection method: A comparison. In *International Conference on Engineering Applications of Neural Networks* (2021), Springer, pp. 399–410.

[21] Angelov, P. P. *Empirical Approach to Machine Learning.* Springer, 2017.

[22] Angelov, P. P., and Filev, D. P. An approach to online identification of takagi-sugeno fuzzy models. *Trans. Sys. Man Cyber. Part B 34*, 1 (Feb. 2004), 484–498.

[23] Angelov, P. P., and Zhou, X. Evolving fuzzy-rule-based classifiers from data streams. *IEEE Transactions on Fuzzy Systems 16*, 6 (Dec. 2008), 1462–1475.

[24] Antonanzas, J., Arias, M., and Bifet, A. Sketches for time-dependent machine learning. *arXiv preprint arXiv:2108.11923* (2021).

[25] Ardia, D., Bluteau, K., and Rüede, M. Regime changes in Bitcoin GARCH volatility dynamics. *Finance Research Letters 29* (Jun. 2019), 266–271.

[26] Ariyo, A. A., Adewumi, A. O., and Ayo, C. K. Stock price prediction using the ARIMA model. In *2014 UKSim-AMSS 16th International Conference on Computer Modelling and Simulation* (2014), IEEE, pp. 106–112.

[27] Armano, G., Marchesi, M., and Murru, A. A hybrid genetic-neural architecture for stock indexes forecasting. *Information Sciences 170*, 1 (2005), 3–33.

[28] Atsalakis, G., and K, V. *Surveying stock market forecasting techniques - Part I: Conventional methods.* Nova Science Publishers, Inc, New York, 01 2013, ch. 4, pp. 49–104.

[29] Atsalakis, G. S., and Valavanis, K. P. Surveying stock market forecasting techniques - Part II: Soft computing methods. *Expert Systems with Applications 36*, 3 PART 2 (4 2009), 5932–5941.

[30] Bacchetta, P., Mertens, E., and Van Wincoop, E. Predictability in financial markets: What do survey expectations tell us? *Journal of International Money and Finance 28*, 3 (2009), 406–426.

[31] Baek, S., Mohanty, S. K., and Glambosky, M. Covid-19 and stock market volatility: An industry level analysis. *Finance Research Letters 37* (2020), 101748.

[32] Baena-Garcia, M., del Campo-Ávila, J., Fidalgo, R., Bifet, A., Gavalda, R., and Morales-Bueno, R. Early drift detection method. In *Fourth international workshop on knowledge discovery from data streams* (2006), vol. 6, pp. 77–86.

[33] Bahri, M., Bifet, A., Gama, J., Gomes, H. M., and Maniu, S. Data stream analysis: Foundations, major tasks and tools. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 11*, 3 (2021), e1405.

[34] Bakirov, R., Fay, D., and Gabrys, B. Automated adaptation strategies for stream learning. *Machine Learning* (2021), 1–34.

[35] Baldi, P., Sadowski, P., and Whiteson, D. Searching for exotic particles in high-energy physics with deep learning. *Nat Commun 5* (07 2014).

[36] Ballings, M., Van Den Poel, D., Hespeels, N., and Gryp, R. Evaluating multiple classifiers for stock price direction prediction. *Expert Systems with Applications 42*, 20 (2015), 7046–7056.

[37] Barddal, J. P., Gomes, H. M., Enembreck, F., and Pfahringer, B. A survey on feature drift adaptation: Definition, benchmark, challenges and future directions. *Journal of Systems and Software 127* (2017), 278–294.

[38] Barros, R. S., Cabral, D. R., Gonçalves Jr, P. M., and Santos, S. G. RDDM: Reactive drift detection method. *Expert Systems with Applications 90* (2017), 344–355.

[39] Barros, R. S. M., and Santos, S. G. T. C. A large-scale comparison of concept drift detectors. *Information Sciences 451-452* (2018), 348–370.

[40] Baruah, R. D., and Angelov, P. Evolving fuzzy systems for data streams: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 1*, 6 (11 2011), 461–476.

[41] Basseville, M., Nikiforov, I. V., et al. *Detection of abrupt changes: theory and application*, vol. 104. prentice Hall Englewood Cliffs, 1993.

[42] Bayes, T. An essay towards solving a problem in the doctrine of chances. 1763. *MD computing: computers in medical practice 8*, 3 (1991), 157–171.

[43] Bettman, J. L., Sault, S. J., and Schultz, E. L. Fundamental and technical analysis: substitutes or complements? *Accounting & Finance 49*, 1 (2009), 21–36.

[44] Bifet, A. Classifier Concept Drift Detection and the Illusion of Progress. In *International Conference on Artificial Intelligence and Soft Computing* (2017), Springer, pp. 715–725.

[45] Bifet, A., de Francisci Morales, G., Read, J., Holmes, G., and Pfahringer, B. Efficient online evaluation of big data stream classifiers. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining* (2015), pp. 59–68.

[46] Bifet, A., and Gavalda, R. Learning from time-changing data with adaptive windowing. In *Proceedings of the 2007 SIAM international conference on data mining* (2007), vol. 7, SIAM, pp. 443–448.

[47] Bifet, A., and Gavaldà, R. Adaptive learning from evolving data streams. In *Advances in Intelligent Data Analysis VIII: 8th International Symposium on Intelligent Data Analysis, IDA 2009, Lyon, France, August 31 - September 2, 2009. Proceedings.* Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 249–260.

[48] Bifet, A., Gavaldà, R., Holmes, G., and Pfahringer, B. *Machine learning for data streams: with practical examples in MOA*. MIT press, 2018.

[49] Bifet, A., Holmes, G., Kirkby, R., and Pfahringer, B. MOA: Massive Online Analysis. *Journal of Machine Learning Research 11* (Aug. 2010), 1601–1604.

[50] Bifet, A., Holmes, G., and Pfahringer, B. Leveraging bagging for evolving data streams. In *Joint European conference on machine learning and knowledge discovery in databases* (2010), Springer, pp. 135–150.

[51] Bifet, A., Holmes, G., Pfahringer, B., and Frank, E. Fast perceptron decision tree learning from evolving data streams. In *Pacific-Asia conference on knowledge discovery and data mining* (2010), Springer, pp. 299–310.

[52] Bifet, A., Holmes, G., Pfahringer, B., Kranen, P., Kremer, H., Jansen, T., and Seidl, T. Moa: Massive online analysis, a framework for stream classification and clustering. *Journal of Machine Learning Research - Proceedings Track 11* (2010), 44–50.

[53] Bisoi, R., and Dash, P. K. A hybrid evolutionary dynamic neural network for stock market trend analysis and prediction using unscented kalman filter. *Applied Soft Computing 19* (2014), 41–56.

[54] Black, F. Noise. *The Journal of Finance 41*, 3 (Jul. 1986), 529–543.

[55] Bollen, J., Mao, H., and Zeng, X. Twitter mood predicts the stock market. *Journal of computational science 2*, 1 (2011), 1–8.

[56] Bollerslev, T. Generalized autoregressive conditional heteroskedasticity. *Journal of econometrics 31*, 3 (1986), 307–327.

[57] Bollerslev, T., Chou, R. Y., and Kroner, K. F. Arch modeling in finance: A review of the theory and empirical evidence. *Journal of econometrics 52*, 1-2 (1992), 5–59.

[58] Booth, A., Gerding, E., and McGroarty, F. Automated trading with performance weighted random forests and seasonality. *Expert Systems with Applications 41*, 8 (6 2014), 3651–3661.

[59] Boser, B. E., Guyon, I. M., and Vapnik, V. N. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory* (1992), ACM, pp. 144–152.

[60] Boulbazine, S., Cabanes, G., Matei, B., and Bennani, Y. Online semi-supervised growing neural gas for multi-label data classification. In *2018 International Joint Conference on Neural Networks (IJCNN)* (2018), pp. 1–8.

[61] Box, G. E., Jenkins, G. M., Reinsel, G. C., and Ljung, G. M. *Time series analysis: forecasting and control.* John Wiley & Sons, 2015.

[62] Brabazon, A., and O'Neill, M. *Biologically inspired algorithms for financial modelling.* Springer Science & Business Media, 2006.

[63] Bracke, P., Datta, A., Jung, C., and Sen, S. Machine learning explainability in finance: an application to default risk analysis. Working paper, Bank of England, 2019.

[64] Brailsford, T. J., and Faff, R. W. An evaluation of volatility forecasting techniques. *Journal of Banking & Finance 20*, 3 (1996), 419–438.

[65] Breiman, L. Bagging predictors. *Machine learning 24*, 2 (1996), 123–140.

[66] Breiman, L. Random forests. *Machine Learning 45*, 1 (Oct. 2001), 5–32.

[67] Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. *Classification and regression trees.* Routledge, 2017.

[68] Brereton, R. G. The Mahalanobis distance and its relationship to principal component scores. *Journal of Chemometrics 29*, 3 (Mar. 2015), 143–145.

[69] Brighton, H., and Mellish, C. Advances in instance selection for instance-based learning algorithms. *Data Min. Knowl. Discov. 6*, 2 (Apr. 2002), 153–172.

[70] Brock, W., Lakonishok, J., and LeBaron, B. Simple technical trading rules and the stochastic properties of stock returns. *The Journal of finance 47*, 5 (1992), 1731–1764.

[71] Brockwell, P. J., and Davis, R. A. *Time series: theory and methods.* Springer Science & Business Media, 2009.

[72] Bryll, R., Gutierrez-Osuna, R., and Quek, F. Attribute bagging: improving accuracy of classifier ensembles by using random feature subsets. *Pattern recognition 36*, 6 (2003), 1291–1302.

[73] Brzezinski, D., and Stefanowski, J. Reacting to different types of concept drift: The accuracy updated ensemble algorithm. *IEEE Transactions on Neural Networks and Learning Systems 25*, 1 (1 2014), 81–94.

[74] Burnham, K. P., and Anderson, D. R. Multimodel inference: understanding AIC and BIC in model selection. *Sociological methods & research 33*, 2 (2004), 261–304.

[75] Cano, A. A survey on graphic processing unit computing for large-scale data mining. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 8*, 1 (2018), e1232.

[76] Carnein, M., and Trautmann, H. Optimizing data stream representation: An extensive survey on stream clustering algorithms. *Business & Information Systems Engineering 61*, 3 (2019), 277–297.

[77] Cassales, G., Gomes, H., Bifet, A., Pfahringer, B., and Senger, H. Improving the performance of bagging ensembles for data streams through mini-batching. *Information Sciences 580* (2021), 260–282.

[78] Cauwenberghs, G., and Poggio, T. Incremental and decremental support vector machine learning, 2000.

[79] Cavalcante, R. C., Brasileiro, R. C., Souza, V. L. F., Nobrega, J. P., and Oliveira, A. L. I. Computational Intelligence and Financial Markets: A Survey and Future Directions. *Expert Systems with Applications 55* (8 2016), 194–211.

[80] Cavalcante, R. C., and Oliveira, A. L. An autonomous trader agent for the stock market based on online sequential extreme learning machine ensemble. In *2014 International Joint Conference on Neural Networks (IJCNN)* (2014), IEEE, pp. 1424–1431.

[81] Cerqueira, V., Torgo, L., and Mozetič, I. Evaluating time series forecasting models: an empirical study on performance estimation methods. *Machine Learning 109*, 11 (Nov. 2020), 1997–2028.

[82] Chang, C.-C., and Lin, C.-J. LIBSVM: A library for support vector machines. *ACM Trans. Intell. Syst. Technol. 2*, 3 (May 2011), 27:1–27:27.

[83] Chen, C., Dongxing, W., Chunyan, H., and Xiaojie, Y. Exploiting social media for stock market prediction with factorization machine. In *2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)* (2014), vol. 2, IEEE, pp. 142–149.

[84] Chen, H., and Singal, V. Role of speculative short sales in price formation: The case of the weekend effect. *The Journal of Finance 58*, 2 (2003), 685–705.

[85] Chen, K., Koh, Y. S., and Riddle, P. Proactive drift detection: Predicting concept drifts in data streams using probabilistic networks. In *2016 International Joint Conference on Neural Networks (IJCNN)* (2016), IEEE, pp. 780–787.

[86] Chikushi, R. T. M., de Barros, R. S. M., da Silva, M. G. N., and Maciel, B. I. F. Using spectral entropy and bernoulli map to handle concept drift. *Expert Systems with Applications 167* (Apr. 2021), 114114.

[87] Chintapalli, S., Dagit, D., Evans, B., Farivar, R., Graves, T., Holderbaugh, M., Liu, Z., Nusbaum, K., Patil, K., Peng, B. J., et al. Benchmarking streaming computation engines: Storm, flink and spark streaming. In *2016 IEEE international parallel and distributed processing symposium workshops (IPDPSW)* (2016), IEEE, pp. 1789–1792.

[88] Chiu, C. W., and Minku, L. L. Diversity-Based Pool of Models for Dealing with Recurring Concepts. In *Proceedings of the International Joint Conference on Neural Networks* (Oct. 2018), vol. 2018-July, Institute of Electrical and Electronics Engineers Inc.

[89] Chiu, C. W., and Minku, L. L. A diversity framework for dealing with multiple types of concept drift based on clustering in the model space. *IEEE Transactions on Neural Networks and Learning Systems* (2020).

[90] Chordia, T., Goyal, A., Lehmann, B. N., and Saar, G. High-frequency trading. *Journal of Financial Markets 16*, 4 (11 2013), 637–645.

[91] Choudhury, S., Ghosh, S., Bhattacharya, A., Fernandes, K. J., and Tiwari, M. K. A real time clustering and SVM based price-volatility prediction for optimal trading strategy. *Neurocomputing 131* (5 2014), 419–426.

[92] Ciaburro, G., and Venkateswaran, B. *Neural Networks with R: Smart models using CNN, RNN, deep learning, and artificial intelligence principles.* Packt Publishing Ltd, 2017.

[93] Cohen, J. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement 20*, 1 (1960), 37–46.

[94] Comuzzi, M., and Patel, A. How organisations leverage big data: A maturity model. *Industrial Management & Data Systems* (2016).

[95] Cordón, O., Gomide, F., Herrera, F., Hoffmann, F., and Magdalena, L. Ten years of genetic fuzzy systems: Current framework and new trends. In *Fuzzy Sets and Systems* (2004), vol. 141, pp. 5–31.

[96] Cristianini, N., Shawe-Taylor, J., et al. *An introduction to support vector machines and other kernel-based learning methods.* Cambridge university press, 2000.

[97] Cryer, J. D., and Chan, K.-S. *Time series analysis: with applications in R*, vol. 2. Springer, 2008.

[98] Cumby, R., Figlewski, S., and Hasbrouck, J. Forecasting volatilities and correlations with EGARCH models. *The Journal of Derivatives 1*, 2 (1993), 51–63.

[99] da Costa, F. G., Duarte, F. S., Vallim, R. M., and de Mello, R. F. Multidimensional surrogate stability to detect data stream concept drift. *Expert Systems with Applications 87* (2017), 15–29.

[100] da Costa, F. G., Rios, R. A., and de Mello, R. F. Using dynamical systems tools to detect concept drift in data streams. *Expert Systems with Applications 60* (2016), 39–50.

[101] Dai, D. On highdimensional Mahalanobis distances. In *School of Business and Economics, Department of Economics and Statistics.* Doctoral Thesis. Linnaeus University, Spain, 2017.

[102] Dai, Q., Singleton, K. J., and Yang, W. Is regime-shift risk priced in the us treasury market? Tech. rep., Working paper, New York University and Stanford University, 2003.

[103] Dang, X. H., Lee, V., Ng, W. K., Ciptadi, A., and Ong, K. L. An em-based algorithm for clustering data streams in sliding windows. In *International conference on database systems for advanced applications* (2009), Springer, pp. 230–235.

[104] Das, R. T., Ang, K. K., and Quek, C. IeRSPOP: A novel incremental rough set-based pseudo outer-product with ensemble learning. *Applied Soft Computing Journal 46* (9 2016), 170–186.

[105] Dash, R., Dash, P. K., and Bisoi, R. A self adaptive differential harmony search based optimized extreme learning machine for financial time series prediction. *Swarm and Evolutionary Computation 19* (2014), 25–42.

[106] Davies, G. Regime changes in the financial markets, 2016.

[107] de Barros, R. S. M., and de Carvalho Santos, S. G. T. An overview and comprehensive comparison of ensembles for concept drift. *Information Fusion 52* (2019), 213–244.

[108] de Mello, R. F., Vaz, Y., Grossi, C. H., and Bifet, A. On learning guarantees to unsupervised concept drift detection on data streams. *Expert Systems with Applications 117* (2019), 90–102.

[109] De Prado, M. L. *Advances in financial machine learning.* John Wiley & Sons, 2018.

[110] Deboeck, G. *Trading on the edge: neural, genetic, and fuzzy systems for chaotic financial markets.* Wiley, 1994.

[111] Dehuri, S., Mohapatra, C., Ghosh, A., and Mall, R. Comparative study of clustering algorithms, 2006.

[112] Dellaert, F. The expectation maximization algorithm. Tech. rep., Georgia Institute of Technology, 2002.

[113] Derrac, J., Triguero, I., Garcia, S., and Herrera, F. Survey of new approaches on prototype selection and generation. Tech. rep., Technical Report, Department of Computer Science and Artificial Intelligence, 2011.

[114] Dias, J. G., Vermunt, J. K., and Ramos, S. Clustering financial time series: New insights from an extended hidden markov model. *European Journal of Operational Research 243*, 3 (2015), 852–864.

[115] Diler, A. Predicting direction of ISE national-100 index with back propagation trained neural network. *Journal of Istanbul Stock Exchange 7*, (25– 26) (2003), 65–81.

[116] Din, S. U., and Shao, J. Exploiting evolving micro-clusters for data stream classification with emerging class detection. *Information Sciences 507* (2020), 404–420.

[117] Ding, X., Zhang, Y., Liu, T., and Duan, J. Using structured events to predict stock price movement: An empirical investigation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (2014), pp. 1415–1425.

[118] Ditzler, G., Roveri, M., Alippi, C., and Polikar, R. Learning in non-stationary environments: A survey. *IEEE Computational Intelligence Magazine 10*, 4 (Nov. 2015), 12–25.

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

[120] Dror, G., Raijman, D., and Ulitsky, I. Analysis of gene expression data spring semester. Tel Aviv University, 3 2005. Lecture 6.

[121] Duda, P., Jaworski, M., and Rutkowski, L. On ensemble components selection in data streams scenario with reoccurring concept-drift. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)* (2017), IEEE, pp. 1–7.

[122] Eatwell, J., Milgate, M., and Newman, P. *Time series and statistics.* Springer, 1990.

[123] Efron, B. Bootstrap methods: another look at the jackknife. In *Breakthroughs in statistics*. Springer, 1992, pp. 569–593.

[124] Elwell, R., and Polikar, R. Incremental learning of concept drift in nonstationary environments. *IEEE transactions on neural networks 22*, 10 (10 2011), 1517–31.

[125] Engle, R. Risk and volatility: Econometric models and financial practice. *American economic review 94*, 3 (2004), 405–420.

[126] Engle, R. F. Autoregressive conditional heteroscedasticity with estimates of the variance of united kingdom inflation. *Econometrica: Journal of the econometric society* (1982), 987–1007.

[127] Fama, E. F. The behavior of stock-market prices. *The journal of Business 38*, 1 (1965), 34–105.

[128] Fama, E. F. Efficient Capital Markets: A Review of Theory and Empirical Work. *The Journal of Finance 25*, 2 (1970), 383.

[129] Ferreira, F. G., Gandomi, A. H., and Cardoso, R. T. Artificial intelligence applied to stock market trading: A review. *IEEE Access 9* (2021), 30898–30917.

[130] Figlewski, S. Forecasting volatility. *Financial markets, institutions & instruments 6*, 1 (1997), 1–88.

[131] Fišer, D., Faigl, J., and Kulich, M. Growing neural gas efficiently. *Neurocomputing 104* (2013), 72–82.

[132] Frias-Blanco, I., del Campo-Ávila, J., Ramos-Jimenez, G., Morales-Bueno, R., Ortiz-Diaz, A., and Caballero-Mota, Y. Online and non-parametric drift detection methods based on Hoeffding's bounds. *IEEE Transactions on Knowledge and Data Engineering 27*, 3 (2014), 810–823.

[133] Friedman, M. Nobel lecture: inflation and unemployment. *Journal of political economy 85*, 3 (1977), 451–472.

[134] Friesen, G. C., Weller, P. A., and Dunham, L. M. Price trends and patterns in technical analysis: A theoretical and empirical examination. *Journal of Banking and Finance 33*, 6 (Jun. 2009), 1089–1100.

[135] Fritzke, B. Growing cell structures—a self-organizing network for unsupervised and supervised learning. *Neural Networks 7*, 9 (1994), 1441–1460.

[136] Fritzke, B. A growing neural gas network learns topologies. In *Advances in Neural Information Processing Systems 7* (1995), MIT Press, pp. 625–632.

[137] Fritzke, B. Growing self-organizing networks-why? In *ESANN* (1996), vol. 96, Citeseer, pp. 61–72.

[138] Frömmel, M., and Lampaert, K. Does frequency matter for intraday technical trading? *Finance Research Letters 18* (2016), 177–183.

[139] Gama, J. *Knowledge Discovery from Data Streams*, 1st ed. Chapman & Hall/CRC, 2010.

[140] Gama, J., and Kosina, P. Tracking recurring concepts with meta-learners. In *Lecture Notes in Computer Science* (10 2009), vol. 5816 LNAI, Springer, Berlin, Heidelberg, pp. 423–434.

[141] Gama, J., and Kosina, P. Recurrent concepts in data streams classification. *Knowledge and Information Systems 40*, 3 (9 2014), 489–507.

[142] Gama, J., Medas, P., Castillo, G., and Rodrigues, P. Learning with drift detection. In *Advances in Artificial Intelligence – SBIA 2004: 17th Brazilian Symposium on Artificial Intelligence, Sao Luis, Maranhao, Brazil, September 29-Ocotber 1, 2004. Proceedings.* Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 286–295.

[143] Gama, J., Sebastiao, R., and Rodrigues, P. P. On evaluating stream learning algorithms. *Machine learning 90*, 3 (2013), 317–346.

[144] Gama, J. a., Žliobaitundefined, I., Bifet, A., Pechenizkiy, M., and Bouchachia, A. A survey on concept drift adaptation. *ACM Comput. Surv. 46*, 4 (Mar. 2014).

[145] Garcia, R., Luger, R., and Renault, E. Empirical assessment of an intertemporal option pricing model with latent variables. *Journal of Econometrics 116*, 1-2 (2003), 49–83.

[146] Gârleanu, N., and Pedersen, L. H. Efficiently inefficient markets for assets and asset management. *The Journal of Finance 73*, 4 (2018), 1663–1712.

[147] Gemaque, R. N., Costa, A. F. J., Giusti, R., and dos Santos, E. M. An overview of unsupervised drift detection methods. *WIREs Data Mining and Knowledge Discovery 10*, 6 (2020), e1381.

[148] Geva, T., and Zahavi, J. Empirical evaluation of an automated intraday stock recommendation system incorporating both market data and textual news. *Decision Support Systems 57* (Jan. 2014), 212–223.

[149] Ghesmoune, M., Lebbah, M., and Azzag, H. A new growing neural gas for clustering data streams. *Neural Networks 78* (2016), 36–50. Special Issue on "Neural Network Learning in Big Data".

[150] Ghoddusi, H., Creamer, G. G., and Rafizadeh, N. Machine learning in energy economics and finance: A review. *Energy Economics 81* (2019), 709–727.

[151] Gomes, H. M., Barddal, J. P., Enembreck, F., and Bifet, A. A Survey on Ensemble Learning for Data Stream Classification. *ACM Computing Surveys 50*, 2 (2017), 1–36.

[152] Gomes, H. M., Bifet, A., Read, J., Barddal, J. P., Enembreck, F., Pfharinger, B., Holmes, G., and Abdessalem, T. Adaptive random forests for evolving data stream classification. *Machine Learning* (6 2017), 1–27.

[153] Gomes, H. M., Read, J., and Bifet, A. Streaming random patches for evolving data stream classification. In *2019 IEEE International Conference on Data Mining (ICDM)* (2019), IEEE, pp. 240–249.

[154] Gomes, H. M., Read, J., Bifet, A., Barddal, J. P., and Gama, J. Machine learning for streaming data: state of the art, challenges, and opportunities. *ACM SIGKDD Explorations Newsletter 21*, 2 (2019), 6–22.

[155] Gomes, H. M., Read, J., Bifet, A., and Durrant, R. J. Learning from evolving data streams through ensembles of random patches. *Knowledge and Information Systems* (2021), 1–29.

[156] Gomes, J. B., Gaber, M. M., Sousa, P. A. C., and Menasalvas, E. Mining recurring concepts in a dynamic feature space. *IEEE Transactions on Neural Networks and Learning Systems 25*, 1 (1 2014), 95–110.

[157] Gomes, J. B., Menasalvas, E., and Sousa, P. A. C. Tracking recurrent concepts using context. In *Rough Sets and Current Trends in Computing: 7th International Conference, RSCTC 2010, Warsaw, Poland, June 28-30,2010. Proceedings*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 168–177.

[158] Gomes, J. B., Menasalvas, E., and Sousa, P. A. C. Learning recurring concepts from data streams with a context-aware ensemble. In *Proceedings of the 2011 ACM Symposium on Applied Computing - SAC '11* (New York, New York, USA, 2011), ACM Press, p. 994.

[159] Gonçalves Jr, P. M., Souto, R., and De Barros, M. RCD: A recurring concept drift framework. *Pattern Recognition Letters 34* (2013), 1018–1025.

[160] Gonçalves, P. M., de Carvalho Santos, S. G., Barros, R. S., and Vieira, D. C. A comparative study on concept drift detectors. *Expert Systems with Applications 41*, 18 (2014), 8144–8156.

[161] Grün, B. Model-Based Clustering. *Handbook of Mixture Analysis* (Feb. 2019), 157–192.

[162] Gu, X., Angelov, P. P., Ali, A. M., Gruver, W. A., and Gaydadjiev, G. Online evolving fuzzy rule-based prediction model for high frequency trading financial data stream. In *2016 IEEE Conference on Evolving and Adaptive Intelligent Systems (EAIS)* (5 2016), IEEE, pp. 169–175.

[163] Gu, X., Angelov, P. P., Kangin, D., and Principe, J. C. A new type of distance metric and its use for clustering. *Evolving Systems 8*, 3 (2017), 167–177.

[164] Guidolin, M., and Timmermann, A. An econometric model of nonlinear dynamics in the joint distribution of stock and bond returns. *Journal of Applied Econometrics 21*, 1 (Jan. 2006), 1–22.

[165] Halstead, B., Koh, Y. S., Riddle, P., Pears, R., Pechenizkiy, M., Bifet, A., Olivares, G., and Coulson, G. Analyzing and repairing concept drift adaptation in data stream classification. *Machine Learning* (2021), 1–35.

[166] Hamilton, J. D. A new approach to the economic analysis of nonstationary time series and the business cycle. *Econometrica: Journal of the econometric society* (1989), 357–384.

[167] Hamilton, J. D. Regime switching models. In *Macroeconometrics and time series analysis*. Springer, 2010, pp. 202–209.

[168] Hamilton, J. D. Macroeconomic regimes and regime shifts. *Handbook of macroeconomics 2* (2016), 163–201.

[169] Hammerschmid, R., and Lohre, H. Regime shifts and stock return predictability. *International Review of Economics and Finance 56* (Jul. 2018), 138–160.

[170] Hassan, M. R., Ramamohanarao, K., Kamruzzaman, J., Rahman, M., and Hossain, M. M. A hmm-based adaptive fuzzy inference system for stock market forecasting. *Neurocomputing 104* (2013), 10–25.

[171] Hearst, M., Dumais, S., Osman, E., Platt, J., and Scholkopf, B. Support vector machines. *Intelligent Systems and their Applications, IEEE 13*, 4 (Jul. 1998), 18–28.

[172] Henrique, B. M., Sobreiro, V. A., and Kimura, H. Literature review: Machine learning techniques applied to financial market prediction. *Expert Systems with Applications 124* (2019), 226–251.

[173] Hoeffding, W. Probability inequalities for sums of bounded random variables. In *The collected works of Wassily Hoeffding*. Springer, 1994, pp. 409–426.

[174] Hoepner, A. G., McMillan, D., Vivian, A., and Wese Simen, C. Significance, relevance and explainability in the machine learning age: an econometrics and financial data science perspective. *The European Journal of Finance 27*, 1-2 (2021), 1–7.

[175] Hoi, S. C., Sahoo, D., Lu, J., and Zhao, P. Online learning: A comprehensive survey. *Neurocomputing 459* (2021), 249–289.

[176] Hossain, A., and Nasser, M. Comparison of the finite mixture of ARMA-GARCH, back propagation neural networks and support-vector machines in forecasting financial returns. *Journal of Applied Statistics 38*, 3 (2011), 533–551.

[177] Hosseini, M. J., Ahmadi, Z., and Beigy, H. Pool and accuracy based stream classification: A new ensemble algorithm on data stream classification using recurring concepts detection. In *2011 IEEE 11th International Conference on Data Mining Workshops* (Dec. 2011), pp. 588–595.

[178] Hosseini, M. J., Ahmadi, Z., and Beigy, H. New management operations on classifiers pool to track recurring concepts. In *Lecture Notes in Computer Science* (2012), vol. 7448 LNCS, Springer, Berlin, Heidelberg, pp. 327–339.

[179] Hsu, M. W., Lessmann, S., Sung, M. C., Ma, T., and Johnson, J. E. Bridging the divide in financial market forecasting: machine learners vs. financial economists. *Expert Systems with Applications 61* (2016), 215–234.

[180] Hsu, S.-H., Hsieh, J. P.-A., Chih, T.-C., and Hsu, K.-C. A two-stage architecture for stock price forecasting by integrating self-organizing map and support vector regression. *Expert Systems with Applications 36*, 4 (2009), 7947–7951.

[181] Hu, H., Kantardzic, M., and Sethi, T. S. No Free Lunch Theorem for concept drift detection in streaming data classification: A review. In *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, no. 2 in 10. Wiley-Blackwell, Mar. 2020.

[182] Hu, Y., Liu, K., Zhang, X., Xie, K., Chen, W., Zeng, Y., and Liu, M. Concept drift mining of portfolio selection factors in stock market. *Electronic Commerce Research and Applications 14*, 6 (2015), 444–455.

[183] Huang, C. L., and Tsai, C. Y. A hybrid SOFM-SVR with a filter-based feature selection for stock market forecasting. *Expert Systems with Applications 36*, 2 PART 1 (2009), 1529–1539.

[184] Huang, W., Nakamori, Y., and Wang, S.-Y. Forecasting stock market movement direction with support vector machine. *Computers & Operations Research 32* (2005), 2513–2522.

[185] Hulten, G., Spencer, L., and Domingos, P. Mining time-changing data streams. *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '01* (2001), 97–106.

[186] Hynar, M., Burda, M., and Sarmanova, J. Unsupervised clustering with growing self-organizing neural network–a comparison with non-neural approach. In *Dateso* (2005), Citeseer, pp. 58–68.

[187] Hyndman, R. J., and Athanasopoulos, G. *Forecasting: principles and practice.* OTexts, 2018.

[188] Imbrea, A.-I. Automated machine learning techniques for data streams. *arXiv preprint arXiv:2106.07317* (2021).

[189] Jankowski, N., and Grochowski, M. *Comparison of Instances Seletion Algorithms I. Algorithms Survey.* Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 598–603.

[190] Jin, L. Comparing autocorrelation structures of multiple time series via the maximum distance between two groups of time series. *Journal of Statistical Computation and Simulation 85*, 17 (Nov. 2015), 3535–3548.

[191] Jorion, P. Predicting volatility in the foreign exchange market. *The Journal of Finance 50*, 2 (1995), 507–528.

[192] Jorion, P. 1. risk and thrnover in the foreign exchange market. In *The microstructure of foreign exchange markets*. University of Chicago Press, 2009, pp. 19–40.

[193] Käding, C., Rodner, E., Freytag, A., and Denzler, J. Fine-tuning deep neural networks in continuous learning scenarios. In *Computer Vision – ACCV 2016 Workshops* (Cham, 2017), C.-S. Chen, J. Lu, and K.-K. Ma, Eds., Springer International Publishing, pp. 588–605.

[194] Kale, S. 'I put my life savings in crypto': how a generation of amateurs got hooked on high-risk trading. *The Guardian* (2021).

[195] Kara, Y., Acar Boyacioglu, M., and Baykan, O. K. Predicting direction of stock price index movement using artificial neural networks and support vector machines: The sample of the Istanbul Stock Exchange. *Expert Systems with Applications 38*, 5 (2011), 5311–5319.

[196] Karnick, M., Ahiskali, M., Muhlbaier, M. D., and Polikar, R. Learning concept drift in nonstationary environments using an ensemble of classifiers based approach. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)* (6 2008), pp. 3455–3462.

[197] Kasabov, N., and Filev, D. Evolving intelligent systems: Methods, learning, applications. In *2006 International Symposium on Evolving Fuzzy Systems* (Sept 2006), pp. 8–18.

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

[199] Kim, K.-J. Financial time series forecasting using support vector machines. *Neurocomputing 55*, 1-2 (2003), 307–319.

[200] Kim, K.-J., and Han, I. Genetic algorithms approach to feature discretization in artificial neural networks for the prediction of stock price index. *Expert Systems with Applications 19*, 2 (2000), 125–132.

[201] Koh, Y. S., Huang, D. T. J., Pearce, C., and Dobbie, G. Volatility drift prediction for transactional data streams. In *2018 IEEE International Conference on Data Mining (ICDM)* (2018), IEEE, pp. 1091–1096.

[202] Kohonen, T. Self-organized formation of topologically correct feature maps. *Biological cybernetics 43*, 1 (1982), 59–69.

[203] Kohonen, T. Learning vector quantization. In *Self-organizing maps*. Springer, 1995, pp. 175–189.

[204] Kohonen, T. Self-organizing maps, spring series in information sciences, vol. 30, 1995.

[205] Kolter, J. Z., and Maloof, M. A. Dynamic Weighted Majority: An Ensemble Method for Drifting Concepts. *Journal of Machine Learning Research 8* (2007), 2755–2790.

[206] Korycki, Ł., and Krawczyk, B. Streaming decision trees for lifelong learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases* (2021), Springer, pp. 502–518.

[207] Krawczyk, B., Minku, L. L., Gama, J., Stefanowski, J., Woźniak, M., and Wó Zniak, M. Ensemble learning for data stream analysis: A survey. *Information Fusion 37* (2017), 132–156.

[208] Kristjanpoller, W., and Minutolo, M. C. A hybrid volatility forecasting framework integrating GARCH, artificial neural network, technical analysis and principal components analysis. *Expert Systems with Applications 109* (Nov. 2018), 1–11.

[209] Kritzman, M., Page, S., and Turkington, D. Regime shifts: Implications for dynamic strategies (corrected). *Financial Analysts Journal 68*, 3 (2012), 22–39.

[210] Kumar, M., and M., T. Forecasting Stock Index Movement: A Comparison of Support Vector Machines and Random Forest. *SSRN Electronic Journal* (2006).

[211] Kumar, M., and Thenmozhi, M. Forecasting Stock Index Movement: a Comparision of Support Vector Machines and Random Forest. *Forest, Indian Institute of Capital Markets 9th Capital Markets Conference Paper.* (2005), 1–16.

[212] Ładyżyński, P., Żbikowski, K., and Grzegorzewski, P. Stock trading with random forests, trend detection tests and force index volume indicators. In *Artificial Intelligence and Soft Computing: 12th International Conference, ICAISC 2013, Proceedings, Part II* (Berlin, Heidelberg, Jun. 2013), Springer Berlin Heidelberg, pp. 441–452.

[213] Lam, E., and Ossinger, J. Bitcoin ($BTC USD) Cryptocurrency Price Outlook: Futures a Warning to JPMorgan. *Bloomberg* (2021).

[214] Lam, M. Neural network techniques for financial performance prediction: integrating fundamental and technical analysis. *Decision Support Systems 37*, 4 (2004), 567–581. Data mining for financial decision making.

[215] Laney, D. 3D data management: Controlling data volume, velocity and variety. *META Group Research Note 6* (2001), 70.

[216] Laskov, P., Gehl, C., Krüger, S., and Müller, K.-R. Incremental support vector learning: Analysis, implementation and applications. *J. Mach. Learn. Res. 7* (Dec. 2006), 1909–1936.

[217] Leung, K. M. Naive bayesian classifier. *Polytechnic University Department of Computer Science/Finance and Risk Engineering 2007* (2007), 123–156.

[218] Li, A. W., and Bastos, G. S. Stock market forecasting using deep learning and technical analysis: a systematic review. *IEEE Access 8* (2020), 185232–185242.

[219] Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research 18*, 1 (2017), 6765–6816.

[220] Li, P., Wu, M., He, J., and Hu, X. Recurring drift detection and model selection-based ensemble classification for data streams with unlabeled data. *New Generation Computing* (2021), 1–36.

[221] Li, P., Wu, X., and Hu, X. Mining Recurring Concept Drifts with Limited Labeled Streaming Data. *ACM Transactions on Intelligent Systems and Technology 3*, 2 (2012), 1–32.

[222] Lim, C. P., and Harrison, R. F. Online pattern classification with multiple neural network systems: An experimental study. *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews 33*, 2 (5 2003), 235–247.

[223] Lin, Z. Modelling and forecasting the stock market volatility of SSE Composite Index using GARCH models. *Future Generation Computer Systems 79* (2018), 960–972.

[224] Linda, O., and Manic, M. Gng-svm framework - classifying large datasets with support vector machines using growing neural gas. In *Neural Networks, 2009. IJCNN 2009. International Joint Conference on* (Jun. 2009), pp. 1820–1826.

[225] Liu, C. F., Yeh, C. Y., and Lee, S. J. Application of type-2 neuro-fuzzy modeling in stock price prediction. *Applied Soft Computing Journal 12*, 4 (2012), 1348–1358.

[226] Liu, H., and juan Ban, X. Clustering by growing incremental self-organizing neural network. *Expert Systems with Applications 42*, 11 (2015), 4965–4981.

[227] Lo, A. W. The Adaptive Markets Hypothesis: Market Efficiency from an Evolutionary Perspective *. Tech. rep., MIT, 2004.

[228] Lo, A. W. Reconciling efficient markets with behavioral finance: the adaptive markets hypothesis. *Journal of investment consulting 7*, 2 (2005), 21–44.

[229] Lo, A. W., and MacKinlay, A. C. *A Non-Random Walk Down Wall Street.* Princeton University Press, Dec. 2011.

[230] Lo, A. W., Mamaysky, H., and Wang, J. Foundations of Technical Analysis: Computational Algorithms, Statistical Inference, and Empirical Implementation. *The Journal of Finance 55*, 4 (8 2000), 1705–1765.

[231] Lopez, M. A., Lobato, A. G. P., and Duarte, O. C. M. A performance comparison of open-source stream processing platforms. In *2016 IEEE Global Communications Conference (GLOBECOM)* (2016), IEEE, pp. 1–6.

[232] Louppe, G., and Geurts, P. Ensembles on random patches. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases* (2012), Springer, pp. 346–361.

[233] Lu, J., Liu, A., Dong, F., Gu, F., Gama, J., and Zhang, G. Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering 31*, 12 (2019), 2346–2363.

[234] Lughofer, E. *Evolving fuzzy systems-methodologies, advanced concepts and applications*, vol. 53. Springer, 2011.

[235] Lughofer, E., and Angelov, P. Handling drifts and shifts in on-line data streams with evolving fuzzy systems. *Appl. Soft Comput. 11*, 2 (Mar. 2011), 2057–2068.

[236] Lughofer, E., Cernuda, C., Kindermann, S., and Pratama, M. Generalized smart evolving fuzzy systems. *Evolving Systems 6*, 4 (2015), 269–292.

[237] MacQueen, J., et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability* (1967), no. 14 in 1, Oakland, CA, USA, pp. 281–297.

[238] Malkiel, B. *A Random Walk Down Wall Street.* WW Norton & Company, 1973.

[239] Malkiel, B. G. The efficient market hypothesis and its critics. *Journal of economic perspectives 17*, 1 (2003), 59–82.

[240] Manapragada, C., Webb, G. I., Salehi, M., and Bifet, A. Emergent and unspecified behaviors in streaming decision trees. *CoRR abs/2010.08199* (2020).

[241] Manku, G. S., and Motwani, R. Approximate frequency counts over data streams. In *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases* (2002), Elsevier, pp. 346–357.

[242] Martinetz, T., and Schulten, K. A "Neural-Gas" Network Learns Topologies. In *Artificial Neural Networks*, vol. 1. North-Holland, 1991, pp. 397–402.

[243] Masegosa, A. R., Martínez, A. M., Ramos-López, D., Langseth, H., Nielsen, T. D., and Salmerón, A. Analyzing concept drift: A case study in the financial sector. *Intelligent Data Analysis 24*, 3 (2020), 665–688.

[244] Maslov, A., Pechenizkiy, M., Žliobaitė, I., and Kärkkäinen, T. Modelling recurrent events for improving online change detection. In *Proceedings of the 2016 SIAM International Conference on Data Mining* (2016), SIAM, pp. 549–557.

[245] McMillan, D., Speight, A., and Apgwilym, O. Forecasting uk stock market volatility. *Applied Financial Economics 10*, 4 (2000), 435–448.

[246] McNicholas, P. D. Model-based clustering. *Journal of Classification 33*, 3 (2016), 331–373.

[247] Mehmood, E., and Anees, T. Challenges and solutions for processing real-time big data stream: A systematic literature review. *IEEE Access 8* (2020), 119123–119143.

[248] Menasalvas, E., Sousa, P. A. C., and Lisboa, U. N. D. Tracking Recurrent Concepts Using Context in Memory-constrained Devices. In *Fourth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies, UBICOMM 2010, Florence, Italy.* (2010).

[249] Mendes, C. A. T., Gattass, M., and Lopes, H. FGNG: A fast multi-dimensional growing neural gas implementation. *Neurocomputing 128* (2014), 328–340.

[250] Menhaj, M. B., and Jahanian, H. R. An analytical alternative for som. In *IJCNN'99. International Joint Conference on Neural Networks. Proceedings (Cat. No. 99CH36339)* (1999), vol. 3, IEEE, pp. 1939–1942.

[251] Mermillod, M., Bugaiska, A., and Bonin, P. The stability-plasticity dilemma: Investigating the continuum from catastrophic forgetting to age-limited learning effects. *Frontiers in psychology 4* (2013), 504.

[252] Miguel Ángel, A., João Bártolo, G., and Ernestina, M. Predicting recurring concepts on data-streams by means of a meta-model and a fuzzy similarity function. *Expert Systems With Applications 46* (2015), 87–105.

[253] Minku, L. L. Transfer learning in non-stationary environments. In *Learning from Data Streams in Evolving Environments*. Springer, 2019, pp. 13–37.

[254] Minku, L. L., White, A. P., and Yao, X. The impact of diversity on online ensemble learning in the presence of concept drift. *IEEE Transactions on Knowledge and Data Engineering 22*, 5 (2010), 730–742.

[255] Miranda, M. J., and Fackler, P. L. *Applied computational economics and finance.* MIT press, 2004.

[256] Mitsyn, S., and Ososkov, G. The growing neural gas and clustering of large amounts of data. *Optical Memory and Neural Networks 20*, 4 (2011), 260–270.

[257] Montiel, J., Halford, M., alaneu Alan, Saulo Martiello Mastelini mastelini, F., Bolmier, G., Vaysse, R., Zouitine, A., Murilo Gomes, H., Read, J., Bifet, A., Martiello Mastelini, S., Sourty, R., and Abdessalem, T. River: machine learning for streaming data in Python. *Journal of Machine Learning Research 22* (2021), 1–8.

[258] Montiel, J., Mitchell, R., Frank, E., Pfahringer, B., Abdessalem, T., and Bifet, A. Adaptive xgboost for evolving data streams. In *2020 International Joint Conference on Neural Networks (IJCNN)* (2020), IEEE, pp. 1–8.

[259] Münnix, M. C., Shimada, T., Schäfer, R., Leyvraz, F., Seligman, T. H., Guhr, T., and Stanley, H. E. Identifying States of a Financial Market. *Scientific Reports 2*, 1 (12 2012), 644.

[260] Murphy, J. J. *Technical analysis of the financial markets: A comprehensive guide to trading methods and applications.* Penguin, 1999.

[261] Nakano, M., Takahashi, A., and Takahashi, S. Bitcoin technical trading with artificial neural network. *Physica A: Statistical Mechanics and its Applications 510* (2018), 587–609.

[262] Namitha, K., and Santhosh Kumar, G. Learning in the presence of concept recurrence in data stream clustering. *Journal of Big Data 7*, 1 (Dec. 2020), 75.

[263] Neely, C. J., Rapach, D. E., Tu, J., and Zhou, G. Forecasting the equity risk premium: The role of technical indicators. *Management Science 60*, 7 (2014), 1772–1791.

[264] Neto, A., and Barreto, G. Opposite maps: Vector quantization algorithms for building reduced-set svm and lssvm classifiers. *Neural Processing Letters 37*, 1 (2013), 3–19.

[265] Nguyen, H.-L., Woon, Y.-K., and Ng, W.-K. A survey on data stream clustering and classification. *Knowledge and information systems 45*, 3 (2015), 535–569.

[266] Nguyen, H.-L., Woon, Y.-K., Ng, W.-K., and Wan, L. Heterogeneous ensemble for feature drifts in data streams. In *Advances in Knowledge Discovery and Data Mining* (Berlin, Heidelberg, 2012), P.-N. Tan, S. Chawla, C. K. Ho, and J. Bailey, Eds., Springer Berlin Heidelberg, pp. 1–12.

[267] Nooralishahi, P., Seera, M., and Loo, C. K. Online semi-supervised multi-channel time series classifier based on growing neural gas. *Neural Computing and Applications 28*, 11 (2017), 3491–3505.

[268] Nosratabadi, S., Mosavi, A., Duan, P., Ghamisi, P., Filip, F., Band, S. S., Reuter, U., Gama, J., and Gandomi, A. H. Data science in economics: comprehensive review of advanced machine learning and deep learning methods. *Mathematics 8*, 10 (2020), 1799.

[269] Nti, I. K., Adekoya, A. F., and Weyori, B. A. A systematic review of fundamental and technical analysis of stock market predictions. *Artificial Intelligence Review 53*, 4 (Apr. 2020), 3007–3057.

[270] Oza, N. C., and Russell, S. J. Online bagging and boosting. In *International Workshop on Artificial Intelligence and Statistics* (2001), PMLR, pp. 229–236.

[271] Ozbayoglu, A. M., Gudelek, M. U., and Sezer, O. B. Deep learning for financial applications: A survey. *Applied Soft Computing 93* (2020), 106384.

[272] Pan, S. J., and Yang, Q. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering 22*, 10 (2009), 1345–1359.

[273] Park, C.-H., and Irwin, S. H. What do we know about the profitability of technical analysis? *Journal of Economic surveys 21*, 4 (2007), 786–826.

[274] Park, S.-H., Lee, J.-H., Song, J.-W., and Park, T.-S. Forecasting change directions for financial time series using hidden Markov model. In *International Conference on Rough Sets and Knowledge Technology* (2009), Springer, pp. 184–191.

[275] Patel, J., Shah, S., Thakkar, P., and Kotecha, K. Predicting stock and stock price index movement using trend deterministic data preparation and machine learning techniques. *Expert Systems with Applications 42*, 1 (2015), 259–268.

[276] Patel, J., Shah, S., Thakkar, P., and Kotecha, K. Predicting stock market index using fusion of machine learning techniques. *Expert Systems with Applications 42*, 4 (3 2015), 2162–2172.

[277] Pavlidis, N. G., Plagianakos, V. P., Tasoulis, D. K., and Vrahatis, M. N. Financial forecasting through unsupervised clustering and neural networks. *Operational Research 6*, 2 (2006), 103–127.

[278] Pękalska, E., Duin, R. P., and Paclík, P. Prototype selection for dissimilarity-based classifiers. *Pattern Recognition 39*, 2 (2006), 189–208.

[279] Pesaranghader, A., and Viktor, H. L. Fast hoeffding drift detection method for evolving data streams. In *Joint European conference on machine learning and knowledge discovery in databases* (2016), Springer, pp. 96–111.

[280] Pettenuzzo, D., and Timmermann, A. Predictability of stock returns and asset allocation under structural breaks. *Journal of Econometrics 164*, 1 (Sep. 2011), 60–78.

[281] Pietruczuk, L., Rutkowski, L., Jaworski, M., and Duda, P. A method for automatic adjustment of ensemble size in stream data mining. In *2016 International Joint Conference on Neural Networks (IJCNN)* (2016), pp. 9–15.

[282] Piger, J. Econometrics: Models of regime changes, 2009.

[283] Pratama, M., Anavatti, S. G., Angelov, P. P., and Lughofer, E. PANFIS: A novel incremental learning machine. *IEEE Transactions on Neural Networks and Learning Systems 25*, 1 (1 2014), 55–68.

[284] Pratama, M., Anavatti, S. G., Joo, M., and Lughofer, E. D. pclass: An effective classifier for streaming examples. *IEEE Transactions on Fuzzy Systems 23*, 2 (Apr. 2015), 369–386.

[285] Pratama, M., Lu, J., Lughofer, E., Zhang, G., and Anavatti, S. Scaffolding type-2 classifier for incremental learning under concept drifts. *Neurocomputing 191* (2016), 304–329.

[286] Pratama, M., Lu, J., Lughofer, E., Zhang, G., and Er, M. J. Incremental Learning of Concept Drift Using Evolving Type-2 Recurrent Fuzzy Neural Network. *IEEE Transactions on Fuzzy Systems* (2016), 1–1.

[287] Pratama, M., Lughofer, E., Er, J., Anavatti, S., and Lim, C.-P. Data driven modelling based on Recurrent Interval-Valued Metacognitive Scaffolding Fuzzy Neural Network. *Neurocomputing 262* (2017), 4–27.

[288] Preis, T., Schneider, J. J., and Stanley, H. E. Switching processes in financial markets. *Proceedings of the National Academy of Sciences 108*, 19 (2011), 7674–7678.

[289] Prudent, Y., and Ennaji, A. An incremental growing neural gas learns topologies. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.* (2005), vol. 2, IEEE, pp. 1211–1216.

[290] Ramírez-Gallego, S., Krawczyk, B., García, S., Woźniak, M., and Herrera, F. A survey on data preprocessing for data stream mining: Current status and future directions. *Neurocomputing 239* (2017), 39–57.

[291] Rather, A. M., Agarwal, A., and Sastry, V. Recurrent neural network and a hybrid model for prediction of stock returns. *Expert Systems with Applications 42*, 6 (2015), 3234–3241.

[292] Read, J. Concept-drifting data streams are time series; the case for continuous adaptation. *arXiv preprint arXiv:1810.02266* (2018).

[293] Read, J., Rios, R. A., Nogueira, T., and de Mello, R. F. Data Streams Are Time Series: Challenging Assumptions. In *Lecture Notes in Computer Science* (Oct. 2020), vol. 12320 LNAI, Springer, Cham, pp. 529–543.

[294] Rice, J. R. The algorithm selection problem. In *Advances in computers*, vol. 15. Elsevier, 1976, pp. 65–118.

[295] Rish, I., et al. An empirical study of the naive Bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence* (2001), vol. 3, pp. 41–46.

[296] Robins, A. Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science 7*, 2 (1995), 123–146.

[297] Rodríguez-González, A., Ángel García-Crespo, Colomo-Palacios, R., Guldrís Iglesias, F., and Gómez-Berbís, J. M. Cast: Using neural networks to improve trading systems based on technical analysis by means of the rsi financial indicator. *Expert Systems with Applications 38*, 9 (2011), 11489–11500.

[298] Rokach, L. Ensemble-based classifiers. *Artificial Intelligence Review 33*, 1-2 (Feb. 2010), 1–39.

[299] Rossi, A. L. D., De Souza, B. F., Soares, C., de Leon Ferreira de Carvalho, A., and Ponce, C. A guidance of data stream characterization for meta-learning. *Intelligent Data Analysis 21*, 4 (2017), 1015–1035.

[300] Rundo, F., Trenta, F., di Stallo, A. L., and Battiato, S. Machine learning for quantitative finance applications: A survey. *Applied Sciences 9*, 24 (2019), 5574.

[301] Rüping, S. Incremental learning with support vector machines. In *icdm* (2001), IEEE, p. 641.

[302] Ryll, L., and Seidens, S. Evaluating the performance of machine learning algorithms in financial market forecasting: A comprehensive survey. *arXiv preprint arXiv:1906.07786* (2019).

[303] Sahoo, D., Pham, Q., Lu, J., and Hoi, S. C. Online deep learning: Learning deep neural networks on the fly. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence IJCAI 2018* (Jul. 2018), pp. 2660–2666.

[304] Sakthithasan, S., and Pears, R. Capturing recurring concepts using discrete fourier transform. *Concurrency and computation: practice and experience 28*, 15 (2016), 4013–4035.

[305] Samitas, A., Kampouris, E., and Kenourgios, D. Machine learning as an early warning system to predict financial crisis. *International Review of Financial Analysis 71* (2020), 101507.

[306] Sateesh Babu, G., Suresh, S., and Huang, G.-B. Meta-cognitive Neural Network for classification problems in a sequential learning framework. *Neurocomputing 81* (2011), 86–96.

[307] Schulmeister, S. Profitability of technical stock trading: Has it moved from daily to intraday data? *Review of Financial Economics 18*, 4 (2009), 190–201.

[308] Serbera, J. P., and Paumard, P. The fall of high-frequency trading: A survey of competition and profits. *Research in International Business and Finance 36* (2016), 271–287.

[309] Sestito, S., and Dillon, T. S. *Automated knowledge acquisition.* Prentice-Hall, Inc., 1994.

[310] Sethi, T. S., and Kantardzic, M. On the reliable detection of concept drift from streaming unlabeled data. *Expert Systems with Applications 82* (2017), 77–99.

[311] Shaker, A., and Hüllermeier, E. Survival analysis on data streams: Analyzing temporal events in dynamically changing environments. *International Journal of Applied Mathematics and Computer Science 24*, 1 (2014), 199–212.

[312] Shaker, A., and Hüllermeier, E. Recovery analysis for adaptive learning from non-stationary data streams: Experimental design and case study. *Neurocomputing 150*, Part A (2015), 250–264.

[313] Shintate, T., and Pichl, L. Trend Prediction Classification for High Frequency Bitcoin Time Series with Deep Learning. *Journal of Risk and Financial Management 12*, 1 (Jan. 2019), 17.

[314] Shirai, Y., and Tsujii, J.-i. *Artificial Intelligence: Concepts, techniques and applications.* John Wiley & Sons, Inc., 1984.

[315] Shleifer, A. *Inefficient markets: An introduction to behavioural finance.* Oup Oxford, 2000.

[316] Si, J., Lin, S., and Vuong, M.-A. Dynamic topology representing networks. *Neural Networks 13*, 6 (2000), 617–627.

[317] si Zhang, S., wei Liu, J., and Zuo, X. Adaptive online incremental learning for evolving data streams. *Applied Soft Computing 105* (2021), 107255.

[318] Sidhu, P., and Bhatia, M. A novel online ensemble approach to handle concept drifting data streams: diversified dynamic weighted majority. *International Journal of Machine Learning and Cybernetics 9*, 1 (2018), 37–61.

[319] Silva, B., Marques, N., and Panosso, G. Applying neural networks for concept drift detection in financial markets. In *CEUR Workshop Proceedings* (2012), vol. 960, pp. 43–47.

[320] Silva, J. A., Faria, E. R., Barros, R. C., Hruschka, E. R., Carvalho, A. C. d., and Gama, J. Data stream clustering: A survey. *ACM Computing Surveys (CSUR) 46*, 1 (2013), 1–31.

[321] Sirignano, J., and Cont, R. Universal features of price formation in financial markets: perspectives from deep learning. *Quantitative Finance* (2019).

[322] Smith, T., and Alahakoon, D. Growing self-organizing map for online continuous clustering. In *Foundations of Computational Intelligence Volume 4*. Springer, 2009, pp. 49–83.

[323] Stein, J. C. Efficient Capital Markets, Inefficient Firms: A Model of Myopic Corporate Behavior*. *The Quarterly Journal of Economics 104*, 4 (11 1989), 655–669.

[324] Straat, M., Abadi, F., Göpfert, C., Hammer, B., and Biehl, M. Statistical mechanics of on-line learning under concept drift. *Entropy 20*, 10 (2018), 775.

[325] Street, W. N., and Kim, Y. A streaming ensemble algorithm (sea) for large-scale classification. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining* (2001), ACM, pp. 377–382.

[326] Suárez Cetrulo, A. L., Burnham-King, L., Haugthon, D., and Carbajo, R. S. Wind Power Forecasting Using Ensemble Learning for Day-Ahead Energy Trading. SSRN Electronic Journal [Preprint] doi:10.2139/ssrn.4015233, Feb. 2022.

[327] Suárez-Cetrulo, A. L., and Cervantes, A. Estudio e implementación en MOA de nuevos algoritmos de aprendizaje incremental basados en Support Vector Machines. In *Department of Computer Science*. BSc. Thesis. Universidad Carlos III de Madrid, Spain, 2013.

[328] Suárez-Cetrulo, A. L., and Cervantes, A. An Online Classification Algorithm for Large Scale Data Streams: iGNGSVM. *Neurocomputing* (2017).

[329] Suárez-Cetrulo, A. L., Cervantes, A., and Quintana, D. Incremental Market Behavior Classification in Presence of Recurring Concepts. *Entropy 21*, 1 (Jan. 2019), 25.

[330] Suárez-Cetrulo, A. L., Kumar, A., and Miralles-Pechuán, L. Modelling the COVID-19 Virus Evolution With Incremental Machine Learning. In *The 29th Irish Conference on Artificial Intelligence and Cognitive Science 2021 (AICS 2021).* (2022), CEUR-WS.org, p. 12.

[331] Sun, Y., Tang, K., Zhu, Z., and Yao, X. Concept drift adaptation by exploiting historical knowledge. *IEEE transactions on neural networks and learning systems 29*, 10 (2018), 4822–4832.

[332] Syed, N. A., Huan, S., Kah, L., and Sung, K. Incremental learning with support vector machines. In *Proc. Int. Conf. on Artificial Intelligence (IJCAI-99)* (1999).

[333] Syed, N. A., Liu, H., Huan, S., Kah, L., and Sung, K. Handling concept drifts in incremental learning with support vector machines. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-99* (1999), ACM Press, pp. 317–321.

[334] Szadkowski, R., Drchal, J., and Faigl, J. Continually trained life-long classification. *Neural Computing and Applications* (2021), 1–18.

[335] Tang, H., Chiu, K.-C., and Xu, L. Finite mixture of ARMA-GARCH model for stock price prediction. In *Proceedings of the Third International Workshop on Computational Intelligence in Economics and Finance (CIEF'2003), North Carolina, USA* (2003), Citeseer, pp. 1112–1119.

[336] Tay, F. E., and Cao, L. Application of support vector machines in financial time series forecasting. *Omega 29*, 4 (8 2001), 309–317.

[337] Taylor, S. J. *Modelling financial time series.* World Scientific Publishing Company, 2008.

[338] Teixeira, L. A., and de Oliveira, A. L. I. A method for automatic stock trading combining technical analysis and nearest neighbor classification. *Expert Systems with Applications 37*, 10 (2010), 6885–6890.

[339] Tetlock, P. C., Saar-Tsechansky, M., and Macskassy, S. More than words: Quantifying language to measure firms' fundamentals. *The journal of finance 63*, 3 (2008), 1437–1467.

[340] Tieppo, E., Santos, R. R. d., Barddal, J. P., and Nievola, J. C. Hierarchical classification of data streams: a systematic literature review. *Artificial Intelligence Review* (2021), 1–40.

[341] Tomek, I., et al. An experiment with the edited nearest-neighbor rule. *Systems, Man and Cybernetics, IEEE Transactions on SMC-6*, 6 (Jun. 1976), 448–452.

[342] Tong, S., and Koller, D. Support vector machine active learning with applications to text classification. *J. Mach. Learn. Res. 2* (Mar. 2002), 45–66.

[343] Trajdos, P., and Kurzynski, M. Soft confusion matrix classifier for stream classification. In *International Conference on Computational Science* (2021), Springer, pp. 3–17.

[344] Triguero, I., Derrac, J., Garcia, S., and Herrera, F. Prototype generation for nearest neighbor classification: Survey of methods. In *Technical Report, Department of Computer Science and Artificial Intelligence*. University of Granada, Spain, Tech. Rep., 2011.

[345] Triguero, I., Derrac, J., Garcia, S., and Herrera, F. A taxonomy and experimental study on prototype generation for nearest neighbor classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 42*, 1 (2011), 86–100.

[346] Tsaih, R., Hsu, Y., and Lai, C. C. Forecasting S&P 500 stock index futures with a hybrid AI system. *Decision Support Systems 23*, 2 (6 1998), 161–174.

[347] Tsang, E., and Chen, J. *Detecting regime change in computational finance: data science, machine learning and algorithmic trading.* CRC Press, 2020.

[348] Tsay, R. S. *Analysis of financial time series*, vol. 543. John wiley & sons, 2005.

[349] Tsinaslanidis, P. E., and Kugiumtzis, D. A prediction scheme using perceptually important points and dynamic time warping. *Expert Systems with Applications 41*, 15 (2014), 6848–6860.

[350] Tsymbal, A. The Problem of Concept Drift: Definitions and Related Work. *Technical Report: TCD-CS-2004-15, Department of Computer Science Trinity College, Dublin* (2004).

[351] Urquhart, A., and McGroarty, F. Are stock markets really efficient? evidence of the adaptive market hypothesis. *International Review of Financial Analysis 47* (2016), 39–49.

[352] Vallim, R. M., and De Mello, R. F. Proposal of a new stability concept to detect changes in unsupervised data streams. *Expert systems with applications 41*, 16 (2014), 7350–7360.

[353] van Rijn, J. N., Holmes, G., Pfahringer, B., and Vanschoren, J. Algorithm selection on data streams. In *International Conference on Discovery Science* (2014), Springer, pp. 325–336.

[354] van Rijn, J. N., Holmes, G., Pfahringer, B., and Vanschoren, J. The online performance estimation framework: heterogeneous ensemble learning for data streams. *Machine Learning 107*, 1 (2018), 149–176.

[355] Vanschoren, J. Meta-learning: A survey. *arXiv preprint arXiv:1810.03548* (2018).

[356] Vanschoren, J., Van Rijn, J. N., Bischl, B., and Torgo, L. Openml: networked science in machine learning. *ACM SIGKDD Explorations Newsletter 15*, 2 (2014), 49–60.

[357] Vanstone, B., and Finnie, G. An empirical methodology for developing stockmarket trading systems using artificial neural networks. *Expert Systems with Applications 36*, 3, Part 2 (2009), 6668–6680.

[358] Vapnik, V., and Lerner, A. Y. Recognition of patterns with help of generalized portraits. *Avtomat. i Telemekh 24*, 6 (1963), 774–780.

[359] Varmuza, K., and Filzmoser, P. *Introduction to multivariate statistical analysis in chemometrics.* CRC press, 2016.

[360] Vella, V., and Ng, W. L. Enhancing risk-adjusted performance of stock market intraday trading with Neuro-Fuzzy systems. *Neurocomputing 141* (2014), 170–187.

[361] Veloso, B., Gama, J., Malheiro, B., and Vinagre, J. Hyperparameter self-tuning for data streams. *Information Fusion 76* (2021), 75–86.

[362] Villmann, T., and Bauer, H.-U. Applications of the growing self-organizing map. *Neurocomputing 21*, 1-3 (1998), 91–100.

[363] Wakabayashi, K., and Miura, T. Data stream prediction using incremental hidden Markov models. In *International Conference on Data Warehousing and Knowledge Discovery* (2009), Springer, pp. 63–74.

[364] Wares, S., Isaacs, J., and Elyan, E. Data stream mining: methods and challenges for handling concept drift. *SN Applied Sciences 1*, 11 (2019), 1–19.

[365] Webb, G. I., Hyde, R., Cao, H., Nguyen, H. L., and Petitjean, F. Characterizing concept drift. *Data Mining and Knowledge Discovery 30*, 4 (7 2016), 964–994.

[366] Webb, R. I., Ryu, D. D., Ryu, D. D., and Han, J. The price impact of futures trades and their intraday seasonality. *Emerging Markets Review 26* (2016), 80–98.

[367] Widmer, G., and Kubat, M. Learning in the presence of concept drift and hidden contexts. *Machine Learning 23*, 1 (1996), 69–101.

[368] Wilcoxon, F. Individual comparisons by ranking methods. *Biometrics Bulletin 1*, 6 (1945), 80–83.

[369] Wong, W., Xu, L., and Yip, F. Financial prediction by finite mixture GARCH model. In *Proceedings of International Conference on Neural Information Processing (ICONIP 98)* (1998), vol. 3, p. 1351–1354.

[370] Wu, O., Koh, Y. S., Dobbie, G., and Lacombe, T. Nacre: Proactive recurrent concept drift detection in data streams. In *2021 International Joint Conference on Neural Networks (IJCNN)* (2021), IEEE, pp. 1–8.

[371] Wu, O., Koh, Y. S., Dobbie, G., and Lacombe, T. Probabilistic exact adaptive random forest for recurrent concepts in data streams. *International Journal of Data Science and Analytics* (2021), 1–16.

[372] Wu, X., Li, P., and Hu, X. Learning from concept drifting data streams with unlabeled data. *Neurocomputing 92* (2012), 145–155.

[373] Xing, E. P., Ng, A. Y., Jordan, M. I., and Russell, S. Distance metric learning, with application to clustering with side-information. In *Advances in Neural Information Processing Systems* (2003).

[374] Xu, Y., Shen, F., and Zhao, J. An incremental learning vector quantization algorithm for pattern classification. *Neural Computing and Applications 21*, 6 (2012), 1205–1215.

[375] Yang, H., Huang, K., King, I., and Lyu, M. R. Localized support vector regression for time series prediction. *Neurocomputing 72*, 10-12 (2009), 2659–2669.

[376] Yang, Y., Wu, X., and Zhu, X. Mining in anticipation for concept change: Proactive-reactive prediction in data streams. *Data Mining and Knowledge Discovery 13*, 3 (2006), 261–289.

[377] Yao, J., Tan, C. L., and Poh, H.-L. Neural Networks for Technical Analysis: a Study on Klci. *International Journal of Theoretical and Applied Finance 02*, 02 (4 1999), 221–241.

[378] Yu, Y., Duan, W., and Cao, Q. The impact of social and conventional media on firm equity value: A sentiment analysis approach. *Decision support systems 55*, 4 (2013), 919–926.

[379] Zhang, S.-s., Liu, J.-w., and Zuo, X. Adaptive online incremental learning for evolving data streams. *Applied Soft Computing 105* (2021), 107255.

[380] Zhang, X., Li, Y., Wang, S., Fang, B., and Philip, S. Y. Enhancing stock market prediction with extended coupled hidden Markov model over multi-sourced data. *Knowledge and Information Systems 61*, 2 (2019), 1071–1090.

[381] Zhao, P., Cai, L. W., and Zhou, Z. H. Handling concept drift via model reuse. *Machine Learning 109*, 3 (2020), 533–568.

[382] Zheng, J., Shen, F., Fan, H., and Zhao, J. An online incremental learning support vector machine for large-scale data. *Neural Computing and Applications 22*, 5 (2013), 1023–1035.

[383] Zheng, X., Li, P., Hu, X., and Yu, K. Semi-supervised classification on data streams with recurring concept drift and concept evolution. *Knowledge-Based Systems 215* (2021), 106749.

[384] Žliobaitė, I., Bifet, A., Read, J., Pfahringer, B., and Holmes, G. Evaluation methods and decision theory for classification of streaming data with temporal dependence. *Machine Learning 98*, 3 (2014), 455–482.

[385] Žliobaite, I., Budka, M., and Stahl, F. Towards cost-sensitive adaptation: When is it worth updating your predictive model? *Neurocomputing 150*, Part A (2015), 240–249.

[386] Zubaroğlu, A., and Atalay, V. Data stream clustering: a review. *Artificial Intelligence Review 54*, 2 (Feb. 2021), 1201–1236.

This page is intentionally left blank.