Models and Algorithms for Concept Drift Detection, Adaptation, and Resolution in Streaming Data

Ali Alizadeh Mansouri

A Thesis
In the Department
of
Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy (Computer Science) at
Concordia University
Montréal, Québec, Canada

December 2024

CONCORDIA UNIVERSITY
SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By: Mr. Ali Alizadeh Mansouri

Entitled: Models and Algorithms for Concept Drift Detection, Adaptation, and Resolution in Streaming Data

and submitted in partial fulfillment of the requirements for the degree of

Doctor Of Philosophy

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

Dr. Geoffrey C. Dover — Chair

Dr. Benjamin C. M. Fung — External Examiner

Dr. Abdelwahab Hamou-Lhadj — Arm's Length Examiner

Dr. Tristan Glatard — Examiner

Dr. Adam Krzyzak — Examiner

Dr. Nematollaah Shiri — Thesis Supervisor (s)

Approved by _____

Chair of Department or Graduate Program Director

2024-12-16

Date of Defence

Mourad Debbabi — Dean, Faculty of Engineering and Computer Science

Abstract

Models and Algorithms for Concept Drift Detection, Adaptation, and Resolution in Streaming Data

Ali Alizadeh Mansouri, Ph.D.
Concordia University, 2025

The evolution of streaming data during long periods of time presents significant challenges for maintaining the accuracy and efficiency of predictive models due to concept drift—where changes in data distribution can lead to performance degradation. In this research, we study the problems of concept drift detection (CDD) and adaptation (CDA). Unlike traditional approaches that treat CDD and CDA independently and in isolation, often under non-streaming, static conditions, we propose a novel methodology based on multivariate vector error-correction analysis of feature importance measures (FIMs). The FIMs provided a solid foundation that allowed us to reformulate concept drift detection and adaptation in streaming data.

We additionally introduce, formalize, and develop the notion of concept drift resolution (CDR) as an innovative model preference technique. This solution further enhances the overall performance by effectively using multiple models undergoing concept drift, including the main learner and the proposed CDA model. The results of our numerous experiments and analyses indicate that the proposed CDD method significantly reduces computation time, particularly in applications experiencing abrupt drifts, while our CDA model delivers notable improvements in prediction accuracy and F1 score on both gradual drift and abrupt drift datasets, outperforming existing methods on varying drift rates and characteristics of concept drift.

By utilizing FIMs as a common basis, we develop a unified framework that integrates CDD, CDA, and CDR tasks, thus bridging the gap between detection and adaptation. Extensive experiments validate the effectiveness of our proposed methods, demonstrating their applicability in various real-world and synthetic benchmark datasets. This work not only advances the understanding of concept drift in streaming data but also provides a general solution framework that balances performance with interpretability, thus paving the way for development of more reliable and explainable data-driven applications and systems.

Acknowledgments

I would like to express my deepest gratitude to my advisor, Dr. Nematollaah Shiri, for his invaluable guidance, patience, and support throughout my research. Your insightful feedback and encouragement have been a key factor in shaping this work, and I am profoundly thankful for your mentorship.

I extend my sincere appreciation to the members of my thesis committee for their valuable insights, comments, and feedback at different stages of my program. Your expertise and thoughtful suggestions have meaningfully contributed to the quality and direction of my research.

I am also deeply thankful to my coworkers at Intact—the *Explorers*—especially Laurent, Stéphane, HamidReza, and Derek for creating a positive, supportive and collaborative environment during the final stages of my studies. The hands-on experience I gained with real-world problems, techniques, and applications while working with you proved invaluable in shaping both my research and professional growth.

A special heart-felt thanks to my family—my mom, Roya, my dad, Farzad, and my sister, Maryam—for their unwavering love, patience, belief, and confidence in me. Your constant support and encouragement have been the light in the darkest moments, and I am forever grateful for your presence in my life. Mom, this would not have been possible without your sacrifices, love, and prayers. I dedicate this work to you.

Lastly, I also dedicate this work to the memory of my late grandmother *RuhAngiz*, who was a source of strength, inspiration, and warmth throughout my life. Her love and wisdom continue to guide me. This accomplishment is as much hers as it is mine.

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

AAI	average accuracy improvement 48, 49
ADF	augmented Dickey-Fuller xi, 61--63, 75, 83--87
AFI	average F-score improvement 48, 49, 51
AI	artificial intelligence 7
ATC	air traffic controller 89, 90
CD	concept drift viii, xi, 2, 3, 5, 10, 11, 15, 18--36, 38, 42, 48, 53--58, 60, 61, 66, 67, 73--76, 86--90, 92, 97, 119
CDA	concept drift adaptation iii, vii--x, xii, 2--5, 10, 11, 18, 23, 26, 31--36, 52, 53, 55, 76, 79, 81, 82, 87--90, 96, 97, 99--108, 117, 118, 120
CDD	concept drift detection iii, vii--x, xii, 2--5, 10, 11, 15, 18, 21--32, 34--39, 42, 44, 47, 48, 51--53, 55, 73, 76, 79, 84, 85, 87--90, 92, 96, 97, 99--108, 117, 118, 120
CDD&A	concept drift detection and adaptation v, viii, 1--6, 11, 17, 18, 20, 23--26, 29, 32, 35, 36, 53--56, 60, 74, 75, 79, 88--90, 96--98, 117--120
CDR	concept drift resolution iii, vii, ix, x, xii, 4, 5, 10, 11, 36, 76, 88--90, 92, 96--118, 120
CPD	change-point detector 42
DCS	dynamic classifier selection 22
DevOps	development and operations 7
DMS	driver monitoring system 89, 90
DSMS	data stream management system ix, 7, 8, 17, 18, 21, 25, 26, 34, 35, 56, 57
ECSMiner	ECSMiner 24, 25, 31
ELEC	electricity dataset ix, 44--46, 49, 50, 62--65, 67, 69--72, 98, 100--107, 109--116

ELM	extreme learning machine 2, 29, 30, 36, 37, 39, 41, 53, 54
EnLAuDD	Ensemble Learning Augmented Drift Detection ix, xi, 4, 5, 38, 40, 41, 43--53
FIM	feature importance measure iii, ix, xi, 3--5, 36, 54--61, 64--76, 79--82, 88, 97, 117, 119, 120
FNR	false negative rate 35
FPR	false positive rate 35
GBDT	gradient-boosting decision tree ix, 54--59, 66, 73, 79, 81, 99
GMM	Gaussian mixture model 24, 28
IoT	Internet-of-Things 8
kDD	knowledge discovery in databases 6, 7, 11
Learn^{++}.NSE	Learn^{++}.NSE x, xii, 22, 33, 97, 108--117
ML	machine learning 7, 33--35
MOA	Massive Online Analysis 44
MSE	mean squared error 41, 79, 80
NOAA	weather dataset ix, 44--46, 49, 50, 62--65, 67--72, 98, 100--116
OS-ELM	online sequential extreme learning machine ix--xii, 29, 30, 37--39, 41, 43, 48--51, 97, 108--117
PDF	probability density function 12
RBM	restricted Boltzmann machine 22, 30
RBM-IM	RBM for IMbalanced data streams 22, 30
RCB	rotating checkerboard iii, ix, xi, 43--46, 48--50, 61, 62, 68, 97, 108
RCB-C	RCB-constant 43, 45, 49, 61--66, 69--72, 97, 100--107, 109--116
RCB-E	RCB-exponential 43, 45, 47, 49, 61--66, 69--72, 97, 100--107, 109--116
RCB-P	RCB-pulse 43, 45, 47, 49, 61--65, 67, 69--72, 97, 99--107, 109--116
RCB-S	RCB-sinusoidal 43, 45, 47, 49, 61--65, 67, 69--72, 97, 100--107, 109--116
ROC AUC	area under the receiver operating characteristic curve ix, x, 98, 102, 105, 108, 111, 117

SEA Streaming Ensemble Algorithm ix, xi, 43, 45--51, 61, 62, 66, 68, 73, 97, 100--105, 108--114

SEA-1 Streaming Ensemble Algorithm 43, 45, 49, 61, 63--65, 69--72, 97, 98, 106, 107, 115, 116

SEA-2 Streaming Ensemble Algorithm 43, 45, 49, 61, 63--65, 69--72, 97, 98, 106, 107, 115, 116

SEA-3 Streaming Ensemble Algorithm 43, 45, 49, 61, 63--65, 69--72, 98, 106, 107, 115, 116

SMOTE synthetic Minority Over-sampling Technique 22

SPE stream processing engine 8, 26, 34

STL seasonal-trend decomposition using locally estimated scatterplot smoothing (LOESS) 85

SVM support vector machine 117

TCN Temporal Convolutional Network 32

TNR true negative rate 35

TPR true positive rate 17, 35

VAR vector autoregression 59--65, 67, 121

VEC vector error-correction 59--61, 64, 65, 67, 75, 121

If we lived on a planet where nothing ever changed [...] there'd be no impetus for science. And if we lived in an unpredictable world where things changed in random or complex ways [...] again, there'd be no such thing as science. But we live in an in-between universe where things change, all right but according to patterns, rules or as we call them, laws of nature.

—Carl Sagan, *Cosmos: A Personal Voyage (1980)*

Chapter 1

Introduction

With continuous advances in computing technologies, mining techniques are gaining increased popularity for discovering hidden patterns in streaming data, where unbounded data arrives at fast speeds for extended periods of time. Emerging streaming data analytics must detect pattern changes in the distribution of one or more variables that may affect applications' performance and adapt accordingly when such changes occur. This is referred to as *concept drift* [35, 61, 9]. If not detected and dealt with, concept drift may result in deterioration of the learner's inference or prediction performance. Examples of real-life applications showing this behavior include healthcare [82, 9], industrial sensor grids [89, 9], environmental sensing [21, 70], smart cities and homes, [69], network infrastructure monitoring [84], business, e-commerce and insurance [1, 2, 15, 63, 9], and finance [42, 74, 9], to name a few.

As a simple example, consider a classifier whose task is to discriminate apples from oranges on a conveyor belt. The classifier is trained on samples of apples and oranges of certain cultivars. If the training data consists solely of red and yellow *Honey Crisp* apples, but during testing, green *Granny Smith* apples are introduced, it is likely that the classifier's accuracy will decrease. An expert noticing the changes will need to stop the classification task, re-train the classifier by including new green apples in the train set, and resume the classification task once a new model is ready. Assuming that the test data is an unbounded stream of apples on a conveyor belt, the concept drift detection and adaptation (CDD&A) tasks performed by the expert may result in classification errors on some data, cause delay in re-training and re-classification of data, or result in loss of test data (i.e., apples) due to shortage of temporary storage.

A major challenge in tackling the CDD&A problems is selection and analysis of the available information. There have chronologically been three major categories of techniques based on the kind of *information used* for CDD&A tasks. Data feature analysis methods, such as [28,

51], focus on independent features in raw data. This, however, has the risk of overlooking the association between the predictor and response. Next generation of CDD&A solutions, such as [81], have relied on the learner's predictability performance evaluation and feedback. They are more accurate than the raw data analyzers, but still do not consider possible correlations between the features and target variables. Both of these categories of techniques suffer from using the same learner for the main classification and the CDD&A tasks. This potentially limits their effectiveness because the learner designated for CDD&A might perform poorly on its main classification task, and vice versa.

More recent CDD&A techniques use auxiliary models that run in parallel to the learner. These models have the advantages of being light-weight, decoupled, designed for the CDD&A task while respecting the stream processing requirements. Furthermore, they can analyze the evolving correlation of features and target variables more effectively than previous techniques. For example, Yang et al. [97] used an online sequential extreme learning machine (ELM) model [60] for concept drift detection (CDD).

Conventionally, CDD&A problems have often been studied independently and separately, in particular, for streaming data where data is often non-stationary. As a result, existing solutions tend to prioritize performance over explainability/interpretability or vice versa.

Therefore, in this research we look for answers to the following questions:

1. What would be a suitable model of concept drift (CD) that provides information beneficial to both CDD and CDA problems in *a single unified setting*?
2. Can this model balance the trade-off between effectiveness and efficiency in detecting and adapting to concept drifts in streaming data?
3. Would the information provided by this model be a faithful representation of the evolving correlation between the features and target?
4. How can streaming data applications undergoing concept drifts make use of this information to improve their performance without compromising interpretability and explainability?
5. Given multiple solutions addressing the CDD&A problems which may perform differently over the data stream, how is it possible to *resolve* CD by leveraging information from each solution thus maintain optimal performance of the application overall?

The first question is concerned with possibility and potential benefits of studying both CDD and concept drift adaptation (CDA) problems from a common perspective, and if we could find a

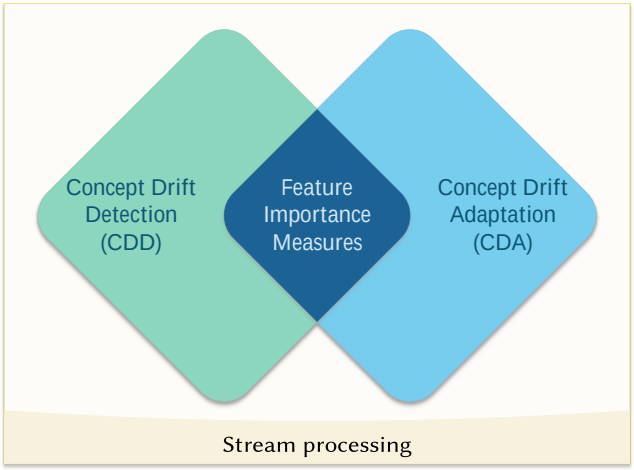


Figure 1.1: Unified view of the CDD&A problems.

basis on which to build solutions for each problem. The second research question addresses data stream processing application's requirements. The third question is concerned about effectiveness of a viable solution to the CDD&A problems. The fourth question pertains to explainability of the evolving nature of data and the model's response to it, which is a key requirement in many applications, such as healthcare, finance, and insurance. Lastly, the fifth research question is about the synergy between the CDD and CDA problems, and how they can be used to resolve the CD problem for the most effective and efficient solution.

In this research, we focused on feature analysis and study both CDD and CDA problems as different facets of the same problem in a unified and generic framework. This is done by providing a basis to address several sub problems involved in concept drift for streaming data. Therefore, with the aim of looking for such a common basis as the source of information for CDD and CDA, we study models that capture the dynamic relationship between raw data features and target. This unified view of the CDD&A problems is shown in Figure 1.1.

Furthermore, we propose novel methods for CDD and CDA problems based on the multivariate vector error-correction analysis of feature importance measures (FIMs). We show that the proposed methods are effective in detecting and adapting to concept drifts in streaming data. Moreover, we propose and demonstrate the synergy between the CDD and CDA problems by using the same basis for both problems in a unified setting. Additionally, we propose a novel method for concept drift resolution (CDR) based on the recent performance of the main learner improved by the proposed CDD technique, and performance of the proposed CDA technique. We also present the results of our extensive experiments carried out to validate the proposed

methods.

The contributions of this research are as follows:

1. We propose a novel methodology for studying the CDD&A problems based on the multivariate vector error-correction analysis of feature importance measures (FIMs) of raw data features as the foundation for the proposed methods.

2. We propose a unified framework, called Amytis [1] , for concept drift analysis, detection, adaptation, and resolution in streaming data based on our novel methodology for analysis of FIMs, which is the first of its kind to the best of our knowledge.

3. We propose a robust and novel model for concept drift adaptation (CDA) that is effective, efficient, and explainable in adapting to concept drifts in streaming data.

4. We propose two novel techniques for concept drift detection (CDD), one based on bagging and boosting, and the other by leveraging the FIMs-based model. We show effectiveness of both techniques in detecting concept drifts in streaming data while maintaining interpretability.

5. We propose a novel concept drift resolution (CDR) technique based on the recent performance of the main learner improved by the proposed CDD technique, and performance of the proposed CDA technique, which is the first of its kind to the best of our knowledge as of the time of writing.

6. As a by-product of our research, we provide an up-to-date review and taxonomy of the literature on CDD&A problems.

7. As a proof of concept, we develop a flexible and extensible software framework for concept drift analysis and data stream processing, used in this study to perform the experiments carried out.

The rest of the thesis is organized as follows. In chapter 2, we provide the background, formal problem statement, review and taxonomy of the literature on CD, CDD, and CDA problems. In

[1] *Amytis* was a Median princess, traditionally identified as the daughter of the Median king, Astyages, and the maternal aunt of Cyrus the Great, founder of the Achaemenid Empire. She is most well-known for her marriage to Nebuchadnezzar II, the king of Babylon, in the 6th century BCE. Amytis is often associated with the legendary Hanging Gardens of Babylon, one of the Seven Wonders of the Ancient World. According to some accounts, Nebuchadnezzar II built the gardens to remind Amytis of the green hills and valleys of her homeland in Media, as she missed them in the flat, arid landscape of Babylon. Her story is a blend of history and legend, but she remains an iconic figure linked to one of history's most famous architectural marvels. The name *Amytis* is derived from the Old Persian word *Umati*, meaning "having good thought".

chapter 3, we present the proposed Ensemble Learning Augmented Drift Detection (ENLAUDD) technique for CDD. In chapter 4, we present our novel methodology for FIMS analysis and evaluation to study their correlation with the performance of the application and assess their viability as the basis for the proposed methods.

In chapter 5, we present AMYTIS, a unified framework for CDD&A problems, and develop the CDA and CDD techniques based on the FIMS analysis foundation. Additionally, we present the proposed CDR technique based on the recent performance of the main learner improved by the proposed CDD technique, and performance of the proposed CDA technique. We will also present the experimental setup, the datasets used, the evaluation metrics, and the results of the experiments. In chapter 6, we provide a summary of the research followed by a discussion and conclusions, and the future work.

... one glance at [a book] and you're inside the mind of another person, maybe somebody dead for thousands of years. [...] Writing is perhaps the greatest of human inventions, binding together people who never knew each other, citizens of distant epochs. Books break the shackles of time.

—Carl Sagan, *Cosmos: A Personal Voyage (1980)*

Chapter 2

Background and Related Work

In this chapter, we provide a background on the streaming data analytics and the concept drift problem. We first briefly review the data mining life cycle and pipeline in section 2.1.1, followed by a formal overview of the concept drift problem in section 2.1.2. We will then review the literature on the concept drift problem, a taxonomy of the methods and challenges in concept drift detection and adaptation in section 2.2.

2.1. Definitions

In this section, we review concepts and techniques related to the CDD&A problems over data streams, starting with basics of the data mining life cycle and pipeline.

2.1.1. Data mining life cycle and pipeline

A data mining and knowledge discovery in databases (KDD) task consists of two main phases: the data mining life cycle, and the data pipeline. For this, we use the terminology of Chapman et al. [22] for different phases of the KDD life cycle, and refer to the data mining step of Reinartz [76] as *modeling*.

2.1.1.1 Data mining life cycle

A data mining and KDD project starts with the KDD life cycle, which consists of seven phases [76, 22]. Figure 2.1 summarizes a generic representation of a data mining project. The seven phases of the life cycle are depicted in the top part of the figure. The first phase, shown on the top and adopted from [76, 22, 106], is the data mining life cycle. The artifacts (models, etc.) of this phase are then deployed inside a data pipeline, shown at the bottom and adopted from [86], which includes a data stream management system architecture.

As the first phase, *business understanding* focuses on the project's objectives, requirements, questions to be answered, the project's life cycle, and deployment feasibility. The goal of *data understanding* is familiarity with data by gaining initial insights into the data such as available quality, quantity, and attributes of data based on the objectives identified in the first phase. *Data preparation* aims at preparing and building the final dataset for the subsequent steps, and includes tasks such as data selection, cleaning, transformation, integration, etc. *Data explorations*'s objective is to gain better insights into data by using descriptive statistics and visualization tools. While initial statistics and insights are obtained in *data understanding* phase, the output dataset from the *data preparation* phase can help business analysts and engineers to better understand the data as well as to form hypotheses on data patterns and knowledge to be extracted. In the *modeling* phase, statistical and algorithmic tools and techniques are used to build abstract models of data. Moreover, the parameters of these models are fine-tuned, and different models and techniques are experimented with if more than one has been deemed appropriate for the current data mining task in one of the previous steps. In the *evaluation* phase, the models built as well as the entire data mining process are evaluated from a business perspective to ensure that the results answer the business questions and assess if they can be deployed in a data mining pipeline. The results of a data mining life cycle are finally *deployed* either in the form of reports and presentations, or incorporated into a repetitive data mining pipeline [1]. In case of the latter, automated or manual continuous monitoring and evaluation of the results and extracted knowledge becomes a crucial part of the deployment phase in data mining pipelines involving stream processing, because changes to data patterns may occur over time.

Each phase in the life cycle may require moving back to the previous phases and re-evaluating the decisions made, steps taken, and tasks performed. For example, the results of the evaluation phase may require revisions on questions asked in business understanding, tasks performed in

[1] Deploying data mining models in a production or *prod* environment, also referred to as *operationalization*, is a crucial step in the data mining life cycle. However, it is not always considered as a separate phase in the life cycle, and involves work from the ML, DevOps, and AI engineering teams rather than the data scientists. Therefore, it is outside the scope of this thesis.

data preparation, or the techniques used in modeling. This is shown in Figure 2.1 by backward arrows between phases.

In case of a streaming big data analytics project, the models resulting from the data mining life cycle process are often deployed in the stream processing engine (SPE) of the data stream management system (DSMS), or the parallel and high performance computing components of the cloud in a data mining pipeline. The bottom part of Figure 2.1 presents a generic view of such a data mining pipeline.

The pipeline consists of three tiers. In *data acquisition*, raw data of various often heterogeneous sources are generated at large volumes, fast rates, and virtually unbounded. Examples of such sources of data are Internet-of-Things (IoT) devices; arrays of sensors deployed in a building, city, environment, or manufacturing; smart vehicles; healthcare devices; user interactions; social media; and financial transactions. Location-wise, data producers are usually located far from the cloud where the data mining results and models are often developed and deployed.

These vast unbounded heterogeneous streams of data are then integrated using message brokers such as *Apache Kafka* [7] which feed the streams to a DSMS on the next step.

The heart of the streaming tier is a data stream management system (DSMS), which receives streams of data from the message broker, and processes the streams of a certain window size of data on the working memory.

The stream processing engine (SPE) of the DSMS contains a stream processor and a data mining processor. The stream processor performs tasks such as cleaning and filtering, which can be considered as continuous standing queries from a database perspective. Such tasks are usually computationally inexpensive and are the focus of the stream processing data analytics research community as well as the industries. The specific processes can be a partial result of the data mining life cycle phase. For example, the cleanup task can be an automated version of the cleanup performed by the data experts in the data preparation phase. The data mining component of SPE performs data analytics such as prediction (classification, regression, etc.) or inference (clustering, frequent item-set mining, anomaly detection, etc.). As mentioned above, the models are usually the main outcomes of the life cycle. Examples of SPEs are *Azure IoT Edge* [11] for the edge, or *Apache Flink* [6] for the cloud.

SPE can also interact with a real-time limited storage [105], such as *Redis* [75]. The entire stream processing phase can be executed either at the edge/core of the network, on the cloud, or both.

Data streams are then sent to the cloud for offline processing. Parallel computing frameworks and paradigms, such as *Apache Spark* [8] and map-reduce respectively, and high-performance computing frameworks and techniques, such as *AWS HPC* [10] and deep learning respectively, are used to extract higher-level patterns from the collected data over large periods of time.

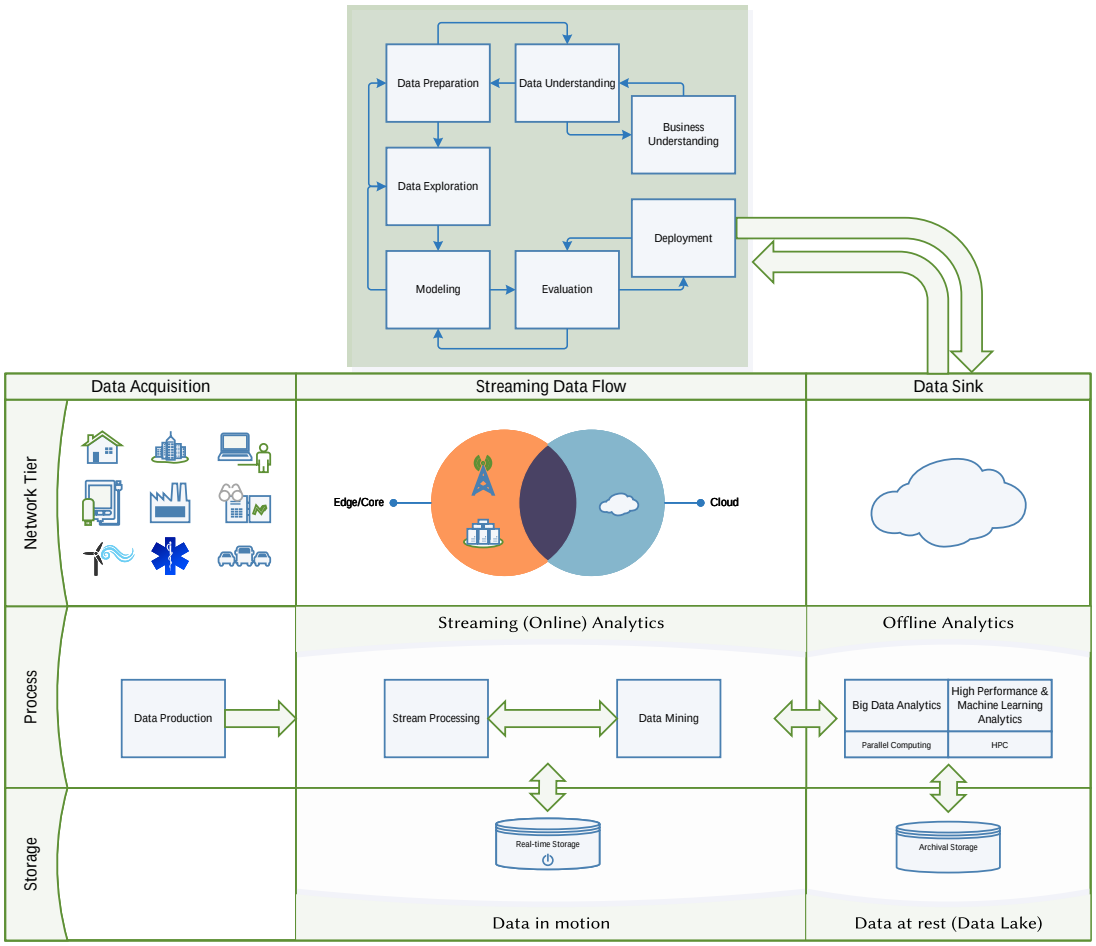


Figure 2.1: A high-level view of a data mining process.

Such streaming data pipeline may repeat for extended periods of time. However, data is likely to change over time, and so do the patterns and the knowledge inherent in data that the models try to capture. In other words, the patterns in data running through the pipeline which the deployed analytical methods try to extract may become different than those used to train the model earlier. As a result, the performance of the models deteriorate over time when such changes occur. As mentioned previously, this is referred to as concept drift.

To mitigate the adverse effects of concept drift, conventional data mining applications require continuous monitoring and evaluation of the deployed models. This usually has to be done by a domain expert or a data scientist, especially in case of more complicated modeling techniques. This could lead to detection and/or adaptation of the models to the new patterns in data. In mainstream industry, this means that the data mining life cycle has to be repeated fully or partially. The results of the streaming or offline analytical components are used to either perform another complete cycle of data mining in the cloud, or only start at certain phases. Existing models may be re-evaluated, fine-tuned if necessary, or new ones may be developed to be re-deployed to the pipeline. This is shown using the back-and-forward arrows between the life cycle and pipeline phases. Notice how the deployment phase of the life cycle directly leads to the pipeline, but the pipeline (on the cloud side) leads back to the entire life cycle, and not just to the deployment phase, because the life cycle may be re-started at any phase.

This issue becomes more intricate if the application employs an ensemble learning technique such as bagging, boosting, or stacking, where multiple models are trained and combined to make a decision. This can be due to base learners being complex and computationally expensive to re-train. Furthermore, the base learners may perform differently over time in the face of changes in data as each base learner is trained on different subsets of data and may capture different patterns in data. Therefore, aggregating the base learners for prediction may not be as straightforward as when the base learners are trained on stationary data. This problem of deciding between various differently-performing learners can also arise with entirely different learners, such as a main learner addressing the main learning task, and a CDD or CDA learner addressing the CD problem. We refer to this problem as *concept drift resolution (CDR)* in this thesis, which has not been addressed in the literature. We will provide a detailed explanation of the CDR problem in chapter 5.

In summary, there are three major drawbacks with existing data mining processes that run over extended periods of time due to changes in data patterns. First, a domain expert has to continuously monitor and evaluate performance of the analytical models trained in the data mining life cycle for changes in data patterns. Second, part or all of the data mining life cycle phases have to be repeated either after detecting of concept drift or every once in a while. This is especially crucial for the modeling phase, where training of the analytical models is often expensive in terms of computing resources. Third, the decision-making process of the

ensemble learning or CDD&A techniques may become more complicated when different learners are affected by concept drift.

The CD problem encompasses these three downsides. The main objective of CDD is the detection of pattern changes in data automatically over time to reduce the time and effort investment of domain experts for continuous monitoring of model performance as much as possible. Similarly, CDA aims at automating one or more phases of the data mining life cycle *when deployed in the stream processing pipeline*. Lastly, CDR aims at automating the decision-making process when multiple learners are used in the ensemble learning or CDD&A techniques while maintaining an optimal overall performance of the application.

Žliobaitė et al. [106] make a similar suggestion for incorporating CDD&A as ''Adaptive Data Mining''. However, their illustration does not clearly differentiate between the existing process of updating and re-deploying the analytical models versus incorporating CDD techniques.

2.1.2. Concept Drift

Data evolves over time. While the goal of KDD methods is extraction of patterns from data, these patterns do not stay the same over long periods of time. Therefore, the abstract models of data built on the extracted knowledge become obsolete, and will have to be either re-created or adjusted. This phenomenon is referred to as *concept drift* or *concept shift* due to data being *non-stationary* [35, 61, 9]. In accordance with the literature on concept drift, we investigate concept drift only in the domain of classification type of applications, where the dependent random variable y is discrete, and we do this from a probabilistic perspective. However, many of the concepts and processes discussed can be applied in a regression task as is or with some adjustments. In such cases, we will refer to the task generically as *learning*, and the model as the *learner*. We first describe the problem of pattern classification and the Bayesian decision theory briefly in section 2.1.2.1. Then in section 2.1.2.2, we will formalize the concept drift problem based on the classification problem. We will further define concept drift, and explaining using Bayesian decision theory, how and why it leads to the deterioration of the performance of the classifier. In section 2.1.2.3, we will study different types of concept drift, and provide a working example of concept drift in a classifier setting.

2.1.2.1 The classification problem and Bayesian decision theory

In a pattern classification task, we decide among multiple possible values of a categorical dependent random variable y —also called *state of nature* [31] or *response* [48]—based on observations or measurements of a d-dimensional *feature vector* $\mathbf{x}$ of independent continuous

random variables in Euclidean space $\mathbb{R}^d$, called the *feature space*.

Before making any measurements of the feature vector $\mathbf{x}$ for a new test instance, it is essential to know the frequency of each class to aid in determining the class of the new test instance. We call this the *class prior* for category y_j, denoted as $P(y_j)$, which represents user's prior knowledge of how likely category y_j is to occur[2]. Our classification decision would then be choosing a category y_j which has the largest $P(y_j)$. However, we manage to gather a vector x_0 of measurements for our new test instance, using which we like to better decide the category to which the test instance belongs. This will be the *posterior* probability density $P(y_j|\mathbf{x})$ of class y_j being the true class after having observed the feature vector x_0.

The classification problem is to decide which one of the values—also called *categories*, *classes*, *labels*, *targets*, or *target values*—of y is more likely to be the actual state of nature given an instance vector of measurements, also called *features* or *attributes*, as a *test* instance x_0. The decision is made based on the posterior probability of the class y_j being the actual class. The joint probability density of observing measurements of feature vector $\mathbf{x}$ for category y_j, denoted as $p(y_j, \mathbf{x})$, can be used to derive the posterior in *Bayes' formula*, shown in eq. (1), which asserts how likely the new observation may belong to a class after making its measurements:

$$P(y_j|\mathbf{x}) = \frac{p(\mathbf{x}|y_j)P(y_j)}{p(\mathbf{x})} \tag{1}$$

Assume that $\{y_1, \ldots, y_c\}$ is the set of c classes that the dependent variable y can take. The variability of different measurements for some category y_j is denoted as $p(\mathbf{x}|y_j)$, and is called the *class-conditional* (or *state-conditional*) probability density function (PDF). It is the *likelihood* of y_j being the true category the greater $p(\mathbf{x}|y_j)$ is over a range of measurements. In other words, the class-conditional PDF tells us how likely (or unlikely) it is to see a certain variation of measurements for each class.

$p(\mathbf{x})$ is called the *evidence* and is used as a scaling factor such that the posterior probability always sums to 1. However, it does not affect the posterior in any way.

In summary, the posterior probability $P(y_j|\mathbf{x})$ that class y_j is the true state of nature after observing a vector $\mathbf{x}$ of measurements depends on how likely it is to make those measurements in category y_j (likelihood) and how likely class y_j can occur based on our prior knowledge (class prior). The posterior is further scaled by the loss function of the application. The classification decision will ultimately be made based on the this scaled value to minimize the Bayes risk, assigning the observation to the most likely category. This will be the best decision that any

[2]Following the notations of Duda et al. [31], we use a lower-case $p(\cdot)$ to denote a probability density function, and an upper-case $P(\cdot)$ to denote a probability mass function.

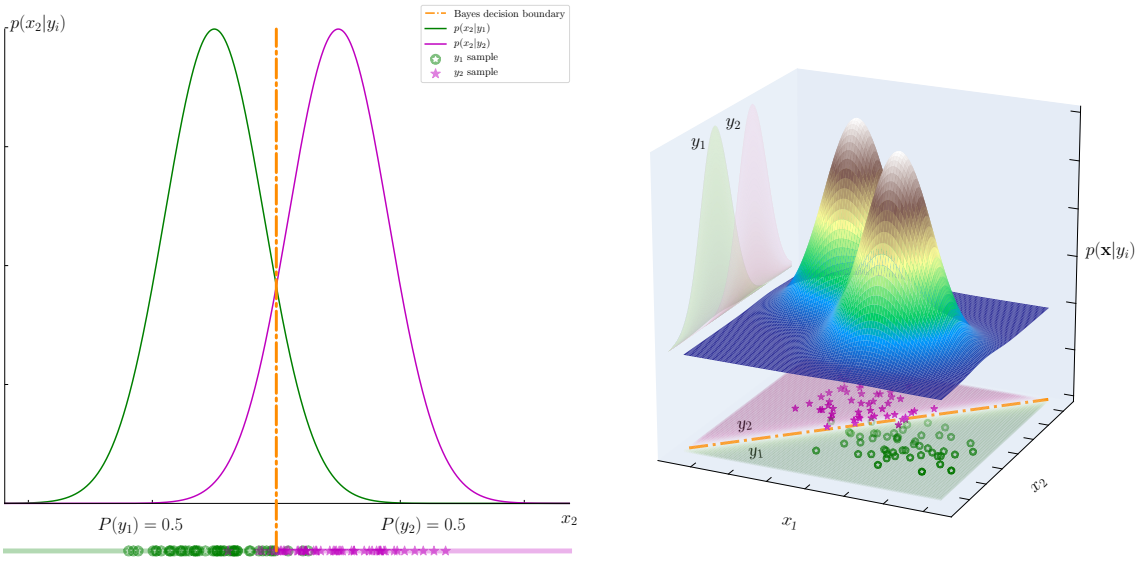


Figure 2.2: Depictions of a simple binary classification problem for classifying *apples* from *oranges*. The left 1-d plot is of one feature (x_2 as *volume*), and the 2-d plot on the right is of two features (x_1 as *weight* and x_2 as *volume*). The Bayes decision boundary, shown as an orange point in 1-d and a dashed orange line in 2-d, is the region where measurements are equally likely to belong to each class; adopted from [31].

classifier can take to achieve the best performance [31, 48]. All values in the feature space where feature values are equally likely to be in two or more categories are called the *Bayes decision boundary*. For a binary classification problem, the Bayes decision boundary constitutes values of 50% probability. This threshold is used in the Bayes classifier for making predictions.

Figure 2.2 depicts a simple binary classification problem, which could apply to our running example from chapter 1. The class conditionals $p(\mathbf{x}|y_j)$ are multivariate normal, and the class priors are $\mathrm{P}(y_1) = \mathrm{P}(y_2) = 0.5$, with y_1 representing *apples* and y_2 representing *oranges*. This means that samples belonging to each class of fruits are equally likely to be observed. Features x_1 and x_2 could be *weight* and *volume* of the fruits. True distribution hyperparameters are known here, which is the reason we can see the exact Bayes decision boundary and decision regions. However, in real-world applications, the decision boundaries and regions will have to be estimated based on the samples, observations, or measurements alone.

2.1.2.2 The concept drift problem

An assumption often made in pattern classification in data is that the distribution of the class priors $\mathrm{P}(y)$, the likelihood $p(\mathbf{x}|y_j)$, and even the evidence $\mathrm{P}(\mathbf{x})$ do not change between train

and test times. That means the assumed distributions stay the same between the time the set of feature vectors X_{train} and the set of their associated labels Y_{train} are analyzed, and the time created models are used to take an action in decision making $\{X_{test}, y_{test}\}$. Intuitively this is because the training and test observations have usually been gathered relatively around the same time in which the true underlying patterns of data remain stationary: the rules of game were not expected to change. However, especially with the advent of stream processing pipelines and increase in processing and storage capabilities of computing systems, the classifiers are now deployed to make decisions over longer periods after the initial training was performed. In this case, the assumption that data distribution does not change may not always hold true. Therefore, conventional classifiers would perform sub-optimal in decision-making when data patterns do change. In such cases, we refer to this problem as the *concept drift problem*.

Formally, *concept drift* is defined as a change in the joint probability distribution of the dependent variable y and feature vector $\mathbf{x}$ between two points t_0 and t_1 in time, as shown in eq. (2) [35, 50, 61]. That means the joint probability density of making an observation that bears a class y_j *and* has measurements of feature vector $\mathbf{x}$ has changed over time.

$$p_{t_0}(y_j, \mathbf{x}) \neq p_{t_1}(y_j, \mathbf{x}) \tag{2}$$

Since the joint probability density $p(y_j, \mathbf{x})$ can be expressed based on both marginals $p(\mathbf{x})$ and $\mathrm{P}(y_j)$, we can derive the posterior $\mathrm{P}(y_j|\mathbf{x})$ as Bayes' formula, leading to eq. (3).

$$p(y_j, \mathbf{x}) = \mathrm{P}(y_j|\mathbf{x})p(\mathbf{x}) = p(\mathbf{x}|y_j)\mathrm{P}(y_j) \tag{3}$$

Recall from the previous section that the Bayes decision rule to minimize the probability of error selects class y_j for the most likely posterior $\mathrm{P}(y_j|\mathbf{x})$. That means the classification decision is based on the posterior. Therefore, we are only interested in changes of the distributions that affect the posterior.

Since the Bayes classifier provides the best theoretical performance any classifier can achieve [48], any change in the probabilities affecting the posterior in general results in sub-optimal performance of the classifier.

Based on Bayes' formula in eq. (1), any change in the posterior can be attributed to either a change in the true distribution of the likelihood $p(\mathbf{x}|y_j)$, the class priors $\mathrm{P}(y)$, or both. However, as mentioned previously, the evidence $p(\mathbf{x})$ does not affect the posterior. Even though the distribution of the measurements among *all* classes may change over time, they do not affect the classifier's decision-making ability as long as the changes are not class-specific. In summary, changes in the distributions of the class-conditionals, the priors, or a combination of the two

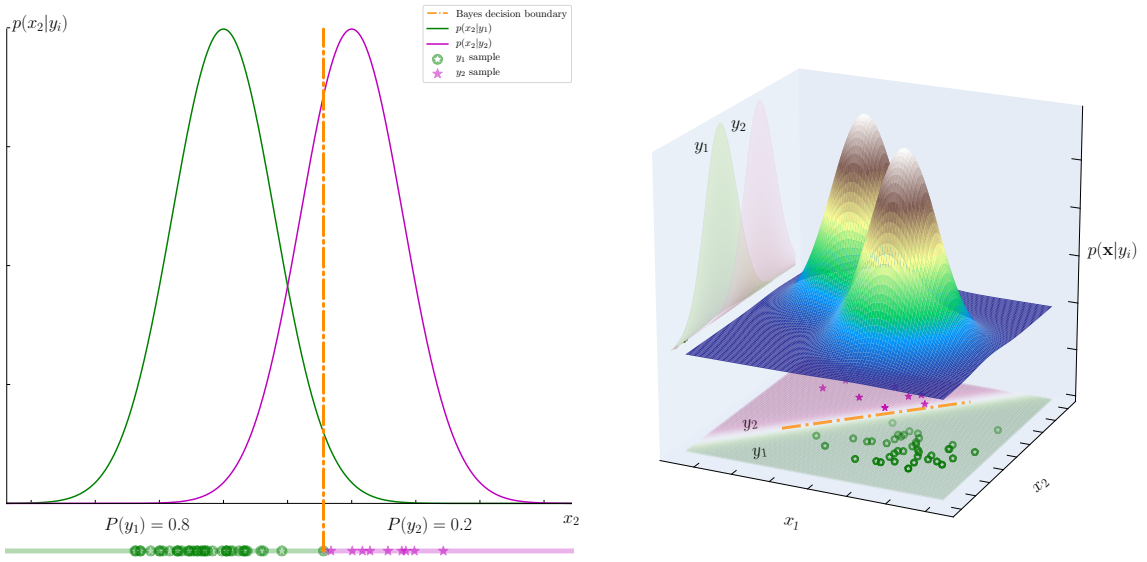


Figure 2.3: Depictions of a simple binary classification problem for classifying *apples* from *oranges* with unequal priors. This has led to a shift of the decision boundary, hence a concept drift.

lead to changes in the posterior, which result in a sub-optimal performance of the classifier.

The provided definition from a Bayesian probabilistic perspective has been the focus of research that adopt statistical analysis methods for CDD, as we will review in the upcoming sections. Alternatively, we can define concept drift to be any changes that lead to changes in the Bayesian decision boundary, because such changes directly affect the quality of the decisions which results in an obsolete model of data.

Lastly, we can also define concept drift from an application perspective as any changes in the data distributions that lead to deterioration of the performance of the classifier. This is intuitive because changes in the posterior affect the decision making quality of the classifier. However, here the focus is on the evaluation of the model and how the classifier performs on test instances. Many research works adopt this definition and provide solutions based on it, as we will review later.

Referring back to our running example of Figure 2.2, at some point in time there could be changes to class priors. For instance, *oranges* become less likely to be seen during summer. In this example, the priors are now $P(y_1) = 0.8$ and $P(y_2) = 0.2$. This has resulted in the change of the posterior, leading to the shift of the decision boundary and the occurrence of CD, as shown in Figure 2.3.

2.1.2.3 Types of concept drifts

Concept drift may occur due to changes in one or more probability distributions of Bayes' formula of eq. (1). This has led to a variety of terminologies regarding each type of drift. One or a combination of these types has also been the focus of different research in the literature.

1. *Real drift.* Gama et al. [35] and Lu et al. [61] consider the problem of concept drift for changes that directly affect the decision boundary, hence the decision capability of the classifier. Such changes are referred to as *real drift*. They further distinguish between changes in different probability densities, and consider changes in class-conditionals $p(\mathbf{x}|y_j)$ as real drift. Khamassi et al. [50] follow the same definition; however, they incorrectly refer to changes in the probability densities as actual values.

2. *Virtual drift.* This term has undergone the greatest variability in definitions across the literature. For example, Gama et al. [35] and Lu et al. [61] refer to virtual drift as changes in the distribution of the evidence $p(\mathbf{x})$. As mentioned in the previous section, the evidence is only used to scale the posterior to sum to 1 as a probability density. However, its changes *may* affect the posterior if there are changes in some class-conditional $p(\mathbf{x}|y_j)$ as well. Still, virtual drift has been addressed in the literature because (i) statistical analysis methods are computationally efficient and easy to infer, and (ii) there could be some delay between the time test instances X_{test} are provided for prediction, and the time the true class labels y_{test} are available for evaluation of the model. This results in an unsupervised or semi-supervised setting at least for one time step until class labels become available [66]. Khamassi et al. [50], on the other hand, note that virtual drift is any change in the class-conditionals $p(\mathbf{x}|y_j)$ which does *not* cause changes in the posterior. This can only apply if the class-conditionals for *all* classes y_j change similarly; otherwise, the posterior would be affected. By definition, a single ubiquitous change of all class-conditionals is equivalent to a change in the evidence, as shown in eq. (4):

$$p(\mathbf{x}) = \sum_{j=1}^{c} p(\mathbf{x}|y_j)\mathrm{P}(y_j) \tag{4}$$

3. *Drift in class priors.* This type of drift occurs if the probability densities of the priors $\mathrm{P}(y_j)$ change. As we saw previously, such changes could affect the distribution of the posterior, hence the decision boundary of the classifier. However, Gama et al. [35] do not consider it as one possible type of drift. Khamassi et al. [50] do mention it as a drift type, but incorrectly specify cases where certain class prior changes, such as class imbalance, may *not* lead to a change in posterior. They categorize this type of change in priors as

virtual drift. They do not discuss how or when a change of priors may affect the posterior and the decision making. The class priors directly determine the posterior alongside class-conditionals, and therefore are needed to be analyzed as the prior knowledge of the population. Overlooking them may even lead to wrong conclusions such as (informally) the base rate bias or false positive paradox. This is the reason precision plays an important role alongside true positive rate in evaluation metrics (see section 2.2.3 for more details). In short, only proper evaluation metrics such as precision signify the consideration of class priors as a type of drift.

4. *Population drift.* This type of drift is pointed out by Gama et al. [35], but is left out in the survey of Khamassi et al. [50]. It covers all changes in the hidden context [96], such as unmeasured and unmeasurable variables, as well as noise. While such changes may adversely affect the decision quality of the classifier over time, they are not normally covered in conventional data analysis methods due to being the irreducible error [48]. Therefore, consideration of population drift comes down to the definition of the concept drift one adopts. From the probabilistic perspective, hidden or unmeasured variables are left out of the set of variables affecting the posterior. However, from an application-oriented perspective, *any* deterioration of the performance of the classifier is the result of concept drift, be it systematic or not. Despite this, all research works in the literature that adopt an application-oriented approach to CDD&A by using performance evaluation feedback, only consider real and/or virtual drifts as the causes of increased overall loss or error in the classifier when concept drift occurs.

In summary, long-running data stream management systems have to watch for and detect changes in data over time and adapt their analytical models accordingly, to which we broadly refer as ''Concept Drift Detection and Adaptation over Data Streams''. Sections 2.1.1 and 2.1.2 covered the CD problem from practical and theoretical perspectives, respectively. Figure 2.4 depicts the high-level view of a taxonomy of the concept drift problem. We consider the problem setting and requirements in section 2.2.1 first. When facing CD in a DSMS, there are two sub-problems to tackle: concept drift detection (CDD) and concept drift adaptation (CDA), which are often studied independently in the related literature. We will review these sub-problems in sections 2.2.2.2 and 2.2.2.3, respectively.

2.2. Methods

In what follows we will discuss different aspects of the concept drift problem and provide a survey of related work. We examine the characteristics of possibly evolving data and the stream

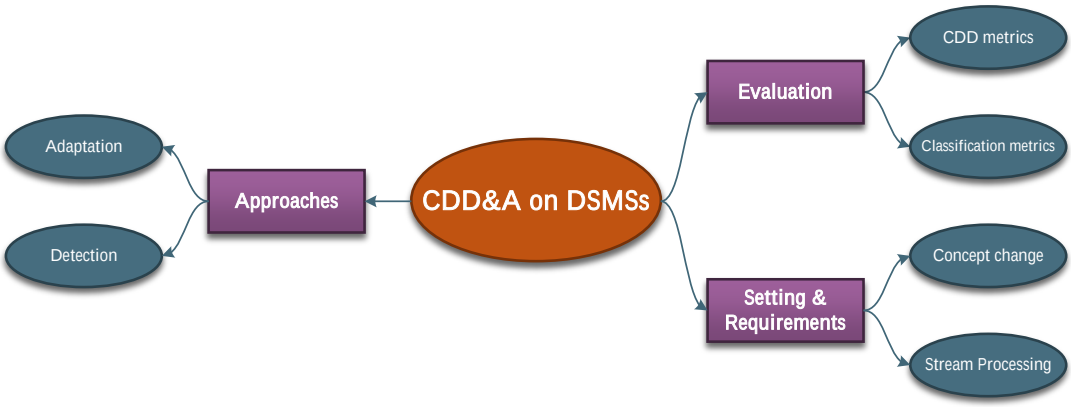


Figure 2.4: The top level view of a taxonomy of the concept drift detection and adaptation over data streams problem.

processing domain in section 2.2.1. We review two overlapping problems of CDD and CDA in sections 2.2.2.2 and 2.2.2.3, respectively. Lastly, we will review the evaluation criteria and indicators of these techniques in section 2.2.3.

2.2.1. Concept drift setting and requirements

We will review the settings and requirements from two points of view: a theoretical perspective in section 2.2.1.1 that focuses on possibly evolving data characteristics, and a practical perspective in section 2.2.1.2 that focuses on the constraints and challenges faced in stream processing as the problem domain.

2.2.1.1 A concept drift perspective

A taxonomy of the concept change perspective is presented in Figure 2.5.

A change in concept to be learned can be attributed to its rate of changes, periodicity, monotonicity, predictability, feature properties, and the drift nature. These are characteristics of possibly evolving data to be modeled.

The *rate of changes* can be either *abrupt* or *gradual*. Gradual changes happen when the distribution of the features shifts *relatively* slowly from the original distribution on which the model was built and trained. For example, the set of features for weather prediction shifts

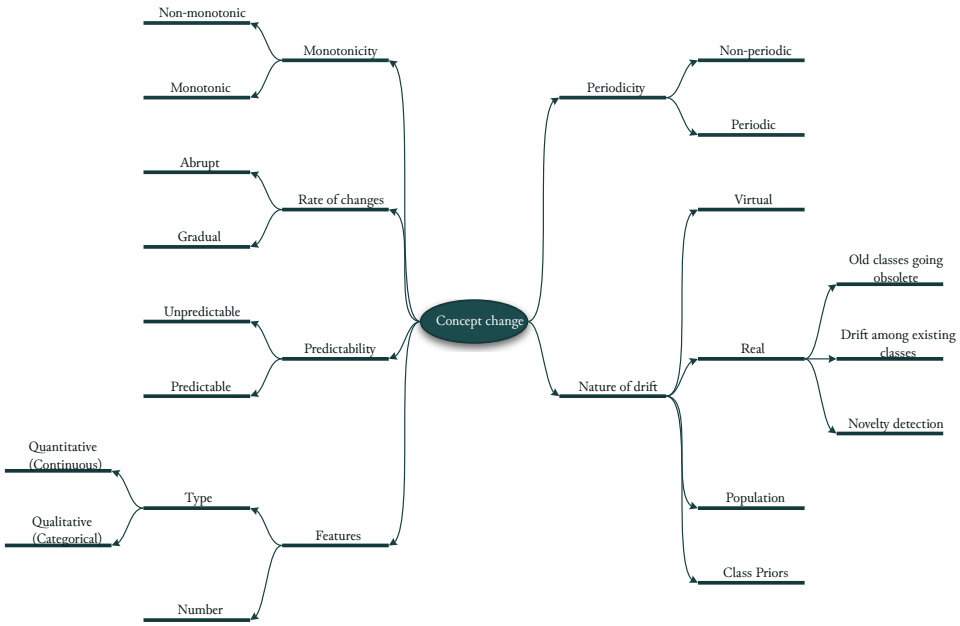


Figure 2.5: A taxonomy of the concept drift problem from a theoretical perspective.

slowly between seasons. Such changes can result in a gradual deterioration in the learner's performance. Since these effects are not too severe in a short time frame, the learner has a higher chance of adapting itself rather than throwing most or all of its learned knowledge away to start training from scratch.

Periodic CDs repeats during different time intervals, and may be periodic in differing temporal resolutions. Periodicity can be either a seasonal component of a time-series model if it repeats at the same frequency, or a cyclic structure otherwise [68]. A CDD&A model may be trained to detect or predict CD at one or more temporal resolutions. In this case, availability of enough training data can be of great advantage to the CDD&A solution's initial training.

Concept changes can be *predictable* regardless of their periodicity. The predictability information can be incorporated in the design of the CDD&A solution to make it more proactive.

Monotonicity is a desirable feature of data in many applications due to differentiability and integrability of the monotonic domain. Monotonic CD, whether occurring across the entire time domain or within specific sub-intervals, is generally easier to model or manage. Additionally, an ever-increasing or decreasing CD exhibits predictable characteristics.

The curse of *dimensionality* is one of the challenges in CDD&A. Upon deterioration of performance, it is difficult for the CDD&A methods to tell if this is due to emergence of CD, lack of

leaner's ability to generalize because of large number of dimensions, or both. Therefore, most research works consider a limited set of features. Moreover, as each feature in the feature vector $\mathbf{x}$ can be *quantitative* (continuous) or *qualitative* (discrete), most research works on CDD&A methods consider only a subset of continuous features because the latter is easier to model and analyze, better supported by statistical methods, and does not require additional preprocessing steps which are specific to qualitative features, such as encoding.

The *nature of drift* can be any of the CD types discussed in section 2.1.2.3. Furthermore, real concept drift can be attributed to drift among existing classes, old classes becoming obsolete, or emergence of new classes; the latter is also known as *novelty detection*. More specifically, novelty detection refers to estimating the probability of a new observation belonging to the same distribution as that of the training data [78].

Let us consider the above characteristics of CD for two real-world CD examples, shown in table 2.1. Deterioration of chemical sensors over time is gradual, non-periodic, monotonically decreasing, hence predictable to some degree, and involves 128 real-valued features [89, 73]. It can be categorized as real-drift, because deterioration of the sensors affects the performance of the classifier negatively over time. Weather features [32], on the other hand, are periodic on different time resolutions (days, weeks, months, etc.), predictable, and non-monotonic. The rate of drift may be considered gradual or abrupt depending on the level of time resolution, although it is often treated as gradual. It is also an example of real-drift because the decision boundary needs continuous adjustment to account for change of days, etc.

Table 2.1: CD characteristics for two real-world CD examples.

CD example	Rate of drift	Periodic	Predictable	Monotonic	Nature of drift
Deterioration of chemical sensors [89, 73]	gradual	✘	✓	✓	real drift
Weather features [32]	gradual	✓	✓	✘	real drift

2.2.1.2 A stream processing perspective

A taxonomy of the stream processing perspective is presented in Figure 2.6.

If the focus of the CD problem is on stream processing, the emphasis is on common big data characteristics such as volume, velocity, variety (heterogeneity) as well as data stream-specific properties such as unboundedness. On the other hand, the DSMS and its attributes such as limited available resources and limited access to archival storage might be of interest in dealing with the CD problem [105].

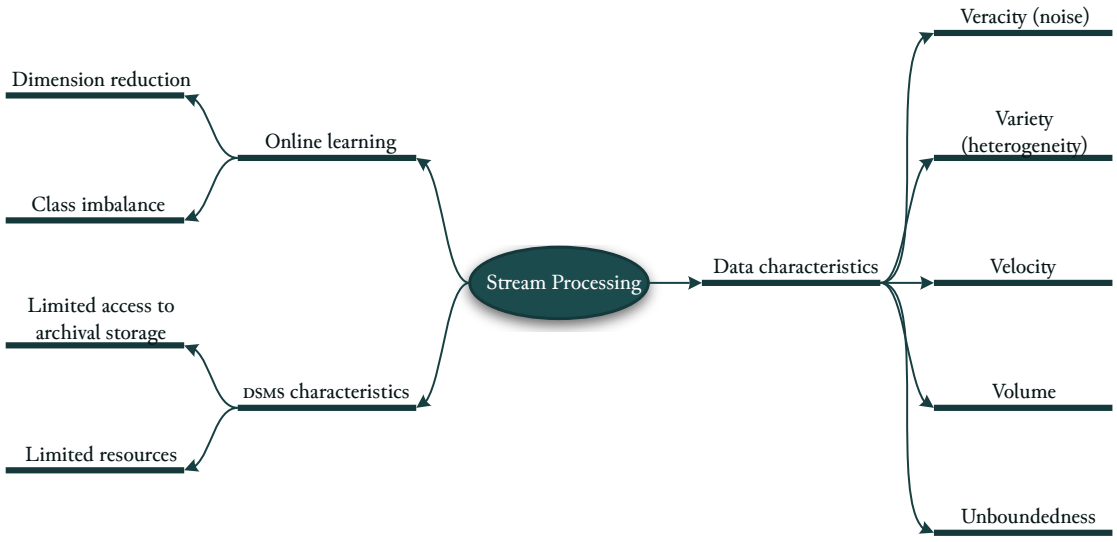


Figure 2.6: A taxonomy of the concept drift problem from an application stream processing perspective.

Lastly, the concept drift detection or adaptation method may focus on the challenges faced in online learning. These may include online dimensionality reduction to avoid the curse of dimensionality in dealing with CD, or dealing with class imbalance, especially in novelty and outlier detection. More specifically, an imbalanced domain contains a large number of instances of one class, and very few of the other (usually the positive instances). Examples of data stream classification applications with imbalanced domains include financial fraud detection [25], network intrusion detection, and email spam filtering. Techniques used to deal with the minority class imbalance usually adopt a sampling approach such as over-sampling, bootstrapping, or other parametric or non-parametric sampling approaches [92].

Ditzler and Polikar [30] extend their Learn^{++}.NSE [32] technique to imbalanced domains using two approaches: one that incorporates synthetic Minority Over-sampling Technique (SMOTE) [23] to over-sample minority class data before applying Learn^{++}.NSE, and the other that uses bootstrapped bagging on majority class data to create sub-ensembles and then balances the classification accuracy in favor of minority class data. This technique leverages different types of information available (raw data and prediction performance) without assuming a stationary minority class. However, it tackles the problem on a window-based basis and does not consider changes to class imbalance in the long run. The performance of the proposed data-based with performance-based solutions, nor the performance of these techniques over different rates of CD were not compared.

Wang et al. [93] monitor the data stream's distribution of class labels and prediction performance

to detect class imbalance, and apply over-sampling to minority class data or under-sampling to majority class data accordingly. Lu et al. [62] adopt a similar approach to Learn^{++}.NSE, but perform under-bagging (under-sampling and bagging) at every batch to adjust the bias from the majority class to the minority class. Dal Pozzolo et al. [25] tackle class imbalance under CD by aggregating two classifiers trained on existing and delayed labeled data. Li et al. [59] propose an ensemble-based method using bagging on the most recent batch of data to handle class imbalance under CD. Zyblewski et al. [107] present another ensemble-based technique based on [85] and dynamic classifier selection (DCS), where bootstrapping takes into account the minority and majority classes separately to properly handle class imbalance. All these techniques suffer from similar shortcomings of [30].

More recently, Korycki and Krawczyk [53] employed a restricted Boltzmann machine (RBM) that accounts for class imbalance in the loss function and generatively samples minority class data. This technique, called RBM-IM, monitors changes to the RBM's per-class error trends for CDD. Changes to both minority class data and the network's error trends (as a measure of CD) in long term can be modeled using this technique. However, it does not address different rates of drift. Moreover, the RBM network suffers from the complexity of the inherent recurrent neural networks: it under-performs on smaller-sized datasets due to underfitting, and may also require tuning the network's parameters, which are costly in general because of the training needed at every window.

2.2.2. Approaches to CDD&A solutions

There are general criteria that influence the design of CDD&A techniques. However, specific considerations impact the design of CDD and CDA methods individually. We will explore these aspects in the following subsections.

2.2.2.1 General Criteria

The general criteria in designing CDD&A techniques are depicted in Figure 2.7.

Problem type

Most CDD&A solutions typically employ a supervised learning approach. This is because, as previously mentioned, they treat the CD problem as a change in the posterior distribution within Bayes' formula. In the context of an online data stream, a learner deployed in the pipeline to predict test data only has access to a window of test data $X_{test}(t_i)$ at time t_i, whether it processes

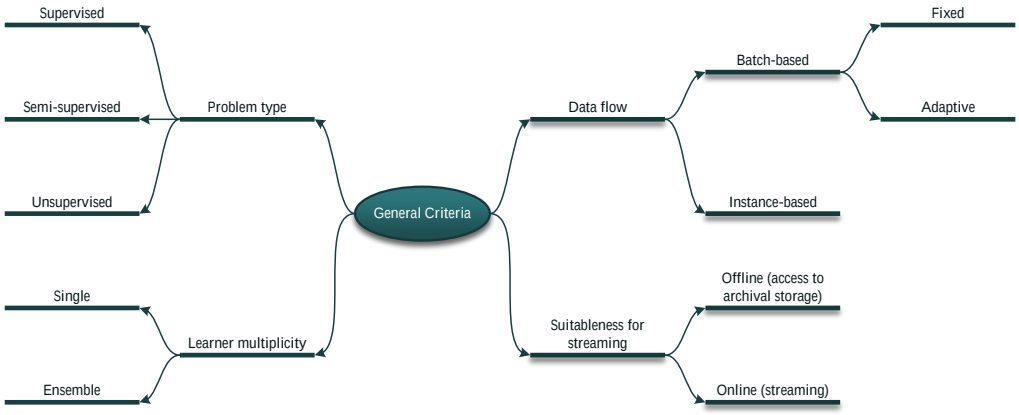


Figure 2.7: A taxonomy of the general criteria in designing concept drift detection and adaptation techniques.

that window as individual instances or as a batch. Clearly, ground truth $y(t_i)$ is to be provided with some delay δ. Otherwise, we would not need a learner in the first place if $\delta = 0$. More specifically, δ represents the delay between when the learner and CDD&A method receive the test data $X_{test}(t_i)$ and the time they receive the ground truth $y(t_i)$ corresponding to that test data. The CD problem itself can be addressed as supervised, unsupervised, or semi-supervised, based on the duration of δ, explained below.

- If $y(t_i)$ is received at the same time step (t_i) that the learner performed its classification task, we would be dealing with a *supervised* CD problem. The CDD&A method at hand could evaluate the predictability performance of the learner, and use this information for the CDD&A task [43, 97, 94]. In this case, $\delta < \underset{i \neq j}{} |t_i - t_j|$.
- If true target labels are received with at least one time step delay ($\delta \geq \underset{i \neq j}{} |t_i - t_j|$), the CDD&A method has to either wait until $y(t_i)$ are received with delay, making it a delayed supervised CD problem [66], or use only test instances $X_{test}(t_i)$ for CDD&A. The latter case would be an *unsupervised* CD problem. As noted in section 2.1.2.1, the independent variable $\mathbf{x}$ alone is insufficient to detect real CD and may result in the detection of or adaptation to only virtual drift. It is possible that the CDD&A method can use only $X_{test}(t_i)$ form an initial hypothesis of a CD and then confirm its prediction once it receives the ground truth.
- Lastly, the CDD&A method may only receive a subset of $y(t_i)$ corresponding to the given

$X_{test}(t_i)$, leading to a *semi-supervised* CD problem [66]. In this scenario, the CDD&A method can leverage the available ground truth to generalize the relationship between the independent and dependent variables to the remaining independent variable instances, allowing it to infer whether CD might have occurred. Subsequently, the delayed availability of the full ground truth may convert the problem into a delayed supervised CDD&A problem.

Ditzler and Polikar [29] proposed a semi-supervised ensemble method to model labeled training data using a set of Gaussian mixture models (GMMs) plus an extra GMM for an unlabeled test set under CD. The distance of hyperparameters of the training and test GMMs are then used as the basis for CDD. While intuitive, this method makes strong assumptions on the underlying distribution of data (being Gaussian), requires experimentation with the initialization of the number of clusters, and the unlabeled test data is essentially the new batch of data received to classify, as is common in most other research works. The unlabeled data is not used to adapt or train the learner either. Haque et al. [43] presented a semi-supervised K-means clustering method based on ECSMiner [65] (which focused on novelty detection, see section 2.2.2.2). Their method, however, does address the problem of novelty-detection in a semi-supervised fashion, similar to ECSMiner, rather than generalizing knowledge of the partially labeled data to unlabeled data. More recently, [104] proposed a CDD method based on [43], but use Jensen-Shannon divergence to measure the similarity of current classifier confidence score with those of the recent batches to detect recurring CDs. The latter technique, however, relies on the semi-supervised K-means clustering approach of ECSMiner, thus does not address the semi-supervised learning problem under CD.

Data flow

The CDD&A methods may work on *single instances* of the data stream [66, 94] or they may have to process data in *batches* [101, 97]. This is often not a requirement of the underlying DSMS, as modern DSMSs support both types of stream flow. Most research assumes that data is received in batches. This is because these methods—whether supervised or unsupervised—require at least one batch of the data stream for training. Additionally, algorithms are often trained incrementally or online, batch by batch. As a result, these techniques frequently assume that all instances within a single batch are stationary, and may even disregard the temporal order within the batch for the sake of simplicity. Those that are instance-based may reduce the delay between the time at which data is received and the time CD is detected, provided that the method is actually processing data instances individually and not buffering to be processed in batches. The other difference between true instance-based and batch-based methods is that,

most batch-based methods assume no CD of any type in a single batch of data, while it may not be true in instance-based methods where CD might occur (or start to occur) at any instance.

Another parameter in CDD&A is the length of the window of the stream, or the batch size, which could be *fixed*, e.g. [97, 94], or *varying*, e.g. [13, 43]. The latter requires adaptation during processing. Adaptive window methods strive for smaller batch sizes when the algorithm considers a low probability of CD in the upcoming batches. This eliminates the need for large numbers of training instances. When an algorithm anticipates a CD in the near future, such as during periodic drifts, it can increase the window size to include more instances. This approach enhances the accuracy of CDD&A. Conversely, reducing the window size when CD is unlikely leads to smaller batch sizes, improving efficiency in memory usage and processing power.

Learner multiplicity

Single learners such as [94] use a single classifier model to search through the hypothesis space. The performance evaluation of the same single classifier can be used for CDD&A. *Ensemble* learners such as [65, 57, 98], on the other hand, build a number of base estimators either independently (averaging methods such as bagging) or sequentially (boosting methods). Most CDD methods work independently of the multiplicity of the estimator, e.g. [97, 53].

As is the case with the main learning task, ensemble methods in CDD&A demand more computing resources than their single-model counterparts.

Suitability for data streaming

The traditional CDD&A methods that have access to archival or historical data are not suitable for detection or adaptation in streaming. This is because timely access to historical data is often not feasible in a data stream processing environment. On the other hand, if the method only requires access to online working data in the streaming window, then it can be used in stream processing engine of a data stream management system.

2.2.2.2 Concept Drift Detection

The goal of concept drift detection (CDD) is to automatically identify pattern changes in data over time without requiring the involvement or effort of domain experts. Changes in data can be caused due to (i) a change in the posterior of Bayes' formula in eq. (1), or (ii) a change in the Bayes decision boundary from a Bayesian probabilistic perspective, or (iii) a change in

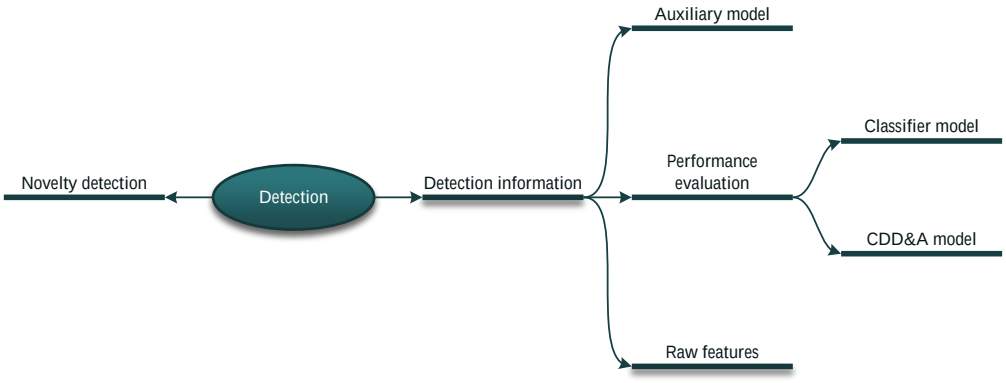


Figure 2.8: A taxonomy of the concept drift detection methods.

deterioration of the performance of the learner from a practical perspective.

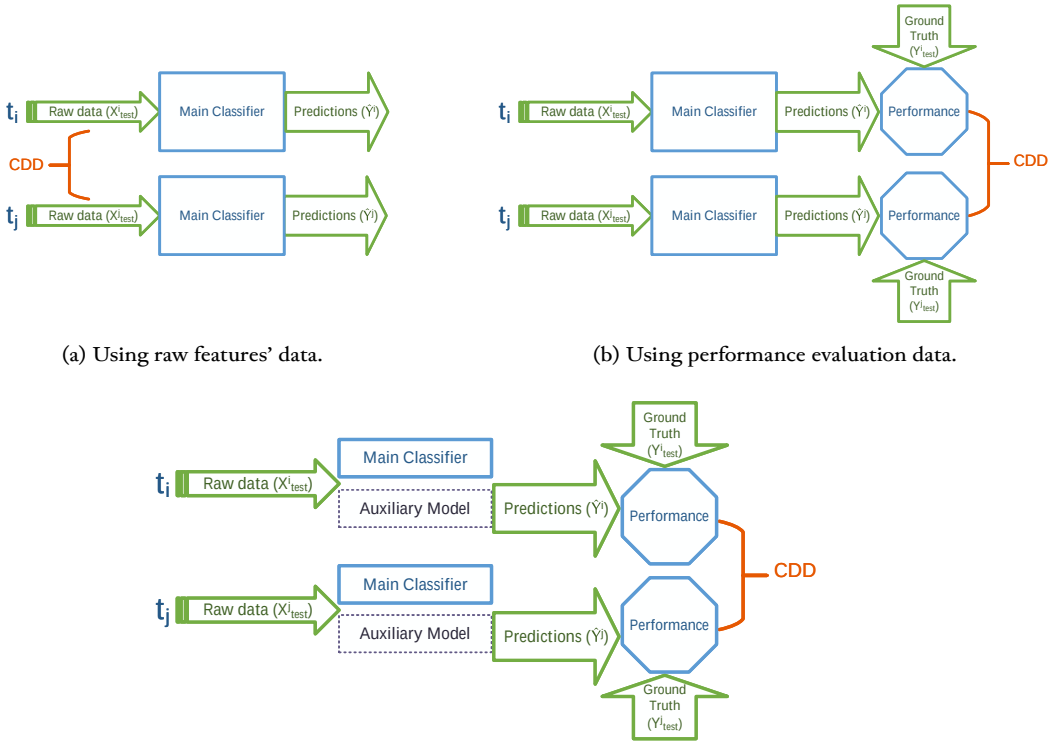
It is worth noting that CDD is crucial in applications where pattern changes are expected (see section 2.2.1.1) and certain actions must be taken once CD is detected. Examples include network intrusion detection [84] and finance, banking and insurance fraud detection [1, 42], to name a few. Selecting a proper CDD method becomes more challenging in the case that CD or its properties are unpredictable. In such cases, we may instead opt for passive CDA in the DSMS.

Figure 2.8 shows a taxonomy of the CDD methods. We will discuss different aspects of CDD methods in the following subsections.

Detection information

There are three major categories of methods based on the kind of information used for the task of CDD, as shown in Figure 2.9.

- Data feature detectors analyze only independent features in raw data, i.e., test data $X_{test}(t_i)$ (Figure 2.9a). These methods mainly consider the formal definition of CD, which on the basis of data flows can be classified into sequential and data distribution-based methods (section 2.2.2.1). Sequential raw data detectors receive test data instances one at a time, and determine a CD if a certain number of new instances are deemed to deviate from the original distribution of data. Data distribution-based methods work on batches (windows) of data, comparing the distribution of data in two windows at different times,



(a) Using raw features' data.

(b) Using performance evaluation data.

(c) Using information from an auxiliary model.

Figure 2.9: Concept drift detection using different sources of detection information.

using statistical methods and probability measures. Since these methods rely only on raw data measurements, they do not have to wait for ground truth to become available. This results in performing data analysis earlier than done by supervised methods. This has the advantage that such methods can work in an unsupervised or semi-supervised manner. These methods also have the advantage of providing insight on analyzed data because of using well-established statistical methods. A downside of these methods is that, being unsupervised, they may detect only virtual drift or detect drift in independent features that are unrelated to targets y. Raw data detectors also need true target labels for the purpose of evaluation. Furthermore, there could also be some delay until enough data instances or batches is collected by the method in order to be able to perform its analysis more accurately. This has the advantage of not having to wait for ground truth in the first place. For instance, Ditzler and Polikar [28] measure the Hellinger distance of consecutive batches of data and alarm CDs, including virtual, if the divergence is higher

than an adaptive threshold. A more recent example is Yu et al. [98] who use a GMM with sliding windows to monitor changes in the distribution of unlabeled data streams over time.

- Many CD detection methods rely on the learner's predictability performance evaluation and feedback (Figure 2.9b). These methods are based on the practical definition of CD, that is, deterioration of the learner's performance. Generally, such methods wait for the ground truth to become available (or assume that they are immediately available). Next, they evaluate the performance of the learner using standard indicators such as accuracy, precision, F-score, etc. With sufficient evaluation information, performance-based techniques identify a drift if the learner's performance falls below a fixed or adaptive threshold. In case of ensemble learners, the error threshold can be calculated based on the average, majority, or temporally-weighted deterioration of the base learners. Learner's performance-based detectors can properly detect real drifts as they also take the information about the target variable into account. However, they treat the CD problem as a black box: the mere deterioration of the learner's performance or its error rate does not provide any insights about the actual features, dependent variable or variables, their distributions, or their correlation. An example of a CDD&A technique relying on this approach is [43], where classifier confidence scores are used as input to a ''change point'' detector. The changes beyond a given threshold are inferred to as CD. To detect CD, Yu et al. [99] use a two-step detection test based on the classifier's confusion matrix. The two-step detection evaluation has the advantage of reducing the number of false positives. Similar to [43], Li et al. [57] weight imbalanced data samples using the classifier's prediction confidence to detect CD. Wang et al. [94] propose a CDD technique for detection of abrupt drifts by comparing the distribution of the classifier's accuracy between two time windows. The method is based on the assumption that the classifier's accuracy is a good indicator of CD, but is not as effective for gradual drifts. In the multi-stream framework of Yu et al. [98], the CDD technique in part monitors the performance of an ensemble of classifiers and alarms CD if the ensemble's error rate exceeds a given threshold.

- Few studies on CDD employ an auxiliary model in addition to the main learner (Figure 2.9c). This auxiliary model is designed to track changes either in the data or through the model's prediction performance. Changes in the parameters of these models are then monitored and used to detect CDs. For example, Yang et al. [97] proposed a CDD technique in which they use an auxiliary online sequential extreme learning machine (OS-ELM) model [60]. Since passive or blind adaptation in the form of re-modeling the main learner is costly at every time step in terms of computing resources, the auxiliary model aims to prevent blind adaptation at each time step by instead re-training the

simpler OS-ELM model. This is due to the fact that training the ELM model [47] requires only random assignment of the model's parameters or analytical calculation in place of iterative backpropagation. The premise of the proposed solution in [97] is that the auxiliary model's parameters change whenever a CD occurs. Based on this premise, the method re-builds the auxiliary OS-ELM model at every time step. It alarms a CD and re-models the main classifier whenever the difference of the model parameters is above a set threshold. The technique uses output weights matrix β and Euclidean distance as the model parameters and distance measure, respectively. Smaller threshold values correspond to more sensitivity to changes in the model parameters. The threshold is calculated based on the desired main classifier accuracy. Therefore, the method works fully supervised in this manner. In short, the sensitivity to changes in the auxiliary model parameters is dynamically updated over time based on the predictability performance feedback of the main model. The method, on the one hand, uses an auxiliary model to abstract the features and labels correlation, and hence it is more informative than raw data-based detectors. On the other hand, it is based on the learner's predictability performance (and can be labeled as such). While this method is intuitive and interesting, it suffers from certain shortcomings. For instance, it is not applicable to periodic drifts and is prone to fail if its validation window parameter is not set properly based on (only one) level of periodicity. Moreover, the parameters have to be determined and experimentally set for different applications based on prior knowledge. Korycki and Krawczyk [53] proposed a CDD technique, named RBM-IM, that employs an RBM as an auxiliary model to model changes to data over time (see also section 2.2.1.2). Besides the types of neural networks used, RBM-IM differs from OS-ELM [97] in how changes to data are modeled, monitored and used for CDD. RBM-IM monitors and detects significant changes to trends of the network's errors using a statistical test, whereas OS-ELM measures the changes to the weight matrix of the network's hidden layer and alarms CD if the changes are above a given threshold. The primary advantage of RBM-IM over OS-ELM lies in its improved long-term modeling of CD, specifically by accounting for changes in the error trends of the auxiliary model. It is also more robust to data characteristics such as noise and class imbalance. OS-ELM, on the other hand, has the advantage of using a simpler ELM model that does not require loss function optimization for training, resulting in a much less computationally costly model that is also less likely to underfit when trained on smaller-size datasets. In a more recent study, Li et al. [58] monitor the error rates of a pre-trained prototypical neural network model, treating the problem as few-shot learning. The auxiliary model can detect CD and its type, including abrupt and gradual.

Novelty Detection

Novelty detection in the context of this research is a form of anomaly detection where new observations, henceforth referred to as novelties, may occur but do not belong to the same distribution as of recent batches data. The difference with outlier detection is that novelties are not considered abnormal, and they may form a small yet dense cluster, which may even persist and replace the original data over time as a form of gradual drift. In the context of CDD in the literature, novelty detection often involves a change in the class priors $P(y)$ in the Bayes' formula, resulting in either class imbalance or even the introduction of new classes of data.

Most novelty detection techniques follow an unsupervised or semi-supervised approach, and assign recently arrived data to a new class if sufficient data is distanced enough from existing groups. For example, Masud et al. [65] use an ensemble-based method, called ECSMiner, and K-means clustering to detect outliers that can potentially be declared a novel class. Some novelty detectors, e.g. [78, 101], try to find a geo-spatial boundary delimiting the distribution of existing data, and either classify or assign a probability to new observations based on where and how far they lie relative to this boundary. Haque et al. [43] improve ECSMiner's novelty detection by introducing a change point detector based on the ensemble's base classifier confidence scores. More recently, Zheng et al. [104] proposed an improvement to ECSMiner to detect recurring CDs. They used Jensen–Shannon divergence to measure the similarity of current classification scores with those of the recent batches. Similar scores in terms of this measure are deemed recurring.

2.2.2.3 Concept Drift Adaptation

While human brain is capable of adapting to changing environments, adaptation to concept changes is challenging for the analytical learner component. It is often the case that a streaming data application demands maintenance of the analytical model over time once the learned information begins to become obsolete. Figure 2.10 shows a taxonomy of the concept drift adaptation (CDA) methods.

Adaptation type

The goal of CDA is to adapt the learner to new data patterns either continuously or as needed. The analytical model may be adjusted to changes continuously at each time step or at fixed intervals, without being explicitly informed of the occurrence of CD. This is referred to as *blind* or *passive* adaptation. This approach can be more beneficial to applications that process data with unpredictable or gradual CDs. It is also possible that the application simply is not

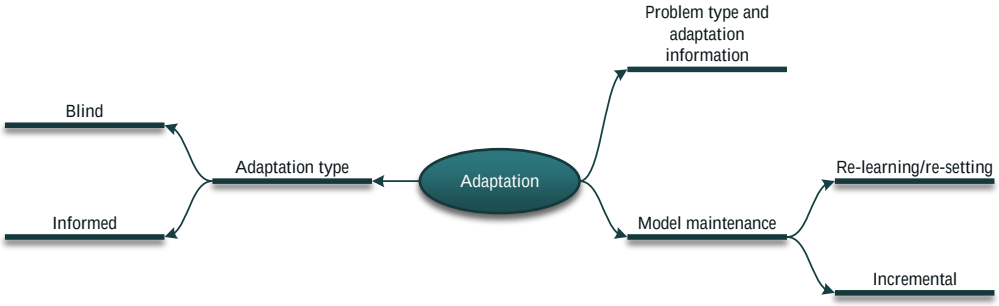


Figure 2.10: A taxonomy of the concept drift adaptation methods.

interested in learning *when* CD occurs, but rather in maintaining the model's performance over time. Methods that adopt blind adaptation may also experience delays in responding to CD if they perform model maintenance at fixed intervals, which could occur some time after the drift.

Due to high computational costs associated with frequent remodelings over time, applications that adopt this approach seek two desirable properties in the model: (i) the model training should not be computationally costly (to meet application requirements), and (ii) the model should be easy to maintain incrementally and online (to meet stream processing requirements). Ensemble methods are favored in this context and are popular approaches among both the research community and industry, as they fulfill both criteria and are well-known for their robustness to CD, as well as their ability to maintain high accuracy over time. For instance, Krawczyk and Woźniak [54] employ a weighted learning and forgetting adaptation mechanism based on the recency of data batches. Zhang et al. [103] use reinforcement learning to dynamically weight an ensemble of two Temporal Convolutional Networks (TCNs) to adapt to changes. While effective in adaptation to CD, this approach uses the same ensemble of base learners for both CDA and the main learning task. Another limitation of this approach is that the two TCNs are not updated during the stream, and maintenance only adjusts their weights for prediction.

Informed adaptations are usually implemented when a CDD method is in place to notify the system of a potential CD first, and then the CDA method to maintain the learner accordingly. This allows the CDD&A technique to maintain the main learner in a timely manner while avoiding unnecessary re-training of the model. However, few studies—for instance [97, 94, 98, 58]—adopted this approach, which for CDA is often in the form of re-training the learner from scratch. In other words, there is no underlying common abstraction or information leveraged for the CDD and CDA tasks. This is important because the CDA method essentially becomes limited to and dependent on the performance of the CDD technique used. This in turn affects

the performance of the CDA method when the CDD method fails to detect CD accurately or timely. Additionally, this approach may lead to unnecessary remodelings when the CDD technique detects virtual drifts or false positives.

Problem type and information

The purpose of a CDA method is to maintain the learner's performance over time by adapting to new patterns in the data. This can be done unsupervised (or semi-supervised) for a while depending on how fast the environment is changing. However, the method must eventually get access to the target labels to adapt the learner to current state of the data. Majority of research in CDA, however, assume that the target labels are available at the same time as the data, and the learner is re-trained at each time step. This leads to another shortcoming of existing CDA methods, which is inability to maintain the application's performance beyond one time step.

Model maintenance

Regardless of the type of adaptation presented earlier, the analytical model must be maintained after the occurrence of CD to adapt to new patterns. In conventional machine learning (ML), the learner is re-trained or re-modeled, either offline or online, using newly gathered training data. Since training a new model is a resource consuming task for many conventional learners such as deep neural networks, certain CDA methods opt for incremental learning to avoid re-modeling the learner from scratch as long as possible.

Ensemble methods such as bagging and boosting gained popularity for stream data processing because of their suitability for incremental learning. Examples include [43, 88, 91, 104, 29, 32, 66, 73, 52, 65, 81, 89, 98, 58, 103]. However, maintaining the ensemble of learners is challenging in the long run because of the risk of overfitting and possibly having high variance as the result, especially if data is imbalanced in terms of CD (see section 2.2.1.2). Moreover, adding base models to the boosted ensemble over time incurs computational overhead. Therefore, ensemble model maintenance for CDA has been a research topic over the last decade with the aim of decreasing variance and computational overhead.

Elwell and Polikar [32] proposed a weighting scheme, named Learn^{++}.NSE, as a maintenance technique that weights the base models according to their recent performance (prediction error and time added) on the stream. This weighting is later used in majority voting of the ensemble for prediction. This method achieves low variance, thus high prediction accuracy especially on datasets with gradual drift. It, however, does not address the computational overhead issue mentioned earlier.

Wang et al. [91] proposed a gradient boosting ensemble maintenance method to tackle both issues of using ensembles on data streams stated. They provide two base model pruning strategies: one that removes obsolete base models based on a loss improvement criterion they proposed with the aim of achieving high accuracy, and another based on the Kolmogorov-Smirnov test with a set confidence limit with the aim of keeping some sub-optimal base models, hence preserving diversity and low variance in the long run. An interesting observation in this work is how the deterministic strategy achieved better results (in terms of prediction accuracy) on datasets that exhibit abrupt drifts, while the statistical strategy had better results on datasets showing gradual drifts.

In short, blind adaptation with re-modeling is the most common method in stream data processing which does not take advantage of neither CDD nor incremental learning. Assuming the application is demanding, a reasonable DSMS solution tackles the CD problem by incorporating a CDD method to make an informed decision on when to maintain the model as well as supporting online and incremental learning to maintain the learner as long as possible.

2.2.3. Evaluation

Since a DSMS that encounters CD has to deal with a variety of problems, the evaluation metrics will depend on which individual or combination of these problems are tackled in a certain research work. These problems include the online data stream management, ML and data analytics, CDD, and CDA. Therefore, we can consider the evaluation metrics in three categories:

- *Online data stream management:* As DSMSs process unbounded streams of data, a desired solution should use limited memory and processing power. This in particular is important if stream processing is done at the edge side of the network. Other constraints include fast response times and limited network bandwidth. Based on these conditions and constraints, the evaluation metrics for a method focusing on online data stream management include processing and memory complexity as well as network bandwidth if the method works in a distributed manner or is deployed on more than one node of the network.

- *Data analytics and ML:* Data analytics resides at the core of the SPE of the DSMS. Conventional data analytics metrics that are derived from a confusion matrix include accuracy, true positive rate (TPR) (a.k.a. sensitivity, recall or hit rate), true negative rate (TNR) (a.k.a. specificity), false negative rate (FNR) (a.k.a. miss rate), and false positive rate (FPR) (a.k.a. fall-out). These metrics are commonly used for a learner in supervised

learning problems. Methods tackling unsupervised learning types of problems use metrics such as homogeneity, completeness, and Silhouette Coefficient [77] for evaluation.

- *Concept drift detection and adaptation:* As mentioned previously, CD can be defined in terms of the learner's predictability performance, leading to evaluation metrics of the previous category. However, methods tackling CDD may opt for performance indicators more directly relevant to these tasks. To do this, they treat the CD problem as a binary classification one, considering CDs as positive instances and the rest of data as negative instances. This requires the availability of datasets with explicitly annotated CDs. However, such datasets with annotated CDs are scarce. Therefore, some research in CDD&A evaluate the performance of their proposed methods on synthetic datasets using CDD metrics, and evaluate the predictability performance of the main classifier using conventional ML metrics (see above) on real-world datasets. Common metrics of this category include TPR (the fraction of correctly identified CDs), FNR (the fraction of incorrectly unidentified CDs), and FPR (the fraction of incorrectly identified CDs). It should be noted that certain CDs (mainly abrupt drifts) occur only occasionally within data streams. That means, we have very few positive instances (CDs) versus a large number of negative instances (regular data). This leads to an imbalanced dataset of CDs, which poses further challenges to the CDD problem. It is therefore crucial to use the metrics discussed here appropriately to ensure that the results accurately reflect the actual operational performance of the CDD method. For example, in a dataset that contains 100 instances with only 3 true CDs, a method might label all 100 instances as CDs. This would result in 100% accuracy and TPR (or 0% FNR) for detecting CDs, but still lead to a 97% FPR or just a 3% TNR.

2.3. Conclusion

In this chapter we reviewed the literature related to concept drift, concept drift detection, and concept drift adaptation. We discussed the requirements of CD in the context of DSMS applications and the challenges imposed by these requirements. We also reviewed and provided a taxonomy of the different solution approaches to CDD and CDA problems and the evaluation metrics used in those solutions.

Even though the literature on CDD and CDA has been growing a lot over the last decades, there are still challenges that need to be addressed in the context of stream processing. These challenges include the need for a more efficient and accurate CDD method that can detect CD in a timely manner. To address this challenge, we will introduce our proposed solution for CDD using an auxiliary ensemble of ELM models, and discuss the implementation details and experimental

results in chapter 3.

Importantly, a methodical comprehensive study is needed to find a common source of information that can be leveraged for both CDD and CDA tasks. In chapter 4 we will present a novel methodology to address this challenge by evaluating the feasibility of feature importance measures as a common source of information for both CDD and CDA tasks. We will then propose a comprehensive framework, called Amyris, founded on the feature importance measures methodology with CDD and CDA solutions in chapter 5.

As reviewed in this chapter, the CDD&A techniques that attempt to tackle both CDD and CDA tasks constrain the performance of the CDA technique by making it dependent on the CDD solution. The question of how to leverage multiple models in the CDD and CDA tasks that *may not necessarily belong to the same ensemble or be of the same type* and taking into account their *recent performance over the stream* has been largely unexplored in the literature. We identify this problem as *concept drift resolution (CDR)* for which we propose a novel solution to it in chapter 5.

There is a theory which states that if ever anyone discovers exactly what the Universe is for and why it is here, it will instantly disappear and be replaced by something even more bizarre and inexplicable. There is another theory which states that this has already happened.

—Douglas Adams, *The Hitchhiker's Guide to the Galaxy (1979)*

Chapter 3

An Ensemble Learning Augmentation Method for Concept Drift Detection

Even though a variety of methods, some of which we reviewed in chapter 2, have been developed to address concept drift detection (CDD) over the past two decades, further research is needed to support emerging applications which require to handle CDD stream data. Most existing methods utilize only a portion of available information for this task. Gama et al. [35] review the methods that focused on the predictive performance of the main analytical model. Additionally, a common shortcoming of existing CDD methods is that they do not consider the rate of drift as a key design factor, which we believe can contribute to different CDD solutions for different application features and requirements.

Another shortcoming of CDD methods is that they often assume that properly labeled data becomes available almost instantly at every window of streaming data. This assumption, however, is often unrealistic in many applications, such as insurance, healthcare analytics, industrial sensor grids, environmental sensing, smart city, network infrastructure monitoring and sensing, and social media.

Yang et al. [97] proposed a CDD method which uses an online sequential extreme learning machine (OS-ELM) model [60] for CDD. The changes to the output weight matrix β of an extreme learning machine (ELM) [47] model is then used to infer drifts in the stream. Since passive or blind adaptation and thus re-modeling the main classifier at every time step is computationally expensive, they proposed an auxiliary model to re-train the simpler OS-ELM model. This helps because training the (ELM) model requires only random assignment of the model's parameters or analytical calculation as opposed to using iterative back propagation.

The premise of the proposed solution in [97] is that the parameters of the auxiliary model change whenever a concept drift occurs. Based on this premise, the method re-builds the auxiliary OS-ELM model at every time step. It alarms a CD and re-models the main classifier whenever the difference of the model parameters exceeds a set threshold. They use output weights matrix β and Euclidean distance as the model parameters and distance measure, respectively. Smaller threshold values correspond to more sensitivity to changes in the model parameters. The threshold is computed based on the desired main classifier accuracy. Therefore, the method works fully supervised in this manner. In short, the sensitivity to changes in the auxiliary model parameters is dynamically updated over time based on the predictability performance feedback of the main model.

While this method is intuitive and interesting, it suffers from certain shortcomings, as follows. It is not applicable to periodic drifts and is prone to fail if its validation window parameter is not set properly based on (only one) level of periodicity. Moreover, the parameters have to be determined and experimentally set for different applications based on prior knowledge.

In this study, we tackle the CDD problem with constraints and requirements imposed by big data stream processing applications such as high detection accuracy, consumption of as much information as is available at each streaming window to build a more accurate detection model, and cope with different drift rate scenarios. More specifically, we proposed [3] an online incremental streaming algorithm, called Ensemble Learning Augmented Drift Detection (ENLAUDD), which employs an ensemble of weak (cheap) classifiers to model concept as a one-dimensional aggregation of changes in base classifiers. An efficient change-point detection algorithm is then applied on this one-dimensional signal to detect CD as drastic changes of the aggregated signal. Moreover, we consider two approaches in our proposed algorithm, one based on *bagging* and the other based on *boosting* with verification with the main classifier to handle data streams that exhibit different concept drift rates.

We developed and evaluated the performance of ENLAUDD using real-life and synthetic data streams exhibiting abrupt and gradual concept drifts. The results of our numerous experiments indicate that bootstrap aggregation of an ensemble of weak classifiers trained as auxiliary models alongside the main classifier leads to lower average variance in CDD. The proposed algorithm allows using any change-point detection algorithm, especially light-weight methods such as Z-test, on the aggregated signal to detect extreme deviations as an indication of concept drift.

3.1. Concept Drift Detection Method

In this section, we present our solution for concept drift detection, described as Algorithm 1. This algorithm uses Algorithm 2 which updates and maintains the ensemble model.

Algorithm 1: EnLAUDD concept drift detection algorithm

Input: Batches of data $B = \{X^t_{N\times D}, y^t_N\}, t = 1, 2, ...$, ensemble size M, bootstrap sample size n, drift rate $\upsilon \in \Upsilon = \{$'Abrupt', 'Gradual'$\}$, probability of verification with the main classifier ρ

Output: CD: a boolean value indicating the presence of concept drift in the current batch X^t

Initialize the ensemble $\Gamma^{\oplus}$ on the first batch of data.
Maintain the ensemble using bagging or boosting technique by calling `MaintainEnsemble`(B, $\Gamma^{\oplus}$, M, n, υ)
Compute the model parameter differences of the ensemble's base models.
Compute the losses of the ensemble's base models based on their predictions of the current batch of data.
Compute the ensemble's aggregated change as the weighted average of base model parameter changes where weights are adjusted rankings of base models according to their losses.
if *B is training data* **then**
    Train the main classifier on B
else
    Feed the aggregated value to the change point detector.
    if *concept drift is detected* **then**
        Verify the loss of the main classifier with a probability of ρ.
        Maintain the main classifier by training it on B.
return CD

Inspired by the OS-ELM method [97], we employ an ensemble of simple basic ELM models to be trained incrementally over the course of data streaming.

The EnLAUDD algorithm proceeds as follows. On the first batch of the stream, we train M basic ELM classifiers to construct the ensemble framework. The main classifier is also trained on this batch of data. For the rest of the stream, we update the basic models using bootstrap sampling (with replacement). The main classifier is also trained on subsequent batches of data during the training phase of the stream. Starting with the test phase, we perform CDD, described in

Algorithm 2: MaintainEnsemble

Input: Batches of data $B = \{X^t_{N\times D}, y^t_N\}, t = 1, 2, ...$, ensemble set $\Gamma^{\oplus}$, ensemble size M, bootstrap sample size n, drift rate $\upsilon \in \Upsilon = \{\text{'Abrupt', 'Gradual'}\}$

Output: maintained ensemble set $\Gamma^{\oplus}$

1 **if** *υ is 'Abrupt'* **then**
2 | Train each base model in $\Gamma^{\oplus}$ on a bootstrap sample of size n from B
3 **else if** *υ is 'Gradual'* **then**
4 | Add a new base model to the ensemble.
5 | Train the newly added base model of the previous step on a bootstrap sample of size n from B
6 **return** $\Gamma^{\oplus}$

section 3.1.1, after training the ensemble on the most recent batch of data. In case a concept drift is detected, then the main classifier will be re-modeled to enable adaptation to the changed environment. Therefore, re-modeling is performed only when required and not necessarily on every batch of data stream.

The algorithm has four parameters: the size M of the ensemble to create and maintain, the bootstrap sample size n, the drift rate υ, and the probability ρ of verification of occurrence of concept drift with the main classifier.

After receiving the first batch of data (line 1 in Algorithm 1), the algorithm starts by initializing an ensemble of base ELM classifiers. For the first batch only, the ensemble is trained on the entire batch of data $\{\mathbf{x}^t, y^t\}$. In this step, we also store the set of weight matrices of the collection of M base models of the ensemble as β.

Starting with batch 2, we apply bagging or boosting techniques depending on the prior information provided about the rate of drift υ. This is done by making a call to Algorithm 2 in line 2 of Algorithm 1. If we believe that the data stream exhibits more abrupt drifts, we perform bagging by taking a bootstrap sample (with replacement) of every incoming batch B of data as S and training each base model $\gamma^{\oplus}_m$ on sampled data S (line 2 of Algorithm 2).

On the other hand, if the data stream is suspected to have gradual or continuous drifts, we apply a boosting method by adding a new base model $\gamma^{\oplus,t}_m$ for this batch to the ensemble, taking a bootstrap sample (with replacement) of the incoming batch B of data as S, and training the new base model $\gamma^{\oplus,t}_m$ on sampled data S (line 5 of Algorithm 2).

In line 3, we compute the matrix norm of the weight parameters β for each base model $\gamma^{\oplus}_m$ at time t and β of the same base model at time $t-1$. This is the change δ^t_m in the model parameters

of each base model after being trained on new data, and the set of these changes at time t is denoted as Δ^t. One of the shortcomings of OS-ELM [97] is that their proposed auxiliary model, being a simple base model with high bias, is too sensitive to changes in data. To partially mitigate this sensitivity issue in ENLAUDD, unlike Yang et al. [97] who used Euclidean norm, we adopted the minimum norm shown in eq. (5) in order to minimize the overall sensitivity of the ensemble classifiers. N_h denotes the number of neurons in the hidden layer in each ELM model of the ensemble, and U is the dimensionality of response y.

$$\left\|\beta_m^t - \beta_m^{t-1}\right\|_{-\infty} = \min\left\{\sum_{j=1}^{U}\left|\beta_{m_{j,i}}^t - \beta_{m_{j,i}}^{t-1}\right|\right\}, \quad \forall i \in \{1, \dots, N_h\} \tag{5}$$

With the ensemble trained on the sampled data, we then compute the training residuals by making predictions on the set of predictor variables $\{\mathbf{x}^t\}$ in sampled data S as $\hat{y}_m^t$ for each ensemble model $\gamma_m^{\oplus}$ (line 4). Training residual ε_m^t for model $\gamma_m^{\oplus}$ is then computed using mean squared error (MSE) of the predictions $\hat{y}_m^t$ and true target labels y^t, as shown in eq. (6).

$$\frac{1}{n}\sum_{n=1}^{n}\left(\hat{y}_{m,i}^t - y_i^t\right)^2 \tag{6}$$

After obtaining the set of training residuals ε^t for the ensemble, we proceed to compute the set of coefficients ω^t for the base models. We rank all base models in increasing values of their associated MSE from eq. (6). This rank is next used as the base model's coefficient ω_m^t to compute the weighted average over all ensemble model's norms δ_m^t as the aggregation of the ensemble parameters $\bar{\Delta}^t$ in line 5. In other words, the better a base model is trained on the current batch, the higher its change of parameters will affect the aggregation of the ensemble. One advantage of this aggregation of the ensemble is that $\bar{\Delta}^t \in \mathbb{R}^1$, meaning that we have reduced the M-dimensional norm vector Δ^t to only one dimension. This will make the task of change-point detection in the next step simpler and more efficient.

3.1.1. Concept Drift Detection

The steps of Algorithm 1 above are performed regardless of training or test phase of the data stream processing. During the training phase, the algorithm stops and returns without detecting any concept drifts.

During the test phase, we feed $\bar{\Delta}^t$ to a change-point detector (CPD). This is done incrementally at each window of data as soon as we compute $\bar{\Delta}^t$. The change-point detector will then use the set of $\bar{\Delta}$ values received so far as a temporal signal. We have adopted this approach based on the *Z*-test peak detector in [16]. We construct a temporal signal of $\bar{\Delta}$ values incrementally over time, and perform a detection on a newly computed value of $\bar{\Delta}^t$ to see if it deviates from the existing signal. Our intuition is that if there are changes in the distribution of data batches *B*, meaning data has incurred concept drift, such changes affect the ensemble's base model parameters, causing $\bar{\Delta}^t$ to deviate from the distribution of the set $\bar{\Delta}^u, \forall u < t$. Therefore, the CPD component is tasked with verifying if the current $\bar{\Delta}^t$ has deviated by a number of standard deviations from the moving mean of the distribution of the previously seen aggregation values. Hence we consider a positive response from the CPD as a confirmation of the presence of CD.

If based on prior information, if data stream does not exhibit varying rates of drifts, we verify an increase of loss of the main classifier compared with its loss of the previous batch with a probability ρ, where $\rho = 1$ means that we perform the verification at every time step, and $\rho = 0$ means that no verification is performed and that we solely determine the outcome of CDD based on the CPD output. We consider the response to this verification as a confirmation of concept drift in the data.

In this case, the main classifier is re-modeled with the entire batch of data *B* to adapt to changes that have been detected in the data stream, and the algorithm terminates and signals concept drift. We experimentally found that the values of 5, 2.4, and 1 for the *lag*, *threshold*, and *influence* parameters, respectively, yielded better results, as referenced in [16].

3.1.2. Complexity Analysis

The primary factors affecting the computational cost of the proposed CDD algorithm, ENLAUDD, are the batch size *N*, the feature dimension *D*, the ensemble size *M*, the bootstrap sample size *n*, and the number of neurons in the hidden layer N_h of each ELM base model. When accounting for the target dimensionality *U*, which represents the number of response variables, we assume that it is much smaller than the input feature dimension *D* (i.e., $U \ll D$).

The per-batch complexity of the algorithm is $O(M \cdot N \cdot D \cdot N_h)$, indicating that the computational cost is proportional to the combined batch size and feature dimension. This demonstrates that the algorithm is scalable for large datasets and high-dimensional data, even when considering multi-output responses.

3.2. Experiments and Results

All experiments reported in this section were performed on a PC with an i7-10750H CPU @ 2.60GHz and 32GB of RAM. The reported results are the average and standard deviation of measurements of 30 runs. Each run of ENLAUDD with bagging took approximately 8 and 10.5 seconds on average on real-world and synthetic datasets, respectively. With boosting, ENLAUDD took approximately an average of 15 and 45.5 seconds on those datasets in the same order. OS-ELM took on average approximately 162, 120, and 91 seconds on these datasets. For all experiments in ENLAUDD, we set $N_h = 25$, $M = 200$, and n to be 40% of the dataset size N, i.e., $n = 0.4 \times N$. We set the parameters of OS-ELM to those suggested in [97].

3.2.1. Datasets

We used a total of nine synthetic and real-world benchmark datasets for a more thorough evaluation of the performance of the proposed algorithm using data with different characteristics.

Synthetic or virtually generated datasets have the advantage of specifying the precise attributes of the concept drift, such as drift type (abrupt or gradual), periodicity and duration. We have conducted experiments on the following synthetic datasets:

- *Rotating checkerboard (RCB)* was introduced in [55]. We have used the parameters in [32] for this dataset, and conducted our experiments on four rotation speed variations of this dataset that result in different rates of concept drift: RCB-constant (RCB-C), RCB-pulse (RCB-P), RCB-exponential (RCB-E), and RCB-sinusoidal (RCB-S), each dataset with a batch size of 400 for a total of 1024 batches.

- *Streaming Ensemble Algorithm (SEA) dataset* was originally introduced in [85]. It has three continuous features, two of which affect the decision boundary while the third one is noise. The two classes are determined by comparing the sum of the relevant features with a threshold θ. In our experiments, we set θ as done in [85, 32] for two variants of SEA-1 and SEA-2. We also experimented on a third variant of this dataset as SEA-3 with values of $\theta = 8.0, 9.0, 7.5, 9.0$. This threshold changes suddenly throughout the dataset, resulting in abrupt drifts over the change points. Since concept drifts in this dataset are discrete, ground truth of precise instants of concept drifts is available to us. This allows us to evaluate the performance of the techniques in terms of sensitivity (recall) and positive predictive value (precision). Each dataset contains 200 batches with a batch size of 250 instances.

We also evaluated the performance of EnLAUDD using the following two real-world datasets, both of which exhibit graduate drifts.

- *Bellevue weather dataset (NOAA)*. This periodic data was collected at Offutt Air Force Base in Bellevue, Nebraska [87], and spans over 50 years (1949–1999). Data consists of eight features, which are daily weather measurements, and two classes (''rain'' and ''no rain''). This dataset has 605 batches, each with 30 instances. We used the first 36 batches for training.
- *Electricity dataset (ELEC)*. This dataset was introduced in [44], and used for the first time in [36] to evaluate CDD techniques. It was collected from 07/May/1996 to 05/Dec/1998 in New South Wales, Australia, and consists of eight features affecting the change of electricity price—a binary class label that is ''up'' or ''down''. This dataset contains a total of 944 batches with a batch size of 48 instances, and the first 56 batches were used for training. In our experiments we used the normalized version of this dataset from the Massive Online Analysis (MOA) framework [14].

3.2.2. Comparison of bagging vs. boosting and effects of verification with the main classifier

We evaluated our algorithms through experiments using all nine datasets with two approaches of our algorithm—bagging and boosting—each with different probabilities of $\rho = 0$, $\rho = 1$, and $\rho = 0.5$ for verification with the main classifier. The goal of this study was to compare the performance of bagging vs. boosting approaches on datasets with different rates of drift (i.e., gradual versus abrupt) as well the effects of verification with the main classifier on the performance of these two approaches. The results are shown in Table 3.1, where A ($\rho = 0$) denotes boosting without verification, A ($\rho = 0.5$) denotes boosting with 50% chance of verification, A ($\rho = 1$) denotes boosting with verification on all time steps, B ($\rho = 0$) denotes bagging without verification, B ($\rho = 0.5$) denotes bagging with 50% chance of verification, and B ($\rho = 1$) denotes bagging with verification on all time steps. Figure 3.1 depicts this comparison of the EnLAUDD approaches and their aforementioned variations in terms of the main classifier accuracy.

From the results, we observe that boosting with at least some verification outperforms all the other variations on most datasets. More specifically, on datasets exhibiting gradual drift (RCB and real-world), the boosting technique clearly outperforms bagging on all these datasets. We believe this is because adding a fresh new base classifier every batch of data to adapt to this newly-arrived batch (boosting) allows the ensemble to adapt to continuous changes better

Table 3.1: Experimental results of EnLAUDD boosting (A) and bagging (B) approaches with different probabilities of verification with the main classifier (ρ) on RCB, real-world, and SEA datasets. The results are reported as the mean (standard deviation) of 30 runs. The best accuracy and detection F1 score are shown in bold.

		A ($\rho = 0$)	A ($\rho = 0.5$)	A ($\rho = 1$)	B ($\rho = 0$)	B ($\rho = 0.5$)	B ($\rho = 1$)
RCB-C	Detections	97.80 (6.52)	45.67 (6.52)	46.83 (4.26)	116.83 (9.97)	29.17 (3.13)	57.60 (5.75)
	Accuracy	70.83 (1.20)	71.27 (1.20)	71.41 (0.94)	70.79 (1.03)	70.84 (1.27)	70.65 (1.04)
RCB-P	Detections	98.43 (7.50)	45.13 (7.50)	44.40 (3.94)	114.87 (7.48)	27.43 (4.36)	55.57 (4.88)
	Accuracy	72.67 (1.47)	72.08 (1.47)	72.16 (1.61)	70.45 (1.96)	70.93 (2.75)	70.35 (2.67)
RCB-E	Detections	98.37 (6.52)	48.67 (6.52)	48.27 (4.03)	115.60 (9.43)	28.17 (3.74)	56.73 (5.38)
	Accuracy	71.63 (1.20)	71.45 (1.20)	71.51 (1.19)	70.89 (1.31)	71.00 (1.28)	71.11 (1.08)
RCB-S	Detections	96.80 (6.27)	46.93 (6.27)	44.83 (4.57)	116.27 (9.23)	28.00 (4.07)	54.57 (4.85)
	Accuracy	72.15 (2.26)	72.38 (2.26)	72.11 (1.22)	70.59 (2.26)	70.18 (1.94)	70.53 (1.66)
NOAA	Detections	102.60 (16.75)	47.10 (5.76)	54.10 (10.71)	101.23 (7.63)	48.47 (4.81)	47.23 (4.77)
	Accuracy	61.54 (1.50)	61.02 (1.42)	61.59 (1.64)	60.38 (1.23)	60.22 (1.40)	59.67 (1.85)
ELEC	Detections	122.83 (5.62)	59.03 (4.69)	61.47 (4.49)	149.90 (7.58)	71.67 (5.19)	73.33 (6.27)
	Accuracy	74.55 (0.82)	74.53 (0.77)	74.32 (0.85)	74.43 (0.75)	74.41 (0.86)	74.07 (0.75)
SEA-1	Detections	25.73 (3.17)	12.40 (3.17)	12.00 (2.10)	25.70 (4.32)	7.00 (1.69)	13.03 (2.51)
	Accuracy	93.36 (0.49)	93.09 (0.49)	93.39 (0.41)	93.27 (0.60)	93.30 (0.53)	93.39 (0.49)
	Detection F1 score	0.11 (0.06)	0.19 (0.12)	0.21 (0.16)	0.12 (0.06)	0.12 (0.12)	0.21 (0.13)
SEA-2	Detections	24.73 (4.02)	12.77 (4.02)	12.80 (2.20)	25.33 (3.92)	6.57 (1.71)	13.10 (2.95)
	Accuracy	93.31 (0.44)	93.39 (0.44)	93.10 (0.59)	93.33 (0.68)	93.35 (0.51)	93.21 (0.52)
	Detection F1 score	0.12 (0.06)	0.18 (0.11)	0.15 (0.12)	0.12 (0.06)	0.17 (0.15)	0.16 (0.10)
SEA-3	Detections	24.63 (3.15)	12.30 (3.15)	11.90 (2.31)	25.30 (3.53)	7.43 (2.30)	12.97 (2.60)
	Accuracy	93.25 (0.45)	93.24 (0.45)	93.09 (0.67)	93.20 (0.52)	93.33 (0.52)	93.21 (0.60)
	Detection F1 score	0.09 (0.06)	0.20 (0.13)	0.20 (0.12)	0.12 (0.07)	0.13 (0.16)	0.20 (0.11)

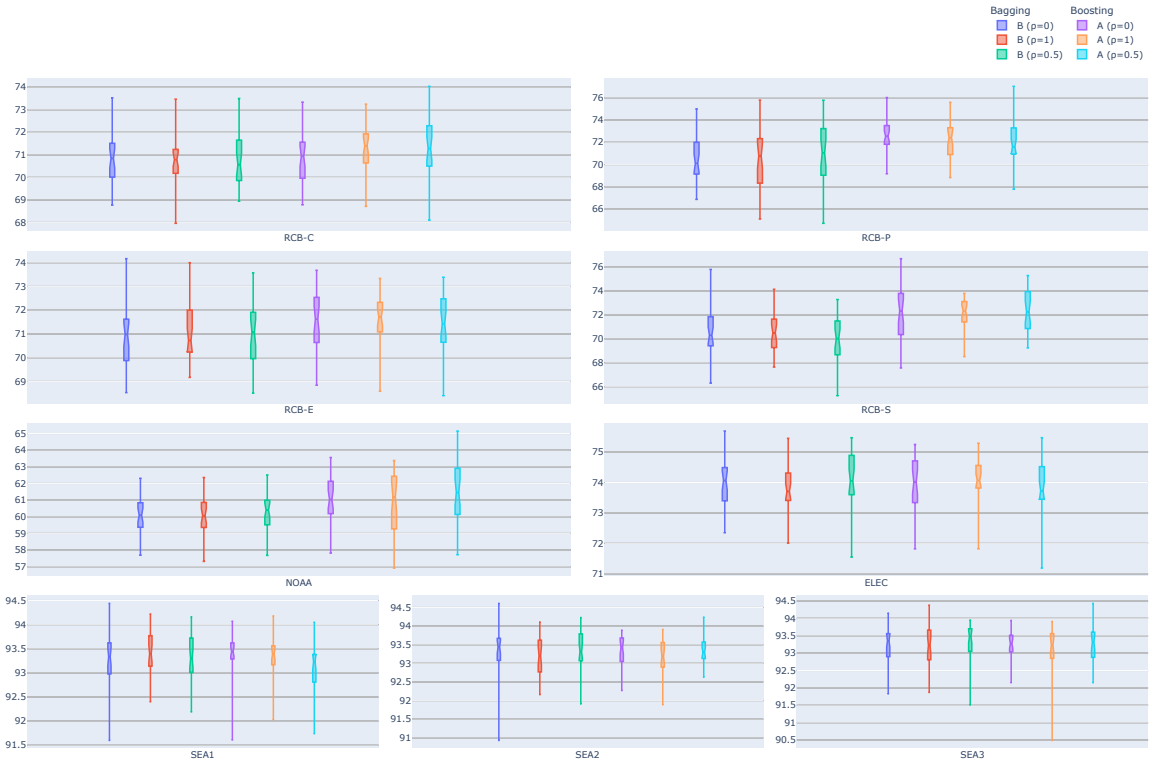


Figure 3.1: Comparison of EnLAUDD bagging and boosting approaches on RCB, NOAA, ELEC, and SEA datasets according to the main classifier accuracy (vertical axis).

and thus perform better in the long run of CDD than re-training all base models every batch (bagging). We suspect that bagging performs worse because the base models of the ensemble cannot catch up in learning continuously-changing concepts.

On the other hand, on datasets exhibiting abrupt drift (e.g., SEA datasets in table 3.1), the two ensemble approaches perform comparably well. This is because concept drifts are now discreet, which provides the bagging ensemble enough time to adapt its base models over time until the occurrence of the next abrupt change.

Regarding the effects of verification with the main classifier on the performance of these two approaches, we observe that some verification is beneficial in general to the performance of our CDD approaches. Bagging did not perform well on any of the datasets without any verification, and boosting only performed better in terms of average overall accuracy on few datasets without verification (this is discussed later). This implies that verification with the main classifier is an important step when the CDD techniques produce false positives on detection of concept drift. Furthermore, differences in achieved accuracy on datasets with continuous drifts that have approximately constant rates are insignificant when verification is done with probabilities $\rho = 1$ and $\rho = 0.5$. We suggest using verification with the probability $\rho = 0.5$ when no prior knowledge is available about the concept drift rate, and fine tune ρ through further experiments, if possible.

Lastly, we observe that on RCB-P, RCB-E, and RCB-S datasets, boosting achieved greater or comparable accuracy without any verification. This is because these three datasets have varying speeds of concept drift (''acceleration'' and ''deceleration'') causing the main classifier to fall behind in reporting a correct loss (or lack of it thereof) in accuracy after a single batch of data. In other words, we observe that the main classifier's accuracy loses its validity as confirmation to existence of concept drift when varying rates of concept drift exist in data. We conclude that, if prior knowledge about existence of varying rates of drifts is provided, it is better not to verify a change of loss of the main classifier and rely solely on the speculation of the ensemble on emergence or continuation of concept drifts.

3.2.3. Comparison with other classifiers

We developed two baseline classifiers to further study and measure the performance improvements of the proposed ENLAUDD algorithm:

- a naïve classifier that *naïvely* assumes absence of drift in data, and performs as a conventional classifier in a stationary environment,

- a persistent forecast classifier that assumes concept drift continuously occurring at every time step. Therefore, it does not perform any CDD but simply remodels the main classifier at every time step.

Intuitively, the two baseline classifiers mentioned above represent two extremes in performance yield in a changing environment. An ideal classifier operating in a non-stationary environment would fall somewhere in between these two extremes. Such a classifier should be equipped with a concept drift detector that actively, if not proactively, monitors the environment (i.e., incoming data) to react ''promptly'' to changes. The appropriate course of action would typically involve remodeling the main classifier only when necessary, thereby avoiding unnecessary training costs during periods of data stability. In addition to the aforementioned baseline models, we compare the performance of our proposed ENLAUDD algorithm with OS-ELM [97]. As for the main classifier, all these CDD methods were run alongside a decision tree learner for all the datasets considered. For ENLAUDD, we employed the boosting approach (A) with verification with the main classifier with $\rho = 0.5$ as this performed best overall in Table 3.1 experiments.

Table 3.2 demonstrates and compares the results of these experiments. Average accuracy improvement (AAI) measures the average improvement of accuracy over OS-ELM for each category of datasets (RCB, real-world, and SEA). Similarly, average F-score improvement (AFI) indicates the average improvement of F1-score for the SEA datasets where ground truth for concept drift times are available. Figure 3.2 demonstrates a visual comparison of the CDD techniques in terms of the main classifier accuracy. To be specific, it compares naïve, persistent forecast, ENLAUDD (boosting approach with $\rho = 0.5$), and OS-ELM models on the benchmark datasets in terms of the main classifier accuracy.

On gradual drift datasets, i.e., RCB and real-world datasets, ENLAUDD achieves comparable results to OS-ELM but with significantly smaller number of detections, hence fewer re-modelings of the main classifier. This is because the ensemble of base models in ENLAUDD is able to adapt better to changes in the environment than OS-ELM by employing a weighted aggregation of changes in the base models. This allows the ensemble to keep a memory of recurring patterns in periodic datasets. Since we employed training loss in weighting and then aggregating the ensemble base models, any base model that has the lowest loss will be given the highest weight in aggregation of the base model changes signal. That means that if the data stream exhibits recurring patterns, base models already trained on repeating patterns will be assigned a higher vote, hence keeping a memory of recurring patterns in periodic datasets. This also allows the drift detection algorithm to adapt better to ongoing changes by training an individual base model on each incoming batch of data stream.

We further observed that verification with the main classifier is correlated to the *rate* at which data changes. More specifically, verification is beneficial to drift detection when the changes

Table 3.2: Experimental results on RCB, real-world, and SEA datasets. The results are reported as the mean (standard deviation) of 30 runs. The best accuracy and detection F1 score between ENLAUDD and OS-ELM are highlighted in bold.

		Naïve	Persistent Forecast	ENLAUDD–A ($\rho = 0.5$)	OS-ELM
RCB-C	Detections	0.00 (0.00)	1,024.00 (0.00)	45.67 (4.17)	80.13 (27.86)
	Accuracy	50.08 (0.00)	60.83 (0.00)	71.27 (1.31)	**71.48** (2.17)
RCB-P	Detections	0.00 (0.00)	1,024.00 (0.00)	45.13 (3.25)	77.10 (27.23)
	Accuracy	72.93 (0.00)	62.02 (0.00)	72.08 (2.13)	**75.13** (4.62)
RCB-E	Detections	0.00 (0.00)	1,024.00 (0.00)	48.67 (4.34)	87.57 (29.40)
	Accuracy	49.36 (0.07)	60.88 (0.00)	71.45 (1.17)	**71.72** (1.96)
RCB-S	Detections	0.00 (0.00)	1,024.00 (0.00)	46.93 (4.57)	86.77 (27.42)
	Accuracy	55.45 (0.25)	61.60 (0.02)	72.38 (1.76)	**72.42** (2.83)
	AAI			-0.89	
NOAA	Detections	0.00 (0.00)	569.00 (0.00)	47.10 (5.76)	57.10 (42.29)
	Accuracy	70.44 (0.00)	66.59 (0.16)	**61.02** (1.42)	61.00 (1.77)
ELEC	Detections	0.00 (0.00)	888.00 (0.00)	59.03 (4.69)	118.03 (15.16)
	Accuracy	74.88 (0.04)	80.34 (0.10)	**74.53** (0.77)	73.18 (1.03)
	AAI			0.68	
SEA-1	Detections	0.00 (0.00)	200.00 (0.00)	12.40 (2.55)	15.10 (5.83)
	Accuracy	89.67 (0.35)	93.90 (0.07)	**93.09** (0.54)	92.27 (1.04)
	Detection F1 score	0.00 (0.00)	0.03 (0.00)	**0.19** (0.12)	0.09 (0.09)
SEA-2	Detections	0.00 (0.00)	200.00 (0.00)	12.77 (1.99)	14.37 (4.45)
	Accuracy	89.60 (0.30)	93.92 (0.09)	**93.39** (0.38)	92.53 (0.86)
	Detection F1 score	0.00 (0.00)	0.03 (0.00)	**0.18** (0.11)	0.09 (0.09)
SEA-3	Detections	0.00 (0.00)	200.00 (0.00)	12.30 (2.31)	13.23 (5.21)
	Accuracy	89.80 (0.36)	93.94 (0.06)	**93.24** (0.53)	91.89 (1.36)
	Detection F1 score	0.00 (0.00)	0.03 (0.00)	**0.20** (0.13)	0.09 (0.09)
	AAI			1.01	
	AFI			0.10	

Figure 3.2: Comparison of naïve, persistent forecast, EnLAuDD (boosting approach with $\rho = 0.5$), and OS-ELM on RCB, NOAA, ELEC, and SEA datasets according to the main classifier accuracy (vertical axis).

occur slowly over time. However, when the rate of changes is fast, verification may even be disadvantageous. This is because in fast-changing environments, the main classifier does not have enough time to adapt to the changes once the CDD algorithm detects a CD. In other words, by the time the main classifier is remodeled, the data might have changed so much that its verification has lost its validity and utility for the CDD algorithm.

With the ground truth of the SEA datasets available to us, we analyzed the performance of different techniques in terms of recall and precision, and computed the F1 scores. The results show the F1 scores of ENLAUDD to be twice better than that of OS-ELM while performing fewer re-modelings of the main classifier (AFI in Table 3.2). Moreover, ENLAUDD achieved at least six times improvements in terms of F1 score compared to the persistent forecast classifier while requiring re-modeling of the main classifier in at most 6% of the cases to maintain such score in concept drift detection. Recall that a persistent forecast classifier is a conventional high-end extreme of a benchmark that is re-trained blindly and continuously on all data batches, hence demanding the highest computational costs. By achieving six times improvements in detection accuracy and at least 6% reduction in computation cost, we conclude that ENLAUDD is a preferred method to balance prediction quality and computational costs resulting from re-trainings of the main classifier over time.

Lastly, it is worth noting that the ENLAUDD algorithm is significantly more stable overall due to less variance in the number of drift detections and accuracy than the OS-ELM algorithm.

3.3. Conclusion

In this chapter, we tackled the problem of CDD with challenges faced in big data stream processing applications such as high detection accuracy, consumption of as much information as is available at each streaming window to build a more accurate detection model, and coping with multiple drift rate scenarios. As a viable solution to this problem, we proposed an online incremental streaming technique employing an ensemble of base classifiers to model concept as a one-dimensional weighted aggregation of parameter changes in the base models. We also took advantage of an efficient change-point detection algorithm applied on the aforementioned one-dimensional signal to detect concept drift as drastic changes of the weighted aggregated signal. Our proposed technique consists of two approaches, one based on bagging and the other based on boosting with verification capability with the main classifier to handle data streams that exhibit different rates of concept drift.

We conducted a comprehensive study consisting of two baseline naïve and persistent forecast classifiers, variations of our proposed technique, and a state-of-the-art technique on nine

synthetic and real-life streaming benchmark datasets. Using EnLAUDD, our proposed algorithm, we achieved lower average variance in detection of concept drift by using bootstrap aggregation of an ensemble of base classifiers trained as auxiliary models alongside the main classifier. A particular advantage of EnLAUDD is the ability to use any change-point detection algorithm, especially light-weight methods such as Z-test, on the aggregated signal to detect extreme deviations as a sign of concept drift. We also adopted bootstrapping on top of these techniques to lower the overall variance of the detection algorithm. As shown by the results of our experiments [3], this provided CDD prediction performance and higher accuracy of the main classifier. Our results also demonstrated vast improvements of concept drift detection accuracy measured in F1 score. This includes an improvement of at least twice the F1 score over the state-of-the-art method [97] with fewer re-modelings of the main classifier, and at least six times improvement over a persistent forecast classifier with less than 7% of the number of re-modelings on datasets with abrupt drifts, thus providing a better trade-off between prediction quality and computational cost.

In the next chapter, we continue to investigate the research questions posed in chapter 1, that whether there exists a source of information that can be used to address both CDD and CDA problems in the same settings for big data stream processing applications. We will indeed show that the answer to this quest is affirmative, and develop concepts and techniques that can effectively address both CDD and CDA problems in the same settings.

Disorder increases with time because we measure time in the direction in which disorder increases.

—Stephen Hawking, *A Brief History of Time (1988)*

Chapter 4

Multivariate Vector Error-Correction Analysis of Feature Importance Measures

Thus far, we have explored the problem of concept drift detection (CDD) in streaming data. But when we considered the CDA problem, we felt the need for a common ground to study CDD&A in the same settings—if it existed, which we doubted, in particular since the question was not raised in then existing literature. Previous studies mainly focused on CDD&A as two subproblems separately. However, our view is that the two are related and in fact well-rooted in the same concept. Therefore, we hypothesized that these two problems can be studied as facets of the same problem, thus benefit from a common source of information. In this chapter, we propose a novel methodology to analyze and evaluate the viability of models of data as common ground for both concept drift detection and adaptation (CDD&A) tasks. Our analytical study and the proposed methodology pave the way to finding answers to research questions 1–4 posed in chapter 1.

4.1. Introduction

In chapter 3, we introduced Ensemble Learning Augmented Drift Detection (ENLAUDD) and demonstrated its effectiveness: a boosting ensemble of extreme learning machine (ELM) models, to detect concept drift (CD). However, and as noted in [90], these techniques suffer from a major shortcoming of extreme learning machine (ELM) models, namely having high variance in each ELM base model whose hidden layer does not reveal much information beneficial to the concept drift detection and adaptation (CDD&A) tasks.

To mitigate the aforementioned limitations, we posed the following questions:

1. What if we used a slightly more expensive model than ELM if we could afford the computing resources to decrease the variance?
2. Does such a model provide better detection information whose analysis and use could prove beneficial to the task of CDD&A?

The first question addresses data stream processing application's requirements. To this end, we consider gradient boosted decision trees (GBDT) models [34] as a slightly more accurate, more *expensive* model to replace ELM to lower the bias and achieve higher accuracy.

The second question addresses the effectiveness of a viable solution to the CDD&A problem. To this end, we consider feature importance measures (FIMs). Breiman [18, 17] formalized impurity-based feature importance measurements for random forests. Considering an ensemble of classification or regression trees trained on a stationary dataset, distribution of features whose scores are calculated and selected as split nodes do not change over time. However, this may not be true for data streams in which CD occurs [20]. Study of FIMs has been a subject of interest in offline machine learning, but not investigated thoroughly in data stream processing applications, particularly those undergoing CD. Therefore, these FIMs represent a valuable and worthwhile source of information for learning, detecting, adapting to, and even predicting CD in streaming data. Impurity-based importance measurements are prone to high variance as they are calculated on training data only, and miscalculate on continuous and high-cardinality features. As a model-agnostic and more robust alternative, we also consider permutation feature importance measurements. Permutation importance measurement is the decrease in a model's performance when a single feature value is randomly shuffled [18].

Wang et al. [91] demonstrated the effectiveness of GBDT for CDA. Barddal et al. [12] performed *feature selection* by training an Ada boosting ensemble of Hoeffding (online) tree stumps, but its performance could suffer if only a few features are highly predictive. Cassidy and Deviney [20] applied two online feature scoring metrics to an ensemble of online random forests and concluded that these metrics follow virtual CD. Gomes et al. [37] studied CDD using two impurity-based feature scores from an incremental random forest and an ensemble of Hoeffding adaptive trees. However, both measures suffer from the limitations of impurity-based scoring techniques [95].

So far, we considered the constraints and requirements of data stream processing applications and studied the relationship between FIMs—which are analyzed from streaming data that exhibit different characteristics of CD—and the predictability performance metric of the main classifier. The two groups of FIMs that we studied—impurity-based importance measurements

and permutation importance measurements—are computed over an auxiliary GBDT ensemble model that runs in parallel to the main classifier but processes and analyzes the same streaming data. As such, the two models used are decoupled: the main classifier, whose remodeling can be potentially costly, has the task of processing streaming data with the goal of prediction on test instances. The auxiliary GBDT ensemble model is assigned the task of processing the same streaming data with the goal of drift analysis. We specifically study the correlation of drift information, i.e., the two types of FIMs extracted from the auxiliary GBDT ensemble, with the performance of the main classifier. Therefore, no detection or adaptation are performed so that changes in FIMs and the main classifier in the face of CD can be monitored and analyzed with no interference.

The main outcome of this study is providing evidence for strong correlation between FIMs computed from a decoupled, cost-effective model with the performance of a more accurate but more costly model over time which acknowledges data stream processing requirements and encounters different types and rates of CD. Establishing the aforementioned correlation further provides:

- strong evidence to employ FIMs as a viable source of drift information for CDD&A applications, that is, detection of and adaptation to changes reactively,
- better understanding of the behavior of CD in the underlying streaming data and processes, and,
- a way to investigate prediction of CD, that is, detection of and adaptation to changes proactively.

A main contribution of this study is a novel approach on the CDD&A problem-solving methodology. More specifically, we investigate the direct correlation of detection information with the main classifier's predictability performance rather than treating the problem as a black box, comparing the performance metrics of the main classifier *only after* incorporating a CDD&A technique.

4.2. Methodology

We propose the following methodology to study the relationship between the main classifier's performance and the FIMs.

4.2.1. Variables

As reviewed in chapter 2, CD is defined as a change in the joint probability distribution of the dependent variable y and the feature vector $\mathbf{x}$ between the times t_i and t_{i+1}, as expressed in eq. (7) [35, 61, 50].

$$p_{t_{i+1}}(y_j, \mathbf{x}) \neq p_{t_i}(y_j, \mathbf{x}) \tag{7}$$

We propose a data stream processing system where batches of streaming data that might exhibit CD, denoted as $B^t = \{< X^t_{N\times D}, y^t_N >\}$, for $t \geq 1$, are simultaneously provided as test data to the initially trained main classifier as well as the GBDT model, where D is the dimension of feature vector $\mathbf{x} \in X$. The two models are decoupled and do not interact with each other throughout the run of the streams. Figure 4.1 presents the architecture of our proposed data stream processing system.

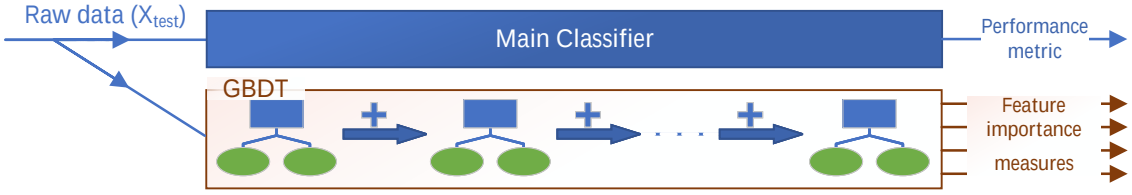


Figure 4.1: Architecture of the proposed data stream processing system to study the relationship of the main classifier's performance metrics with feature importance measures computed from a gradient-boosting decision tree used as an auxiliary model.

The accuracy and F1 scores of the main classifier at each time step t are collected as predictability performance metrics, and eventually modeled as two univariate time series $\mathscr{A}_t$ and $\mathscr{B}_t$, respectively. We compute and model impurity-based feature importance measurements ($\mathscr{G}_{d,t}$) and permutation importance measurements ($\mathscr{H}_{d,t}$), for each dimension $d \in D$ as univariate time series models. We then represent impurity-based importance measurements of feature vector $\mathbf{x}$ as a multivariate time series $\mathscr{G}_t = \{\mathscr{G}_{d,t}\}$, for $1 \leq d \leq D$. Likewise, we represent permutation importance measurements as a multivariate time series, $\mathscr{H}_t = \{\mathscr{H}_{d,t}\}$, for $1 \leq d \leq D$. Lastly, we form four multivariate time series models out of these two importance measurements types and the two predictability performance metrics, as denoted in eq. (8).

$$\begin{aligned} \mathscr{P}_t &= \{\mathscr{G}_t, \mathscr{A}_t\} \\ \mathscr{Q}_t &= \{\mathscr{G}_t, \mathscr{B}_t\} \\ \mathscr{R}_t &= \{\mathscr{H}_t, \mathscr{A}_t\} \\ \mathscr{S}_t &= \{\mathscr{H}_t, \mathscr{B}_t\} \end{aligned} \tag{8}$$

For instance, $\mathscr{P}_t = \{\mathscr{G}_t, \mathscr{A}_t\}$ in eq. (8) denotes the multivariate time series model of impurity-based feature importance measurements and the accuracy of the main classifier.

4.2.2. Hypotheses

Based on the research questions raised in section 4.1, we hypothesize that there exists meaningful relationship between FIMs computed from an auxiliary model and a main classifier's predictability performance as it deteriorates while undergoing CD. Specifically, we consider the following:

- an online, incremental GBDT as the auxiliary model,
- a main classification model that is at least as costly as the auxiliary model in terms of computational resources and data stream processing requirements,
- an impurity-based feature importance measurement $\mathscr{G}_d$, which is the (normalized) total least squares improvement contributed by $\mathbf{x}_d$, as introduced in [34],
- a permutation-based feature importance measurement $\mathscr{H}_d$, which is the change in misclassification after noising feature $\mathbf{x}_d$ of test samples by random permutation [18, 17],
- the main classifier's accuracy and F1 score as the predictability performance metrics, denoted as $\mathscr{A}_t$ and $\mathscr{B}_t$, respectively.

The null and alternative hypotheses for each type of FIMs and prediction performance metrics are stated in null hypotheses 1 to 4 and hypotheses 1 to 4, as follows.

Null hypothesis (H_0) 1 *There is no relationship between impurity-based importance measurements computed by an online, incremental GBDT model and the main classifier's accuracy over time while each model analyzes streaming data exhibiting CD separately.*

Hypothesis (H_1) 1 *There exists statistically significant relationship between impurity-based importance measurements computed by an online, incremental GBDT model and the main classifier's accuracy over time while each model analyzes streaming data exhibiting CD separately.*

Null hypothesis (H_0) 2 *There is no relationship between permutation importance measurements computed by an online, incremental GBDT model and the main classifier's accuracy over time while each model analyzes streaming data exhibiting CD separately.*

Hypothesis (H_1) 2 *There exists statistically significant relationship between permutation importance measurements computed by an online, incremental* GBDT *model and the main classifier's accuracy over time while each model analyzes streaming data exhibiting* CD *separately.*

Null hypothesis (H_0) 3 *There is no relationship between impurity-based importance measurements computed by an online, incremental* GBDT *model and the main classifier's F1 score over time while each model analyzes streaming data exhibiting* CD *separately.*

Hypothesis (H_1) 3 *There exists statistically significant relationship between impurity-based importance measurements computed by an online, incremental* GBDT *model and the main classifier's F1 score over time while each model analyzes streaming data exhibiting* CD *separately.*

Null hypothesis (H_0) 4 *There is no relationship between permutation importance measurements computed by an online, incremental* GBDT *model and the main classifier's F1 score over time while each model analyzes streaming data exhibiting* CD *separately.*

Hypothesis (H_1) 4 *There exists statistically significant relationship between permutation importance measurements computed by an online, incremental* GBDT *model and the main classifier's F1 score over time while each model analyzes streaming data exhibiting* CD *separately.*

4.2.3. Statistical methods

The goal of this study is to investigate if the main classifier's predictability performance metrics as time series models share a common long-term stochastic drift with FIMs as time series models computed from a GBDT constructed and maintained in parallel to the main classifier over the course of the stream. To this end, we have adopted and performed multivariate cointegration analysis as a valid statistical method in econometrics to establish relationships among the aforementioned time series models. The motivation for this choice follows.

Standard correlation statistics such as Pearson correlation coefficient (r), rank correlation coefficients such as Spearman's ρ and Kendall's τ, and Granger causality test can mislead to spurious relationships when data is non-stationary [100, 46, 102, 67]. Therefore, employing these tests over time series initiating from nonstationary drifting data can result in establishing significant correlations that are either meaningless or are not directly causally related due to existence of one or more confounding variables, which is usually the temporal component of the series [39]. In addition, standard detrending techniques such as differencing do not rectify

the problem of spurious relationship when resulting data is still nonstationary or different series are of different orders of integration [38, 26]. Moreover, long-term information of the shared stochastic drift between studied variables (FIMs and performance metrics) may appear in the levels of data, implying nonstationarity. In such cases, differencing leads to loss of the aforementioned common long-term information. Therefore, standard statistical practices such as vector autoregression (VAR) analysis—where the goal is to establish relationships among the original variables in levels—or tests such as Granger causality become invalid [83].

Engle and Granger [33] proposed to consider the presence of cointegration when testing for relationships between time series variables that are integrated of at least order one $I(1)$, which means non-stationary time series variables must be differenced at least once to become stationary. If two or more time series variables share a common stochastic trend and a linear combination of them is a stationary time series or one with a lower common order of integration, they are considered cointegrated. The cointegrating relationships among the variables can thus be modeled as a vector error-correction (VEC) model. The *error-correction* term in a VEC model measures both the deviation from the stationary mean at time $t - 1$ in data as the *error*, and the adjustment speeds at which the series correct to move around the common stochastic trend after a short-term deviation in the last period. This allows for balancing the short-term dynamics of the system with long-term tendencies of time series and their effects on one another. Besides rectifying the issues described earlier above, a VEC can be converted to a VAR model in levels, a.k.a. cointegrated VAR model, to allow simulation and forecast of the future behavior of the series.

Since we deal with streaming data that exhibits CD, it is likely that any time series information produced by analyzing non-stationary streaming data is non-stationary *per se* as well. Therefore, we study the cointegration of the time series variables to avoid incorrect acceptance of spurious results. Of the two popular cointegration tests in econometrics, the Engel-Granger method is used to test the cointegrating relationships between two series at a time. However, it is likely that the FIMs from the auxiliary GBDT model cointegrate together with the performance metric of the main classifier. Therefore, we adopt the Johansen method [49] in our study because it allows for multiple simultaneously cointegrating variables, requires no pretesting, and provides error correction features on the resulting VEC model.

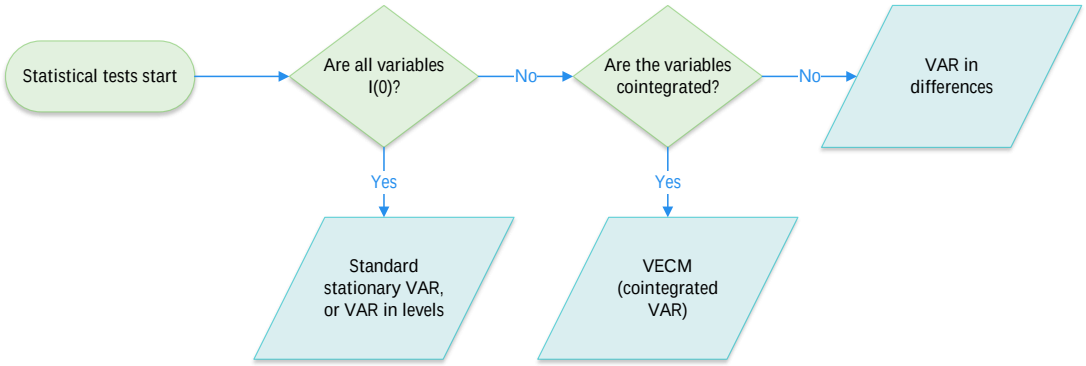


Figure 4.2: Steps of the proposed multivariate cointegration analysis.

4.3. Experiments, Results, and Analyses

4.3.1. Experimental Setup

The steps of our procedure for multivariate cointegration analysis are illustrated in Figure 4.2. We perform this procedure on the four time series models of eq. (8), using data gathered by running streams of all datasets in section 4.3.2 with no CDD&A technique applied in order to analyze the behavior of the main classifier in the face of CD in the long run.

We start by testing each univariate time series model in $\mathscr{P}_t$, $\mathscr{Q}_t$, $\mathscr{R}_t$, and $\mathscr{S}_t$ for stationarity using ADF tests. The null hypothesis of the ADF test is non-stationarity, and the alternative hypothesis is stationarity.

If we can reject the null, we can determine that all univariate variables in each of the multivariate series are stationary. This in turn enables us to form a VAR in levels from the FIMs and the performance metric of that multivariate time series model.

Otherwise, we use the Johansen method [49] to test if all univariate variables in each of the multivariate series are conintegrated. If the tests determine that the impact matrix C of the resulting VEC model has any rank $r > 0$, we conclude that there is statistically significant cointegration between FIMs of that multivariate series and the predictability metric with at least one cointegrating relation (a stationary linear combination) between them. However, if the tests determine that the impact matrix C of the resulting VEC model has rank $r = 0$, the error-correction term disappears, and we can form a VAR in differences out of the FIMs and the performance metric of that multivariate time series model.

4.3.2. Datasets

In our experiments, we used several synthetic and real-world datasets for a more thorough analysis and understanding of the relationship between FIMs and the main classifier's long-time performance using data with different characteristics. These datasets share the same parameters as those used in the experiments reported in chapter 3 except for the SEA-3 dataset, as noted below. We review these datasets again here for completeness. We used the following synthetic datasets:

- *Rotating checkerboard (RCB)* [55]. We have used the parameters in [32], and considered four rates of CD as constant (RCB-C), pulse (RCB-P), exponential (RCB-E), and sinusoidal (RCB-S) each with a batch size of 1024 instances for a total of 400 batches.
- *Streaming Ensemble Algorithm (SEA)* [85]. It has three continuous features, two of which affect the decision boundary while the third one is noise. We used the threshold values of θ considered in [85, 32] for SEA-1 and SEA-2, and used $\theta = 8.0, 9.0, 7.5, 9.0$ for SEA-3. This threshold changes three times suddenly throughout the dataset, resulting in three abrupt drifts. Each dataset consists of 200 batches of streams of size 250 instances each.

We also performed regression analysis on the following two real-world datasets, both of which exhibit gradual periodic drifts.

- *Bellevue weather dataset (NOAA)* [87]. This dataset consists of eight features as daily weather measurements, and two classes (''rain'' and ''no rain''). It has 605 batches, each containing 30 instances, with the first 36 batches used for training.
- *Electricity dataset (ELEC)* [44, 36]. This dataset consists of eight features affecting the change of electricity price and two classes (''up'' and ''down''). It contains a total of 944 batches with a batch size of 48 instances. We used the first 56 batches for training.

4.3.3. Experiments and Results

The results of the augmented Dickey-Fuller (ADF) tests are displayed in Table 4.1. The feature importance is extracted and tested for staionarity for each feature of synthetic and real-world datasets. Since importance measurements series $\mathscr{G}_t$ and $\mathscr{H}_t$ are multivariate, we rejected the null of ADF test for these series only if it could be rejected for all individual importance measurements series comprising $\mathscr{G}_t$ or $\mathscr{H}_t$. The highest significance level achieved is noted in Table 4.1. Overall,

Table 4.1: Augmented Dickey-Fuller (ADF) test stationarity test results.

Dataset	Series	ADF	Significance	Conclusion
RCB-C	$\mathscr{A}_t$	Stationary (I(0))	99%	Standard stationary VAR
	$\mathscr{B}_t$	Stationary (I(0))	99%	Standard stationary VAR
	$\mathscr{G}_t$	Stationary (I(0))	95%	
	$\mathscr{H}_t$	Stationary (I(0))	99%	
RCB-P	$\mathscr{A}_t$	Nonstationary	–	Test for cointegration
	$\mathscr{B}_t$	Nonstationary	–	Test for cointegration
	$\mathscr{G}_t$	Nonstationary	–	
	$\mathscr{H}_t$	Stationary (I(0))	95%	
RCB-E	$\mathscr{A}_t$	Stationary (I(0))	99%	Test for cointegration
	$\mathscr{B}_t$	Stationary (I(0))	99%	Test for cointegration
	$\mathscr{G}_t$	Stationary (I(0))	95%	
	$\mathscr{H}_t$	Inconclusive	–	
RCB-S	$\mathscr{A}_t$	Nonstationary	–	Test for cointegration
	$\mathscr{B}_t$	Nonstationary	–	Test for cointegration
	$\mathscr{G}_t$	Nonstationary	–	
	$\mathscr{H}_t$	Stationary (I(0))	99%	
NOAA	$\mathscr{A}_t$	Stationary (I(0))	99%	Test for cointegration
	$\mathscr{B}_t$	Stationary (I(0))	99%	Test for cointegration
	$\mathscr{G}_t$	Inconclusive	–	
	$\mathscr{H}_t$	Stationary (I(0))	90%	
ELEC	$\mathscr{A}_t$	Stationary (I(0))	95%	Test for cointegration
	$\mathscr{B}_t$	Nonstationary	–	Test for cointegration
	$\mathscr{G}_t$	Nonstationary	–	
	$\mathscr{H}_t$	Inconclusive	–	
SEA-1	$\mathscr{A}_t$	Nonstationary	–	Test for cointegration
	$\mathscr{B}_t$	Nonstationary	–	Test for cointegration
	$\mathscr{G}_t$	Nonstationary	–	
	$\mathscr{H}_t$	Inconclusive	–	
SEA-2	$\mathscr{A}_t$	Nonstationary	–	Test for cointegration
	$\mathscr{B}_t$	Nonstationary	–	Test for cointegration
	$\mathscr{G}_t$	Nonstationary	–	
	$\mathscr{H}_t$	Inconclusive	–	
SEA-3	$\mathscr{A}_t$	Nonstationary	–	Test for cointegration
	$\mathscr{B}_t$	Nonstationary	–	Test for cointegration
	$\mathscr{G}_t$	Nonstationary	–	
	$\mathscr{H}_t$	Inconclusive	–	

Table 4.2: The Johansen method test results for cointegration of feature importance measures (FIMs) with accuracy. EIG and TRC refer to the eigenvalue statistic and trace statistic in the test, respectively.

Dataset	Series	EIG	Sig. lvl.	TRC	Sig. lvl.	Conclusion
RCB-C	$\mathscr{P}_t$	$r(3)$—full rank	99%	$r(3)$—full rank	99%	VEC
	$\mathscr{Q}_t$	$r(3)$—full rank	99%	$r(3)$—full rank	99%	VEC
RCB-P	$\mathscr{P}_t$	$r(1)$	99%	$r(1)$	99%	VEC
	$\mathscr{Q}_t$	$r(3)$—full rank	99%	$r(3)$—full rank	99%	VEC
RCB-E	$\mathscr{P}_t$	$r(3)$—full rank	99%	$r(3)$—full rank	99%	VEC
	$\mathscr{Q}_t$	$r(3)$—full rank	99%	$r(3)$—full rank	99%	VEC
RCB-S	$\mathscr{P}_t$	$r(1)$	99%	$r(1)$	99%	VEC
	$\mathscr{Q}_t$	$r(3)$—full rank	99%	$r(3)$—full rank	99%	VEC
NOAA	$\mathscr{P}_t$	$r(2)$	99%	$r(4)$	99%	VEC
	$\mathscr{Q}_t$	$r(9)$—full rank	99%	$r(9)$—full rank	99%	VEC
ELEC	$\mathscr{P}_t$	$r(3)$	99%	$r(2)$	99%	VEC
	$\mathscr{Q}_t$	$r(7)$	99%	$r(7)$	99%	VEC
SEA-1	$\mathscr{P}_t$	$r(0)$	—	$r(0)$	—	VEC or VAR in diff.
	$\mathscr{Q}_t$	$r(2)$	99%	$r(3)$	99%	VEC
SEA-2	$\mathscr{P}_t$	$r(1)$	99%	$r(1)$	90%	VEC
	$\mathscr{Q}_t$	$r(4)$—full rank	99%	$r(4)$—full rank	99%	VEC
SEA-3	$\mathscr{P}_t$	$r(0)$	—	$r(0)$	—	VEC or VAR in diff.
	$\mathscr{Q}_t$	$r(2)$	99%	$r(3)$	99%	VEC

Table 4.3: The Johansen method test results for cointegration of feature importance measures (FIMs) with F1 score. EIG and TRC refer to the eigenvalue statistic and trace statistic in the test, respectively.

Dataset	Series	EIG	Sig. lvl.	TRC	Sig. lvl.	Conclusion
RCB-C	$\mathscr{R}_t$	$r(3)$—full rank	99%	$r(3)$—full rank	99%	VEC
	$\mathcal{S}_t$	$r(3)$—full rank	99%	$r(3)$—full rank	99%	VEC
RCB-P	$\mathscr{R}_t$	$r(1)$	99%	$r(1)$	99%	VEC
	$\mathcal{S}_t$	$r(3)$—full rank	99%	$r(3)$—full rank	99%	VEC
RCB-E	$\mathscr{R}_t$	$r(1)$	99%	$r(1)$	99%	VEC
	$\mathcal{S}_t$	$r(3)$—full rank	99%	$r(3)$—full rank	99%	VEC
RCB-S	$\mathscr{R}_t$	$r(1)$	99%	$r(1)$	99%	VEC
	$\mathcal{S}_t$	$r(3)$—full rank	99%	$r(3)$—full rank	99%	VEC
NOAA	$\mathscr{R}_t$	$r(2)$	99%	$r(4)$	99%	VEC
	$\mathcal{S}_t$	$r(9)$—full rank	99%	$r(9)$—full rank	99%	VEC
ELEC	$\mathscr{R}_t$	$r(3)$	99%	$r(2)$	99%	VEC
	$\mathcal{S}_t$	$r(7)$	99%	$r(7)$	99%	VEC
SEA-1	$\mathscr{R}_t$	$r(1)$	99%	$r(1)$	99%	VEC
	$\mathcal{S}_t$	$r(2)$	99%	$r(2)$	99%	VEC
SEA-2	$\mathscr{R}_t$	$r(1)$	99%	$r(0)$	–	VEC or VAR in diff.
	$\mathcal{S}_t$	$r(4)$—full rank	99%	$r(4)$—full rank	99%	VEC
SEA-3	$\mathscr{R}_t$	$r(1)$	99%	$r(1)$	99%	VEC
	$\mathcal{S}_t$	$r(3)$	99%	$r(3)$	99%	VEC

the ADF stationarity test results indicate the necessity to test for cointegration of multivariate series $\mathscr{P}_t$, $\mathscr{Q}_t$, $\mathscr{R}_t$, and $\mathscr{S}_t$ on the next step except for the RCB-C dataset. We nonetheless tested this dataset's series alongside the others in order to verify the stationarity test results. Tables 4.2 and 4.3 demonstrate the results of the Johansen method to test for cointegration of impurity-based and permutation importance measurements with accuracy ($\mathscr{P}_t$ and $\mathscr{Q}_t$) and F1 score ($\mathscr{R}_t$ and $\mathscr{S}_t$), respectively.

4.3.3.1 Gradual vs. Abrupt Drifts

As it can be seen in Table 4.1, among datasets with gradual drifts (RCB-C, RCB-P, RCB-E, RCB-S, NOAA, ELEC), we can accept the alternative hypothesis of stationarity of all importance measurements and predictability performance series tested on the RCB-C dataset. The reason for stationarity of all series in this dataset seems to be constant drift rate in this dataset. Therefore, $\mathscr{P}_t$, $\mathscr{Q}_t$, $\mathscr{R}_t$, and $\mathscr{S}_t$ can be modeled using a standard stationary vector autoregression (VAR) or VAR in levels. However, this would not be the case for the other variants of the RCB dataset where the drift rate is non-constant.

On datasets with abrupt drifts (SEA variants, for example), performance metrics were unanimously nonstationary, and impurity-based importance measurements were stationary. Yet, permutation importance measurements were stationary only for a subset of features. It appears that the reason lies in the fact that every abrupt change in these datasets is permanent until the next drift, hence resulting in a lasting deviation from the long-term trend.

The conintegration results shown in Tables 4.2 and 4.3 show that in all datasets exhibiting gradual drift, the cointegration rank of the impact matrix C based on both eigenvalue and trace statistic is $r > 0$. The impact matrix C consolidates the long-term dynamics of each multivariate series: deviation from the stationary mean as *error* and adjustment speeds to *correct* or revert to the stationary mean over time. That means, there is highly significant evidence that there exists at least one cointegrating relation between any type of FIMs analyzed from the simpler gradient-boosting decision tree (GBDT) auxiliary model and both predictability performance metrics of the main classifier in the face of CD.

Concerning our hypotheses stated in section 4.2.2, we reject null hypotheses 1 to 4, and accept the alternative hypotheses 1 to 4 with significant evidence provided by the results on datasets that exhibit gradual drifts.

It can be concluded that these FIMs follow the long-term trend of the performance metrics even if any of these series are non-stationary. Moreover, even if the FIMs deviate from the performance metrics in short-term (*error*), they revert (*correct*) to the long-term mean of the multivariate

series.

The conintegration results shown in Tables 4.2 and 4.3 are consistent with the stationarity results of Table 4.1 on abrupt drift datasets. On these datasets, we can only reject the null hypotheses 2 and 4 and accept alternative hypotheses 2 and 4 for permutation importance measurements. We fail to reject null hypotheses 1 and 3 on abrupt drift datasets. Since the abrupt changes are sudden and persisting, we expect that the changes in importance measurements of features, except the third noise feature on SEA datasets, also persist for a longer time. That results in fewer cointegrating relations between FIMs and performance metrics. Indeed, the experimental results confirm that in series $\mathscr{P}_t$ and $\mathscr{R}_t$ there is hardly any meaningful relationship between impurity-based importance measurements and either accuracy or F1 score.

4.3.3.2 Stable vs. Unstable Drift Rates

The *rate of drift* in RCB-C and RCB-E is constant and monotonic, respectively. All multivariate series are stationary in these two datasets, except for permutation importance measurements ($\mathscr{H}_t$) on RCB-E. This is further supported by cointegration results where the impact matrix C is full rank based on both eigenvalue and trace statistic for both importance measurements types and accuracy.

On the other hand, the *rate of drift* is non-monotonic and periodic in RCB-P and RCB-S, respectively. The stationarity test results show that all series tested are non-stationary except for permutation importance measurements. Conintegration tests results support this observation where C is restricted to reduced rank for impurity-based multivariate series $\mathscr{P}_t$ and $\mathscr{R}_t$.

Out of the two real-world datasets NOAA and ELEC, NOAA has a more consistent periodic behavior, that is, close to constant drift rate, and does not deviate drastically from its mean in long-term. Therefore, it is not surprising that the stationarity of its performance metrics accuracy ($\mathscr{A}_t$) and F1 score ($\mathscr{B}_t$) are significant. Non-stationarity of this dataset's importance measurements is either failed to be rejected or is inconclusive. That means, while there is significant evidence for non-stationarity of some importance measurements, tests results were insignificant for the others. The ELEC dataset, however, experiences a drastic change where data for two features become available partially through the stream. The accuracy of the classifier in the face of these changes and possibly other phenomenon inducing CD is less significantly stationary, and its F1 score is non-stationary. Similar to NOAA, the importance measurements computed on ELEC dataset are either non-stationary or inconclusive.

Our cointegration results are also consistent here, as for NOAA most series tested are stationary with a significance level of 95%, whereas for ELEC only accuracy is stationary with a significance

of 95%. Moreover, cointegration tests found more stationary linear combinations between FIMs and performance metrics for NOAA than ELEC, implying FIMs follow the long-term stochastic trend of performance metrics accuracy and F1 score more stably.

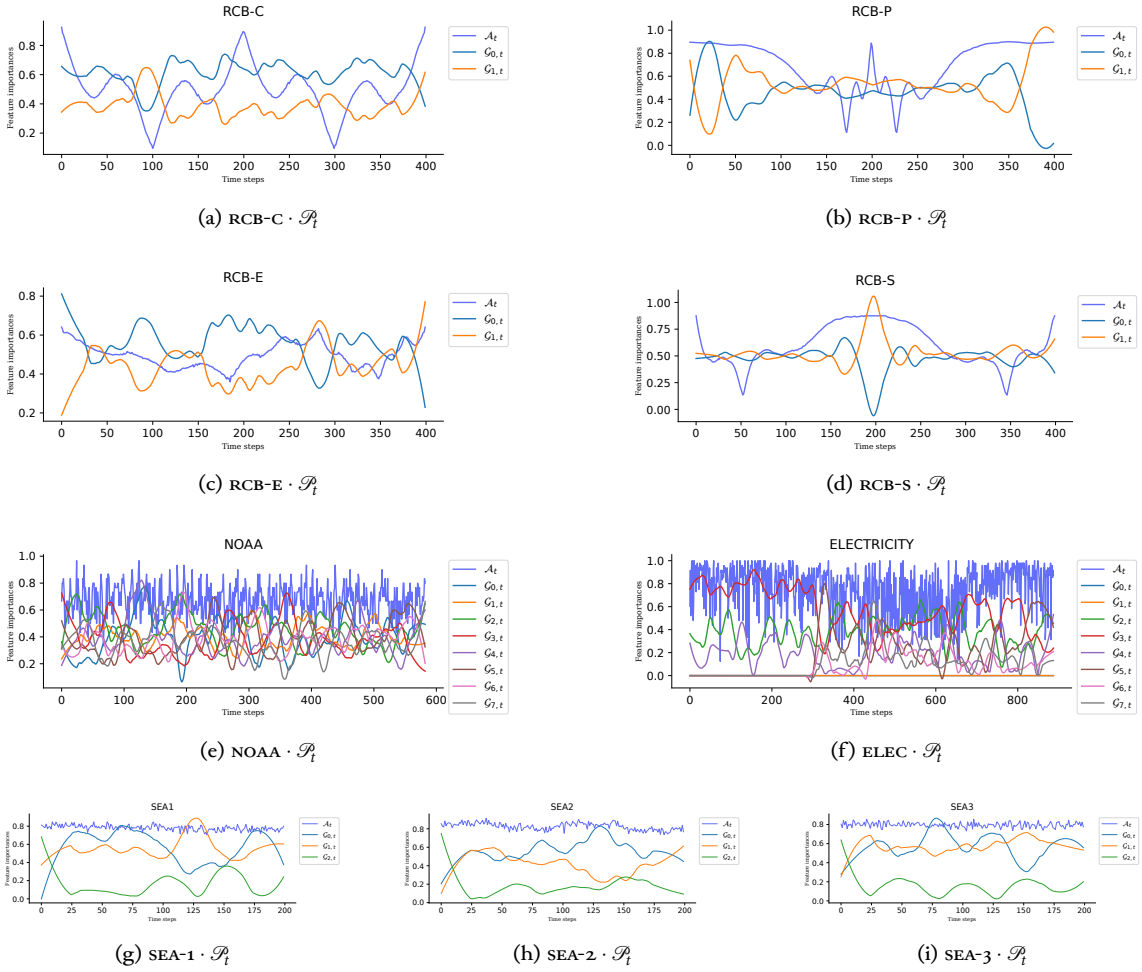
We can conclude that given data with constant or monotonic *rate of drift*, classification models perform more stably in the long run in terms of stationarity of both feature importance measurements analyzed from data, and their predictability performance. In contrast, when provided with data exhibiting non-monotonic or periodic *rate of drift*, both importance measurements and the predictability performance of classification models in long-term become nonstationary and unstable in levels.

4.3.3.3 Impurity-based vs. Permutation Feature Importance Measurements

We observe that the conintegration results indicate that permutation importance measurements are more stable in terms of following long-term trends of performance metrics because the impact matrix C based on both eigenvalue and trace statistic is full rank on all datasets with gradual drift except for ELEC. This means, we can form a stationary VAR in levels with an additional lag out of the VEC model and its error-correction term. Even for ELEC, permutation importance measurements had a higher rank, that is, a larger number of cointegrating relations with the performance metrics. Impurity-based importance measurements achieved this full rank of cointegrating relations on more stable datasets with constant or monotonic rates of drift.

Impurity-based importance measurements have the advantage of being computed as part of the auxiliary model construction; therefore, they are computationally less costly than permutation importance measurements. The former is also more insightful as it abstracts *impurity* of predictor and response variables according to some criterion. In our experiments, impurity-based importance measurements had also much less short-term variance than permutation importance measurements.

We can conclude that impurity-based importance measurements are a viable source of drift information on datasets with constant or monotonic rates of drift with the advantage of less computational overhead, more insight on the behavior of data, and less micro-variation. The test results provide evidence for more consistency and reliability of permutation importance measurements over impurity-based importance measurements if data exhibits periodic or non-monotonic rates of drift, or if this prior information about data is unknown.



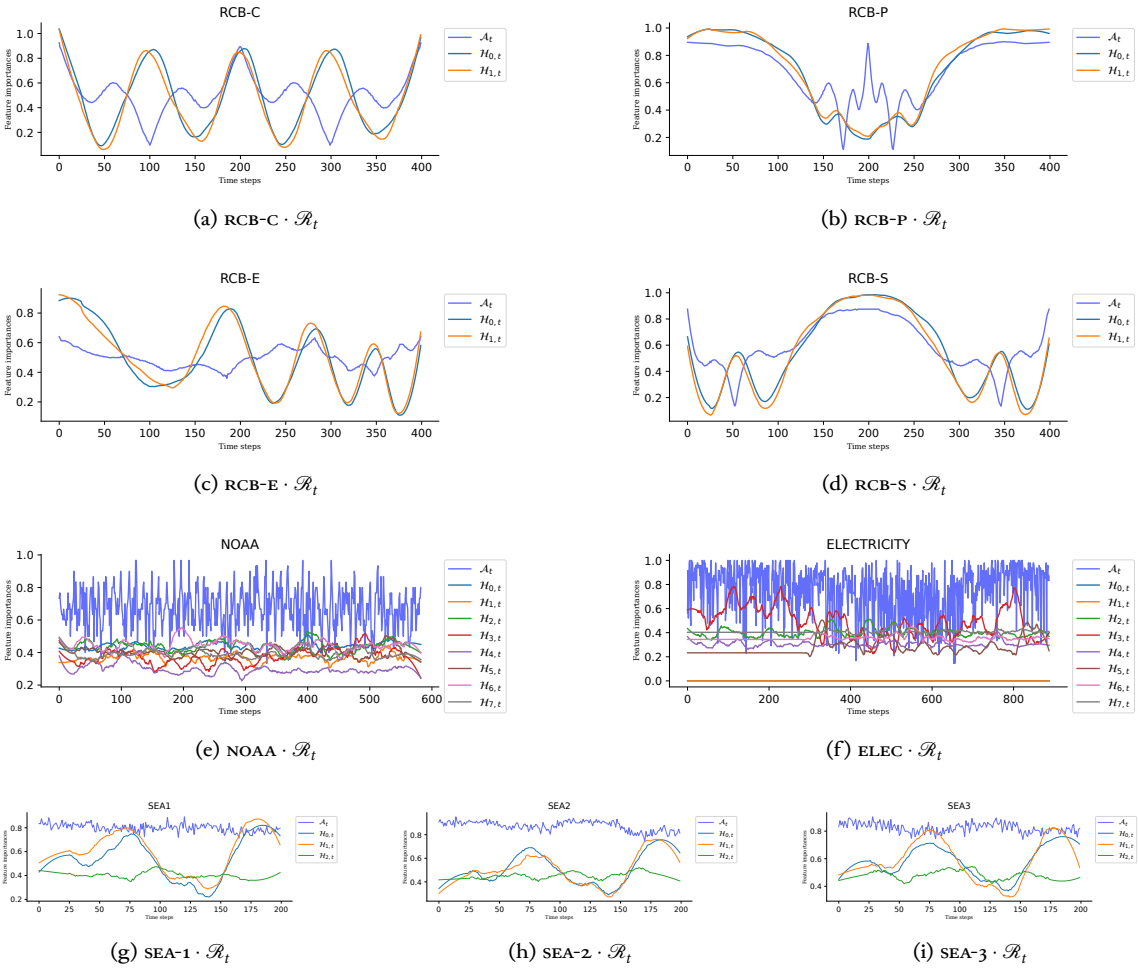
(a) RCB-C · $\mathscr{P}_t$ (b) RCB-P · $\mathscr{P}_t$

(c) RCB-E · $\mathscr{P}_t$ (d) RCB-S · $\mathscr{P}_t$

(e) NOAA · $\mathscr{P}_t$ (f) ELEC · $\mathscr{P}_t$

(g) SEA-1 · $\mathscr{P}_t$ (h) SEA-2 · $\mathscr{P}_t$ (i) SEA-3 · $\mathscr{P}_t$

Figure 4.3: Evolution of the accuracy ($\mathscr{A}_t$) and impurity-based feature importance measures ($\mathscr{G}_t$) undergoing concept drift over time for the datasets.



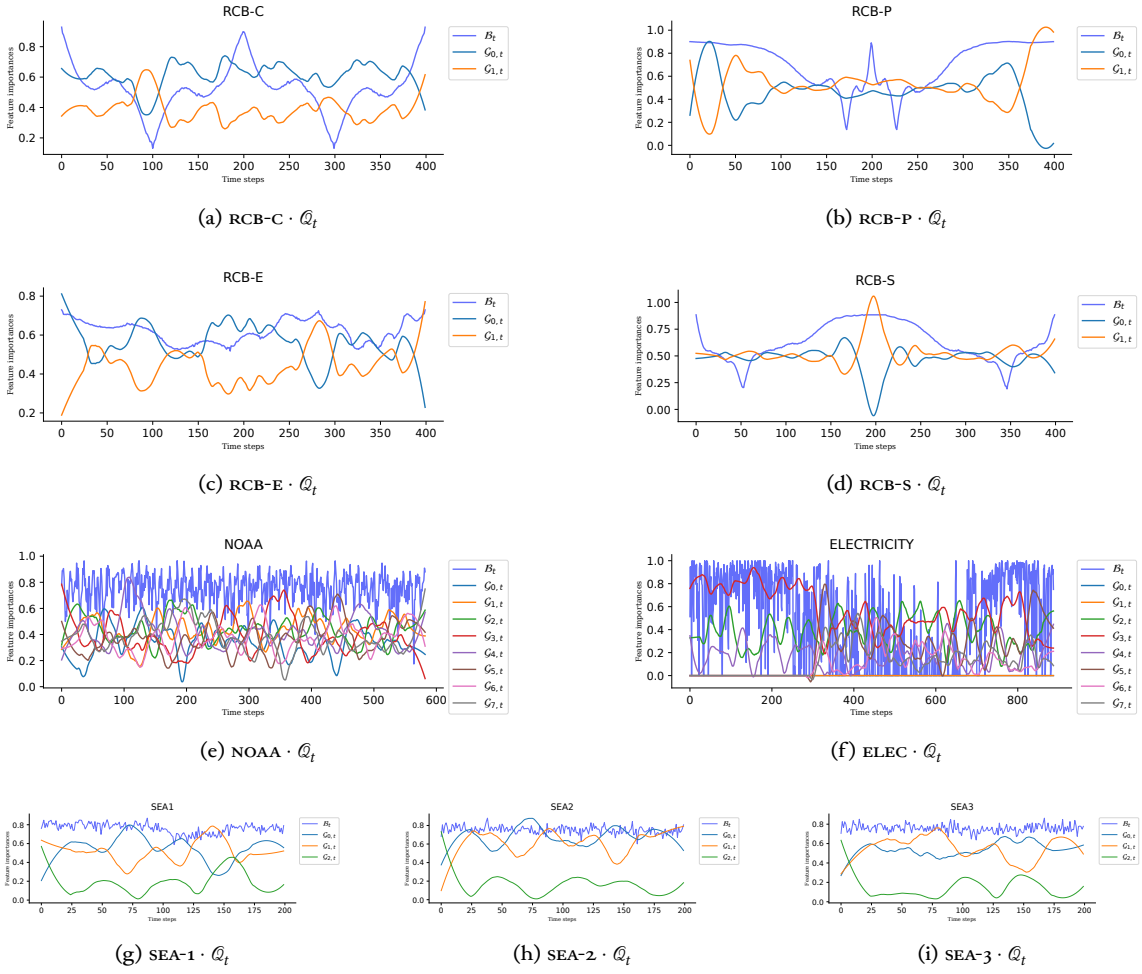
(a) RCB-C · $\mathscr{R}_t$ (b) RCB-P · $\mathscr{R}_t$

(c) RCB-E · $\mathscr{R}_t$ (d) RCB-S · $\mathscr{R}_t$

(e) NOAA · $\mathscr{R}_t$ (f) ELEC · $\mathscr{R}_t$

(g) SEA-1 · $\mathscr{R}_t$ (h) SEA-2 · $\mathscr{R}_t$ (i) SEA-3 · $\mathscr{R}_t$

Figure 4.4: Evolution of the accuracy ($\mathscr{A}_t$) and permutation feature importance measures ($\mathscr{H}_t$) undergoing concept drift over time for the datasets.



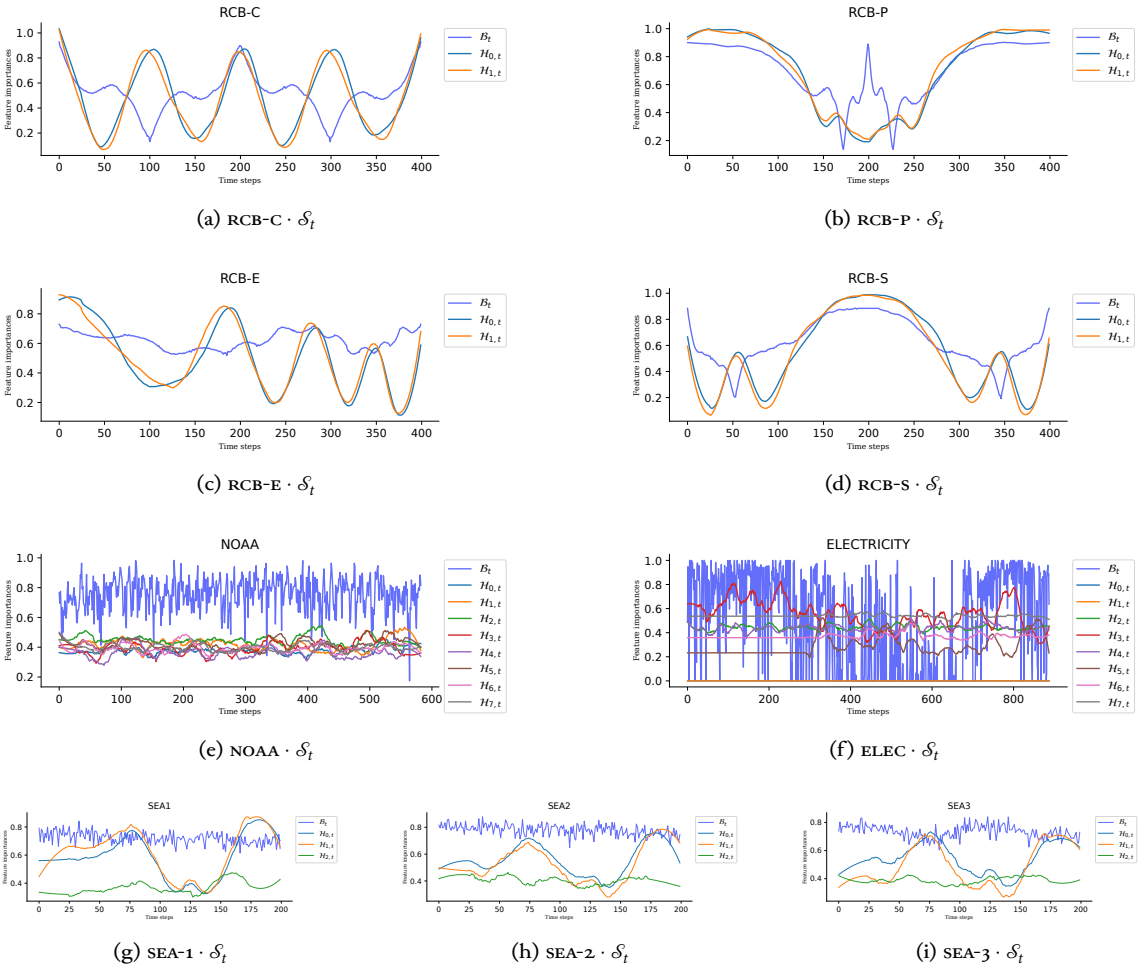
(a) RCB-C · $\mathcal{Q}_t$ (b) RCB-P · $\mathcal{Q}_t$

(c) RCB-E · $\mathcal{Q}_t$ (d) RCB-S · $\mathcal{Q}_t$

(e) NOAA · $\mathcal{Q}_t$ (f) ELEC · $\mathcal{Q}_t$

(g) SEA-1 · $\mathcal{Q}_t$ (h) SEA-2 · $\mathcal{Q}_t$ (i) SEA-3 · $\mathcal{Q}_t$

Figure 4.5: Evolution of the F1 score ($\mathcal{B}_t$) and impurity-based feature importance measures ($\mathcal{G}_t$) undergoing concept drift over time for the datasets.



(a) RCB-C · $\mathcal{S}_t$ (b) RCB-P · $\mathcal{S}_t$

(c) RCB-E · $\mathcal{S}_t$ (d) RCB-S · $\mathcal{S}_t$

(e) NOAA · $\mathcal{S}_t$ (f) ELEC · $\mathcal{S}_t$

(g) SEA-1 · $\mathcal{S}_t$ (h) SEA-2 · $\mathcal{S}_t$ (i) SEA-3 · $\mathcal{S}_t$

Figure 4.6: Evolution of the f1 score ($\mathscr{B}_t$) and permutation feature importance measures ($\mathscr{G}_t$) undergoing concept drift over time for the datasets.

4.3.3.4 Evolution of Feature Importance Measurements

While Tables 4.1 to 4.3 provide a statistical analysis of the relationship between FIMs and the predictability performance metrics of the main classifier, we can further observe the evolution of these series over time more closely in Figures 4.3 to 4.6. For demonstration purposes, we have applied a Savitzky–Golay smoothing filter to impurity-based and permutation importance measurements series $\mathscr{G}$ and $\mathscr{H}$ respectively to reduce noise and emphasize the long-term trends. The statistical tests in Tables 4.1 to 4.3, however, were conducted on the raw series with no smoothing applied.

We can notice how the evolution of either impurity-based or permutation importance measurements provide insight into changes of individual features of data over time, and how they correlate with the classifier's predictive performance. The FIMs in all synthetic datasets (RCB and SEA) provide a faithful, even more clear representation of the classifier's performance metrics.

In NOAA, there is a pronounced drift at around time step 350, which might be due to missing data. While this drift is not much noticeable visually, it affects the classifier's performance nonetheless. This phenomenon, however, is better seen via the lens of FIMs in Figure 4.4e. Features 0–3 (temperature, dew point, sea level pressure, visibility) and 5–7 (maximum sustained wind speed, maximum temperature, minimum temperature) gained importance after this drift, whereas feature 4 (average wind speed) lost some importance.

For instance, in Figures 4.3f, 4.4f, 4.5f and 4.6f, we can observe that (zero-indexed) features 0, 1, 5, 6, and 7 had no impact on the predictability performance of the classifier until around time step 300, likely due to missing data. After that, the importance of features 5–7 increased significantly, whereas the importance of feature 2 decreased. Features 3 and 4 remained consistent, and features 0 and 1 never had any impact on the classifier's performance throughout the stream. This behavior is also reflected in the classifier's performance metrics, where the accuracy and F1 score of the classifier became unstable and deteriorated significantly approximately after time step 300.

Lastly, in SEA datasets, we can observe that the abrupt drifts affect the relevant features 0 and 1 almost equally significantly, whereas the importance of the irrelevant feature 2 (noise) remains low and stable. We conclude that the FIMs provide valuable insight into the behavior of the classifier in the face of CD without prior knowledge of the drifts or the predictive relevance of the features to the target variable.

4.4. Conclusion

In this chapter, we studied the relationship between feature importance measures (FIMs) analyzed from streaming data exhibiting different characteristics of concept drift and the predictability performance metric of the main classifier considering data stream processing application constraints and requirements. We considered two groups of FIMs: impurity-based and permutation feature importance measurements, which are computed over an auxiliary gradient-boosting decision tree (GBDT) ensemble that runs parallel to the main classifier but processes and analyzes the same streaming data more efficiently. As such, the two models used are decoupled: the main classifier has the task of processing the streaming data with the goal of prediction on test instances whose remodeling can be potentially costly. The auxiliary GBDT ensemble has the task of processing the same streaming data with the goal of concept drift detection (CDD) and possibly adaptation of the main classifier to changes in data, but consuming less computational resources and respecting data stream processing constraints. We specifically studied the correlation of detection information, that is, the two types of FIMs extracted from the auxiliary GBDT ensemble, with the performance of the main classifier.

The main outcome of this study is providing evidence for strong correlation between FIMs computed from a decoupled, cost-effective model with the performance of a costly, though more accurate model over time which acknowledges data stream processing requirements and encounters different types and rates of CD.

A key contribution of this study is a novel systematic approach on the concept drift detection and adaptation (CDD&A) problem-solving methodology. Specifically, we investigated the direct correlation between detection information and the predictive performance of the main classifier, rather than treating the problem as a black box. The traditional approach typically evaluates the main classifier's performance only *after* incorporating a CDD&A technique, without considering whether the selected source of information for the CDD&A task has actual meaningful, non-spurious correlation with the classifier's long-term performance. In contrast, our novel approach has enabled us to analyze the long-term dynamics of the performance of the classifier in the presence of CD and its relationship to FIMs as a potential source of detection information.

We further analyzed the aforementioned correlation in regard to major characteristics of data with CD, that is, *rate of drift* (gradual vs. abrupt) and *stability/predictability of drift* (periodicity and monotonicity).

We conclude that the two types of FIMs studied follow the long-term stochastic trend of the performance metrics on gradual drift types of datasets even if any of these FIMs or the performance metrics themselves are nonstationary. Moreover, our results demonstrated that

even if the FIMs deviate from the performance metrics in short-term, they revert to the long-term trend of the performance metrics.

In regard to characteristics of data with CD, our results provide significant evidence that given data with constant or monotonic rate of drift, classification models perform more stably in the long run in terms of stationarity of both FIMs analyzed from data, and their predictability performance. In contrast, when provided with data exhibiting non-monotonic or periodic rates of drift, both FIMs and the predictability performance of classification models in long-term become nonstationary and less stable.

Our results also indicated that impurity-based feature importance measurements are a viable source of detection information on datasets with constant or monotonic rates of drift with the advantage of incurring less computational overhead, more insight on the behavior of data, and less micro-variation. The test results provide evidence for more consistency and reliability of permutation importance measurements over impurity-based importance measurements if data exhibits periodic or non-monotonic rates of drift, or if this prior information about data is unknown.

In summary, by establishing the relationship between FIMs and classifier's performance metrics, we provide:

- strong evidence to employ FIMs as a viable source of detection information for CDD&A applications, that is, detection of and adaptation to changes reactively,
- better understanding of the behavior of CD in the underlying streaming data and processes, and,
- a way to investigate prediction of CD, that is, detection of and adaptation to changes proactively.

We published the results of this study in [4]. In chapter 5, we will leverage the methodology and results of this study to propose a novel CDD&A framework that is founded on the evolution of the FIMs as a common source of information for both CDD&A tasks.

4.4.1. Limitations and Future Work

The augmented Dickey-Fuller (ADF) tests of stationarity are parametric tests which assume that the residuals of the regression model used in the test are normally distributed. In our study, we did not test for the normality of these residuals. Future research could consider using non-parametric tests for stationarity, such as the Phillips-Perron test [72].

Additionally, the Johansen method is vulnerable to data with structural changes. We noticed this shortcoming specifically in datasets exhibiting abrupt drifts. A future study can consider other cointegration tests that are more resilient to structural breaks, such as the Maki cointegration test [64].

In regard to training the cointegrating models, there is need for initial training with relatively large amounts of long-term data in order to achieve an accurate estimation of the cointegrated VEC models. While more training data leading to more accurate modeling of long-term trends stands to be the case, we have demonstrated that this modeling can be performed incrementally and in an online fashion in a data stream processing setting.

In regard to the predictability performance metrics of the main classifier, we can consider other metrics such as prequential AUC [19] that are better suited to online streaming data processing.

A major by-product of the cointegration analysis is the VEC model of the FIMs and performance metrics which consolidates the long-term dynamics of the system. We plan to investigate application of this model to forecast changes in the performance of the classifier when facing CD.

—It's not possible.
—No, it's necessary.

Interstellar (2014)[†]

Chapter 5

Amytis: A Unified Framework for Concept Drift Detection, Adaptation, and Resolution

In this chapter, we present a unified concept drift detection, adaptation, and resolution framework, called *Amytis*, which is based on the multivariate vector error-correction analysis of feature importance measures (FIMs) proposed in chapter 4. The framework is multi-level and uses FIMs as a first-level abstraction of changes (*concept drift*) in data as the common source of information to address both concept drift detection and concept drift adaptation tasks as two facets of the same problem. In this chapter, we will provide details of this framework, its components, and the methodology we used to implement it. Additionally, we introduce the notion of concept drift resolution (CDR) in the context of concept drift in the stream together with a novel solution for the first time to the best of our knowledge.

We use the same CD problem formulation and notations as in chapters 3 and 4, but unlike the previous chapters, we will use superscript instead of subscript, e.g. y^t instead of y_t, to denote the time index of the data streams.

[†]This dialogue between the characters Cooper (a human pilot) and CASE (a robot with advanced AI) occurs when Cooper is trying to dock a spacecraft with a rotating space station. The scene is a metaphor for the challenges of adapting to never-before-seen situations, and conflicting decisions between the human and the machine due to the latter's inability to improvise and adapt to the new environment.

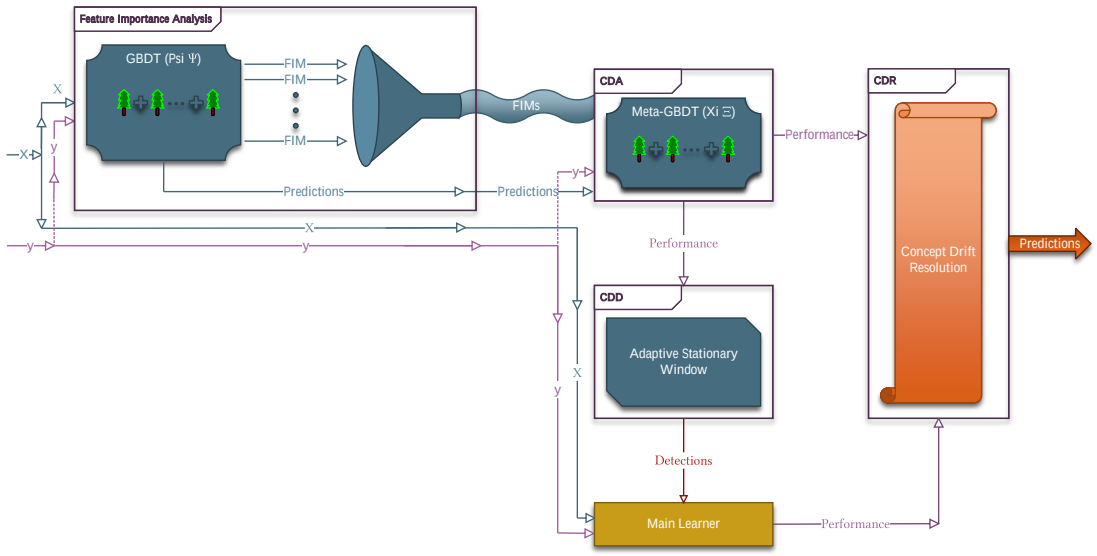


Figure 5.1: The architecture of the proposed unified framework.

5.1. Proposed Framework

The high-level view of the proposed framework is shown in Figure 5.1. The inference (testing phase) and training phase are presented in algorithms 3 and 4, respectively. The train phase is performed whenever the ground truth y^t becomes available for time t. The framework consists of the following five main components:

- The Main Learner
- Feature Importance Analysis
- Concept Drift Adaptation
- Concept Drift Detection
- Concept Drift Resolution

Each of these components is described in detail in the following subsections.

Algorithm 3: Amytis concept drift detection, adaptation, and resolution algorithm—inference phase

Input: Batches of data $B^t = \{X^t_{N\times D}, y^t_N\}$, $B^{t+1} = \{X^{t+1}_{N\times D}\}, t = 1, 2, \ldots$

Compute feature importance measurements $\mathscr{G}^t_{N\times d}$ (eqs. (9) to (12))

Find predictions $\hat{y}^{\Xi,t+1}_N$ by performing concept drift adaptation (algorithm 5)

Store predictions from concept drift adaptation ($\hat{y}^{\Xi,t+1}_N$) as predictions from the algorithm $\hat{y}^{t+1}_N$

if *Concept Drift Detection is enabled* **then**

    Perform concept drift detection (algorithm 6)

    Store whether concept drift was detected as `cd`

if *Concept Drift Resolution is enabled* **then**

    Make inference on the MainLearner on test data $X^{t+1}_{N\times D}$ to get predictions $\hat{y}^{MainLearner^t,t+1}_N$

    Perform concept drift resolution on prediction from the concept drift adaptation ($\hat{y}^{\Xi,t+1}_N$) and predictions from the MainLearner ($\hat{y}^{MainLearner^t,t+1}_N$) (algorithm 7)

    Store resulting predictions from concept drift resolution as predictions from the algorithm $\hat{y}^{t+1}_N$

return $\hat{y}^{t+1}_N$, `cd` *if concept drift detection was performed*

Algorithm 4: Amytis concept drift detection, adaptation, and resolution algorithm—train phase

Input: Batches of data $B^t = \{X^t_{N\times D}, y^t_N\}, t = 1, 2, \ldots$

Maintain Ψ^t and Ξ^t models

Train model Ψ^t incrementally on train data B^t

Train model Ξ^t incrementally on the most up-to-date meta-matrix $\mathbf{M}^{\circledcirc,t-1}_{N\times D+1}$

if *Concept Drift Detection is enabled* ***and*** *concept drift* `cd` *has been detected* **then**

    Maintain the MainLearner

if *Concept Drift Resolution is enabled* **then**

    Maintain a recent window of the performance measurements of the Ξ^t model and the main learner on latest ground truth available

5.1.1. The Main Learner

The main learner is a generic classification or regression model used to predict the target y from the feature vector $\mathbf{x}$ at each window of the stream. The main learner is trained mainly on the initial training data. However, depending on the maintenance strategy, it may be retrained on the new data arriving in the stream and is prone to concept drifts, which may affect its performance. We assume that maintaining the main learner is at least as computationally expensive as the CDD&A strategy adopted to necessitate the use of an auxiliary model for CDD&A tasks.

5.1.2. Feature Importance Analysis

The goal of this component is to compute the feature importance measures (FIMs) of the raw data stream features over time. This serves as a first-level abstraction of data, with the aim of capturing the evolving dynamic relationship between the raw data features and target. This step acts as the foundation on which we built the CDD and CDA components.

To this end, we train a gradient-boosting decision tree (GBDT) model, called Ψ, incrementally and online on raw data features and target. The learning task of this model is the same as the main learner: the predictor is $\mathbf{x}$ and the response is y. However, unlike the main learner which is concerned with minimizing the classification error, we are interested in the feature importance scores of $\mathbf{x}$ when constructing and maintaining this model. Thus, we analyze the FIMs of the raw data features from the Ψ model.

Specifically, we compute and model impurity-based feature importance measurements as $\mathscr{G}_d^t$ and permutation importance measurements $\mathscr{H}_d^t$ for each dimension $d \in D$ as univariate time series models.

In the following discussion, we drop the index t for brevity, assuming all variables are at time t of the stream.

The impurity-based feature importance measurement $\mathscr{G}_d$ is the normalized average total least squares improvement, denoted as FIM in eqs. (9) to (12), contributed by $\mathbf{x}_d$ across all M^Ψ trees in the ensemble Ψ [71].

Let us assume that $\mathbf{x}_d$ is the feature that resulted in the best split in node $c_{d,s}$ in tree $s \in \Psi$, that is, the Friedman's mean squared error (MSE) criterion [34, p. 1202, eq. (35)] reduced the most for this feature during training.

Let us also consider N_c as the number of samples in node $c_{d,s}$, $N_{c,l}$ as the number of samples in

the left child node of $c_{d,s}$, $N_{c,r}$ as the number of samples in the right child node of $c_{d,s}$, and $C_{d,s}$ as the set of all nodes $c_{d,s}$ that split on feature $\mathbf{x}_d$ in tree s. Lastly, FMSI_c, $\text{FMSI}_{c,l}$, and $\text{FMSI}_{c,r}$ are the Friedman's MSE criterion values (impurities) of nodes $c_{d,s}$ and its left and right child nodes, respectively.

The feature importance $\text{FIM}_{d,s,c}$ of feature $\mathbf{x}_d$ in node $c_{d,s}$ of tree s is computed as the reduction in the Friedman's MSE criterion in node $c_{d,s}$ after splitting on feature $\mathbf{x}_d$, weighted by the number of samples in node $c_{d,s}$ and adjusted by the weighted sum of the impurities of the left and right child nodes of $c_{d,s}$. This is shown in eq. (9).

$$\text{FIM}_{d,s,c} = N_c \times \text{FMSI}_c - N_{c,l} \times \text{FMSI}_l - N_{c,r} \times \text{FMSI}_r \tag{9}$$

The importance $\text{FIM}_{d,s}$ of feature $\mathbf{x}_d$ in tree s is then computed as the sum of the feature importances of all nodes in $C_{d,s}$, as shown in eq. (10).

$$\text{FIM}_{d,s} = \sum_{c \in C_{d,s}} \text{FIM}_{d,s,c} \tag{10}$$

The average importance FIM_d of feature $\mathbf{x}_d$ is then computed across all trees s in the ensemble Ψ, as shown in eq. (11).

$$\text{FIM}_d = \frac{1}{M^\Psi} \sum_{s=1}^{M^\Psi} \text{FIM}_{d,s} \tag{11}$$

The impurity-based feature importance measurement $\mathcal{G}_d$ is then computed as the normalized average of the feature importances FIM_d across all trees in the ensemble Ψ, as shown in eq. (12).

$$\mathcal{G}_d = \frac{\text{FIM}_d}{\sum_{d=1}^{D} \text{FIM}_d} \tag{12}$$

These impurity-based importance measurements of feature vector $\mathbf{x}$ are then represented as a multivariate time series $\mathcal{G}_t = \{\mathcal{G}_d^t\}$, for $1 \leq d \leq D$.

Likewise, we denote permutation importance measurements as a multivariate time series $\mathcal{H}_t = \{\mathcal{H}_d^t\}$, for $1 \leq d \leq D$. The permutation-based feature importance measurement $\mathcal{H}_d^t$ is the change in misclassification after noising feature $\mathbf{x}_d$ of test samples at time t by random permutation [18, 17].

5.2. Concept Drift Adaptation

The goal of this step is to proactively and continuously track the changes in the FIMs as a reflection of the changes in the concept. We are interested in analyzing the FIMs from the previous step and their association with the targets y. Such model is used to adjust for the changes in the concept as the importance of features in relation to the target evolves over time. This allows the model to learn the evolving patterns as a second-level abstraction of data, and make robust temporally-adaptive predictions over the course of the stream.

The idea behind our CDA algorithm, described in algorithm 5, is to learn the evolution of the concept by modeling the changes in the FIMs of the raw data features, analyzed from the Ψ model during the feature importance analysis phase. This multi-level analysis is similar to stacked generalization, where the out-of-sample predictions of the first-level models are used as input to the second-level model. However, stacked generalization techniques such as the super learner [56] are challenging to implement in streaming data applications due to high computational cost of training and maintaining the models. Moreover, each time window of the stream can potentially contain a different distribution of the data, which makes it difficult to train a single model that can perform and generalize well across all time windows. Lastly, cross-validation is not feasible in streaming data applications, as it requires the entire dataset to be available at once, and must ensure to preserve the temporal order of the data.

Therefore, we consider a second-level meta-learner Ξ with the task of adapting to the evolution of the FIMs of the raw data features as well as adjusting for the out-of-sample prediction errors of the Ψ model, both in relation to targets y. To this end, we create a meta-matrix $\mathbf{M}^{\circledcirc}$ from the FIMs in the previous step as the first d columns. This will provide the meta-learner Ξ with the necessary information for the first learning task: modeling the changes in the FIMs of the raw data features. The last column/feature of the meta-matrix is the out-of-sample predictions of the Ψ model $\hat{y}_{\Psi}$ at time t on test data at time $t+1$. This allows the meta-learner to learn and correct possible errors in the predictions of the Ψ model. Overall, the meta-matrix $\mathbf{M}^{\circledcirc}$ is used as the predictor for the GBDT model Ξ.

Let $\mathbf{M}^{\circledcirc} \in \mathbb{R}^{N\times(d+1)}$ be a matrix defined as the horizontal concatenation of $\mathcal{G}_{N\times d}$ and $\hat{y}^{\Psi}_{N\times 1}$, where $\mathcal{G}_{N\times d}$ is the matrix whose rows consist of vectors of FIMs computed at time t, i.e., $\mathcal{G}_d$ or $\mathscr{H}_d$, identical for N samples of the current batch, and $\hat{y}^{\Psi}_{N\times 1}$ is a column vector of the out-of-sample (time $t+1$) predictions of the Ψ model. This is shown in eq. (13).

$$\mathbf{M}^{\circledcirc} = \begin{bmatrix} \mathcal{G}_{N\times d} & \hat{y}^{\Psi}_{N\times 1} \end{bmatrix} \in \mathbb{R}^{N\times(d+1)} \tag{13}$$

The response for the Ξ model is target y at time $t+1$. Therefore, training is done only when

Algorithm 5: Concept Drift Adaptation inference

Input: Feature importance measurements $\mathcal{G}^{t}_{N\times d}$, models Ψ^{t} and Ξ^{t}, test data $X^{t+1}_{N\times D}$

1 Make inference on Ψ^{t} model on test data $X^{t+1}_{N\times D}$ to get predictions $\hat{y}_N^{\Psi^t,t+1}$

2 Build the meta-matrix $\mathbf{M}^{\circledcirc,t}$ using feature importance measurements $\mathcal{G}^{t}_{N\times d}$ and predictions from the Ψ^{t} model ($\hat{y}_N^{\Psi^t,t+1}$) (eq. (13))

3 Make inference on Ξ^{t} model on meta-matrix $\mathbf{M}^{\circledcirc,t}$ to get predictions $\hat{y}_N^{\Xi^t,t+1}$

4 **return** *Predictions from the Ξ^{t} model ($\hat{y}_N^{\Xi,t+1}$)*

ground truth y^{t+1} becomes available. It is worth mentioning that even though the dependent variable of the Ξ model is the same as that of the Ψ model, the objectives of the two models are different. The learning objective of the Ψ model is analysis and computation of the FIM s, whereas the learning objective of the Ξ model is to adjust for changes in the concept by correcting the errors in the out-of-sample predictions of the Ψ model augmented by FIMs computed in the previous step. This is accomplished by tracking the changes in the FIMs of the raw data features and the out-of-sample predictions of the Ψ model.

At inference, the predictions of the Ξ model on test data are used as surrogate to the predictions of the main classifier. This proactive approach serves as our first and major line of defense against concept drifts in the stream. The components used so far, that is, the feature importance analysis using the Ψ model, and the concept drift adaptation using the Ξ model, are both decoupled from the main classifier and run in parallel to it.

5.3. Concept Drift Detection

In a streaming data application we might be interested in knowing when a concept drift has occurred in addition to adapting to it. We propose a novel technique to detect concept drifts in the stream based on the same FIMs from the Ψ ensemble and their modeling in the Ξ ensemble. This is done by analyzing the performance of the Ξ model whenever the ground truth y becomes available. While analysis of the performance of the main classifier has been well studied in the literature, our proposed solution technique offers several advantages over existing methods. First, it is based on analysis of the FIMs of the raw data features, which are more informative and robust than considering the raw data features themselves. Second, the FIMs are decoupled from the main classifier, and are more likely to be less affected by the concept drifts than the main classifier. Lastly, consistent monitoring of the main classifier's performance requires continuous training of it at every time window can be computationally expensive, depending on the main

classifier's complexity. On the other hand, the Ψ and Ξ models are more lightweight, trained incrementally with low computational cost, and are independent of the main classifier.

The intuition behind the proposed CDD technique is detecting whether an adaptive window within a recent detection window is stationary or not. This is done by finding and maintaining the largest stationary window within the detection window, and determining if the adaptive window can get minimized due to non-stationarity at the end of the detection window. This is formalized in algorithm 6. Statements in the algorithm starting with ▷ are comments.

While the proposed change-point detection algorithm is general enough to be applied to any time series data, we focus more on a recent window over the performance of the model Ξ up to the point at which ground truth is available. This performance time series model is denoted as $\mathscr{P}^{t-1}$ in algorithm 6. In other words, we look for detecting *recent* period of non-stationarity in the performance of the Ξ model as a reflection of recent changes in the concept. Non-stationary period is determined using augmented Dickey-Fuller (ADF) test [27].

The detection window is a recent window of $\mathscr{P}^{t-1}$, denoted as $\mathscr{K}^{t-1} = \mathscr{P}^{t-1-k:t-1}$, where $\left|\mathscr{K}^{t-1}\right| = k$. The size of the detection window k determines the number of recent performance measurements of the Ξ model to consider for detecting concept drifts. The adaptive window $\mathscr{W}^{t-1}$ within the detection window $\mathscr{K}^{t-1}$ changes size based on stationary analysis where $\left|\mathscr{W}^{t-1}\right| = \omega$. $\mathscr{W}^{t-1}$ is defined in eq. (14).

$$\mathscr{W}^{t-1} = \mathscr{K}^{t-1-k+p_1:t-1-k+p_2}, 10 \leq l \leq \omega = p_2 - p_1 \leq m \leq k \tag{14}$$

In eq. (14):

- l is the minimum size of the adaptive window to reach conclusive stationarity results with a minimum value of 10 for the augmented Dickey-Fuller (ADF) test,
- m is the maximum size of the adaptive window,
- p_1 and p_2 are the indices of the adaptive window within $\mathscr{K}^{t-1}$.

Algorithm 6: Concept Drift Detection

Input: predictions from the Ξ^{t-1} model on data with latest available ground truth $(\hat{y}_N^{\Xi^{t-1},t-1}, y_N^{t-1})$, growth rate $(r^{\oplus})$, shrink rate $(r^{\ominus})$, minimum window size (l), maximum window size (m), regression type (ρ), maximum lag (q), significance level (α), and detection window size (k)

▷ *Initilize window indices and concept drift flag*

$p_1 \leftarrow 0; p_2 \leftarrow 0;$ cd $\leftarrow$ False

▷ *Update the performance model $\mathscr{P}_N^{t-1}$ using the model Ξ^{t-1}'s predictions on the latest ground truth available*

$\mathscr{P}_N^{t-1} \leftarrow \left[\mathscr{P}_N^{t-1}, \text{PerformanceMeasure}(\hat{y}_N^{\Xi^{t-1},t-1}, y_N^{t-1})\right]$

▷ *Create the detection window $\mathscr{K}_N^{t-1}$ from the latest k performance measures*

$\mathscr{K}_N^{t-1} \leftarrow \mathscr{P}_N^{t-1-k:t-1}$

▷ *Initilize the stationary window $\mathscr{W}_N^{t-1}$ to the first l performance measures in the detection window $\mathscr{K}_N^{t-1}$*

$\mathscr{W}_N^{t-1} \leftarrow \mathscr{K}_{t-1-k:t-1-k+l}$

▷ *Adaptive stationary window test over the detection window $\mathscr{K}_N^{t-1}$*

while $p_2 < \left|\mathscr{K}_N^{t-1}\right|$ **do**

  ▷ *Increase the adaptive window size $\left|\mathscr{W}_N^{t-1}\right| = \omega = p_2 - p_1$ by $r^{\oplus}$ until the window size is greater than l*

  while $p_2 - p_1 < l$ **do**

    $p_2 \leftarrow p_2 + r^{\oplus}$

  ▷ *Verify that the window size ω is within the bounds*

  if $l \nleq p_2 - p_1$ ***or*** $p_2 - p_1 \nleq m$ ***or*** *the constraints in eqs. (16) and (17) are not satisfied* **then**

    return *Inconclusive*

  ▷ *Update the stationary window $\mathscr{W}_N^{t-1}$ to the current window indices in the detection window $\mathscr{K}_N^{t-1}$*

  $\mathscr{W}_N^{t-1} \leftarrow \mathscr{K}_{t-1-k+p_1:t-1-k+p_2}$

  ▷ *Test for stationarity of $\mathscr{W}_N^{t-1}$ using ADF test*

  p-value $\leftarrow \text{ADF}(\mathscr{W}_N^{t-1}, \rho, q)$

  ▷ *If the data is stationary, increase the window size ω as long as the window remains stationary and ω is within the bounds*

  while *p-value* $< \alpha$ ***and*** $l \leq p_2 - p_1 \leq m$ ***and*** $p_2 + r^{\oplus} < \left|\mathscr{K}_N^{t-1}\right|$ **do**

    $p_2 \leftarrow p_2 + r^{\oplus}$

    $\mathscr{W}_N^{t-1} \leftarrow \mathscr{K}_{t-1-k+p_1:t-1-k+p_2}$

    p-value $\leftarrow \text{ADF}(\mathscr{W}_N^{t-1}, \rho, q)$

  ▷ *If the window size ω did not grow due to non-stationarity, it is an indication of a change-point*

  if *p-value* $> \alpha$ **then**

    ▷ *Shrink the window size ω as we pass through a non-stationary region but only if the window remains within the bounds*

    if $p_2 - (p_1 + r^{\ominus}) \geq l$ **then**

      $p_1 \leftarrow p_1 + r^{\ominus}$

    else if $p_2 + r^{\ominus} < \left|\mathscr{K}_N^{t-1}\right|$ **then**

      $p_1 \leftarrow p_1 + r^{\ominus}$

      $p_2 \leftarrow p_2 + r^{\ominus}$

  if $l \nleq p_2 - p_1$ ***or*** $p_2 - p_1 \nleq m$ **then**

    Reduce the window size ω as lines 17 to 21

▷ *If the window size ω is shrunk to the minimum window size l or smaller at the end of the detection window $\mathscr{K}_N^{t-1}$, a concept drift is detected*

if $p_2 - p_1 \leq l$ **then** cd $\leftarrow$ True

return cd

Table 5.1: Regression parameter ρ values with corresponding interpretations, model specification of the ADF regression equation, and recommended use cases.

ρ	Interpretation	Model specification	Recommended use case
1	Constant, no deterministic trend	$\Delta\mathscr{W}^{t-1} = c + \gamma\mathscr{W}^{t-2} + \sum_{i=1}^{q} \delta_i \Delta\mathscr{W}^{t-i-1} + \epsilon^t$	Abrupt drifts with long stationary periods and short-term non-stationary phases
2	Constant and deterministic linear trend	$\Delta\mathscr{W}^{t-1} = c + \beta t + \gamma\mathscr{W}^{t-2} + \sum_{i=1}^{q} \delta_i \Delta\mathscr{W}^{t-i-1} + \epsilon^t$	Gradual drifts with constant rate of drift
3	Constant, and deterministic linear and quadratic trends	$\Delta\mathscr{W}^{t-1} = c + \beta t + \theta t^2 + \gamma\mathscr{W}^{t-2} + \sum_{i=1}^{q} \delta_i \Delta\mathscr{W}^{t-i-1} + \epsilon^t$	Gradual drifts with non-constant rate of drift (e.g. accelerating and/or decelerating)

The regression parameter ρ specifies the constant term and the trend order of time series model specification in the ADF test on the adaptive window. Possible values, their interpretations, specifications, and recommended use cases are provided in table 5.1 [80]. We advise a value of $\rho = 1$ for abrupt drift detection when there are long periods of stationarity and short periods of drift in the stream. Values of $\rho = 2$ and $\rho = 3$ are suitable for gradual drift detection as they account for linear and quadratic trends in the stream allowing to capture the gradual change over time. In the case of gradual drifts that exhibit periodic behavior (seasonality), additional steps such as seasonal differencing and seasonal adjustment might be needed to remove the seasonality *before* applying the ADF tests in line 11 of algorithm 6. A common technique for this would be using seasonal-trend decomposition using locally estimated scatterplot smoothing (LOESS) (STL) [24], but it is outside the scope of this study.

The symbols used in the regression model specifications in table 5.1 are as follows:

- c is the constant term (intercept) in the regression model,
- βt is the linear trend term,
- θt^2 is the quadratic trend term,
- $\Delta\mathscr{W}^{t-1}$ is the first difference of the adaptive window $\mathscr{W}^{t-1} - \mathscr{W}^{t-2}$,
- $\gamma\mathscr{W}^{t-2}$ is the lagged level of the adaptive window where γ is the coefficient of the lagged level measuring the tendency of the series to revert to a mean or a trend over time,
- $\sum_{i=1}^{q} \delta_i \Delta\mathscr{W}^{t-i-1}$ are the lagged differences of the series,
- ϵ^t is the error term,
- q is the lag, explained in detail later.

Parameters l and m directly affect sensitivity of the CDD algorithm. The minimum window size l must be set to the expected maximum length of a non-stationary period in the stream. A smaller adaptive window size increases the sensitivity of the algorithm to detect concept drifts, and vice versa. A small value of l allows the adaptive window to shrink more in presence of non-stationarity, which causes the change-point detection to stabilize faster after a change is detected but also increases the risk of false positives. A larger l on the other hand, will make the algorithm more conservative and less sensitive to concept drifts, taking longer to stabilize after a change is detected. The maximum window size m, on the other hand, must be set to the expected minimum length of a stationary period in the stream.

The maximum lag parameter q in the ADF test specifies the maximum number of lags to include in the regression when testing for stationarity, and affects the sensitivity of the change detection. This parameter determines how much the adaptive window $\mathscr{W}^{t-1}$ should be shifted to calculate the correlation for the non-stationarity test. In other words, q decides how much of the past within the adaptive window should be considered to detect a change: the smaller the q, the less sensitive the change detection, because the test considers less of the past. Too large values of q, on the other hand, can be problematic because aggressive *desensitization* leads to too much of the past being processed, which might potentially include an entire non-stationary period. The latter could result in false negatives or delays in detecting changes. If difficult to determine *a priori*, the value of q can be set based on the adaptive window size ω as suggested by Greene [40] and Schwert [79] shown in eq. (15).

$$q = \left\lfloor 12 \left(\frac{\omega}{100} \right)^{1/4} \right\rfloor \tag{15}$$

To ensure the validity and reliability of the non-stationarity test results with sufficient number of samples and to prevent overfitting and multicollinearity, the maximum lag q further constrains the minimum window size l, as follows. If l takes precedence, then the maximum lag q is constrained as eq. (16). If q takes precedence, on the other hand, then the minimum window size l is constrained as eq. (17).

$$q < \frac{l}{2} - \rho - 1 \tag{16}$$

$$l > 2q + 2\rho + 1 \tag{17}$$

Two parameters growth rate $r^{\oplus} \in \mathbb{N}$ and shrink rate $r^{\ominus} \in \mathbb{N}$ are used to increase and decrease the size of the adaptive window, respectively. They can be set according to the characteristics of

CD in the stream if prior knowledge is available. Otherwise, they can be set to default values of 1. Growth rates $r^{\oplus} > 1$ cause the adaptive window to grow faster in absence of concept drifts, suitable when it is known *a priori* that there are long periods of stationarity and concept drifts are rare. Shrink rates $r^{\ominus} > 1$, on the other hand, would cause the adaptive window to shrink faster in presence of concept drifts, which would be more suitable when it is known *a priori* that concept drifts are gradual and/or more frequent.

Lastly, the significance level α is the probability of rejecting the non-stationarity null hypothesis when the adaptive window is actually stationary (Type I error).

The concept drift detection (CDD) algorithm (algorithm 6) starts by initializing window indices p_1 and p_2 representing lower and upper boundaries of the adaptive window $\mathscr{W}^{t-1}$, respectively. The CD flag is initialized to False (undetected) (line 1).

Next, we measure the performance $\mathscr{P}^{t-1}$, for instance accuracy or F_1 score, of model Ξ's predictions $\hat{y}_N^{\Xi^{t-1},t-1}$ from the CDA step provided that ground truth y_N^{t-1} has become available for that time step's predictions (line 2). Lines 3 and 4 create and initialize the detection window $\mathscr{K}^{t-1}$ and adaptive window $\mathscr{W}^{t-1}$, respectively, to the latest k performance measurements from $\mathscr{P}^{t-1}$ (line 3) and the first l of those k performance measurements within the detection window (line 4).

The main loop of the algorithm (line 5) iterates over the detection window $\mathscr{K}^{t-1}$ aiming to find the largest stationary window $\mathscr{W}^{t-1}$ within it. The loop runs until the upper boundary p_2 hits the end of the detection window. At each iteration of the loop, the adaptive window size $\omega = p_2 - p_1$ is first increased by the growth rate $r^{\oplus}$ until it reaches the minimum window size l (lines 6 and 7) to ensure the validity of the adaptive window. If this constraint cannot be satisfied without exceeding the maximum window size m (line 8), the algorithm returns with inconclusive result.

Otherwise, the adaptive window gets adjusted to current indices p_1 and p_2 (line 10). The ADF test is then applied to the adaptive window $\mathscr{W}^{t-1}$ to determine if it is stationary or not (line 11).

If the null hypothesis of the test is rejected and the adaptive window is found to be stationary, we "*inflate*" the window by the growth rate $r^{\oplus}$ and repeat the stationarity test as long as the adaptive window remains stationary and within the bounds (lines 12 to 15).

If at any point we stop due to failing to reject the null hypothesis, indicating non-stationarity of the adaptive window, we "*deflate*" the window by the shrink rate $r^{\ominus}$ to allow the algorithm to slow down and focus on a smaller window as it passes through a non-stationary period (line 16). Deflation of the window, however, is done only if the window is not *compressed* more than the minimum window size l allows (line 17). In the latter case, the algorithm tries to compensate by

increasing the upper boundary p_2 by the same shrink rate $r^{\ominus}$ if this doesn't result in crossing the end of the detection window (line 21).

In summary, the algorithm inflates the window as long as it remains stationary, and deflates it when it becomes non-stationary to a minimum size. *Only then* the minimized window is moved forward to the next position in the detection window until either the window becomes stationary again, or it reaches the end of the detection window. We can deduce that once the algorithm reaches the end of the detection window, the window is greater than the minimum window size l only if it were stationary in the recent past, because stationarity caused the window to inflate. On the other hand, the only way the window could be non-stationary at the end of the detection window is if it were non-stationary in the recent past. The reason is only recent non-stationarity causes the window to deflate to the minimum window size l without room on the upper boundary to inflate or move forward. Therefore, the adaptive window size ω and its comparison with the minimum window size l at the end of the detection window is what we use to determine concept drifts within the detection window (line 24).

5.4. Concept Drift Resolution

In general, the major challenge for a stream processing application undergoing CD is maintaining the performance of its main learning model over time, even with CDD and CDA mechanisms in place. In fact, inclusion of CDD and CDA solutions in parallel could increase the level of complexity in the sense that the application will have to manage the performance of *multiple* models in order to achieve acceptable *overall* performance. This is especially challenging in a non-stationary environment because the performance of uncorrelated models, for instance the main learner and the CDA model, might differ significantly at different points in time. We identify this problem as the concept drift resolution (CDR) problem, which has not been addressed in the literature to the best of our knowledge.

Although the solutions presented in sections 5.2 and 5.3 offer comprehensive approaches to the CDD and CDA problems, they do not address the CDR problem. More specifically, the application that uses Αηγτις or any other CDD&A solution might *still* have to deal with the problem of conventional (main) learner, and various CDD and CDA solutions performing differently due to CD during stream processing.

In this section we propose a solution beyond what CDD and CDA offer to address the CDR challenge: *resolving* the CD by reducing the variance in recent performance of the application. This is done by selecting the model that has outperformed the others *recently*, such that the overall performance of the application is maximized. The proposed solution is model agnostic

and general enough to be applied to any number of diverse models deployed on a stream processing pipeline, whether designed as the main learner, CDA, or CDD models. For simplicity and specificity, in this study we focus on the main learner and the CDA model (Ξ) in this study as one application of the proposed CDR solution.

Thus far, considering the FIMs of the raw data features as a basis to study the CDD and CDA problems, we developed a uniform framework to solve both problems in the same setting. Specifically, the application could adopt a passive approach to continuously track and adapt to changes using the CDA component, and/or a reactive approach to detect the changes and adopt the appropriate strategy using the CDD component, such as remodeling the main learner. However, the application's performance may suffer if it relies only on one of the aforementioned components and that component underperforms at a given time without knowing the exact time of the drift. Therefore, selecting or combining predictions from these models based on their recent performance could potentially improve the overall performance of the application.

The essence of the proposed CDR technique is that the application could take advantage of the information provided by multiple models, the CDA model and the main learner in case of our proposed CDD&A framework, and combine the predictions of both components over time to achieve better performance. For instance, the application could use the CDA component to adapt to the changes in the concept and maintain an acceptable performance until the CDD component detects a drift, and decides that remodeling of the main learner is necessary. Remodeling the main learner in turn might result in a short period of time when the performance of the main learner is better than the performance of the Ξ model. This is because the Ξ model is designed to adapt to the changes in the concept, and might not be able to infer as accurately as a *recently remodeled* and up-to-*recent*-date main learner with lower bias due to the greater complexity of the main learner and its sole objective of the main learning task. Main learner's *burst* of improved performance, however, is ephemeral in presence of CD, and its performance will eventually degrade as the concept continues to drift further over the temporal dimension. This is where the CDA component comes into play again, and the application could switch back to the Ξ model to maintain a reliable performance until the next drift is detected and the main learner is remodeled. In summary, the application can decide which model's predictions to use for the next time step as the final predictions of the application.

As an analogy, consider a narcoleptic pilot of a passenger airliner. The pilot is the main learner, the auto-pilot is the Ξ model (the CDA component), an in-cockpit monitoring system (similar to a driver monitoring system (DMS) for road vehicles) is the CDD component, and lastly the air traffic controller (ATC) is the CDR component. The pilot is responsible for flying the plane with high accuracy as long as they are awake. However, occasionally and depending on the severity

of their condition[1], the pilot gets drowsy or falls sleep, underperforming as a result.

Without the knowledge of the exact time when the pilot will become non-responsive, we have two options. We can wait for the pilot to become non-responsive and then wake them up after the DMS/CDD alarms us. This might result in catastrophic consequences. Alternatively, we can activate the auto-pilot/CDA after take off and rely solely on them to fly the plane. The auto-pilot is trained to adapt to the changes and maintain a reliable performance regardless of whether the pilot becomes non-responsive, but its performance might be suboptimal.

A good balance can be achieved by the ATC/CDR who is responsible for monitoring the performance of the pilot/main learner and the auto-pilot/CDA. The ATC/CDR decides when to switch between the two based on their recent performance. If the pilot has been performing well recently and better than the auto-pilot, the ATC lets the pilot continue to fly the plane. If the pilot becomes non-responsive, for instance, the auto-pilot takes control of the aircraft and/or assists the pilot. The ATC continues to monitor the recent performance of the pilot and the auto-pilot, and switches back to the pilot after the DMS/CDD detects that the pilot is non-responsive, awakes them, and the pilot becomes ready to take over the plane again. It is worth noting that the CDR does not check when the CDD detects a change, but continuously monitors the performance of the CDA and the main learner as there is usually a delay between detection of a drift by the CDD component and the main learner's performance stabilizing after remodeling.

Meanwhile, to the outside observer (that is, the passengers or the *application*), the plane is flying smoothly and safely, and they are unaware of the changes in the cockpit and who may be in charge. This is the essence of the CDR component, which combines the benefits of both the CDD and CDA components.

In order to accomplish this, we propose a technique to analyze the time evolution of the performance of the Ξ model and the main learner. This is done by comparing the recent performance of the main learner with the recent performance of the Ξ model. If former is ''better'' than the latter, likely due to CD in the stream, the application uses the predictions from the Ξ model to maintain a more desired performance until a drift is detected by CDD and the main learner is remodeled accordingly. In other words, the application could use the CDR component to determine the appropriate strategy to decide and adopt based on the time evolution of the performance of the Ξ model (the CDA component) and the main learner (remodeled based on the CDD component's output).

[1] The attentive reader might duly raise the question, ``Why would a remotely sane person in charge of the airliner green-light an unfit pilot in the cockpit in the first place?'' The question is valid and reasonable; however, addressing the relationship between pilot in question and the airliner's CEO satisfactorily is outside the scope of this study.

We will detail the general proposed CDR technique applicable to any number and type of models in the stream processing pipeline. We will then illustrate, through a specific example, the CDR technique applied to the CDA model Ξ and the main learner in the CDD&A framework.

5.4.1. General Concept Drift Resolution Technique

Let $\{P_i^t\}_{i=1}^n$ be a set of n univariate time series models, where P_i^t denotes the i-th time series model representing the performance of the i-th model at time t. The performance of the model is measured using a performance metric, for instance accuracy or F_1 score for a classification task. We analyze the performance of the models over a recent window consisting of u time steps to determine the recent performance of the models, denoted as $\mathscr{R}^t$. The recent performance of the models is then used to decide which model's predictions to use for the next time step as the final predictions of the application.

The trends of the performance measurements alone do not capture how well the two models have been performing recently with sufficient granularity. Therefore, we analyze each univariate time series model P_i^t to compute its recent performance based on two quantities: the trend of the performance measurements over time, calculated as the slope of the least squares solution over a window of u time steps, denoted as β_i^t, and the magnitude of change, calculated as the integral of the performance over the last u time steps, denoted as s_i^t. The recent performance of the model, $\mathscr{R}_i^t$, is then quantified as the magnitude of change scaled by the trend of the performance over the recent window, as shown in eq. (18). At each time step t, we select the model P_i^t with the maximum recent performance $\mathscr{R}_i^t$ value to make predictions for the next time step, thereby maximizing the overall performance. In this section and throughout the remainder of this chapter, the window of u time steps refers to the last u time steps for which the ground truth is available.

$$\mathscr{R}_i^t = \beta_i^t \cdot s_i^t \tag{18}$$

Algorithm 7 provides the general CDR technique to resolve the concept drifts in the stream. The algorithm first iterates over the performance measurements of the models to compute the recent performance of the models $\mathscr{R}^t$ (lines 1 to 4). We provide detail of the computation for β_i^t and s_i^t in sections 5.4.2 and 5.4.3.

The algorithm then selects the model $\mathscr{M}_{i^*}^t$ with the maximum recent performance $\mathscr{R}_i^t$ to make predictions for the next time step (line 5 and eq. (19)).

Algorithm 7: Concept Drift Resolution

Input: Set of univariate time series models $\{P_i^t\}_{i=1}^n$; window size u
Output: Predictions $\hat{y}_{i^*}^t$

1 **for** *each model* P_i^t **do**
2 Compute β_i^t as the slope of least squares fit over recent window $[t-u, t]$ (eq. (29))
3 Compute s_i^t as the magnitude of change of performance over recent window $[t-u, t]$ (eq. (31))
4 Calculate $\mathscr{R}_i^t$ as the product of β_i^t and s_i^t (eq. (18))
5 Select the best model $\mathscr{M}_{i^*}^t$ with maximum $\mathscr{R}_i^t$ (eq. (19))
6 Make predictions $\hat{y}_{i^*}^t$ from the selected model $\mathscr{M}_{i^*}^t$ (eq. (20))
7 **return** $\hat{y}_{i^*}^t$

$$i^* = \arg\max_i \mathscr{R}_i^t \tag{19}$$

The selected model is then used to make predictions for the next time step (line 6 and eq. (20)).

$$\hat{y}_{i^*}^t = \mathscr{M}_{i^*}^t(X^{t+1}) \tag{20}$$

Figure 5.2 demonstrates the concept drift resolution technique on some simulated data for two models $\mathscr{M}_1$ and $\mathscr{M}_2$. The plot shows performance of the two models over time, shown as P_1 and P_2. Recent trends β_1 and β_2, magnitudes of change s_1 and s_2, and the recent performances $\mathscr{R}_1$ and $\mathscr{R}_2$ have been computed for each model over the last $u = 10$ time steps prior to current time $t = 24$.

We can observe from the figure that the two models performed increasingly better from $t = 0$ to $t = 9$, likely due to training incrementally and adapting to the environment. At time $t = 9$, a concept drift occurs, and the performance of $\mathscr{M}_1$ drops significantly while performance of $\mathscr{M}_2$ remains stable. The CDR technique detects the degradation of the performance of $\mathscr{M}_1$ and switches to $\mathscr{M}_2$ for the next time step onwards. At time $t = 14$, another concept drift occurs, and the performance of $\mathscr{M}_2$ deteriorates whereas $\mathscr{M}_1$ starts to become stable again, perhaps due to a detection from the CDD component and a remodeling. Unlike the first CD, however, the CDR technique does not switch back to $\mathscr{M}_1$ immediately, but waits until the performance of $\mathscr{M}_1$ stabilizes and surpasses the performance of $\mathscr{M}_2$ at time $t = 18$. The CDR technique then switches back to $\mathscr{M}_1$ for the next time step onwards as $\mathscr{M}_1$ continues to outperform $\mathscr{M}_2$ consistently during that period. We can also observe that even though $\mathscr{M}_2$ starts to perform

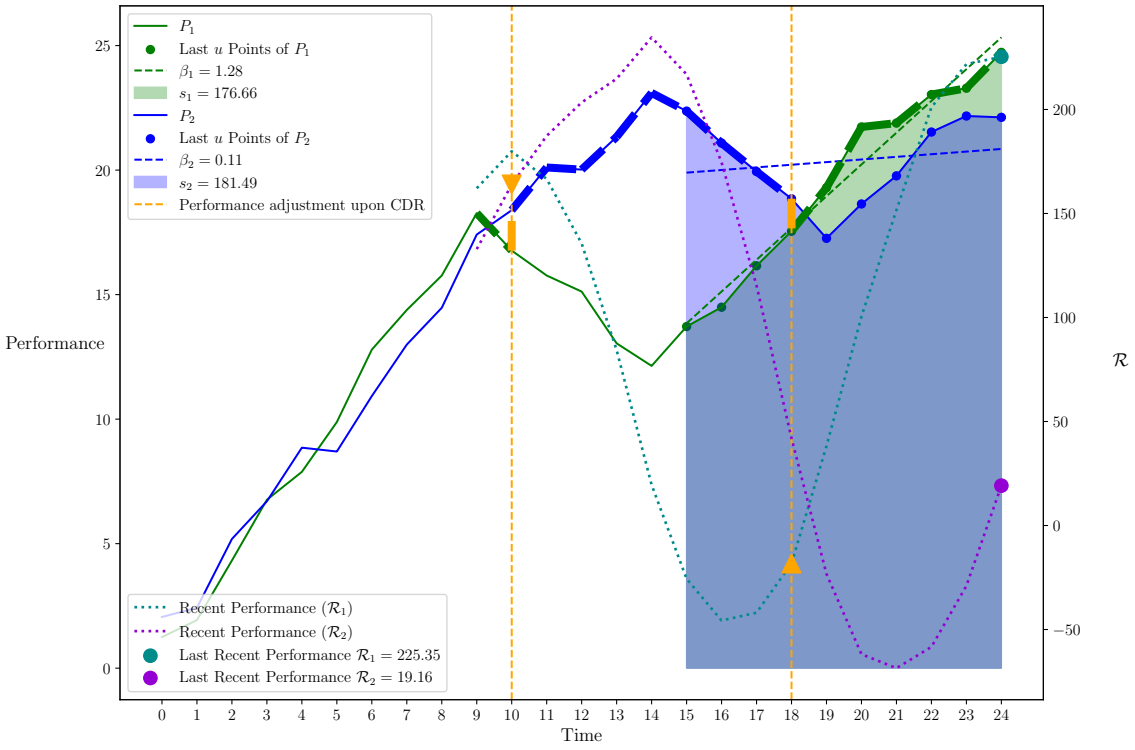


Figure 5.2: Demonstration of the concept drift resolution technique on simulated data. Two concept drift resolutions occur at times $t = 9$ and $t = 14$ when the performance of the two models changes significantly.

better at time $t \geq 19$, the CDR technique does not switch back to $\mathscr{M}_2$ because $\mathscr{M}_1$ continues to outperform $\mathscr{M}_2$.

Without the CDR technique, the application would have to rely on one of the models, either $\mathscr{M}_1$ or $\mathscr{M}_2$. As a result, the application would have to suffer from the performance degradation of the model in use, for example during $t = 9$ to $t = 18$ when $\mathscr{M}_1$ was underperforming, or during $t = 18$ to $t = 24$ when $\mathscr{M}_2$ was underperforming. The CDR technique, instead, allows the application to maintain the optimal level of performance by selecting the model that has been performing better recently, shown as bold dashed lines in the figure.

5.4.2. Trend of Performance Measurements

Given the performance measurements time series P^i, for $1 \leq i \leq u$, we want to fit a linear model, eq. (21), to the last u time steps, where c is the intercept, β is the slope (trend), and ϵ^i is the error term.

$$P^i = c + \beta i + \epsilon^i \tag{21}$$

We create a Vandermonde matrix from the u indexes as eq. (22), as follows:

$$\mathbf{V} = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ \vdots & \vdots \\ 1 & u \end{bmatrix} \tag{22}$$

Equation (23) represents our system, where $x = \begin{bmatrix} c \\ \beta \end{bmatrix}$. The normal equations are given by eq. (24).

$$\mathbf{V}x = P \tag{23}$$

$$\mathbf{V}^T\mathbf{V}x = \mathbf{V}^T P \tag{24}$$

To solve for x, we first construct $\mathbf{V}^T\mathbf{V}$ and $\mathbf{V}^T P$ in eqs. (25) and (26).

$$\begin{aligned} \mathbf{V}^T\mathbf{V} &= \begin{bmatrix} u & \sum_{i=1}^{u} i \\ \sum_{i=1}^{u} i & \sum_{i=1}^{u} i^2 \end{bmatrix} \\ &= \begin{bmatrix} u & \frac{u(u+1)}{2} \\ \frac{u(u+1)}{2} & \frac{u(u+1)(2u+1)}{6} \end{bmatrix} \end{aligned} \tag{25}$$

$$\mathbf{V}^T P = \begin{bmatrix} \sum_{i=1}^{u} P^i \\ \sum_{i=1}^{u} i P^i \end{bmatrix} \tag{26}$$

We then calculate the inverse of $\mathbf{V}^T\mathbf{V}$ in eq. (27),

$$\begin{aligned} \left(\mathbf{V}^T\mathbf{V}\right)^{-1} &= \frac{1}{\det(\mathbf{V}^T\mathbf{V})} \begin{bmatrix} \frac{u(u+1)(2u+1)}{6} & -\frac{u(u+1)}{2} \\ -\frac{u(u+1)}{2} & u \end{bmatrix} \\ &= \begin{bmatrix} \frac{2(2u+1)}{u(u-1)} & -\frac{6}{u(u-1)} \\ -\frac{6}{u(u-1)} & \frac{12}{u(u+1)(u-1)} \end{bmatrix} \end{aligned} \tag{27}$$

where $\det(\mathbf{V}^T\mathbf{V}) = \frac{u^2(u+1)(u-1)}{12}$.

Finally, we solve for x in eq. (28),

$$x = \begin{bmatrix} c \\ \beta \end{bmatrix} = \left(\mathbf{V}^T\mathbf{V}\right)^{-1} \mathbf{V}^T P \tag{28}$$

The trend of the performance measurements at time t is given by the second component of x in eq. (29).

$$\beta^t = [x]_2 \tag{29}$$

5.4.3. Magnitude of Change in Performance Measurements

Given the performance measurements time series P^i, for $1 \leq i \leq u$, we define the magnitude of change in the performance measurements over the last u time steps as eq. (30):

$$s^t = \int_1^u P(t)\, dt \tag{30}$$

The result s^t of this integral can be approximated using the composite trapezoidal rule as eq. (31).

$$s^t \approx \frac{1}{2} \sum_{i=1}^{u} \left(P^i + P^{i+1} \right) \tag{31}$$

5.4.4. Concept Drift Resolution in Amytis

Given the performance measurements of the Ξ model $\mathscr{P}^t$ and the main learner $\mathscr{L}^t$ over the last u time steps, we create the set of univariate time series models $\{P^t\} = \{\mathscr{P}^t, \mathscr{L}^t\}$. We then compute the recent performance of the two models $\mathscr{R}^t_{\mathscr{P}}$ and $\mathscr{R}^t_{\mathscr{L}}$ using the CDR technique (algorithm 7).

The algorithm then makes inference on the model with the maximum recent performance $\mathscr{R}^t$ value for the next time step (eq. (32)).

$$\hat{y}^t = \begin{cases} \hat{y}^t_{\Xi^{t-1}} & \text{if } \mathscr{R}^t_{\mathscr{P}} > \mathscr{R}^t_{\mathscr{L}} \\ \hat{y}^t_{\mathscr{L}^{t-1}} & \text{otherwise} \end{cases} \tag{32}$$

The benefit of accounting for both the recent trend and the magnitude of change in the performance measurements is that it allows the CDR component to make more informed decisions on which model's predictions to use for the next time step. The trend of the performance measurements captures how well the model has been performing recently, while the magnitude of change in the performance measurements captures how much the performance has changed over the last u time steps. For example, this makes the algorithm more resilient to sudden spikes or drops in the performance measurements that might not be indicative of a concept drift but rather a random fluctuation or noise in the stream. Instead, the CDR algorithms waits for some time, depending on the value of u, to see if one model has been consistently outperforming the other before making a decision.

Most importantly, a major benefit of the proposed CDR technique is being model agnostic: not only does the algorithm work independently of either the CDA or the CDD components, it can be used in conjunction with other existing or future supervised CDD&A techniques that can be run in parallel to the main learner. The algorithm can be easily modified to accompany even two or more CDD&A techniques, as it continuously monitors and selects the best model based on their recent performance. This is particularly useful in scenarios where the application has access to multiple models that can be used for the same task, and the performance of these models can vary depending on the context or the data distribution.

5.4.5. Complexity Analysis

In this section, we study the complexity of Amytis, considering each component individually and summarizing the overall time complexity. The analysis is based on the following key parameters:

- N: Number of samples
- D: Number of features
- n: Number of models (typically small and considered constant)
- T: Total number of nodes in the ensemble models (considered constant due to fixed tree depth and number of trees)
- k: Window size used for concept drift detection
- u: Window size used for recent performance for concept drift resolution

5.4.5.1 Feature Importance Measurements

The computation of the impurity-based FIMs $\mathcal{G}_d$ using the GBDT model Ψ involves traversing all nodes in the ensemble. The maximum tree depth h^{Ψ} is a hyperparameter that remains constant throughout the training and test phases. The number of trees M^{Ψ} is also a hyperparameter that can remain constant with a fixed-size ensemble maintenance strategy, e.g., with a growth rate and prune rate of 1. Therefore, the total number of nodes in the ensemble is constant. That is, the time complexity $O(T)$ is $O(1)$, where T is the total number of nodes in the ensemble, and D is the number of features.

5.4.5.2 Concept Drift Adaptation

This component involves the following steps:

1. Making predictions with Ψ^t on test data $X^{t+1}_{N\times D}$ (line 1 of Algorithm 5). Time per sample is $O(M^{\Psi} \cdot h^{\Psi})$, and the total time of this step is $O(N \cdot M^{\Psi} \cdot h^{\Psi})$.
2. Constructing the meta-matrix $\mathbf{M}^{\circledcirc,t}$ (line 2 of Algorithm 5). Time to construct the meta-matrix $\mathbf{M}^{\circledcirc,t}$ is $O(N \cdot D)$. Since $\mathcal{G}^t_{N\times d}$ is identical for all N samples, we can optimize the construction by replicating $\mathcal{G}^t_{1\times d}$ using efficient data structures such as broadcasting

in NumPy, leading to $O(N)$. Then, time to concatenate $\mathcal{G}^t_{N\times d}$ and $\hat{y}_N^{\Psi^t,t+1}$ is $O(N \cdot D)$, resulting in a total time of $O(N \cdot D)$ for this step.

3. Making predictions with Ξ^t on $\mathbf{M}^{\circledcirc,t}$ (line 3 of Algorithm 5). Time per sample is $O(M^{\Xi} \cdot h^{\Xi})$, and the total time of this step is $O(N \cdot M^{\Xi} \cdot h^{\Xi})$.

The total time complexity of the CDA component is $O\Big(N \cdot (M^{\Psi} \cdot h^{\Psi} + M^{\Xi} \cdot h^{\Xi} + D)\Big)$, which can be reduced to $O(N \cdot D)$ by using a fixed-size ensemble maintenance strategy.

5.4.5.3 Concept Drift Detection

The CDD algorithm analyzes the stationarity of a recent performance time series using the ADF test over a detection window of size k. The main loop iterates over the detection window of size k (line 5 of Algorithm 6). Each ADF test within the loop (line 11 of Algorithm 6) has time complexity $O(1)$ due to constant window size ω, and the total number of ADF tests is proportional to k. In this case, the time complexity is linear in window size k.

5.4.5.4 Concept Drift Resolution

In this component, the algorithm computes the recent performance of each model over a window of size u and selects the ''best'' model for predictions. The component involves the following steps from Algorithm 7:

1. For each model i (total n models) (line 1):
   (a) Compute β_i^t (trend): $O(u)$ (line 2).
   (b) Compute s_i^t (magnitude): $O(u)$ (line 3).
   Total complexity per model is $O(u)$.
2. Selecting the best model: $O(n)$ (line 5).
3. Making predictions with the selected model: $O(N)$ (line 6).

Total time complexity of the CDR component is proportional to the window size u and the number of models n, or the number of samples N.

5.4.5.5 Training Phase

The training phase involves incrementally updating the models Ψ^t and Ξ^t with new batches of data, as shown in lines 1 to 5 of Algorithm 4. We can therefore see that the time complexity of this phase is proportional to the combined $N \cdot D$, i.e., the number of samples N and the number of features D by using a fixed-size ensemble maintenance strategy.

5.4.5.6 Overall Complexity

From the above analyses, we can conclude that the overall time complexity is dominated by the combined value $N \cdot D$, specifically from the CDA and training phase.

The Amytis framework demonstrates scalability with respect to both the number of samples N and the number of features D, making it well-suited for large-scale streaming data processing and applications that require efficient handling of high-volume, high-dimensional data. This ensures that computational resources grow proportionally with data size, allowing each component's algorithm to maintain performance as data volumes increase. Additionally, the constant-sized model management (T = constant) prevents resource exhaustion, keeping the processing efficient. Furthermore, the framework's design supports parallelization, particularly in the prediction and training phases, which allows distributing data samples across computing nodes to further improve performance. This is more useful in streaming data applications where the data collected from different sources are to be integrated and processed for CDD&A analyses and decisions. We have not explored this avenue, but it would be a potential research direction.

5.5. Experiments and Results

In this section, we present the experimental results of the proposed framework, Amytis, on synthetic and real-world datasets. We first compare and discuss the performance of different components of the framework, CDD, CDA, and CDR. We then evaluate the performance of the CDD, CDA, and CDR components of the framework, and compare them with two CDD and CDA techniques, OS-ELM [97] and Learn^{++}.NSE [32] respectively. We also compare the performance of the CDR technique with the main learner and the CDA model in the CDD&A framework.

5.5.1. Datasets

In our experiments, we used several synthetic and real-world datasets commonly used in related literature as benchmark (for example in [32, 97, 41, 91]) for a thorough study and analysis of the relationship between FIMs and the main learner's long-term performance in the face of concept drift. These datasets exhibit various characteristics of concept drift, such as abrupt, gradual, and periodic drifts with varying magnitudes, durations, and rates of drift, and are good representatives for evaluating the performance of the proposed CDD, CDA, and CDR techniques.

We already presented these datasets in details in chapters 3 and 5; however, here we provide a brief overview of the datasets used in our study for convenience. We used the following synthetic datasets:

- *Rotating checkerboard (RCB)* [55]. We used the parameters as in [32], and considered the following four rates of CD as constant (RCB-C), pulse (RCB-P), exponential (RCB-E), and sinusoidal (RCB-S). Each variant has 400 batches with a batch size of 1024 instances.
- *Streaming Ensemble Algorithm (SEA)* [85]. It has three continuous features, two of which affect the decision boundary while the third one is noise. We used the threshold values θ as in [85, 32] for SEA-1 and SEA-2. This threshold changes three times suddenly throughout the dataset, resulting in three abrupt drifts. Each dataset consists of 200 batches of streams of size 250 instances for each train and test sets.

  For SEA-3, we used $\theta = 9.5, 7.0, 9.0, 8.0$ with the same batch size and number of batches as SEA-1 and SEA-2 for training, but extended the test set to 650 batches of 250 instances each and mirrored the thetas every 150 batches. This results in a total of 12 abrupt drifts through the test set, with the goal of challenging the CDD&A techniques more than the ones used in related work.

We used the following two real-world datasets on which we performed regression analysis, both of which exhibit gradual periodic drifts.

- *Bellevue weather dataset (NOAA)* [87]. This dataset consists of eight features as daily weather measurements, and two classes (''rain'' and ''no rain''). It has 605 batches, each containing 30 instances, with the first 36 batches used as training set.
- *Electricity dataset (ELEC)* [44, 36]. This dataset consists of five features, affecting the change of electricity price, and two classes (''up'' and ''down''). It contains a total of 944 batches with a batch size of 48 with the first 56 batches used as training set.

Table 5.2: Hyper-parameters for models Ψ and Ξ.

	Ensemble size	Maximum depth	Learning rate	Subsample	Growth rate	Pruning rate
Ψ	10	5	0.1	1.0	1	1
Ξ	12	5	0.1	1.0	1	1

5.5.2. Experimental Setup

We conducted our experiments on a Debian 12 Bookworm desktop computer with an Intel Core i9-14900K CPU and 64 GB of RAM. It is worth noting that the memory utilization by the Amytis framework, excluding the datasets, was less than 220 KB on average. No computations were performed on GPU. The experiments were conducted using Python 3.12.3 and the following libraries: NumPy 1.26.4 [45], scikit-learn 1.5.0 [71], and statsmodels 0.14.2 [80] as well as our implementation of stream processing engine streampy. As the main learner, we used a decision tree classifier with a maximum depth of 5 and a learning rate of 0.1, and a multilayer perceptron with one hidden layer of 100 neurons and a learning rate of 0.001. As the CDR performance indicator metrics (P_i^t), we used accuracy (ACC), F1 score, and area under the receiver operating characteristic curve (ROC AUC) score. We repeated each experiment 30 times and report the average and standard deviation (in parentheses) of the results for each combination of datasets, main learner, and CDR performance metric, for a total of 1620 runs. The results include mean and standard deviation (in parentheses) of the metrics and run time of the algorithms excluding the time taken to load the data and preprocess it. Details of the experiments on the synthetic and real-world datasets are provided in section 4.3.2.

The hyper-parameters of the Ψ and Ξ models are listed in Table 5.2, which we set fixed according to our exploratory experiments for all runs.

- **Ensemble size** is the maximum number of boosting rounds in the GBDT model allowed for Ψ and Ξ models. The few number of boosting rounds is chosen to prevent overfitting and to allow the model adapt quickly to changes in the concept.
- **Maximum depth** is the maximum depth of each decision tree in the GBDT model.
- **Learning rate** is the step size at each iteration of the boosting process.
- **Subsample** is the fraction of the training data to sample at each boosting round of the GBDT model. Commonly used values for GBDT models are less than 1 to prevent overfitting. However, we set this value to 1 because of the limited number of samples available in each batch of the stream.

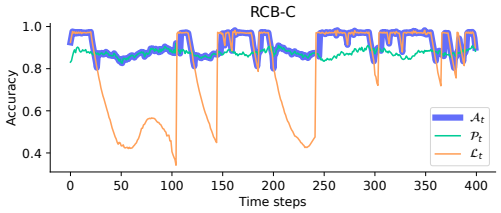
- **Growth rate** and **prune rate** determine the number of trees to add or remove at each time step of the stream, respectively without exceeding the ensemble size. As the ensemble of trees grows throughout the stream, the growth rate is set to 1 to add one tree at each time step. The prune rate is also set to 1 to remove one tree at each time step. This simple maintenance strategy guarantees that the ensemble size remains constant throughout the stream. However, other strategies can be used to maintain the ensemble size, e.g., the one proposed by Wang et al. [91].

5.5.3. Comparison of CDD, CDA, and CDR Techniques

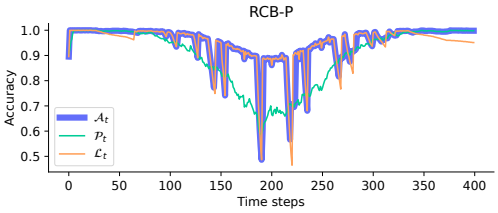
Tables 5.3 and 5.4 show the performance of the CDD, CDA, and CDR components of the Aηγτis framework on the synthetic and real-world datasets using a decision tree and a multilayer perceptron as the main learner, respectively. Figures 5.3 to 5.8 demonstrate the detailed performance of the Aηγτis components for one run of these results for the duration of the stream. The shaded area in the figures represents the standard deviation of the results over 30 runs.

As can be seen from the results, the CDA component outperforms the main learner in all datasets except RCB-P. This does not come as a surprise, as the RCB-P dataset contains a Gaussian pulse as the drift rate, which makes it a challenging type of drift for CDA techniques. The mean performance over the stream, however, does not provide a complete picture of the performance of the models. There are certain times through the stream where the main learner underperforms the CDA component, even if briefly. This is where the CDR component comes into play, which allows detect these instances and switch to the CDA model (like an autopilot) to take charge and maintain a reliable performance until the drift is detected by the CDD component and the main learner is remodeled accordingly. This is evident in Figures 5.3b, 5.4b, 5.5b, 5.6b, 5.7b and 5.8b at times $t = 30$, 220, 266, 277, and 340. On the other hand, after the main learner is remodeled and starts to perform better than the CDA model, the CDR component switches back to the main learner, as seen at times $t = 110$, 145, 224, 271, 314, and 323. This overall improvement on the performance of both the main learner and the CDA model is evident in the results of the CDR component in Tables 5.3 and 5.4, which demonstrates the effectiveness of the CDR component in maintaining a reliable performance throughout the stream.

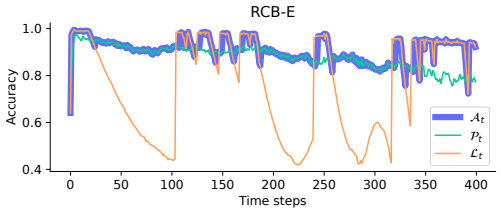
Similarly, the CDR component maintains an overall better performance than the CDA component by using the predictions of the main learner whenever the main learner outperforms the CDA model in all the other datasets, as seen in Tables 5.3 and 5.4 and Figures 5.3 to 5.8.



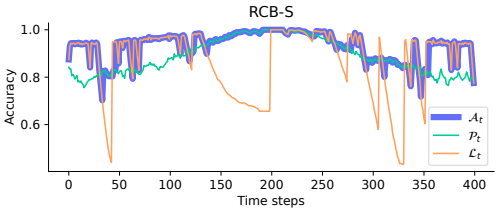
(a) RCB-C · DT · ACC



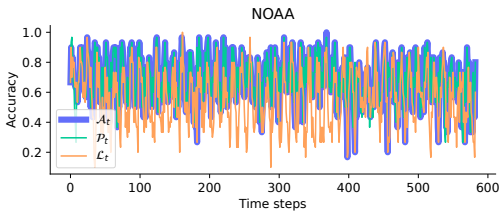
(b) RCB-P · DT · ACC



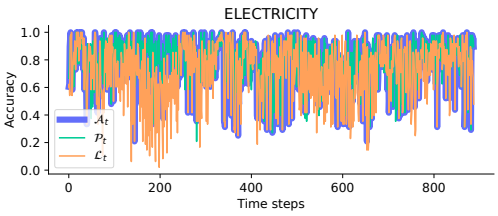
(c) RCB-E · DT · ACC



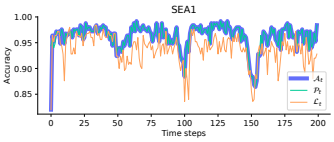
(d) RCB-S · DT · ACC



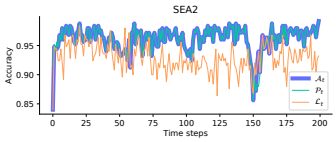
(e) NOAA · DT · ACC



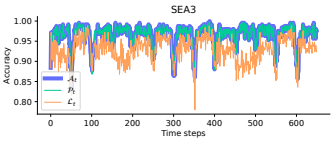
(f) ELEC · DT · ACC



(g) SEA-1 · DT · ACC

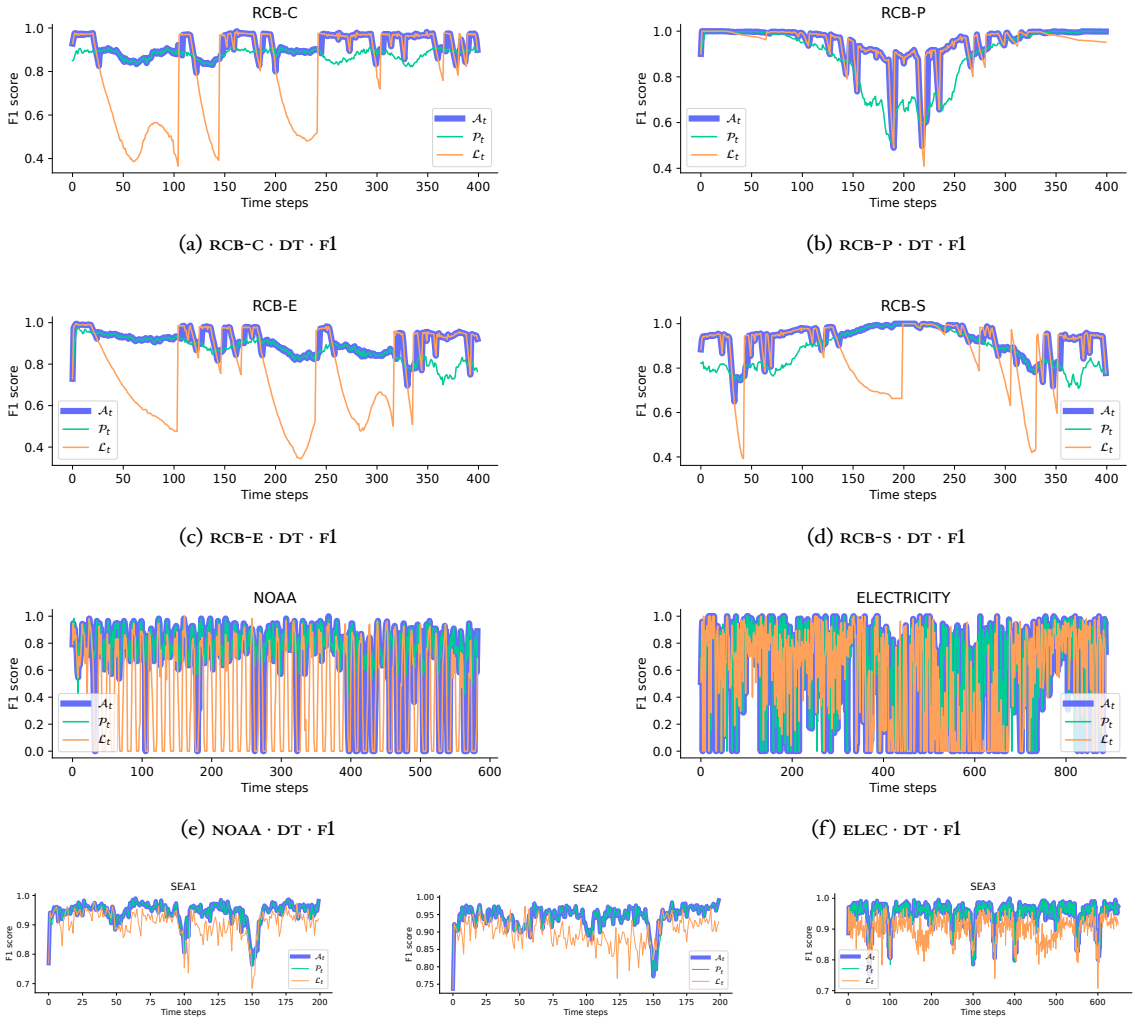


(h) SEA-2 · DT · ACC



(i) SEA-3 · DT · ACC

Figure 5.3: Accuracies of a decision tree as the main learner (CLF · DT–CDD) as $\mathscr{L}_t$, Ξ–CDA as $\mathscr{P}_t$, and the application ($\mathscr{A}$–CDR) as $\mathscr{A}_t$ over the stream for one run on the datasets.



(a) RCB-C · DT · F1

(b) RCB-P · DT · F1

(c) RCB-E · DT · F1

(d) RCB-S · DT · F1

(e) NOAA · DT · F1

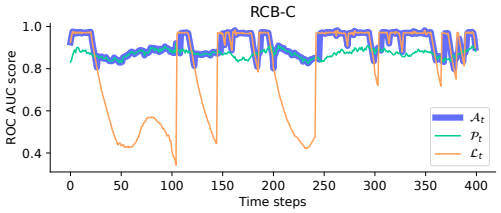
(f) ELEC · DT · F1

(g) SEA-1 · DT · F1

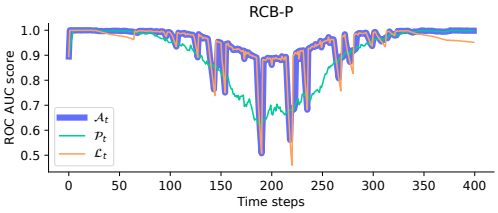
(h) SEA-2 · DT · F1

(i) SEA-3 · DT · F1

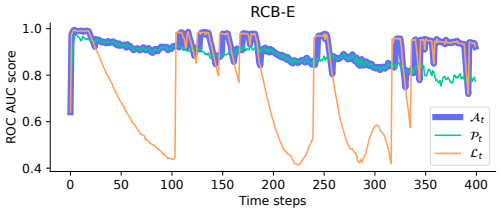
Figure 5.4: F1 scores of a decision tree as the main learner (CLF · DT–CDD) as $\mathscr{L}_t$, Ξ–CDA as $\mathscr{P}_t$, and the application ($\mathscr{A}$–CDR) as $\mathscr{A}_t$ over the stream for one run on the datasets.



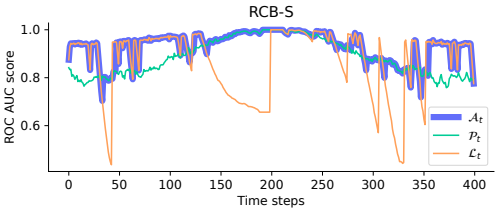
(a) RCB-C · DT · ROC AUC



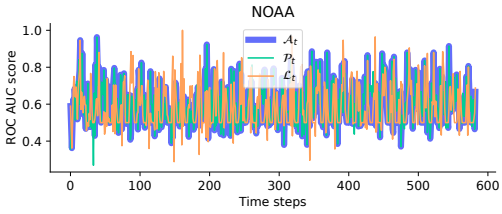
(b) RCB-P · DT · ROC AUC



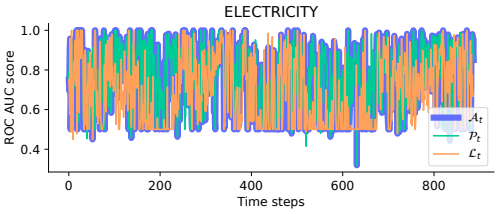
(c) RCB-E · DT · ROC AUC



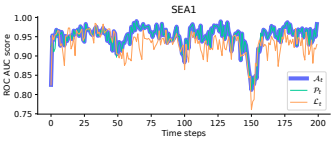
(d) RCB-S · DT · ROC AUC



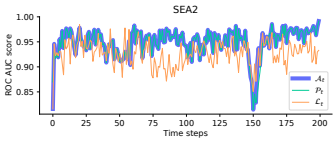
(e) NOAA · DT · ROC AUC



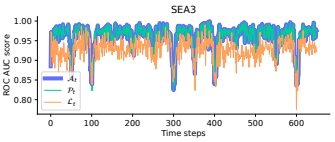
(f) ELEC · DT · ROC AUC



(g) SEA-1 · DT · ROC AUC

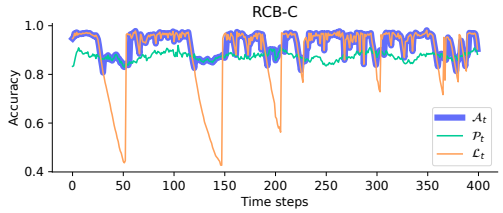


(h) SEA-2 · DT · ROC AUC

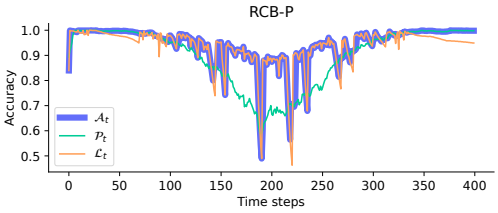


(i) SEA-3 · DT · ROC AUC

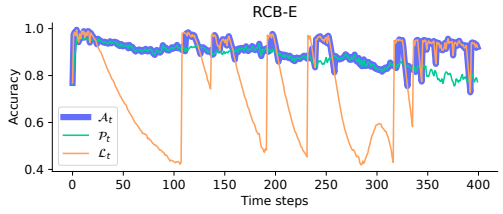
Figure 5.5: ROC AUC scores of a decision tree as the main learner (CLF · DT–CDD) as $\mathscr{L}_t$, Ξ–CDA as $\mathscr{P}_t$, and the application ($\mathscr{A}$–CDR) as $\mathscr{A}_t$ over the stream for one run on the datasets.



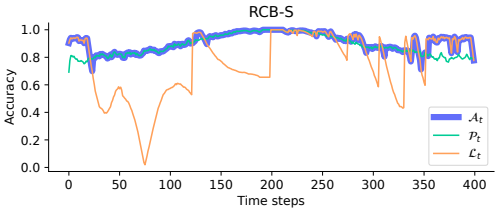
(a) RCB-C · MLP · ACC



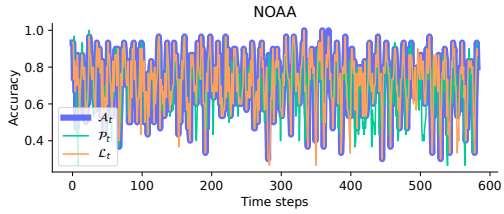
(b) RCB-P · MLP · ACC



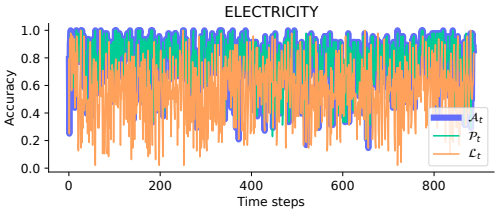
(c) RCB-E · MLP · ACC



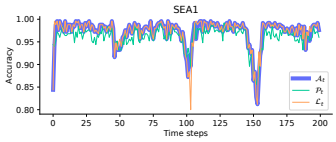
(d) RCB-S · MLP · ACC



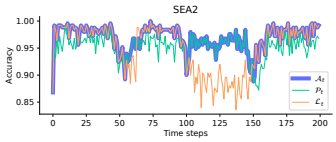
(e) NOAA · MLP · ACC



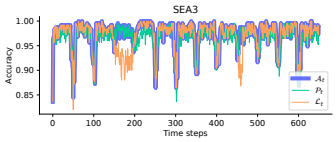
(f) ELEC · MLP · ACC



(g) SEA-1 · MLP · ACC

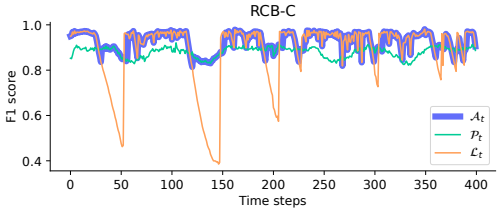


(h) SEA-2 · MLP · ACC

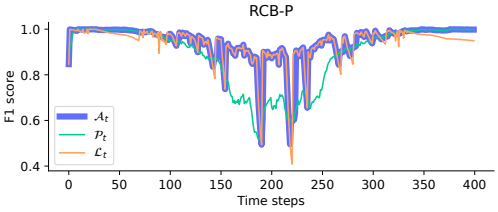


(i) SEA-3 · MLP · ACC

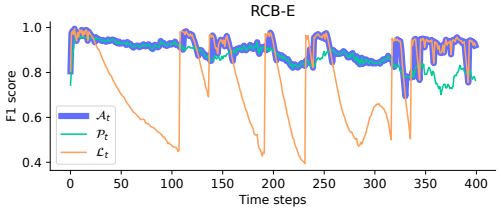
Figure 5.6: Accuracies of a multilayer perceptron as the main learner (CLF · MLP–CDD) as $\mathscr{L}_t$, Ξ–CDA as $\mathscr{P}_t$, and the application ($\mathscr{A}$–CDR) as $\mathscr{A}_t$ over the stream for one run on the datasets.



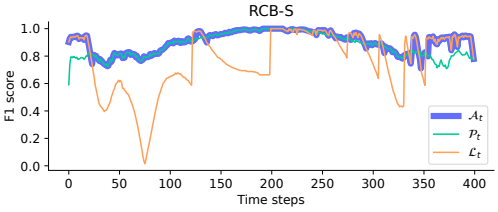
(a) RCB-C · MLP · F1



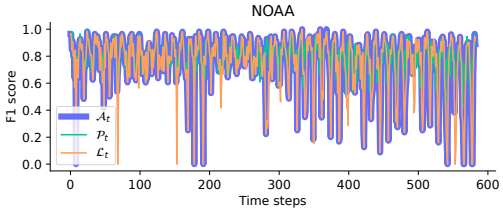
(b) RCB-P · MLP · F1



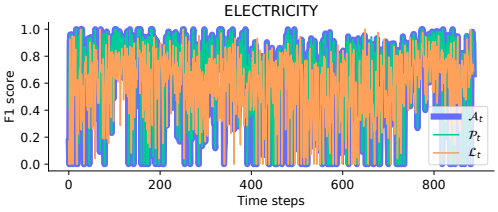
(c) RCB-E · MLP · F1



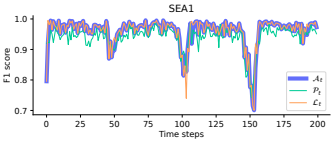
(d) RCB-S · MLP · F1



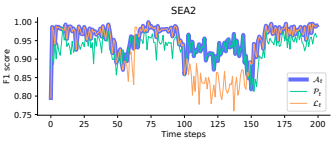
(e) NOAA · MLP · F1



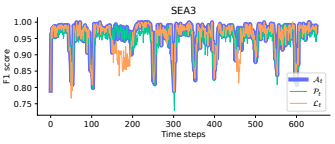
(f) ELEC · MLP · F1



(g) SEA-1 · MLP · F1

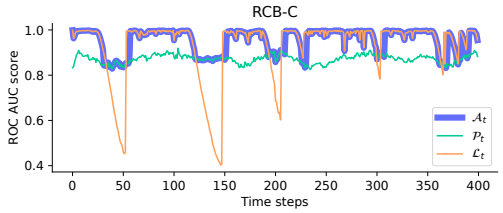


(h) SEA-2 · MLP · F1

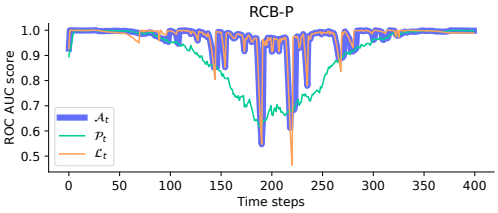


(i) SEA-3 · MLP · F1

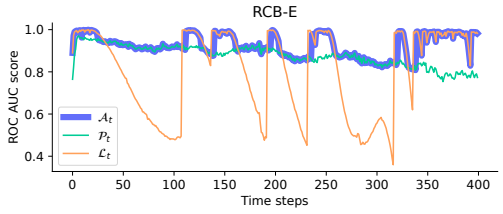
Figure 5.7: F1 scores of a multilayer perceptron as the main learner (CLF · MLP–CDD) as $\mathscr{L}_t$, Ξ–CDA as $\mathscr{P}_t$, and the application ($\mathscr{A}$–CDR) as $\mathscr{A}_t$ over the stream for one run on the datasets.



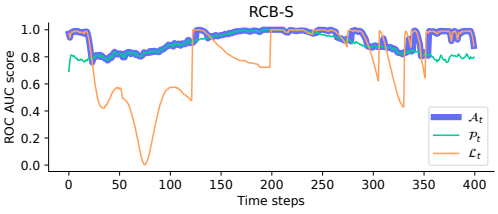
(a) RCB-C · MLP · ROC AUC



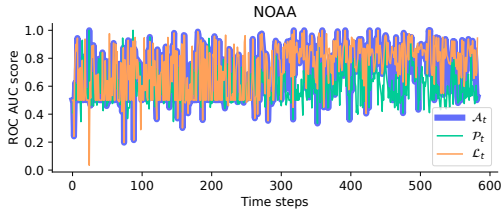
(b) RCB-P · MLP · ROC AUC



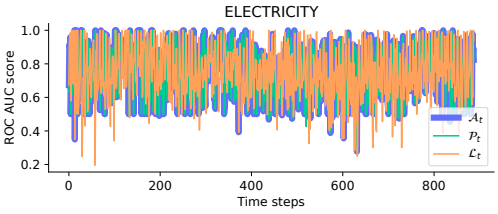
(c) RCB-E · MLP · ROC AUC



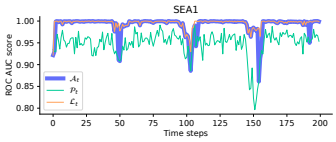
(d) RCB-S · MLP · ROC AUC



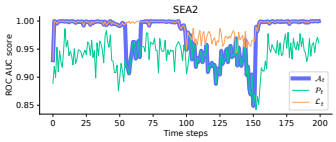
(e) NOAA · MLP · ROC AUC



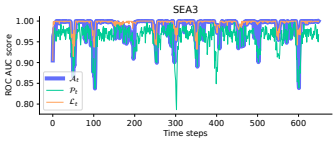
(f) ELEC · MLP · ROC AUC



(g) SEA-1 · MLP · ROC AUC

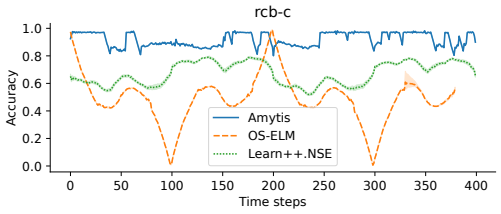


(h) SEA-2 · MLP · ROC AUC

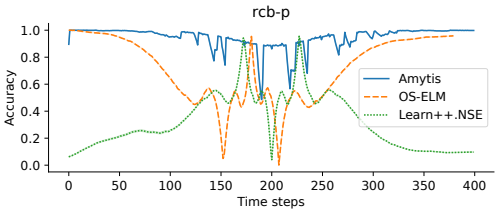


(i) SEA-3 · MLP · ROC AUC

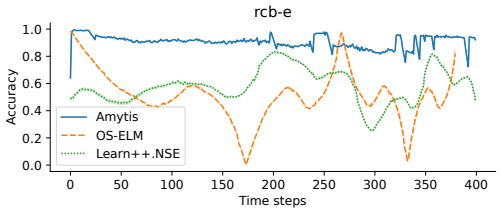
Figure 5.8: ROC AUC scores of a multilayer perceptron as the main learner (CLF · MLP–CDD) as $\mathscr{L}_t$, Ξ–CDA as $\mathscr{P}_t$, and the application ($\mathscr{A}$–CDR) as $\mathscr{A}_t$ over the stream for one run on the datasets.



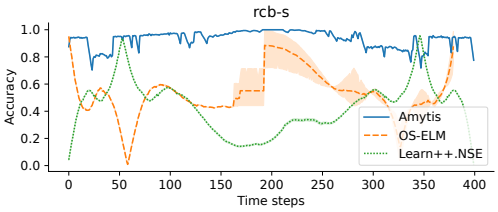
(a) RCB-C · DT · ACC



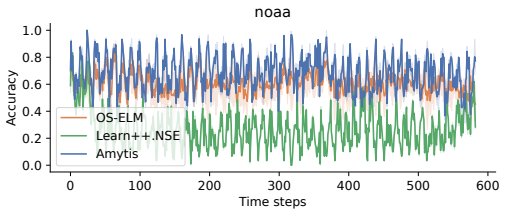
(b) RCB-P · DT · ACC



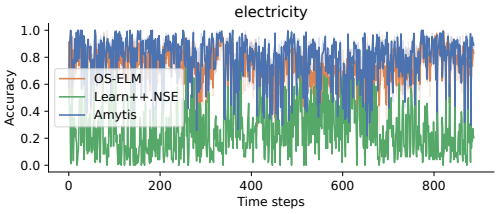
(c) RCB-E · DT · ACC



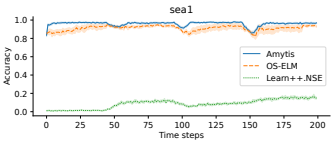
(d) RCB-S · DT · ACC



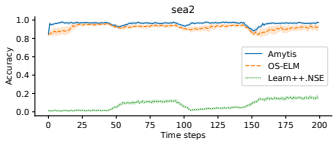
(e) NOAA · DT · ACC



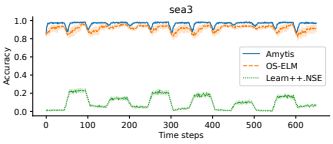
(f) ELEC · DT · ACC



(g) SEA-1 · DT · ACC

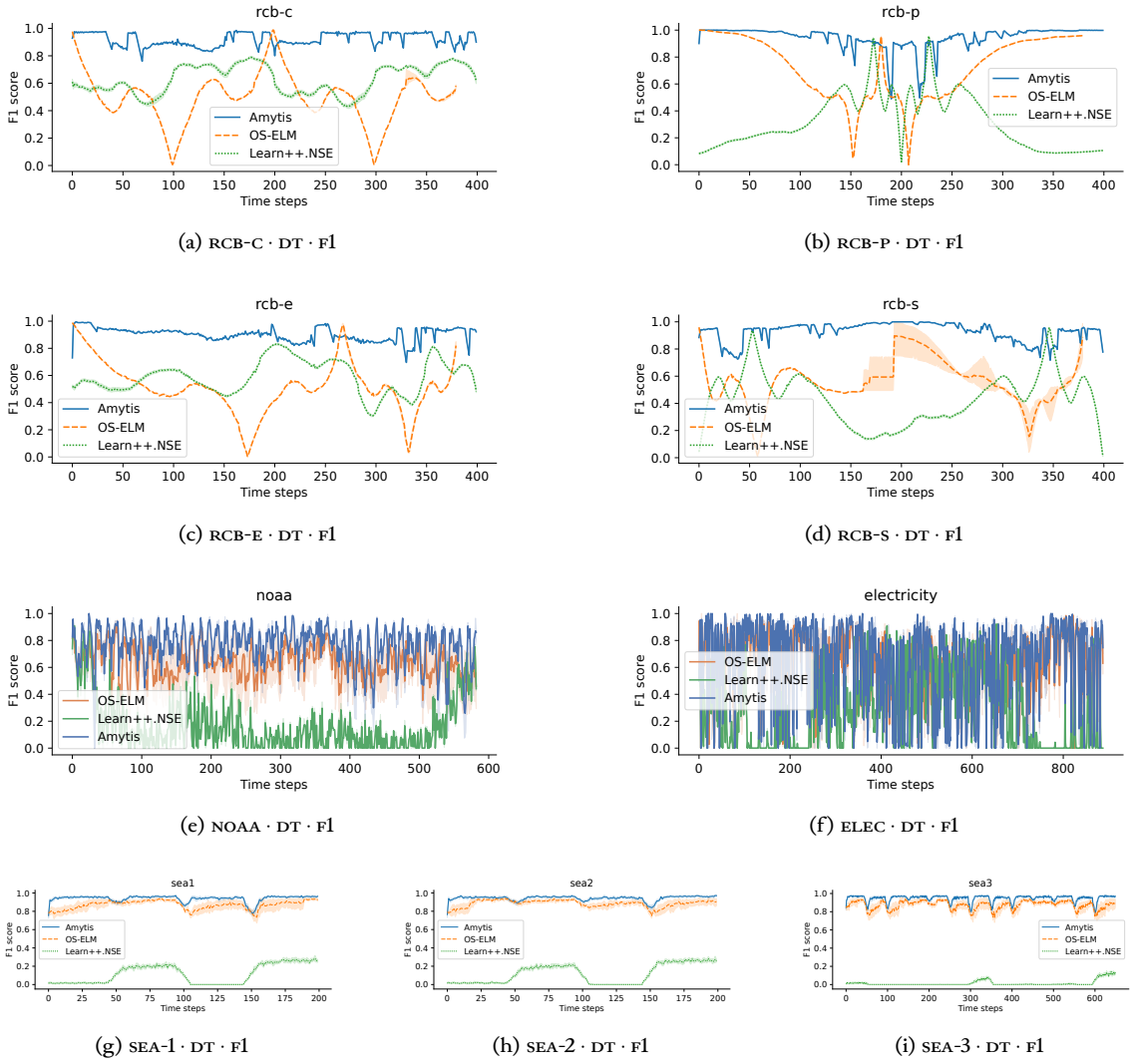


(h) SEA-2 · DT · ACC



(i) SEA-3 · DT · ACC

Figure 5.9: Accuracies of OS-ELM [97], Learn^{++}.NSE [32] and Amytis (CDR–ACC) on the datasets. All three techniques used a decision tree as the main learner (CLF · DT). Amytis's CDR component used accuracy as the performance metric.



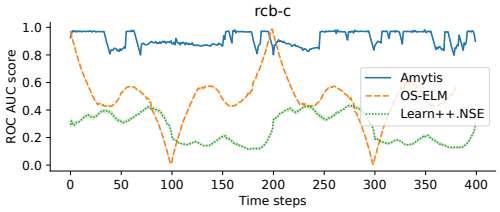
(a) RCB-C · DT · F1 (b) RCB-P · DT · F1

(c) RCB-E · DT · F1 (d) RCB-S · DT · F1

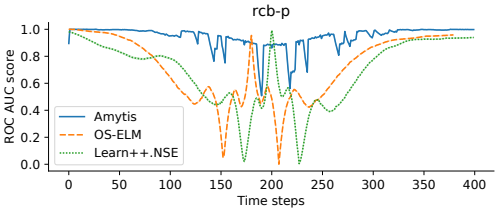
(e) NOAA · DT · F1 (f) ELEC · DT · F1

(g) SEA-1 · DT · F1 (h) SEA-2 · DT · F1 (i) SEA-3 · DT · F1

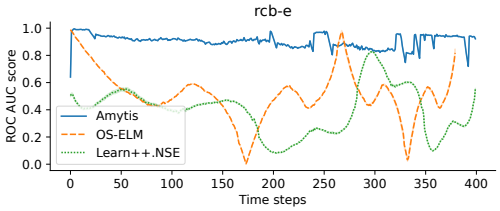
Figure 5.10: F1 scores of OS-ELM [97], Learn^{++}.NSE [32] and Amytis (CDR–ACC) on the datasets. All three techniques used a decision tree as the main learner (CLF · DT). Amytis's CDR component used fluracy as the performance metric.



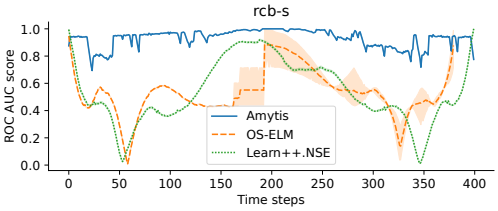
(a) RCB-C · DT · ROC AUC



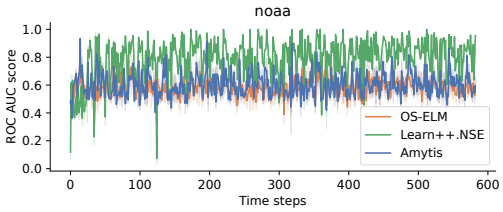
(b) RCB-P · DT · ROC AUC



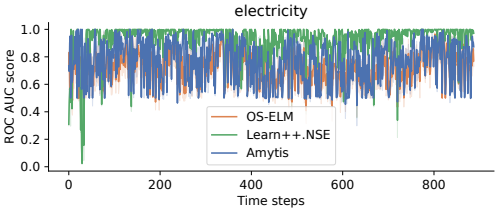
(c) RCB-E · DT · ROC AUC



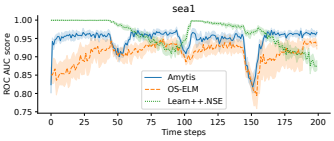
(d) RCB-S · DT · ROC AUC



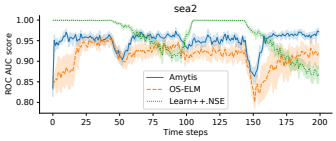
(e) NOAA · DT · ROC AUC



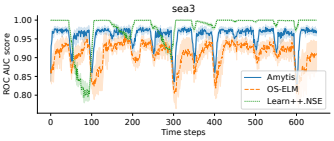
(f) ELEC · DT · ROC AUC



(g) SEA-1 · DT · ROC AUC

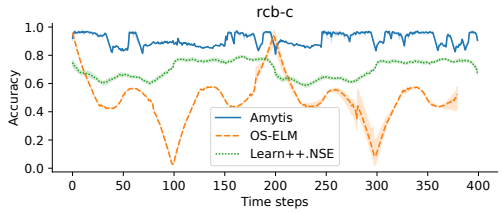


(h) SEA-2 · DT · ROC AUC

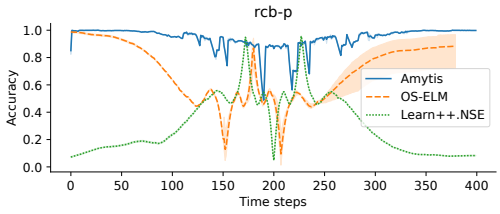


(i) SEA-3 · DT · ROC AUC

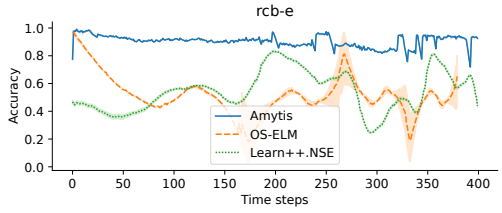
Figure 5.11: ROC AUC scores of OS-ELM [97], Learn^{++}.NSE [32] and Amytis (CDR–ACC) on the datasets. All three techniques used a decision tree as the main learner (CLF · DT). Amytis's CDR component used roc aucuracy as the performance metric.



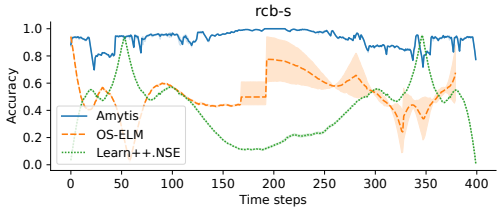
(a) RCB-C · MLP · ACC



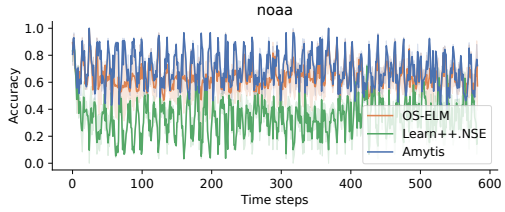
(b) RCB-P · MLP · ACC



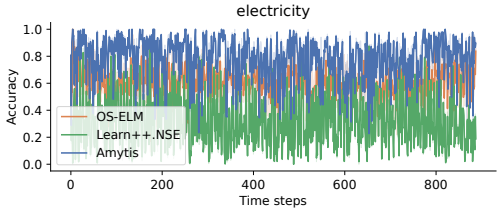
(c) RCB-E · MLP · ACC



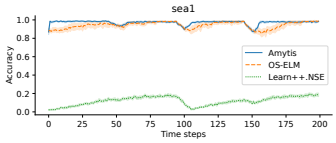
(d) RCB-S · MLP · ACC



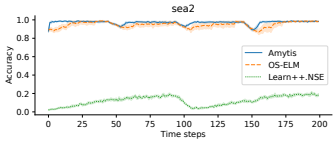
(e) NOAA · MLP · ACC



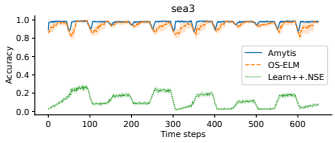
(f) ELEC · MLP · ACC



(g) SEA-1 · MLP · ACC

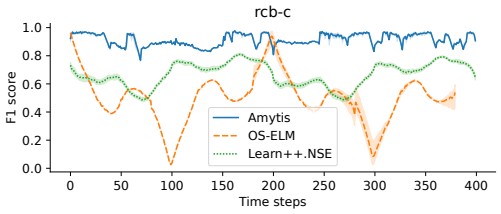


(h) SEA-2 · MLP · ACC

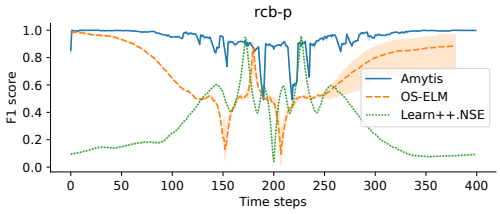


(i) SEA-3 · MLP · ACC

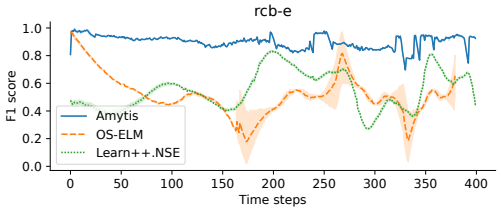
Figure 5.12: Accuracies of OS-ELM [97], Learn^{++}.NSE [32] and Amytis (CDR–ACC) on the datasets. All three techniques used a decision tree as the main learner (CLF · MLP). Amytis's CDR component used accuracy as the performance metric.



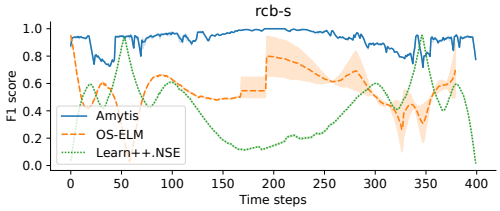
(a) RCB-C · MLP · F1



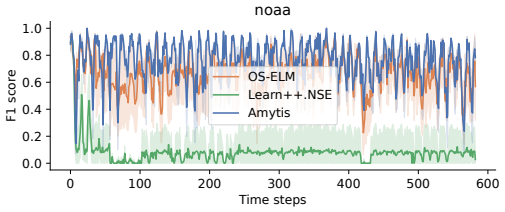
(b) RCB-P · MLP · F1



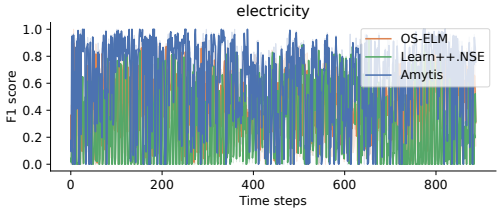
(c) RCB-E · MLP · F1



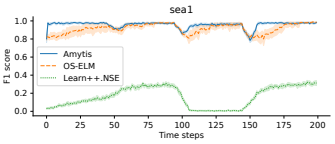
(d) RCB-S · MLP · F1



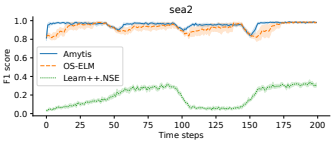
(e) NOAA · MLP · F1



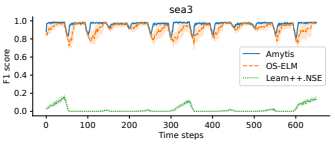
(f) ELEC · MLP · F1



(g) SEA-1 · MLP · F1

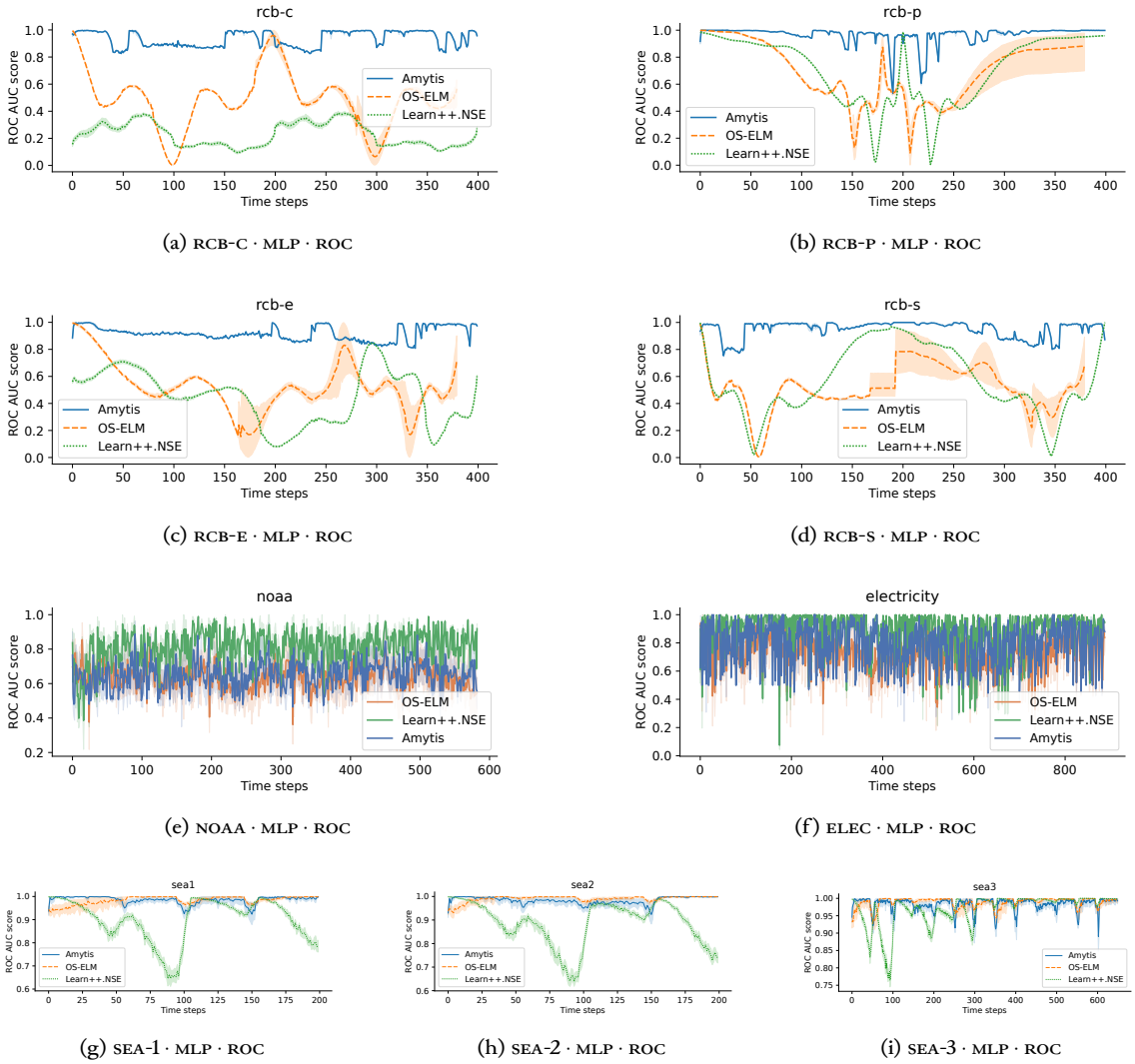


(h) SEA-2 · MLP · F1



(i) SEA-3 · MLP · F1

Figure 5.13: Accuracies of OS-ELM [97], Learn^{++}.NSE [32] and Amytis (CDR–ACC) on the datasets. All three techniques used a decision tree as the main learner (CLF · MLP). Amytis's CDR component used accuracy as the performance metric.



(a) RCB-C · MLP · ROC

(b) RCB-P · MLP · ROC

(c) RCB-E · MLP · ROC

(d) RCB-S · MLP · ROC

(e) NOAA · MLP · ROC

(f) ELEC · MLP · ROC

(g) SEA-1 · MLP · ROC

(h) SEA-2 · MLP · ROC

(i) SEA-3 · MLP · ROC

Figure 5.14: Accuracies of OS-ELM [97], Learn^{++}.NSE [32] and Amytis (CDR–ACC) on the datasets. All three techniques used a decision tree as the main learner (CLF · MLP). Amytis's CDR component used accuracy as the performance metric.

Table 5.3: Mean and standard deviation (in parentheses) of the accuracy (ACC), F1 score (F1), and area under the ROC curve (ROC AUC) of a decision tree as the main learner (CLF · DT–CDD), Ξ–CDA, and the application ($\mathscr{A}$–CDR) on the datasets. For all metrics, the mean and standard deviation over 30 runs are shown with the standard deviation in parentheses. The mean values lie in the interval $[0, 1]$, the higher values are better, and the best values are highlighted in bold. For the application's performance ($\mathscr{A}$–CDR), the best values are highlighted in bold and underlined when the value is greater than or equal to either CLF · DT–CDD or Ξ–CDA.

Dataset	Metric	CLF · DT–CDD	Ξ–CDA	$\mathscr{A}$–CDR
RCB-C	ACC	0.85 (0.0447)	**0.87** (0.0003)	**0.92** (0.0068)
	F1	0.80 (0.0072)	**0.88** (0.0045)	**0.92** (0.0080)
	ROC AUC	0.81 (0.0468)	**0.87** (0.0564)	**0.92** (0.0176)
RCB-P	ACC	**0.94** (0.0060)	0.90 (0.0001)	**0.95** (0.0010)
	F1	**0.93** (0.0209)	0.89 (0.0097)	**0.95** (0.0069)
	ROC AUC	**0.93** (0.0879)	0.90 (0.0360)	**0.95** (0.0499)
RCB-E	ACC	0.74 (0.0446)	**0.88** (0.0002)	**0.91** (0.0042)
	F1	0.69 (0.0093)	**0.87** (0.0189)	**0.90** (0.0066)
	ROC AUC	0.68 (0.0296)	**0.88** (0.0308)	**0.90** (0.0522)
RCB-S	ACC	0.87 (0.0058)	**0.89** (0.0004)	**0.94** (0.0023)
	F1	0.86 (0.0057)	**0.88** (0.0277)	**0.93** (0.0049)
	ROC AUC	0.86 (0.0486)	**0.89** (0.0375)	**0.93** (0.0058)
NOAA	ACC	0.63 (0.0565)	**0.69** (0.0025)	**0.69** (0.0068)
	F1	0.65 (0.1079)	**0.80** (0.0015)	**0.80** (0.0034)
	ROC AUC	0.58 (0.0364)	**0.60** (0.0042)	**0.61** (0.0111)
ELEC	ACC	0.70 (0.0238)	**0.78** (0.0019)	**0.78** (0.0031)
	F1	0.53 (0.0681)	**0.64** (0.0038)	**0.65** (0.0061)
	ROC AUC	0.67 (0.0338)	**0.77** (0.0014)	**0.77** (0.0028)
SEA-1	ACC	0.93 (0.0075)	**0.96** (0.0037)	**0.96** (0.0035)
	F1	0.90 (0.0116)	**0.94** (0.0058)	**0.94** (0.0054)
	ROC AUC	0.92 (0.0067)	**0.95** (0.0037)	**0.95** (0.0033)
SEA-2	ACC	0.93 (0.0072)	**0.96** (0.0028)	**0.96** (0.0027)
	F1	0.90 (0.0107)	**0.94** (0.0038)	**0.94** (0.0038)
	ROC AUC	0.93 (0.0096)	**0.95** (0.0053)	**0.95** (0.0050)
SEA-3	ACC	0.93 (0.0046)	**0.97** (0.0012)	**0.97** (0.0012)
	F1	0.90 (0.0036)	**0.95** (0.0026)	**0.95** (0.0025)
	ROC AUC	0.93 (0.0031)	**0.96** (0.0022)	**0.96** (0.0022)

Table 5.4: Mean and standard deviation (in parentheses) of the accuracy (ACC), F1 score (F1), and area under the ROC curve (ROC AUC) of a multilayer perceptron as the main learner (CLF/MLP–CDD), Ξ–CDA, and the application ($\mathscr{A}$–CDR) on the datasets. For all metrics, the mean and standard deviation over 30 runs are shown with the standard deviation in parentheses. The mean values lie in the interval $[0, 1]$, the higher values are better, and the best values are highlighted in bold. For the application's performance ($\mathscr{A}$–CDR), the best values are highlighted in bold and underlined when the value is greater than or equal to either CLF · DT–CDD or Ξ–CDA.

Dataset	Metric	CLF/MLP–CDD	Ξ–CDA	$\mathscr{A}$–CDR
RCB-C	ACC	0.81 (0.0015)	**0.87** (0.0011)	**0.91** (0.0007)
	F1	0.80 (0.0009)	**0.88** (0.0002)	**0.91** (0.0004)
	ROC AUC	0.84 (0.0021)	**0.87** (0.0004)	**0.94** (0.0005)
RCB-P	ACC	**0.93** (0.0027)	0.90 (0.0037)	**0.94** (0.0007)
	F1	**0.92** (0.0021)	0.89 (0.0003)	**0.94** (0.0005)
	ROC AUC	**0.96** (0.0015)	0.90 (0.0004)	**0.97** (0.0006)
RCB-E	ACC	0.68 (0.0008)	**0.88** (0.0023)	**0.90** (0.0004)
	F1	0.68 (0.0011)	**0.87** (0.0001)	**0.90** (0.0004)
	ROC AUC	0.71 (0.0040)	**0.88** (0.0003)	**0.92** (0.0004)
RCB-S	ACC	0.85 (0.0009)	**0.89** (0.0004)	**0.93** (0.0005)
	F1	0.86 (0.0015)	**0.88** (0.0005)	**0.93** (0.0006)
	ROC AUC	**0.91** (0.0018)	0.89 (0.0003)	**0.95** (0.0005)
NOAA	ACC	0.68 (0.0329)	**0.69** (0.0021)	**0.70** (0.0135)
	F1	0.76 (0.0544)	**0.80** (0.0019)	0.79 (0.0058)
	ROC AUC	**0.63** (0.0916)	0.60 (0.0039)	**0.66** (0.0568)
ELEC	ACC	0.62 (0.0325)	**0.78** (0.0017)	0.77 (0.0030)
	F1	0.36 (0.1189)	**0.64** (0.0036)	0.62 (0.0062)
	ROC AUC	0.77 (0.1598)	0.77 (0.0013)	**0.82** (0.0586)
SEA-1	ACC	0.96 (0.0093)	0.96 (0.0025)	**0.97** (0.0029)
	F1	**0.95** (0.0171)	0.94 (0.0047)	**0.96** (0.0052)
	ROC AUC	**0.99** (0.0038)	0.95 (0.0051)	**1.00** (0.0033)
SEA-2	ACC	0.96 (0.0091)	0.96 (0.0046)	**0.97** (0.0038)
	F1	0.94 (0.0118)	0.94 (0.0056)	**0.96** (0.0053)
	ROC AUC	**0.99** (0.0028)	0.95 (0.0047)	**0.99** (0.0028)
SEA-3	ACC	0.96 (0.0035)	**0.97** (0.0009)	**0.97** (0.0011)
	F1	0.95 (0.0059)	0.95 (0.0018)	**0.96** (0.0015)
	ROC AUC	**0.99** (0.0008)	0.96 (0.0026)	**0.99** (0.0009)

Table 5.5: Mean and standard deviation (in parentheses) of the accuracy (ACC), F1 score (F1), area under the ROC curve (ROC AUC), and run time (in *s*) of OS-ELM [97], Learn^{++}.NSE [32] and Aμγτις(CDR–ACC) on the datasets. All techniques use a decision tree as the main learner. For all metrics, the mean and standard deviation over 30 runs are shown with the standard deviation in parentheses. The mean metric values lie in the interval $[0, 1]$, the higher values are better, and the best values are highlighted in bold. For the run time, the smaller is better.

Dataset	Metric	OS-ELM	Learn^{++}.NSE	Aμγτις
RCB-C	ACC	0.49 (0.0078)	0.68 (0.0059)	**0.92** (0.0068)
	F1	0.50 (0.0036)	0.64 (0.0115)	**0.92** (0.0080)
	ROC AUC	0.49 (0.0079)	0.27 (0.0057)	**0.92** (0.0176)
	run time	34.24 (6.3177)	313.23 (1.9350)	**33.76** (1.5392)
RCB-P	ACC	0.72 (0.0020)	0.32 (0.0007)	**0.95** (0.0010)
	F1	0.72 (0.0036)	0.32 (0.0013)	**0.95** (0.0069)
	ROC AUC	0.72 (0.0019)	0.69 (0.0012)	**0.95** (0.0499)
	run time	40.16 (12.6549)	311.69 (0.4843)	**33.52** (1.6156)
RCB-E	ACC	0.52 (0.0103)	0.58 (0.0053)	**0.91** (0.0042)
	F1	0.52 (0.0105)	0.59 (0.0072)	**0.90** (0.0066)
	ROC AUC	0.52 (0.0104)	0.40 (0.0064)	**0.90** (0.0522)
	run time	44.81 (16.4185)	312.39 (0.4886)	**33.81** (1.6049)
RCB-S	ACC	0.53 (0.0238)	0.43 (0.0029)	**0.94** (0.0023)
	F1	0.56 (0.0164)	0.44 (0.0045)	**0.93** (0.0049)
	ROC AUC	0.53 (0.0239)	0.54 (0.0031)	**0.93** (0.0058)
	run time	**32.11** (1.0334)	311.71 (0.6729)	33.77 (1.6331)
NOAA	ACC	0.61 (0.0244)	0.27 (0.0017)	**0.69** (0.0068)
	F1	0.64 (0.0398)	0.17 (0.0043)	**0.80** (0.0034)
	ROC AUC	0.58 (0.0127)	**0.79** (0.0034)	0.61 (0.0111)
	run time	**32.11** (1.0334)	311.71 (0.6729)	28.76 (1.5131)
ELEC	ACC	0.72 (0.0122)	0.27 (0.0004)	**0.78** (0.0031)
	F1	0.60 (0.0182)	0.29 (0.0018)	**0.65** (0.0061)
	ROC AUC	0.71 (0.0129)	**0.92** (0.0012)	0.77 (0.0028)
	run time	**32.11** (1.0334)	311.71 (0.6729)	46.21 (2.2068)
SEA-1	ACC	0.91 (0.0183)	0.09 (0.0010)	**0.96** (0.0035)
	F1	0.87 (0.0274)	0.12 (0.0031)	**0.94** (0.0054)
	ROC AUC	0.90 (0.0195)	**0.96** (0.0031)	0.95 (0.0033)
	run time	51.51 (23.2786)	27.29 (0.1453)	**17.20** (0.7286)
SEA-2	ACC	0.91 (0.0077)	0.08 (0.0014)	**0.96** (0.0027)
	F1	0.88 (0.0116)	0.12 (0.0026)	**0.94** (0.0038)
	ROC AUC	0.91 (0.0085)	**0.96** (0.0032)	0.95 (0.0050)
	run time	42.89 (12.7864)	27.35 (0.1022)	**17.26** (0.6995)
SEA-3	ACC	0.91 (0.0092)	0.10 (0.0012)	**0.97** (0.0012)
	F1	0.87 (0.0141)	0.01 (0.0004)	**0.95** (0.0025)
	ROC AUC	0.91 (0.0109)	**0.98** (0.0009)	0.96 (0.0022)
	run time	139.55 (39.0372)	151.74 (0.3329)	**37.57** (1.6185)

Table 5.6: Mean and standard deviation (in parentheses) of the accuracy (ACC), F1 score (F1), area under the ROC curve (ROC AUC), and run time (in *s*) of OS-ELM [97], Learn^{++}.NSE [32] and Amytis (CDR–ACC) on the datasets. All techniques use a multilayer perceptron as the main learner. For all metrics, the mean and standard deviation over 30 runs are shown with the standard deviation in parentheses. The mean metric values lie in the interval $[0, 1]$, the higher values are better, and the best values are highlighted in bold. For the run time, the smaller is better.

Dataset	Metric	OS-ELM	Learn^{++}.NSE	Amytis
RCB-C	ACC	0.49 (0.0018)	0.68 (0.0060)	**0.91** (0.0007)
	F1	0.50 (0.0007)	0.64 (0.0106)	**0.91** (0.0004)
	ROC AUC	0.48 (0.0075)	0.27 (0.0083)	**0.94** (0.0005)
	run time	48.17 (1.0212)	331.26 (0.2432)	**33.76** (1.5392)
RCB-P	ACC	0.69 (0.0688)	0.32 (0.0022)	**0.94** (0.0007)
	F1	0.70 (0.0688)	0.32 (0.0021)	**0.94** (0.0005)
	ROC AUC	0.69 (0.0763)	0.69 (0.0008)	**0.97** (0.0006)
	run time	56.16 (1.1571)	332.74 (0.8612)	**33.52** (1.6156)
RCB-E	ACC	0.51 (0.0078)	0.54 (0.0054)	**0.90** (0.0004)
	F1	0.51 (0.0109)	0.55 (0.0082)	**0.90** (0.0004)
	ROC AUC	0.52 (0.0113)	0.45 (0.0044)	**0.92** (0.0004)
	run time	58.18 (1.0589)	330.98 (1.3549)	**33.81** (1.6049)
RCB-S	ACC	0.52 (0.0273)	0.41 (0.0027)	**0.93** (0.0005)
	F1	0.55 (0.0189)	0.42 (0.0023)	**0.93** (0.0006)
	ROC AUC	0.52 (0.0290)	0.57 (0.0012)	**0.95** (0.0005)
	run time	46.40 (0.0947)	336.44 (1.0781)	**33.77** (1.6331)
NOAA	ACC	0.63 (0.0345)	0.34 (0.0947)	**0.70** (0.0135)
	F1	0.66 (0.0547)	0.08 (0.1862)	**0.79** (0.0058)
	ROC AUC	0.63 (0.0357)	**0.81** (0.0686)	0.66 (0.0568)
	run time	39.42 (18.0305)	38.74 (4.0410)	**28.76** (1.5131)
ELEC	ACC	0.60 (0.0153)	0.33 (0.0036)	**0.77** (0.0030)
	F1	0.41 (0.0227)	0.31 (0.0071)	**0.62** (0.0062)
	ROC AUC	0.73 (0.0218)	**0.89** (0.0035)	0.82 (0.0586)
	run time	141.14 (30.6052)	124.52 (3.6285)	**46.21** (2.2068)
SEA-1	ACC	0.94 (0.0107)	0.08 (0.0018)	**0.97** (0.0029)
	F1	0.92 (0.0161)	0.11 (0.0039)	**0.96** (0.0052)
	ROC AUC	0.98 (0.0092)	0.97 (0.0056)	**1.00** (0.0033)
	run time	84.33 (0.0195)	126.41 (0.9534)	**17.20** (0.7286)
SEA-2	ACC	0.95 (0.0120)	0.12 (0.0026)	**0.97** (0.0038)
	F1	0.93 (0.0175)	0.18 (0.0040)	**0.96** (0.0053)
	ROC AUC	**0.99** (0.0034)	0.89 (0.0083)	**0.99** (0.0028)
	run time	84.98 (0.0236)	126.38 (0.8901)	**17.26** (0.6995)
SEA-3	ACC	0.94 (0.0111)	0.09 (0.0014)	**0.97** (0.0011)
	F1	0.92 (0.0158)	0.01 (0.0007)	**0.96** (0.0015)
	ROC AUC	**0.99** (0.0035)	0.98 (0.0010)	**0.99** (0.0009)
	run time	252.91 (0.5583)	162.72 (2.6024)	**37.57** (1.6185)

5.5.4. Comparison of Amytis with Other Techniques

Tables 5.5 and 5.6 show the performance of our proposed Amytis framework compared to the OS-ELM [97] and Learn^{++}.NSE [32] techniques on the datasets using a decision tree and a multilayer perceptron as the main learner, respectively. Figures 5.9 to 5.14 demonstrate detailed performance of the three techniques aggregated over 30 runs throughout the stream.

The results show that Amytis outperforms the other techniques in all RCB datasets for all metrics while achieving a fraction of the run-time, thanks to the light-weight Ψ and Ξ models and their independence from the main learner. In terms of accuracy, Amytis outperformed OS-ELM by +31.94% to +87.76%, and Learn^{++}.NSE by +35.29% to +196.88%. F1 score improvements were as strong, with gains over OS-ELM ranging from +31.94% to +84.00%, and over Learn^{++}.NSE from +43.75% to +196.88%. ROC AUC also saw major improvements, particularly against Learn^{++}.NSE, which is outperformed by Amytis up to +240.74%. Run time improvements were substantial, with up to 89.24% reduction over Learn^{++}.NSE, and more modest reductions compared to OS-ELM, showing Amytis to be superior in efficiency and accuracy for the RCB datasets.

For the real-world datasets NOAA and ELEC, Amytis exhibited moderate to substantial improvements in accuracy and F1 score compared to OS-ELM and Learn^{++}.NSE. For accuracy, Amytis improved over OS-ELM by +8.33% to +13.11%, while the improvements over Learn^{++}.NSE were more significant, reaching +155.56% to +188.89%. F1 score increases were also significant, particularly against Learn^{++}.NSE, with gains of +124.14% to +370.59%. However, there was slight decrease in ROC AUC against Learn^{++}.NSE for both these datasets, with a drop of 22.78% for NOAA and 16.30% for ELEC. We speculate that while Learn^{++}.NSE consistently ranks the predictions correctly and hence a high ROC AUC score, it fails to predict the correct class labels, hence lower accuracy and F1 score. This is likely due to the fact that Learn^{++}.NSE is a CDA technique that relies on the predictions of the main learner, which can be unreliable in the presence of concept drift. On the other hand, Amytis uses the predictions of the main learner only when it outperforms the CDA model, which allows it to maintain a reliable performance throughout the stream. It should be noted that [32] reports higher accuracies on RCB, SEA, and NOAA datasets using a support vector machine (SVM) as the main learner. The paper does not report F1 and ROC AUC scores, which raises the speculation that the SVM model might have been overfitting the data, achieving high accuracy but lacking in generalization ability. In terms of run time, Amytis was generally faster over Learn^{++}.NSE up to 90.77%, however it was slower (43.93%) compared to OS-ELM in the ELEC dataset. This indicates that Amytis can be a desired choice for tasks prioritizing predictive performance, despite a modest increase in run time. In summary, these results suggest that Amytis may benefit further improvements for better handling of imbalanced data in terms of discrimination power.

On abrupt drifts datasets SEA-1, SEA-2, and SEA-3, Amγτis showed remarkable improvement in accuracy and F1 score, particularly compared to Learn^{++}.NSE, with accuracy increase ranging from +5.49% to +1,100.00% and F1 score gains as high as +9,400.00%. Against OS-ELM, the improvements observed were more moderate but still notable, with accuracy and F1 score improvements ranging from +5.49% to +9.20%. Although Amγτis slightly underperformed on ROC AUC against Learn^{++}.NSE, with minor drops in the range 1.04% to 2.04%, its run time was significantly faster up to 75.25% against Learn^{++}.NSE, and 73.06% against OS-ELM. Overall, Amγτis excelled in both predictive performance and efficiency on the SEA datasets. Figures 5.9g to 5.9i, Figures 5.10g to 5.10i, and Figures 5.11g to 5.11i show that Amγτis is faster in response times with less delay in detecting drifts and adapting accordingly compared to OS-ELM and Learn^{++}.NSE. This demonstrates effectiveness of the CDR component in maintaining a reliable performance throughout the stream data processing, particularly in the presence of abrupt drifts.

Furthermore, our results demonstrate consistency of Amγτis's performance across different datasets with both gradual and abrupt drifts, as well as across different main learners. This consistency is attributed to several factors, including independence of the components from the main learner, and the ability of the CDR component to adapt to the changes in the stream by selecting the ''best'' model based on the recent performance of the models. The latter maintains an ''optimal'' level of performance throughout the stream regardless of the type, rate, or duration of the concept drift.

Compared to the OS-ELM and Learn^{++}.NSE techniques, Amγτis maintains low and stable variance in the performance metrics throughout the stream, which is crucial for real-world stream data applications where the performance of the model needs to be reliable, accurate, and consistent.

5.6. Conclusion

In this chapter, we introduced *Amγτis*, a unified and comprehensive concept drift detection and adaptation framework for addressing the corresponding challenges in stream processing applications. By leveraging multivariate vector error-correction analysis of feature importance measures (FIMs) used as a basis in its design and development, Amγτis unifies concept drift detection, adaptation, and resolution into a single, multi-level framework. It provides a novel solution to the concept drift resolution (CDR) problem, which has been overlooked in the literature, by addressing the need for maintaining consistent performance across multiple models in a non-stationary environment.

Amytis effectively manages the performance of the main learner alongside CDD and CDA components, ensuring that the overall framework remains robust even under varying conditions and types of concept drift. Through the CDR component, Amytis can dynamically switch between the models based on their recent performance, thereby reducing variance and improving reliability and accuracy. This adaptability is critical in some existing and emerging real-world applications where the rate, type, and nature of concept drift can vary significantly.

The results of our numerous experiments using real and synthetic benchmark datasets demonstrate that Amytis outperforms existing techniques, particularly in scenarios involving gradual and abrupt drifts, and maintains a consistent level of performance across different datasets and main learners. The ability of Amytis to maintain low and stable variance in performance metrics is especially important for real-world applications, where consistent and reliable performance is crucial.

Amytis demonstrated substantial improvements across most datasets compared to OS-ELM and Learn^{++}.NSE, particularly in accuracy and F1 scores. The accuracy gains over Learn^{++}.NSE ranged from 35.29% to 196.88%, while improvements over OS-ELM were more moderate, ranging from 5.49% to 87.76%. Improvement in F1 score followed a similar pattern, notably over Learn^{++}.NSE and OS-ELM, ranging from 8.33% to 84.00%. Amytis also generally performed better in terms of run time up to 90.77%, especially when compared to Learn^{++}.NSE. However, compared to Learn^{++}.NSE, we observed minor declines in ROC AUC on a few datasets. Overall, Amytis provides a balance between predictive accuracy and computational efficiency.

Overall, the Amytis framework demonstrates an effective solution framework to maintain reliable and consistent performance across various datasets and types of concept drift. By effectively balancing the contributions of the main learner and the CDA component through the CDR mechanism, Amytis addresses the major issue of performance variance that often challenged real-world applications. Compared to other solution techniques that were effective in either CDD or CDA tasks, Amytis achieves a lower and more stable variance in performance metrics of *both* CDD and CDA tasks, which is essential for applications that require dependable, long-term operation under non-stationary conditions. This consistency underscores the framework's potential for broader applicability in environments where concept drift detection, adaptation, and resolution are much needed.

We are currently revising a journal article based on our results in this chapter [5].

With them the seed of Wisdom did I sow,
And with mine own hand wrought to make it grow;
And this was all the Harvest that I reap'd—
"I came like Water, and like Wind I go"

—Omar Khayyám, *The Rubaiyat (1120)*

Chapter 6

Conclusion

In this research, we focused on concept drift (CD) in streaming data and studied methods to overcome the challenges involved. The goal is to ensure accuracy and performance of application over long periods of time using machine learning techniques. Concept drift, if undetected or poorly managed, can severely affect the effectiveness of predictive models, leading to incorrect inferences and decisions. This issue is particularly prevalent in real-world applications such as insurance, healthcare, industrial monitoring, environmental sensing, smart cities, and finance, where data streams are often non-stationary and exhibit varying patterns over time.

6.1. Summary of Contributions

This research addressed several key challenges associated with concept drift detection, adaptation, and resolution in a single unified framework. The primary contributions of this work are as follows:

- **Novel approach to study concept drift detection and adaptation (CDD&A):** We proposed a novel approach based on multivariate vector error-correction analysis of feature importance measures (FIMs). This approach provides a solid basis to analyze streaming data for detecting and adapting to concept drifts by capturing the dynamic relationships between raw data features and target variables.
- **Unified framework—Amytis:** We developed a unified framework named Amytis, using which concept drift analysis, detection, adaptation, and resolution are all studied

and viewed as different facets of the same core concept, namely feature importance measures (FIMs). To the best of our knowledge, this is the first framework in the literature to leverage FIMs analysis in such a comprehensive manner.

- **Robust and explainable concept drift adaptation model:** A new model for concept drift adaptation was proposed, emphasizing effectiveness, efficiency, and explainability. This model is designed to adapt to drifts in streaming data while maintaining explainability, making it suitable for applications requiring interpretability, such as healthcare and finance.
- **Innovative techniques for concept drift detection:** We developed two novel techniques for concept drift detection, one based on ensemble methods and the other based on FIMs analysis. These techniques are effective in identifying drifts in streaming data and are designed to preserve interpretability, making them suitable for real-world applications.
- **Novel concept drift resolution technique:** We addressed the concept drift resolution (CDR) problem and proposed a CDR technique that combines the recent performance of the main learner, improved by the proposed CDD technique, and CDA methods. This technique ensures that the application remains robust and adaptable even in the face of significant concept drifts.
- **Comprehensive development and evaluation:** Extensive experiments were conducted to evaluate the proposed methodologies and models on various real-world and synthetic datasets that exhibit concept drifts with different characteristics and magnitudes of change over time such as gradual, abrupt, non-monotonic, and periodic drifts. The results demonstrate the effectiveness and efficiency of the proposed solutions in handling concept drifts in streaming data.
- **Open-Source software framework:** A flexible and extensible software framework was developed to support concept drift analysis and data stream processing. This framework provides a foundation for future research and development in this area, enabling researchers and practitioners to build and deploy concept drift analysis and evaluation systems.

6.2. Future Directions

The findings of this research have significant implications and applications for various disciplines in which continuous data streams are processed and analyzed. By providing a unified framework

for CDD&A, this work bridges the gap between concept drift detection and adaptation, offering a holistic solution that balances performance with interpretability.

A future direction for this work would be leveraging the vector autoregression and vector error-correction models as byproducts of the proposed methodology to analyze and forecast the relationships between features and target variables in streaming data. This could lead to more accurate and reliable predictions, especially in applications where the underlying data distribution is non-stationary.

We mainly focused on parametric statistical tests and ensemble methods for concept drift detection in this research. Future work could explore the use of non-parametric methods for analytics. Additionally, further research could consider cointegration analysis tests that are more robust to structural changes in the data distribution.

Future improvement would be refining the proposed methods and exploring their applicability in more diverse real-world scenarios. Additionally, extending the software framework to include more advanced features and integrations with popular data processing tools like Apache Kafka and Apache Flink could further enhance its utility.

Finally, ongoing research could explore the possibility of integrating more sophisticated machine learning models and algorithms into the Amȝtis framework, potentially improving its accuracy and adaptability even further.

In conclusion, this thesis contributes to the growing body of knowledge and technology for concept drift analysis by offering novel methodologies, models, and tools that address both theoretical and practical challenges. The proposed solutions pave the way for more reliable, efficient, and explainable data stream processing systems, ultimately benefiting a wide range of applications that rely on streaming data analysis.

References

[1] Aisha Abdallah, Mohd Aizaini Maarof, and Anazida Zainal. "Fraud Detection System: A Survey". In: *Journal of Network and Computer Applications* 68 (June 1, 2016), pp. 90–113. ISSN: 1084-8045. DOI: 10/ggpkf7. URL: http://www.sciencedirect.com/science/article/pii/S1084804516300571 (cit. on pp. 1, 26).

[2] Jan Niklas Adams et al. "Explainable Concept Drift in Process Mining". In: *Information Systems* 114 (Mar. 1, 2023), p. 102177. ISSN: 0306-4379. DOI: 10/gtnj67. URL: https://www.sciencedirect.com/science/article/pii/S0306437923000133 (cit. on p. 1).

[3] Ali Alizadeh Mansouri, Abbas Javadtalab, and Nematollaah Shiri. "An Ensemble Learning Augmentation Method for Concept Drift Detection over Data Streams". In: *Advances in Data Science and Information Engineering*. The 18th Int. Conference on Data Science (ICDATA'22). Las Vegas: Springer, 2022 (cit. on pp. 37, 51).

[4] Ali Alizadeh Mansouri, Abbas Javadtalab, and Nematollaah Shiri. "Streaming Data Analytics for Feature Importance Measures in Concept Drift Detection and Adaptation". In: *Database and Expert Systems Applications*. Ed. by Christine Strauss et al. Lecture Notes in Computer Science. Cham: Springer Nature Switzerland, 2023, pp. 114–128. ISBN: 978-3-031-39847-6. DOI: 10/gs6fsb (cit. on p. 73).

[5] Ali Alizadeh Mansouri and Nematollaah Shiri. "Amytis: A Unified Framework for Concept Drift Detection, Adaptation, and Resolution". Manuscript under revision. 2024 (cit. on p. 120).

[6] *Apache Flink*. Apache Software Foundation. URL: https://flink.apache.org/ (cit. on p. 8).

[7] *Apache Kafka*. Apache Software Foundation. URL: https://kafka.apache.org/ (cit. on p. 8).

[8] *Apache Spark*. Apache Software Foundation. URL: https://spark.apache.org/ (cit. on p. 8).

[9] Shruti Arora, Rinkle Rani, and Nitin Saxena. "A Systematic Review on Detection and Adaptation of Concept Drift in Streaming Data Using Machine Learning Techniques". In: *WIREs Data Mining and Knowledge Discovery* 14.4 (2024), e1536. ISSN: 1942-4795. DOI: 10.1002/widm.1536. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/widm.1536 (cit. on pp. 1, 11).

[10] *AWS HPC*. Amazon Web Services. URL: https://aws.amazon.com/hpc/ (cit. on p. 8).

[11] *Azure IoT Edge*. Microsoft. URL: https://github.com/Azure/iotedge (cit. on p. 8).

[12] Jean Paul Barddal et al. "Boosting Decision Stumps for Dynamic Feature Selection on Data Streams". In: *Information Systems* 83 (July 1, 2019), pp. 13–29. ISSN: 0306-4379 (cit. on p. 53).

[13] Albert Bifet and Ricard Gavaldà. "Learning from Time-Changing Data with Adaptive Windowing". In: *Proceedings of the 2007 SIAM International Conference on Data Mining (SDM)*. Proceedings. Society for Industrial and Applied Mathematics, Apr. 26, 2007, pp. 443–448. ISBN: 978-0-89871-630-6. DOI: 10.1137/1.9781611972771.42. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611972771.42 (cit. on p. 25).

[14] Albert Bifet et al. "MOA: Massive Online Analysis". In: *Journal of Machine Learning Research* 11.52 (2010), pp. 1601–1604. ISSN: 1533-7928. URL: http://jmlr.org/papers/v11/bifet10a.html (cit. on p. 43).

[15] R. P. Jagadeesh Chandra Bose et al. "Dealing With Concept Drifts in Process Mining". In: *IEEE Transactions on Neural Networks and Learning Systems* 25.1 (Jan. 2014), pp. 154–171. ISSN: 2162-2388. DOI: 10/f5ngxj. URL: https://ieeexplore.ieee.org/abstract/document/6634264 (cit. on p. 1).

[16] J.P.G. van Brakel. *Robust Peak Detection Algorithm Using Z-Scores*. Stack Overflow. 2014. URL: https://stackoverflow.com/questions/22583391/peak-signal-detection-in-realtime-timeseries-data/22640362#22640362 (cit. on p. 41).

[17] Leo Breiman. "Manual on Setting up, Using, and Understanding Random Forests v3. 1". In: *Statistics Dept., University of California, Berkeley, CA, USA* 1.58 (2002), pp. 3–42 (cit. on pp. 53, 56, 79).

[18] Leo Breiman. "Random Forests". In: *Machine Learning* 45.1 (Oct. 1, 2001), pp. 5–32. ISSN: 1573-0565 (cit. on pp. 53, 56, 79).

[19] Dariusz Brzezinski and Jerzy Stefanowski. "Prequential AUC: Properties of the Area under the ROC Curve for Data Streams with Concept Drift". In: *Knowledge and Information Systems* 52.2 (Aug. 1, 2017), pp. 531–562. ISSN: 0219-3116. DOI: 10.1007/s10115-017-1022-8. URL: https://doi.org/10.1007/s10115-017-1022-8 (cit. on p. 74).

[20] Andrew Phelps Cassidy and Frank A. Deviney. "Calculating Feature Importance in Data Streams with Concept Drift Using Online Random Forest". In: *2014 IEEE International Conference on Big Data (Big Data)*. 2014 IEEE International Conference on Big Data (Big Data). Oct. 2014, pp. 23–28 (cit. on p. 53).

[21] Paola Alexandra Castro-Cabrera et al. "Supervised and Unsupervised Identification of Concept Drifts in Data Streams of Seismic-Volcanic Signals". In: *Advances in Artificial Intelligence - IBERAMIA 2018*. Ed. by Guillermo R. Simari et al. Lecture Notes in Computer Science. Springer International Publishing, 2018, pp. 193–205. ISBN: 978-3-030-03928-8 (cit. on p. 1).

[22] Peter Chapman et al. *CRISP-DM 1.0: Step-by-Step Data Mining Guide*. 2000. URL: https://www.semanticscholar.org/paper/CRISP-DM-1.0%3A-Step-by-step-data-mining-guide-Chapman-Clinton/54bad20bbc7938991bf34f86dde0babfbd2d5a72 (cit. on pp. 6, 7).

[23] N. V. Chawla et al. "SMOTE: Synthetic Minority Over-sampling Technique". In: *Journal of Artificial Intelligence Research* 16 (June 1, 2002), pp. 321–357. ISSN: 1076-9757. DOI: 10.1613/jair.953. URL: https://www.jair.org/index.php/jair/article/view/10302 (cit. on p. 21).

[24] Robert B Cleveland et al. "STL: A Seasonal-Trend Decomposition Procedure Based on Loess". In: *Journal of Official Statistics* 6.1 (1990), pp. 3–73 (cit. on p. 84).

[25] Andrea Dal Pozzolo et al. "Credit Card Fraud Detection: A Realistic Modeling and a Novel Learning Strategy". In: *IEEE Transactions on Neural Networks and Learning Systems* 29.8 (Aug. 2018), pp. 3784–3797. ISSN: 2162-2388. DOI: 10/gdxz24 (cit. on pp. 21, 22).

[26] Babak Mahdavi Damghani et al. "The Misleading Value of Measured Correlation". In: *Wilmott* 2012.62 (2012), pp. 64–73. ISSN: 1541-8286 (cit. on p. 58).

[27] David A. Dickey and Wayne A. Fuller. "Distribution of the Estimators for Autoregressive Time Series with a Unit Root". In: *Journal of the American Statistical Association* (June 1, 1979). ISSN: 0162-1459. URL: https://www.tandfonline.com/doi/abs/10.1080/01621459.1979.10482531 (cit. on p. 82).

[28] G. Ditzler and R. Polikar. "Hellinger Distance Based Drift Detection for Nonstationary Environments". In: *2011 IEEE Symposium on Computational Intelligence in Dynamic and Uncertain Environments (CIDUE)*. 2011 IEEE Symposium on Computational Intelligence in Dynamic and Uncertain Environments (CIDUE). Apr. 2011, pp. 41–48. DOI: 10/c3cg5t (cit. on pp. 1, 27).

[29] G. Ditzler and R. Polikar. “Semi-Supervised Learning in Nonstationary Environments”. In: *The 2011 International Joint Conference on Neural Networks*. The 2011 International Joint Conference on Neural Networks. July 2011, pp. 2741–2748. DOI: 10/dzfcjg (cit. on pp. 24, 32).

[30] Gregory Ditzler and Robi Polikar. “Incremental Learning of Concept Drift from Streaming Imbalanced Data”. In: *IEEE Transactions on Knowledge and Data Engineering* 25.10 (Oct. 2013), pp. 2283–2301. ISSN: 1558-2191. DOI: 10/f4745g (cit. on pp. 21, 22).

[31] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2nd Edition)*. USA: Wiley-Interscience, 2000. ISBN: 978-0-471-05669-0 (cit. on pp. 11–13).

[32] Ryan Elwell and Robi Polikar. “Incremental Learning of Concept Drift in Nonstationary Environments”. In: *IEEE Transactions on Neural Networks* 22.10 (Oct. 2011), pp. 1517–1531. ISSN: 1045-9227, 1941-0093. DOI: 10/bgxfrz (cit. on pp. 20, 21, 32, 42, 60, 98, 99, 108–113, 116–118).

[33] Robert F. Engle and C. W. J. Granger. “Co-Integration and Error Correction: Representation, Estimation, and Testing”. In: *Econometrica* 55.2 (1987), pp. 251–276. ISSN: 0012-9682. JSTOR: 1913236 (cit. on p. 58).

[34] Jerome H. Friedman. “Greedy Function Approximation: A Gradient Boosting Machine”. In: *The Annals of Statistics* 29.5 (2001), pp. 1189–1232. ISSN: 0090-5364. JSTOR: 2699986 (cit. on pp. 53, 56, 78).

[35] João Gama et al. “A Survey on Concept Drift Adaptation”. In: *ACM Computing Surveys (CSUR)* 46.4 (Jan. 4, 2014), p. 44. ISSN: 0360-0300. DOI: 10.1145/2523813. URL: http://dl.acm.org/citation.cfm?id=2597757.2523813 (cit. on pp. 1, 11, 14, 16, 17, 36, 55).

[36] João Gama et al. “Learning with Drift Detection”. In: *Advances in Artificial Intelligence – SBIA 2004*. Ed. by Ana L. C. Bazzan and Sofiane Labidi. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2004, pp. 286–295. ISBN: 978-3-540-28645-5. DOI: 10/ckzcm4 (cit. on pp. 43, 60, 99).

[37] Heitor Murilo Gomes et al. “Feature Scoring Using Tree-Based Ensembles for Evolving Data Streams”. In: *2019 IEEE International Conference on Big Data (Big Data)*. 2019 IEEE International Conference on Big Data (Big Data). Dec. 2019, pp. 761–769 (cit. on p. 53).

[38] C. W. J. Granger. “Some Properties of Time Series Data and Their Use in Econometric Model Specification”. In: *Journal of Econometrics* 16.1 (May 1, 1981), pp. 121–130. ISSN: 0304-4076 (cit. on p. 58).

[39] C. W. J. Granger and P. Newbold. “Spurious Regressions in Econometrics”. In: *Journal of Econometrics* 2.2 (July 1, 1974), pp. 111–120. ISSN: 0304-4076 (cit. on p. 57).

[40] William H. Greene. *Econometric Analysis*. Prentice Hall, 2003. 1026 pp. ISBN: 978-0-13-110849-3. Google Books: 7i9lQgAACAAJ. URL: https://books.google.ca/books?id=7i9lQgAACAAJ (cit. on p. 85).

[41] Husheng Guo et al. "Concept Drift Detection and Accelerated Convergence of Online Learning". In: *Knowledge and Information Systems* 65.3 (Mar. 1, 2023), pp. 1005–1043. ISSN: 0219-3116. DOI: 10/gtnj7c. URL: https://doi.org/10.1007/s10115-022-01790-6 (cit. on p. 99).

[42] David J. Hand and Niall M. Adams. "Selection Bias in Credit Scorecard Evaluation". In: *Journal of the Operational Research Society* 65.3 (Mar. 1, 2014), pp. 408–415. ISSN: 0160-5682. DOI: 10/f5tmzw. URL: https://doi.org/10.1057/jors.2013.55 (cit. on pp. 1, 26).

[43] Ahsanul Haque, Latifur Khan, and Michael Baron. "Semi Supervised Adaptive Framework for Classifying Evolving Data Stream". In: *Advances in Knowledge Discovery and Data Mining*. Ed. by Tru Cao et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2015, pp. 383–394. ISBN: 978-3-319-18032-8. DOI: 10/gmvzrd (cit. on pp. 23–25, 28, 30, 32).

[44] Michael Harries and New South Wales. *SPLICE-2 Comparative Evaluation: Electricity Pricing*. 1999 (cit. on pp. 43, 60, 99).

[45] Charles R. Harris et al. "Array Programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. ISSN: 1476-4687. DOI: 10/ghbzf2. URL: https://www.nature.com/articles/s41586-020-2649-2 (cit. on p. 100).

[46] Zonglu He and Koichi Maekawa. "On Spurious Granger Causality". In: *Economics Letters* 73.3 (Dec. 1, 2001), pp. 307–313. ISSN: 0165-1765 (cit. on p. 57).

[47] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. "Extreme Learning Machine: Theory and Applications". In: *Neurocomputing*. Neural Networks 70.1 (Dec. 1, 2006), pp. 489–501. ISSN: 0925-2312. DOI: 10/dq3m9m. URL: http://www.sciencedirect.com/science/article/pii/S0925231206000385 (cit. on pp. 29, 36).

[48] Gareth James et al. *An Introduction to Statistical Learning: With Applications in R*. Springer Texts in Statistics. New York: Springer-Verlag, 2013. ISBN: 978-1-4614-7137-0. URL: https://www.springer.com/gp/book/9781461471370 (cit. on pp. 11, 13, 14, 17).

[49] Søren Johansen. "Estimation and Hypothesis Testing of Cointegration Vectors in Gaussian Vector Autoregressive Models". In: *Econometrica* 59.6 (1991), pp. 1551–1580. ISSN: 0012-9682. JSTOR: 2938278 (cit. on pp. 58, 59).

[50] Imen Khamassi et al. “Discussion and Review on Evolving Data Streams and Concept Drift Adapting”. In: *Evolving Systems* 9.1 (Mar. 1, 2018), pp. 1–23. ISSN: 1868-6486. DOI: 10.1007/s12530-016-9168-2. URL: https://doi.org/10.1007/s12530-016-9168-2 (cit. on pp. 14, 16, 17, 55).

[51] Martin Khannouz and Tristan Glatard. “Mondrian Forest for Data Stream Classification under Memory Constraints”. In: *Data Mining and Knowledge Discovery* 38.2 (Mar. 2024), pp. 569–596. ISSN: 1384-5810, 1573-756X. DOI: 10/gtnj7d. URL: https://link.springer.com/10.1007/s10618-023-00970-4 (cit. on p. 2).

[52] T. Al-Khateeb et al. “Stream Classification with Recurring and Novel Class Detection Using Class-Based Ensemble”. In: *2012 IEEE 12th International Conference on Data Mining*. 2012 IEEE 12th International Conference on Data Mining. Dec. 2012, pp. 31–40. DOI: 10/gf6fck (cit. on p. 32).

[53] Łukasz Korycki and Bartosz Krawczyk. “Concept Drift Detection from Multi-Class Imbalanced Data Streams”. In: *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. Apr. 2021, pp. 1068–1079. DOI: 10/gm859d (cit. on pp. 22, 25, 29).

[54] Bartosz Krawczyk and Michał Woźniak. “One-Class Classifiers with Incremental Learning and Forgetting for Data Streams with Concept Drift”. In: *Soft Computing* 19.12 (Dec. 1, 2015), pp. 3387–3400. ISSN: 1433-7479. DOI: 10/f7z3x2. URL: https://doi.org/10.1007/s00500-014-1492-5 (cit. on p. 31).

[55] Ludmila I. Kuncheva. *Combining Pattern Classifiers: Methods and Algorithms*. 2nd ed. John Wiley & Sons, Sept. 9, 2014. 384 pp. ISBN: 978-1-118-31523-1. Google Books: MZgtBAAAQBAJ. URL: https://onlinelibrary.wiley.com/doi/book/10.1002/9781118914564 (cit. on pp. 42, 60, 99).

[56] Mark J. van der Laan, Eric C. Polley, and Alan E. Hubbard. “Super Learner”. In: *Statistical Applications in Genetics and Molecular Biology* 6.1 (Sept. 16, 2007). ISSN: 1544-6115. DOI: 10/cbgkdr. URL: https://www.degruyter.com/document/doi/10.2202/1544-6115.1309/html (cit. on p. 80).

[57] Ang Li et al. “Online Active Learning Method for Multi-Class Imbalanced Data Stream”. In: *Knowledge and Information Systems* 66.4 (Apr. 1, 2024), pp. 2355–2391. ISSN: 0219-3116. DOI: 10.1007/s10115-023-02027-w. URL: https://doi.org/10.1007/s10115-023-02027-w (cit. on pp. 25, 28).

[58] Jinpeng Li et al. “Concept Drift Adaptation by Exploiting Drift Type”. In: *ACM Trans. Knowl. Discov. Data* 18.4 (Feb. 12, 2024), 96:1–96:22. ISSN: 1556-4681. DOI: 10/nd8q. URL: https://doi.org/10.1145/3638777 (cit. on pp. 29, 31, 32).

[59] Zeng Li et al. “Incremental Learning Imbalanced Data Streams with Concept Drift: The Dynamic Updated Ensemble Algorithm”. In: *Knowledge-Based Systems* 195 (May 2020), p. 105694. ISSN: 0950-7051. DOI: 10/gm859c (cit. on p. 22).

[60] Nan-ying Liang et al. “A Fast and Accurate Online Sequential Learning Algorithm for Feedforward Networks”. In: *IEEE Transactions on Neural Networks* 17.6 (Nov. 2006), pp. 1411–1423. ISSN: 1941-0093. DOI: 10/bdjqwh (cit. on pp. 2, 28, 36).

[61] Jie Lu et al. “Learning under Concept Drift: A Review”. In: *IEEE Transactions on Knowledge and Data Engineering* 31.12 (Dec. 2019), pp. 2346–2363. ISSN: 1558-2191. DOI: 10/gjgc2d (cit. on pp. 1, 11, 14, 16, 55).

[62] Yang Lu, Yiu-ming Cheung, and Yuan Yan Tang. “Dynamic Weighted Majority for Incremental Learning of Imbalanced Data Streams with Concept Drift.” In: *IJCAI*. 2017, pp. 2393–2399 (cit. on p. 22).

[63] Abdul Razak M. S. et al. “A Survey on Detecting Healthcare Concept Drift in AI/ML Models from a Finance Perspective”. In: *Frontiers in Artificial Intelligence* 5 (Apr. 17, 2023). ISSN: 2624-8212. DOI: 10/gtnj6t. URL: https://www.frontiersin.org/articles/10.3389/frai.2022.955314 (cit. on p. 1).

[64] Daiki Maki. “Tests for Cointegration Allowing for an Unknown Number of Breaks”. In: *Economic Modelling* 29.5 (Sept. 1, 2012), pp. 2011–2015. ISSN: 0264-9993. DOI: 10/f38xd4. URL: https://www.sciencedirect.com/science/article/pii/S0264999312001162 (cit. on p. 74).

[65] M. Masud et al. “Classification and Novel Class Detection in Concept-Drifting Data Streams under Time Constraints”. In: *IEEE Transactions on Knowledge and Data Engineering* 23.6 (June 2011), pp. 859–874. ISSN: 1041-4347. DOI: 10/bg6qw7 (cit. on pp. 24, 25, 30, 32).

[66] Mohammad M. Masud et al. “Facing the Reality of Data Stream Classification: Coping with Scarcity of Labeled Data”. In: *Knowledge and Information Systems* 33.1 (Oct. 1, 2012), pp. 213–244. ISSN: 0219-3116. DOI: 10/fxf6cw. URL: https://doi.org/10.1007/s10115-011-0447-8 (cit. on pp. 16, 23, 24, 32).

[67] Mariusz Maziarz. “A Review of the Granger-causality Fallacy”. In: *The Journal of Philosophical Economics : Reflections on Economic and Social Issues* VIII.2 (May 20, 2015), pp. 86–105. ISSN: 1843-2298, 1844-8208 (cit. on p. 57).

[68] Andrew V. Metcalfe and Paul S.P. Cowpertwait. *Introductory Time Series with R*. New York, NY: Springer, 2009. ISBN: 978-0-387-88697-8 978-0-387-88698-5. DOI: 10.1007/978-0-387-88698-5. URL: https://link.springer.com/10.1007/978-0-387-88698-5 (cit. on p. 19).

[69] Michalis P. Michaelides et al. “Contaminant Event Monitoring in Intelligent Buildings Using a Multi-Zone Formulation”. In: *IFAC Proceedings Volumes*. 8th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes 45.20 (Jan. 1, 2012), pp. 492–497. ISSN: 1474-6670. DOI: 10/gf6n96. URL: http://www.sciencedirect.com/science/article/pii/S1474667016348029 (cit. on p. 1).

[70] Ivan Adriyanov Nikolov et al. “Seasons in Drift: A Long-Term Thermal Imaging Dataset for Studying Concept Drift”. In: *Thirty-Fifth Conference on Neural Information Processing Systems*. 2021 (cit. on p. 1).

[71] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on pp. 78, 100).

[72] Peter C. B. Phillips and Pierre Perron. “Testing for a Unit Root in Time Series Regression”. In: *Biometrika* 75.2 (1988), pp. 335–346. ISSN: 0006-3444. DOI: 10/bgcrpz. JSTOR: 2336182. URL: https://www.jstor.org/stable/2336182 (cit. on p. 73).

[73] R. Polikar et al. “Learn++: An Incremental Learning Algorithm for Supervised Neural Networks”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 31.4 (Nov. 2001), pp. 497–508. ISSN: 1094-6977, 1558-2442. DOI: 10/b68tgr (cit. on pp. 20, 32).

[74] Victor Ulisses Pugliese, Renato Duarte Costa, and Celso Massaki Hirata. “Comparative Evaluation of the Supervised Machine Learning Classification Methods and the Concept Drift Detection Methods in the Financial Business Problems”. In: *Enterprise Information Systems*. Ed. by Joaquim Filipe et al. Cham: Springer International Publishing, 2021, pp. 268–292. ISBN: 978-3-030-75418-1. DOI: 10/gtnj69 (cit. on p. 1).

[75] *Redis*. Redis Labs. URL: https://redis.io/ (cit. on p. 8).

[76] Thomas Reinartz. *Focusing Solutions for Data Mining: Analytical Studies and Experimental Results in Real-World Domains*. Lecture Notes in Artificial Intelligence. Berlin Heidelberg: Springer-Verlag, 1999. ISBN: 978-3-540-66429-1. DOI: 10.1007/3-540-48316-0. URL: https://www.springer.com/gp/book/9783540664291 (cit. on pp. 6, 7).

[77] Peter J. Rousseeuw. “Silhouettes: A Graphical Aid to the Interpretation and Validation of Cluster Analysis”. In: *Journal of Computational and Applied Mathematics* 20 (Nov. 1, 1987), pp. 53–65. ISSN: 0377-0427. DOI: 10/fdxwqh. URL: http://www.sciencedirect.com/science/article/pii/0377042787901257 (cit. on p. 34).

[78] Bernhard Schölkopf et al. “Estimating the Support of a High-Dimensional Distribution”. In: *Neural Computation* 13.7 (July 1, 2001), pp. 1443–1471. ISSN: 0899-7667. DOI: 10/bxmnv3. URL: https://doi.org/10.1162/089976601750264965 (cit. on pp. 20, 30).

[79] G. William Schwert. “Tests for Unit Roots: A Monte Carlo Investigation”. In: *Journal of Business & Economic Statistics* 7.2 (1989), pp. 147–159. ISSN: 0735-0015. DOI: 10/cz2hgg. JSTOR: 1391432. URL: https://www.jstor.org/stable/1391432 (cit. on p. 85).

[80] Skipper Seabold and Josef Perktold. “statsmodels: Econometric and statistical modeling with python”. In: *9th Python in Science Conference*. 2010 (cit. on pp. 84, 100).

[81] Tegjyot Singh Sethi and Mehmed Kantardzic. “On the Reliable Detection of Concept Drift from Streaming Unlabeled Data”. In: *Expert Systems with Applications* 82 (Oct. 2017), pp. 77–99. ISSN: 0957-4174. DOI: 10/gf5mhf (cit. on pp. 2, 32).

[82] Lavanya Settipalli, G. R. Gangadharan, and Ugo Fiore. “Predictive and Adaptive Drift Analysis on Decomposed Healthcare Claims Using ART Based Topological Clustering”. In: *Information Processing & Management* 59.3 (May 1, 2022), p. 102887. ISSN: 0306-4573. DOI: 10/gtnj6n. URL: https://www.sciencedirect.com/science/article/pii/S0306457322000164 (cit. on p. 1).

[83] Christopher A. Sims, James H. Stock, and Mark W. Watson. “Inference in Linear Time Series Models with Some Unit Roots”. In: *Econometrica* 58.1 (1990), pp. 113–144. ISSN: 0012-9682. JSTOR: 2938337 (cit. on p. 58).

[84] S.J. Stolfo et al. “Cost-Based Modeling for Fraud and Intrusion Detection: Results from the JAM Project”. In: *Proceedings DARPA Information Survivability Conference and Exposition. DISCEX’00*. Proceedings DARPA Information Survivability Conference and Exposition. DISCEX’00. Vol. 2. Jan. 2000, 130–144 vol.2. DOI: 10/cpj827 (cit. on pp. 1, 26).

[85] W. Nick Street and YongSeog Kim. “A Streaming Ensemble Algorithm (SEA) for Large-Scale Classification”. In: *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’01. New York, NY, USA: Association for Computing Machinery, Aug. 26, 2001, pp. 377–382. ISBN: 978-1-58113-391-2. DOI: 10/b776vb. URL: https://doi.org/10.1145/502512.502568 (cit. on pp. 22, 42, 60, 99).

[86] Joe Sullivan. *Enable Rapid Insights and Real-Time Data Analytics Pipelines from Edge to Hybrid Cloud*. May 2, 2019. URL: https://community.hpe.com/t5/AI-Insights/Enable-rapid-insights-and-real-time-data-analytics-pipelines/ba-p/7044733 (cit. on p. 7).

[87] Unknown. *Global Surface Summary of the Day - GSOD*. URL: https://www.ncei.noaa.gov/access/metadata/landing-page/bin/iso?id=gov.noaa.ncdc:C00516 (cit. on pp. 43, 60, 99).

[88] Mark van Heeswijk et al. “Adaptive Ensemble Models of Extreme Learning Machines for Time Series Prediction”. In: *Artificial Neural Networks – ICANN 2009*. Ed. by Cesare Alippi et al. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2009, pp. 305–314. ISBN: 978-3-642-04277-5. DOI: 10/bjczbh (cit. on p. 32).

[89] Alexander Vergara et al. “Chemical Gas Sensor Drift Compensation Using Classifier Ensembles”. In: *Sensors and Actuators B: Chemical* 166-167 (May 20, 2012), pp. 320–329. ISSN: 0925-4005. DOI: 10/tcd. URL: http://www.sciencedirect.com/science/article/pii/S0925400512002018 (cit. on pp. 1, 20, 32).

[90] Jian Wang et al. “A Review on Extreme Learning Machine”. In: *Multimedia Tools and Applications* 81.29 (Dec. 1, 2022), pp. 41611–41660. ISSN: 1573-7721 (cit. on p. 52).

[91] Kun Wang et al. “Evolving Gradient Boost: A Pruning Scheme Based on Loss Improvement Ratio for Learning Under Concept Drift”. In: *IEEE Transactions on Cybernetics* 53.4 (Apr. 2023), pp. 2110–2123. ISSN: 2168-2275. DOI: 10/gm6mrn (cit. on pp. 32, 33, 53, 99, 101).

[92] Shuo Wang, Leandro L. Minku, and Xin Yao. “A Systematic Study of Online Class Imbalance Learning With Concept Drift”. In: *IEEE Transactions on Neural Networks and Learning Systems* 29.10 (Oct. 2018), pp. 4802–4821. ISSN: 2162-2388. DOI: 10/gfcqm8 (cit. on p. 21).

[93] Shuo Wang et al. “Concept Drift Detection for Online Class Imbalance Learning”. In: *The 2013 International Joint Conference on Neural Networks (IJCNN)*. Aug. 2013, pp. 1–10. DOI: 10/gf6nwt (cit. on p. 21).

[94] XueSong Wang et al. “Multiscale Drift Detection Test to Enable Fast Learning in Nonstationary Environments”. In: *IEEE Transactions on Cybernetics* 51.7 (July 2021), pp. 3483–3495. ISSN: 2168-2275. DOI: 10/gm854z (cit. on pp. 23–25, 28, 31).

[95] Allan P. White and Wei Zhong Liu. “Bias in Information-Based Measures in Decision Tree Induction”. In: *Machine Learning* 15.3 (June 1994), pp. 321–329. ISSN: 1573-0565 (cit. on p. 53).

[96] Gerhard Widmer and Miroslav Kubat. “Learning in the Presence of Concept Drift and Hidden Contexts”. In: *Machine Learning* 23.1 (Apr. 1, 1996), pp. 69–101. ISSN: 1573-0565. DOI: 10.1023/A:1018046501280. URL: https://doi.org/10.1023/A:1018046501280 (cit. on p. 17).

[97] Z. Yang et al. “A Novel Concept Drift Detection Method for Incremental Learning in Nonstationary Environments”. In: *IEEE Transactions on Neural Networks and Learning Systems* 31.1 (Jan. 2020), pp. 309–320. ISSN: 2162-2388. DOI: 10/ggk944. URL: https://ieeexplore.ieee.org/abstract/document/8674766 (cit. on pp. 2, 23–25, 28, 29, 31, 36–38, 40, 42, 47, 51, 98, 99, 108–113, 116–118).

[98] En Yu et al. “Online Boosting Adaptive Learning under Concept Drift for Multistream Classification”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 38.15 (15 Mar. 24, 2024), pp. 16522–16530. ISSN: 2374-3468. DOI: 10/gt75cq. URL: https://ojs.aaai.org/index.php/AAAI/article/view/29590 (cit. on pp. 25, 28, 31, 32).

[99] Shujian Yu et al. “Concept Drift Detection and Adaptation with Hierarchical Hypothesis Testing”. In: *Journal of the Franklin Institute* 356.5 (Mar. 2019), pp. 3187–3215. ISSN: 0016-0032. DOI: 10/gnjdpt (cit. on p. 28).

[100] G. Udny Yule. “Why Do We Sometimes Get Nonsense-Correlations between Time-Series?--A Study in Sampling and the Nature of Time-Series”. In: *Journal of the Royal Statistical Society* 89.1 (1926), pp. 1–63. ISSN: 0952-8385. DOI: 10/crfzdj. JSTOR: 2341482. URL: https://www.jstor.org/stable/2341482 (cit. on p. 57).

[101] Poorya ZareMoodi, Sajjad Kamali Siahroudi, and Hamid Beigy. “Concept-Evolution Detection in Non-Stationary Data Streams: A Fuzzy Clustering Approach”. In: *Knowledge and Information Systems* 60.3 (Sept. 1, 2019), pp. 1329–1352. ISSN: 0219-3116. DOI: 10/gf6c67. URL: https://doi.org/10.1007/s10115-018-1266-y (cit. on pp. 24, 30).

[102] Lingxiang Zhang and Xiaotong Zhang. “Spurious Granger Causality between a Broken-Trend Stationary Process and a Stochastic Trend Process”. In: *Mathematics and Computers in Simulation* 81.8 (Apr. 1, 2011), pp. 1673–1681. ISSN: 0378-4754 (cit. on p. 57).

[103] Yifan Zhang et al. “OneNet: Enhancing Time Series Forecasting Models under Concept Drift by Online Ensembling”. In: *Advances in Neural Information Processing Systems* 36 (Dec. 15, 2023), pp. 69949–69980. URL: https://proceedings.neurips.cc/paper_files/paper/2023/hash/dd6a47bc0aad6f34aa5e77706d90cdc4-Abstract-Conference.html (cit. on pp. 31, 32).

[104] Xiulin Zheng et al. “Semi-Supervised Classification on Data Streams with Recurring Concept Drift and Concept Evolution”. In: *Knowledge-Based Systems* 215 (Mar. 2021), p. 106749. ISSN: 0950-7051. DOI: 10/gmvzq5 (cit. on pp. 24, 30, 32).

[105] Morteza Zihayat et al. “Mining High Utility Sequential Patterns from Evolving Data Streams”. In: *Proceedings of the ASE BigData & SocialInformatics 2015*. ASE BD&SI ’15. Kaohsiung, Taiwan: Association for Computing Machinery, Oct. 7, 2015, pp. 1–6. ISBN: 978-1-4503-3735-9. DOI: 10.1145/2818869.2818883. URL: https://doi.org/10.1145/2818869.2818883 (cit. on pp. 8, 20).

[106] Indrė Žliobaitė, Mykola Pechenizkiy, and João Gama. “An Overview of Concept Drift Applications”. In: *Big Data Analysis: New Algorithms for a New Society*. Ed. by Nathalie Japkowicz and Jerzy Stefanowski. Studies in Big Data. Cham: Springer International Publishing, 2016, pp. 91–114. ISBN: 978-3-319-26989-4. DOI: 10.1007/978-3-319-26989-4_4. URL: https://doi.org/10.1007/978-3-319-26989-4_4 (cit. on pp. 7, 11).

[107] Paweł Zyblewski, Robert Sabourin, and Michał Woźniak. “Preprocessed Dynamic Classifier Ensemble Selection for Highly Imbalanced Drifted Data Streams”. In: *Information Fusion* 66 (Feb. 2021), pp. 138–154. ISSN: 1566-2535. DOI: 10/gm854r (cit. on p. 22).