Universidade Estadual de Londrina

GUILHERME YUKIO SAKURAI

PERFORMANCE OF ALGORITHMS AND AN ENSEMBLE STRATEGY FOR DETECTING HETEROGENEOUS DRIFTS

LONDRINA

2025

GUILHERME YUKIO SAKURAI

PERFORMANCE OF ALGORITHMS AND AN ENSEMBLE STRATEGY FOR DETECTING HETEROGENEOUS DRIFTS

Dissertação apresentada ao Programa de Mestrado em Ciência da Computação da Universidade Estadual de Londrina para obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Bruno Bogaz Zarpelão

Coorientador: Prof. Dr. Sylvio Barbon Junior

LONDRINA

2025

Ficha de identificação da obra elaborada pelo autor, através do Programa de Geração Automática do Sistema de Bibliotecas da UEL

S158p Sakurai, Guilherme Yukio .
PERFORMANCE OF ALGORITHMS AND AN ENSEMBLE STRATEGY FOR DETECTING HETEROGENEOUS DRIFTS / Guilherme Yukio Sakurai. - Londrina, 2025.
57 f. : il.

Orientador: Bruno Bogaz Zarpelão.
Coorientador: Sylvio Barbon Junior.
Dissertação (Mestrado em Ciência da Computação) - Universidade Estadual de Londrina, Centro de Ciências Exatas, Programa de Pós-Graduação em Ciência da Computação, 2025.
Inclui bibliografia.

1. Stream Mining - Tese. 2. Concept Drift - Tese. 3. Drift Detection - Tese. 4. Ensemble Learning - Tese. I. Zarpelão, Bruno Bogaz. II. Junior, Sylvio Barbon. III. Universidade Estadual de Londrina. Centro de Ciências Exatas. Programa de Pós-Graduação em Ciência da Computação. IV. Título.

CDU 519

GUILHERME YUKIO SAKURAI

PERFORMANCE OF ALGORITHMS AND AN ENSEMBLE STRATEGY FOR DETECTING HETEROGENEOUS DRIFTS

Dissertação apresentada ao Programa de Mestrado em Ciência da Computação da Universidade Estadual de Londrina para obtenção do título de Mestre em Ciência da Computação.

BANCA EXAMINADORA

Orientador: Prof. Dr. Bruno Bogaz Zarpelão
Universidade Estadual de Londrina

Prof. Dr. Wesley Attrot
Universidade Estadual de Londrina - DC/UEL

Dr. Saulo Martiello Mastelini
Kunumi

Londrina, 04 de Fevereiro de 2025.

Este trabalho é dedicado às crianças adultas
que, quando pequenas, sonharam em se
tornar cientistas.

AGRADECIMENTOS

Expresso minha mais profunda gratidão à minha família. Aos meus pais, Edson Sakurai e Daniella Sakurai, e à minha irmã, Bianca Sakurai, por todo o apoio incondicional. Um agradecimento especial à minha companheira, Luana Ueno, cujo incentivo constante tem sido inestimável ao longo da minha trajetória acadêmica, desde a graduação.

Gostaria também de manifestar meu sincero agradecimento aos meus professores e excepcionais orientadores, Bruno Bogaz Zarpelão e Sylvio Barbon Junior, cujo direcionamento e expertise tornaram esta pesquisa possível.

"A ciência é a forma mais elevada de curiosidade, alimentada pela busca incessante do conhecimento e pelo desejo de compreender o desconhecido"

SAKURAI, G. Y.. **Desempenho de Algoritmos e uma Estratégia de Ensemble para Detecção de Mudanças de Conceito Heterogêneos**. 2025. 59f. Dissertação (Mestrado em Ciência da Computação) – Universidade Estadual de Londrina, Londrina, 2025.

RESUMO

A mineração de dados tem se tornado cada vez mais popular devido à sua capacidade de oferecer algoritmos e metodologias que atendem demandas da Internet das Coisas (IoT) e dos sistemas modernos de aprendizado de máquina. Dentro desse contexto, a detecção de mudanças de conceito é primordial, especialmente para identificar alterações na distribuição dos dados durante a operação de soluções de aprendizado de máquina. Um dos maiores desafios enfrentados por algoritmos de detecção de mudanças de conceito é a capacidade de lidar com diferentes tipos de mudanças, como mudanças abruptas, graduais e incrementais. Este trabalho atua em duas frentes para melhorar a detecção dessas mudanças heterogêneas. Primeiramente, realizamos um estudo que estabelece um *benchmark* para quatro algoritmos de detecção de mudanças de conceito (EDDM, DDM, HDDMW e HDDMA), avaliando seu desempenho em termos de detecção, tempo de resposta e atraso na detecção, utilizando conjuntos de dados sintéticos. Os resultados indicam que o HDDMW oferece o melhor equilíbrio entre os indicadores de desempenho, especialmente na detecção de mudanças de conceito abruptas, embora apresente limitações em termos de tempo de resposta e detecção de mudanças incrementais. Na segunda frente de atuação, propomos o método Self-tuning Drift Ensemble (StDE), um novo algoritmo de *ensemble* para detecção de mudanças de conceito que utiliza mecanismos dinâmicos para adaptar-se às mudanças nas características dos fluxos de dados em tempo real. Diferente de outros detectores de mudanças de conceito baseados em *ensemble*, o StDE ajusta dinamicamente o número de detectores base, mantendo uma solução leve e eficiente. Experimentos realizados em diversos cenários de mudança de conceito demonstram que o método proposto supera os algoritmos estabelecidos, apresentando alta taxa de precisão na detecção de mudanças.

Palavras-chave: Stream Mining, Concept Drift, Drift Detection, Tuning, Ensemble Learning

SAKURAI, G. Y.. **Performance of Algorithms and an Ensemble Strategy for Detecting Heterogeneous Drifts**. 2025. 59p. Master's Thesis (Master in Science in Computer Science) – State University of Londrina, Londrina, 2025.

ABSTRACT

Data mining has become increasingly popular due to its ability to offer algorithms and methodologies that address the challenges of the Internet of Things (IoT) and modern machine learning systems. In this context, concept drift detection is crucial, especially for identifying changes in data distribution during the operation of machine learning solutions. One of the biggest challenges faced by concept drift detection algorithms is the ability to handle different types of drifts, such as abrupt, gradual, and incremental changes. This work addresses this issue in two ways to improve the detection of these heterogeneous drifts. First, we conduct a study that establishes a benchmark for four drift detection algorithms (EDDM, DDM, HDDMW, and HDDMA), evaluating their performance in terms of detection accuracy, response time, and detection delay using synthetic datasets. The results indicate that HDDMW offers the best balance among performance indicators, especially in detecting abrupt drifts, although it has limitations in terms of response time and incremental drift detection. Secondly, we propose the Self-tuning Drift Ensemble (StDE) method, a novel ensemble algorithm for drift detection, which utilizes dynamic mechanisms to adapt to real-time changes in data stream characteristics. Unlike other ensemble-based drift detectors, StDE dynamically adjusts the number of base detectors, maintaining a lightweight and efficient solution. Experiments conducted in various concept drift scenarios demonstrate that the proposed method outperforms established algorithms, showing high precision in drift detection.

Keywords: Stream Mining, Concept Drift, Drift Detection, Tuning, Ensemble Learning

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

StDE	Self-Tuning Drift Ensemble
IoT	Internet of Things
DDM	Drift detection method
EDDM	Early drift detection Method
HDDMA	Hoeffding's drift detection method with A-Test
HDDMW	Hoeffding's drift detection method with W-Test
ADWIN	ADaptive WINdowing
PHT	Page-Hinkley Test
KSWIN	Kolmogorov-Smirnov Windowing
EDIST	Error Distance-based Approach for Drift Detection
STED	Statistical Test Ensemble Detector
STEPD	Statistical Test of Equal Proportions Detection
SLED	Semi-supervised locally-weighted ensemble detector
SDDE	Statistical Drift Detection Ensemble

CONTENTS

1 INTRODUCTION

The rapid development witnessed in recent years in wireless sensor networks, cloud computing, and big data has led to the widespread use of devices that gather, transmit, and process online data across various domains. Examples of applications include weather forecasting, network traffic analysis, power grid control, and stock market trading. These applications may produce data streams with challenging characteristics such as massive size (potentially infinite) and high velocity. As a result, mining algorithms designed for these streams have to meet strict constraints in terms of response latency and storage. Additionally, as these data streams may evolve over time, mining algorithms must be able to incorporate new knowledge as the streams are processed [2, 3].

The dynamic and infinite nature of real-world data streams often results in changes in their behavior, known as concept drifts. Seasonal events, changes in the underlying system configuration, or other unexpected situations can cause these drifts. For instance, in credit card fraud detection systems, user habits and transaction patterns evolve over time, and fraudsters continuously develop new techniques [4]. This ongoing change, referred to as concept drift, necessitates that machine learning solutions adapt to temporal variations to maintain their effectiveness. As these drifts represent changes in data distribution, they must be detected and handled to ensure that mining algorithms can adapt to the new reality they impose.[3, 5, 6]. Concept drift types include: abrupt drifts, where changes occur suddenly; gradual drifts, where changes happen slowly over time; incremental drifts, where small changes accumulate gradually; recurring or periodic drifts, where patterns change cyclically; sudden drifts, where unexpected sharp changes occur; mixed drifts, where different types of drifts happen together; and blip, where there are temporary short-term changes.

Given the diversity of drift types, designing detectors that can cover multiple types is difficult. They may require different detection techniques or particular hyperparameter configurations to be effective. Korycki et al. [7] noted that simple, independent detectors may fail to manage complex drifts effectively.

Ensemble-based approaches offer a potential solution, but some open issues remain. Perez et al. [8] highlighted that while ensemble methods with various classifiers offer advantages, they can introduce significant computational overhead. Du et al. [9] also pointed out scalability issues with multiple detectors, and Komorniczak et al. [5] noted that a fixed number of detectors might limit flexibility. Abbasi et al. [10] added that offline classifiers, typically supervised, might slow down the adaptation to new data. While these approaches handle various drift types well, they can be slow to respond to real-time changes. New solutions are expected to be more adaptive, i.e., able to change their internal

configurations to keep operating with limited memory and high accuracy over unbounded and changing streams that may present heterogeneous drifts.

This work addresses the detection of different types of drift from two approaches. Firstly, we investigate the performance of four commonly used methods (DDM, EDDM, HDDMW, and HDDMA) across hundreds of synthetic datasets to tackle the challenge of selecting a suitable change detection algorithm in a wide range of possible scenarios. These datasets were designed using a tool developed in this work to simulate various types of concept drift, with varying amplitude and duration. The number of drifts and the stream size were also manipulated to increase the diversity. We evaluated the algorithms based on several criteria, including detection accuracy, detection time, and detection delay. By exploring these perspectives, we aimed to provide insights into the strengths and limitations of each method, as well as their suitability for different types of concept drift scenarios.

As a second step, we propose the Self-tuning Drift Ensemble (StDE), an ensemble-based method for concept drift detection. StDE adapts to changes in the data stream and drift characteristics by dynamically adjusting the number and configuration of its base learners based on their voting behavior. To detect drifts, StDE employs a hard voting scheme: a drift is detected when at least a simple majority of the base learners agree. The operation of StDE involves two phases. In the first phase, the Warm Start phase, an ensemble of three detectors is self-tuned over a finite, offline labeled stream. In the second, Online phase, StDE monitors the incoming stream in real-time to detect drifts. The voting results determine the adjustments needed for the ensemble. Unanimous decisions suggest a stable situation, prompting a reduction in the number of detectors to save resources. Conversely, divided decisions indicate uncertainty, leading to the addition of new detectors with slightly modified hyperparameters. Tests conducted on streams with both homogeneous and heterogeneous drifts demonstrated that StDE is robust to various drift types and effectively adjusts the number of detectors in response to stream changes, maintaining a lightweight structure.

The present dissertation seeks to answer the following research question:

"How can existing drift detection methods (namely, DDM, EDDM, HDDMW, and HDDMA) be applied to detect different types of drifts in data streams, considering both homogeneous and heterogeneous drift scenarios?"

Based on the existing literature and preliminary studies, we propose the following hypothesis:

"The application of an ensemble method may enhance drift detection performance by effectively combining the complementary strengths of individual algorithms. An ensemble approach is expected to improve overall detection accuracy, and adapt more efficiently

to varying drift patterns in data streams."

To test this hypothesis, the research establishes the following objectives:

- A systematic comparison among DDM, EDDM, HDDMW, and HDDMA change detectors when processing streams with abrupt, gradual, and incremental drifts. Three different performance perspectives (accuracy, time, and delay) were analyzed.
- The proposal of the Self-tuning Drift Ensemble (StDE) method, which dynamically adjusts its configuration based on real-time data stream characteristics to detect concept drifts efficiently.
- The introduction of a free online tool to generate synthetic streams with different concept drift setups.

The remainder of this work is organized as follows: Chapter 2 presets the theoretical background. Chapter 3 presents the related work. Chapter 4 describes StDE. The experimental results are presented and discussed in Chapter 5. Finally, Chapter 6 provides the concluding remarks.

2 THEORETICAL BACKGROUND

Change detectors are techniques to identify changes in data streams that may indicate shifts in their underlying distribution. In many applications, the ability to quickly detect changes and adapt to them is critical. The problem of change detection becomes even more challenging when dealing with concept drifts, where the concept can change with time. When a concept drift occurs, the relationships between the input features and the output variable may change, leading to a shift in the data distribution. This means that the model trained on the previous data may become less accurate or even useless for making predictions on the new data. Therefore, detecting and adapting to concept drifts in a timely and effective manner is crucial for maintaining model performance and avoiding costly mistakes.

2.1 Stream Mining

Data streams refer to a continuous, potentially infinite flow of data that arrives at high speed and often unpredictably. These streams are increasingly common in various applications, such as sensor networks, social media, financial transactions, and network traffic, where data must be processed and analyzed in real-time [11]. Unlike traditional batch data processing, where data is collected and analyzed in discrete chunks, data stream processing involves the real-time analysis of data as it flows into the system, making it essential to develop efficient methods for handling this dynamic environment [12].

A key challenge in data stream processing is the classification of incoming data points as they arrive, often referred to as stream classification. This involves creating a model that can accurately map new data points to their respective class labels, despite the evolving nature of the data distribution. To formalize this, we can model the generation of a data stream as a random variable S, from which objects $o \in dom(S)$ are randomly drawn. Here, $dom(\cdot)$ represents the domain of a random variable [13]. Let S_t denote a random variable representing the data stream at time t, where t is a positive integer. Each data point $i_t \in dom(S_t)$ is generated by the joint distribution $P_t(i_t, j_t)$, where j_t is the corresponding class label of i_t. The goal of stream classification is to determine a classifier $f_t : dom(S_1) \times \cdots \times dom(S_t) \to \mathcal{L}$ that maps the current and past data instances to their respective class labels, where $\mathcal{L}$ is the set of possible class labels.

2.2 Concept Drift

Changes in data streams in non-stationary environments are reflected through alterations in their probability distributions, which are referred to as concept drifts. For-

mally, let C_t denote the true concept or data distribution at time t. In a typical machine learning setting, we train a model M on a fixed dataset with the assumption that the distribution of the data will remain the same in the future. However, in a streaming scenario, the distribution of the data may shift or evolve over time, leading to a mismatch between the model and the true concept. This drift in the concept can lead to a decrease in the performance of the model over time. A concept drift can manifest in different forms, including abrupt, gradual, and incremental drift [14].

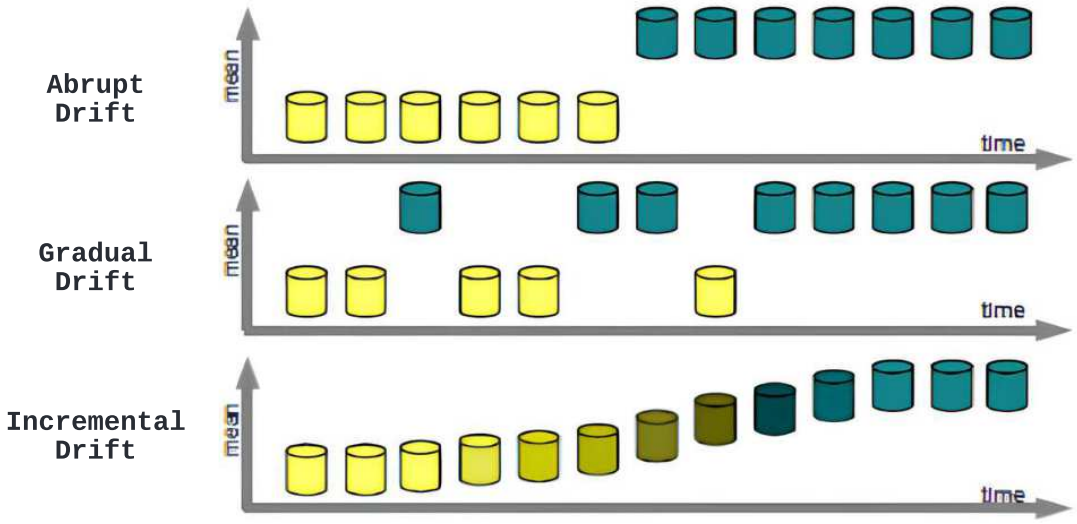

Figure 1 – Types of Concept Drift [1]

As presented in Figure 1, abrupt drift occurs when there is a sudden and significant change in the underlying data distribution. In other words, abrupt drift is characterized by an immediate transition from one concept to another, without any transitional period where both the old and new distributions coexist. The new data distribution is often very different from the old one, and the model that was previously effective may become obsolete. The main challenge with abrupt drift is to quickly detect the change and adapt the model to the new data distribution [14]. An example of abrupt drift could be a change in the way that a sensor is calibrated, resulting in an abrupt shift in the readings.

The second type, gradual drift, occurs when the underlying data distribution slowly changes over time [15]. Gradual drift is marked by a smooth and progressive transition, where the old and new concepts may coexist for a period. This overlap makes it difficult to detect the exact moment when the shift begins, as the change unfolds subtly over time. The main challenge with gradual drift is to continuously monitor the data and update the model accordingly to reflect the evolving distribution. An example of gradual drift could be a change in consumer behavior over time, leading to a slow shift in product preferences.

The last type, incremental drift, refers to a continuous, stepwise change in the data distribution. Although it shares similarities with gradual drift, incremental drift is distinguished by the accumulation of small, consistent changes over time that eventually result in a significant shift. Additionally, there may be a point at which the rate of change increases suddenly. The main challenge with incremental drift is detecting these small changes early enough and adapting the model promptly to avoid performance degradation. An example of incremental drift could be a seasonal shift in a heating system, where the change occurs gradually over several months but accelerates rapidly during the winter.

According to [15, 16], the formal representation of a concept drift between two-time points (t and $t+1$) over a given period of time is:

$$\exists X : P_t(X, y) \neq P_{t+1}(X, y) \tag{2.1}$$

In Equation (2.1), P_t represents the joint distribution between the set of input variables X and the target variable y at time t. Similarly, P_{t+1} denotes the joint distribution at time $t+1$. Concept drift is related to the covariance shift in the distribution, which means that a change in the joint probability of X and y indicates the occurrence of concept drift at time t. Hence, $P_t(X, y)$ can be defined as a composition of $P_t(X) \times P_t(y|X)$. Change detection algorithms can detect concept drift by monitoring changes in the joint distribution $P_t(X, y)$ over time. If the distribution of the input variables or the conditional distribution of the output variables change significantly over time, it can indicate the occurrence of concept drift. Change detection algorithms typically monitor the data statistical properties, such as the mean, variance, or distribution, to detect changes in the joint distribution. When a significant change is detected, the algorithm can trigger the retraining of the model or some other action to adapt to the new concept.

2.2.1 Concept Drift Detection Algorithms

In this section, we discuss some of the key concept drift detection algorithms from the literature: DDM [17], EDDM [18], HDDMA, and HDDMW [19]. These algorithms are widely recognized for their effectiveness in monitoring changes in data streams and are frequently used in the field of data stream mining.

2.2.1.1 DDM

The Drift Detection Method (DDM) employs statistical tests to determine whether a drift has occurred, based on the difference between the observed error rate and the expected error rate relative to a predefined threshold [17]. In this context, the error rate refers to the proportion of incorrect predictions made by a machine learning model when classifying incoming data points. Specifically, as new data points arrive, the algorithm

tracks the model's performance by calculating the error rate, which is the ratio of incorrectly classified instances to the total number of instances observed up to that point.

DDM operates under the assumption that the data are generated by a stationary process, meaning that the underlying data distribution remains constant over time. Under this assumption, the algorithm continuously monitors the error rate, anticipating a decrease as the model is exposed to more data, provided that the data distribution remains unchanged. To effectively analyze the model's performance dynamics, the algorithm calculates both the average error rate and its variance.

When the observed error rate significantly exceeds a calculated threshold determined using the Hoeffding bound, which provides a probabilistic guarantee on the difference between the observed and true error rates, the algorithm signals a drift. This threshold is adaptive, adjusting as more data points are processed, thus enhancing the algorithm's detection accuracy while minimizing false alarms. The adaptiveness of DDM ensures that the threshold remains sensitive to real changes in data distribution. Notably, DDM is computationally efficient and has demonstrated strong performance in practice, particularly in detecting abrupt drifts where the data distribution shifts suddenly.

2.2.1.2 EDDM

The Early Drift Detection Method (EDDM) is a modified technique that enhances the DDM approach [18]. It is specifically designed to detect drifts as early as possible, even before significant changes in the data distribution occur.

The algorithm computes the minimum distance between the reference distribution—a baseline representation of the data distribution under normal conditions—and the observed distribution in each window—a subset of data points taken from the stream within a specific timeframe or number of samples. The size of the window is typically determined based on the specific application and desired sensitivity, striking a balance between detecting subtle changes and maintaining computational efficiency.

When the minimum distance exceeds a predetermined threshold, the algorithm signals a drift. This threshold is calculated using statistical tests, such as the Student's t-test and the Chi-squared test, which are particularly sensitive to small changes in data distribution. EDDM utilizes different statistical tests compared to DDM and has demonstrated effectiveness in detecting both abrupt and gradual drifts.

2.2.1.3 HDDMA and HDDMW

Drift detection algorithms created according to the Hoeffding Drift Detection Method (HDDM) modify the original DDM using Hoeffding's inequality [19]. HDDMA is one of these algorithms and uses the moving average of the observed error rate to estimate

the true error rate of the classifier. The algorithm calculates the average error rate over a sliding window of fixed size and compares it to Hoeffding's bound. If the observed error rate exceeds the bound, the algorithm signals a drift.

The HDDMW algorithm is similar to the HDDMA algorithm but uses a moving weighted average of the observed error rate instead. The algorithm assigns a weight to each data point based on its recency, with more recent data points having higher weights. The weighted average is then used to estimate the true error rate of the classifier and then compared to Hoeffding's bound. HDDMW is effective in detecting both abrupt and gradual drifts, and it outperforms DDM and HDDMA in some scenarios.

3 RELATED WORK

In this section, we introduce the related work corresponding to the two primary objectives of this study. Sections 3.1 and 3.2 will present the relevant research on the concept drift algorithms and the ensemble-based concept drift detectors, respectively. This structured approach aims to provide a comprehensive understanding of the current state of the art in these areas and to highlight the contributions and innovations introduced by our work.

3.1 Concept Drift Algorithms

Each change detection algorithm has its advantages and drawbacks, and the choice of a particular one will depend on the specific characteristics of the data stream and the requirements of the application [20, 18, 19]. In the context of data stream mining, monitoring the statistical properties of the data stream is one of the most commonly used strategies to detect and identify different types of concept drift. However, there is no single algorithm that works well for all types of drift, and different strategies may be more appropriate for different types of drift. For example, Adaptive Windowing (ADWIN), based on monitoring the mean and variance of the data stream, can be effective for detecting abrupt drifts, where the statistical properties of the data change suddenly [21]. On the other hand, monitoring the distribution of the data stream can be more effective for detecting gradual drifts, where the statistical properties of the data change slowly over time, such as the early drift detection method (EDDM) [18]. Thus, there are various types of related research on drift detection [22, 23, 24] with different biases and contributions.

Poenaru-Olaru et al. [25] assessed the reliability of concept drift detectors by exploring how late they were in reporting drifts and how many false alarms they signaled. The study compared the performance of the most popular drift detectors belonging to two different groups, error rate-based detectors and data distribution-based detectors, on both synthetic and real-world data, providing an insightful discussion of the methods. However, the study only covered abrupt and gradual drift, excluding incremental drift. As incremental changes are prevalent in real-world applications, especially in streaming data scenarios, their absence in the study opens the door for further investigation. Another point to consider is that one of the metrics explored to evaluate the drift detectors was latency, which ranges between 0 and 1 to show how late the detector manages to detect the drift. Using latency as a metric to evaluate drift detectors might not provide a comprehensive analysis of the detector's delay performance. The detectors that operate on a sliding window of data are grounded on detecting the drift as soon as it occurs, regardless

of how long it takes to make a prediction on the data within the window.

Gonçalves Jr. et al. [26] evaluated the performance of eight distinct concept drift detectors using artificial datasets that were subject to both abrupt and gradual concept drifts, as well as real-world datasets. The experiments compared the accuracy, evaluation time as well as false alarms and miss detection rates. As with Poenaru-Olaru et al. [25], incremental change was not considered as a type of drift to be explored. However, an important metric was used to enable a fair comparison among drift detectors—the distance to the drift point, which provides information on which method most accurately identified the position of the drift.

Considering the different applications of drift detectors, Barros and Santos [22] compared their configurations of the concept drift detectors in artificial datasets. The main goal was to adequately measure the performance of concept drift detectors and pinpoint the common challenges in detecting drifts closer to their correct positions. Furthermore, the study aimed to determine the parameters that had the most significant impact on the predictive accuracy of the methods. Following the previous surveys, the authors only considered abrupt and gradual changes, limiting the scope of the experiments.

In summary, while the reviewed studies explore contexts and types of drift similar to those in our work, they often lack diversity in their experimental design, leading to results that may not fully capture the complexities and nuances of real-world scenarios. Many of these studies use a limited set of performance metrics, which may overlook critical aspects of drift detection, such as detection time and delay, especially in varied drift conditions.

Our study addresses these gaps by conducting a comprehensive benchmarking analysis of four widely-used drift detection algorithms— DDM, EDDM, HDDMA, and HDDMW —across scenarios involving different types of drifts: abrupt, gradual, and incremental. By systematically evaluating these algorithms using a broad range of performance metrics, including detection accuracy, time, and delay, our research provides a more detailed and nuanced understanding of their strengths and limitations. This approach not only enhances the generalizability of our findings but also offers practical insights for selecting the most appropriate algorithm depending on the specific characteristics of the drift encountered in real-world data streams. Thus, our work contributes to the field by offering a more robust benchmark that can be leveraged in diverse applications, ultimately leading to more effective and reliable drift detection in practice.

3.2 Ensemble-based concept drifts detectors

The identification and adaptation to concept drifts in data streams have been explored through both single-model and ensemble-based approaches. Ensemble methods, in

particular, offer a promising solution because they combine multiple base learners, each of which may excel at detecting different types of drifts. By leveraging the diverse strengths of these base learners, ensembles can provide a more robust and comprehensive detection mechanism, capable of handling the wide variety of drifts that can occur in data streams. This section reviews various techniques, with a particular focus on ensemble-based methods for detecting concept drifts. While strategies for prediction and adaptation, as well as single-model detectors, are important, they fall outside the scope of our proposal. Instead, we concentrate on how ensembles can effectively identify drifts within data streams.

Most ensemble drift detectors are based on supervised learning, particularly using machine learning algorithms. Deckert [27] introduces a Batch Weighted Ensemble framework that relies on supervised learning models to improve performance over single detectors. This approach integrates multiple classifiers and employs a weighted voting scheme based on the performance of individual classifiers. Abbasi et al. [10] use an ensemble of supervised machine learning algorithms to handle concept drift in dynamic social big data streams. They combine several classifiers such as K-Nearest Neighbor, Random Forest, and Multilayer Perceptron to adapt to changes in the data stream efficiently. Ramane et al. [28] introduce an ensemble learning framework integrating two detectors and a classifier to manage concept drift. They proposed dissimilarity-based and performance-based drift detection methods that, when combined, provide effective drift detection. Mavromatis et al. [29] presented a framework named LE3D, capable of identifying irregularities in sensor streams. Operating as an ensemble framework, decisions are based on three estimators: ADaptive WINdowing (ADWIN), Page-Hinkley Test (PHT), and Kolmogorov-Smirnov Windowing (KSWIN).

It is important to mention that relying on machine learning algorithms require implementing offline and online phases, creating computationally costly solutions, and even violating stream mining constraints. For example, Abbasi et al. [10] noted that reliance on offline classifiers may slow down the adaptation to new data and Deckert [27] does not address potential challenges in scalability and computational overhead when using traditional machine learning.

For this reason, we consider that solutions based on drift detectors fit stream constraints more properly. The first work grounded on an ensemble of drift detectors was proposed by Khamassi et al. [30]. The authors discussed integrating drift detection methods and proposed the Error Distance-based Approach for Drift Detection (EDIST), a new method that monitors the distance between two consecutive classification errors. This proposal differs from ours due to the use of a fixed number of detectors with fixed hyperparameters and the combination of classification procedures.

Du et al. [9] proposed a selective detector ensemble using several detectors like

Drift Detection Method (DDM), Early Drift Detection Metod (EDDM), ADWIN, and Statistical Test of Equal Proportions Detection (STEPD) to identify concept drifts. Their approach leverages supervised learning to maintain model accuracy but has a high level of false positives due to the early-find-early-report strategy, which prioritizes early signals from individual detectors. Samant et al. [31] proposed a selective ensemble method that dynamically chooses the most relevant detectors (ADWIN, DDM, and EDDM) based on the performance of a machine learning classifier. Their proposal combines detectors and classifiers, utilizing the classification output as an oracle for adapting the detectors.

Korycki et al. [7] discussed the use of an ensemble drift detection with feature subspaces (EDFS) that operate independently to monitor and detect drifts in data streams, focusing on the simplicity and efficiency of individual detectors. Perez et al. [8] introduced the Statistical Tests Ensemble Detector (STED), which detects changing data distributions using the results of Brown-Forsythe, O'Brien, and ANOVA statistical tests combined with two voting strategies. Zhang et al. [32] proposed a drift detection method that monitors changes in data distribution based on a semi-supervised locally-weighted ensemble detector (SLED), where the relative performance among its base detectors is characterized by a set of weights learned in a semi-supervised manner. SLED effectively deals with situations where different types of drift coexist, only abrupt and gradual ones are considered though. Komorniczak et al. [5] proposed the Statistical Drift Detection Ensemble (SDDE), which utilizes a fixed set of detectors to manage various types of drifts, including sudden, incremental, and gradual changes, using a Gaussian Naïve Bayes (NB) classifier.

Supported by the current literature review on ensembles of drift detectors shown in Table 1, we observed that while existing approaches provide valuable insights, there is still a need for further research to develop methods that can flexibly detect a wide range of drifts while maintaining a lightweight and efficient structure. Many of the existing ensembles focus on specific types of drifts or require extensive computational resources, limiting their applicability in dynamic and resource-constrained environments.

In contrast, our proposed Self-tuning Drift Ensemble (StDE) addresses these limitations by offering a versatile ensemble framework capable of dynamically integrating multiple drift detectors. This allows for the detection of various types of drifts—abrupt, gradual, and incremental—in real-time, adapting to the evolving characteristics of the data stream. The StDE's flexibility in adding or removing detectors based on the stream's behavior ensures that it remains both effective and efficient, providing a robust solution that can outperform existing methods in diverse scenarios. By leveraging this dynamic approach, our work not only fills a critical gap in the literature but also paves the way for more adaptive and scalable drift detection strategies that are better suited to the complexities of real-world data streams.

Author	**Technique**	**Base Learners**	**# Learners**
[27]	Supervised	Decision Trees	HP (2)
[9]	Drift Detectors	DDM, EDDM, ADWIN, STEPD	HP (10)
[30]	Hybrid	EDIST and Hoeffding Tree	HP (2)
[33]	Drift Detectors	HDDM, DDM, ECDD and ADWIN	Fixed (3)
[7]	Drift Detectors	EDFS and Hoeffding Tree	HP (10)
[8]	Supervised	Brown-Forsythe, O'Brien and ANOVA	Fixed (3)
[32]	Semi-supervised	SLED	HP (10)
[10]	Supervised	RF, XGB, and MLP	HP (3)
[31]	Drift Detectors	ADWIN, DDM and EDDM	Fixed (3)
[28]	Hybrid	DDBM and LPBM	Fixed (2)
[5]	Hybrid	Gaussian NB	HP (30)
[29]	Hybrid	ADWIN, KSWIN and PHT	Fixed (3)
StDE	Drift Detectors	DDM and EDDM	Dynamic

Table 1 – A summary of ensemble drift detectors from different techniques (Supervised, Semi-supervised, Hybrid, and Drift Detectors), sorted by year, considering the base learners used and the number (#) of base learners. The number of base learners was organized as hyperparameter-based (HP), fixed, and dynamic numbers of learners. The number in parentheses represents the standard value or the explored value in the paper.

4 STDE: SELF-TUNING DRIFT ENSEMBLE

In this section, we describe our proposed approach, the Self-tuning Drift Ensemble (StDE), designed to enhance the predictive performance of drift detectors through a voting-based and windowed ensemble method. The core idea behind StDE is to provide a robust solution that effectively handles various types of drifts in data streams. As illustrated in Figure 2, StDE consists of two main phases: the Warm Start phase and the Online phase. During the Warm Start phase, the approach performs hyperparameter tuning for DDM and EDDM to establish optimal settings. The subsequent Online phase involves dynamically managing the drift detectors based on the evolving data stream behavior, enabling the ensemble to adapt and self-tune in response to different drift types. This dynamic and self-tuning capability ensures that StDE remains effective and adaptable under varying conditions, making it well-suited for diverse drift scenarios.

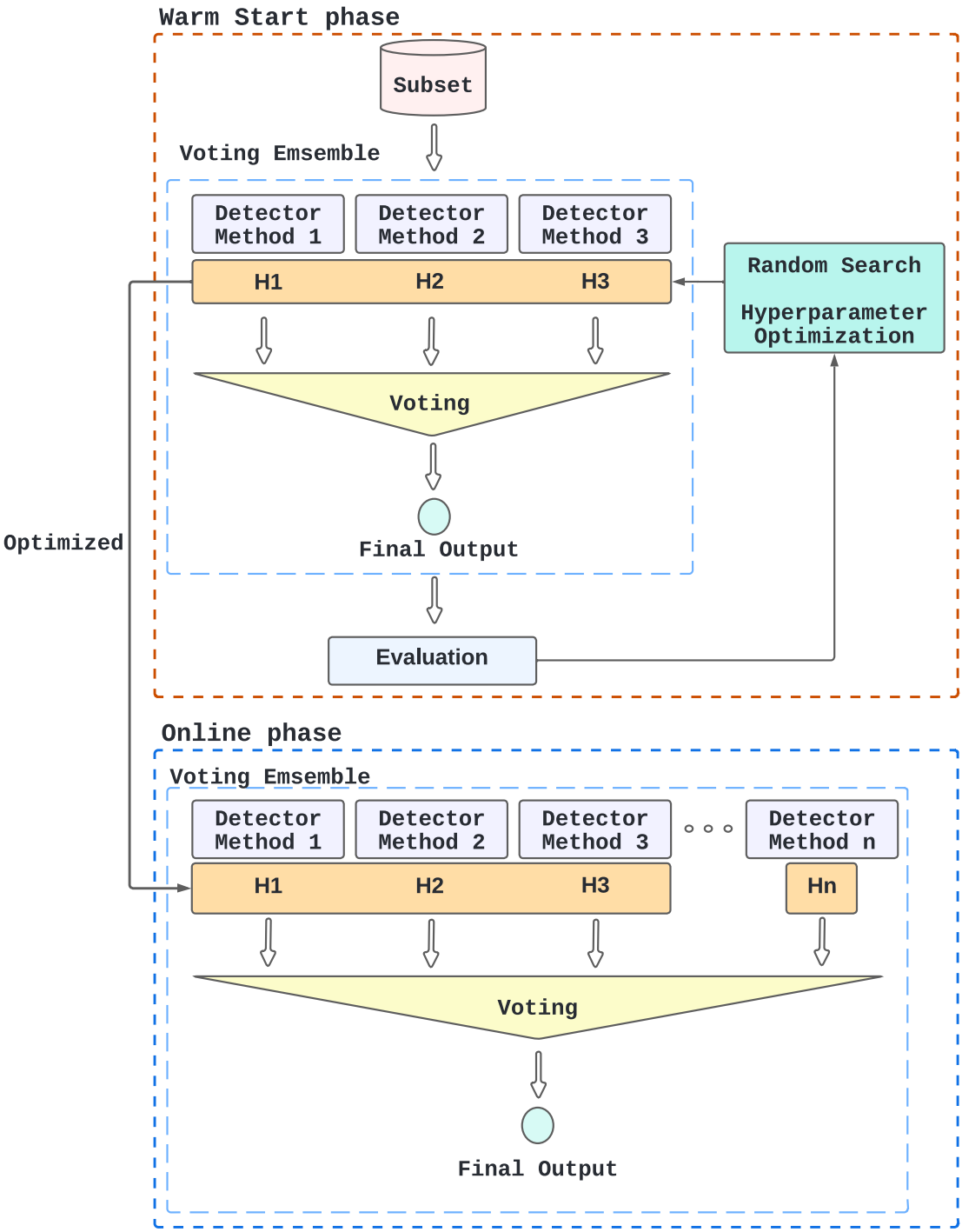

Figure 2 – Self-tuning Drift Ensemble (StDE) overview.

4.1 Voting and Windowing Mechanisms

We propose a voting ensemble method that incorporates multiple instances of DDM and EDDM. Each detector independently monitors the data stream and votes on whether a concept drift has occurred. The final decision is based on the majority vote among these detectors.

In our ensemble algorithm for drift detection, a windowing mechanism is implemented to determine whether successive bits in a data stream represent the same drift or different ones. The size of the window is a parameter that influences the sensitivity and robustness of our drift detection process. A smaller window can detect abrupt changes quickly, enhancing the system's responsiveness to sudden shifts in the data stream. However, this may also increase the likelihood of false positives. Conversely, a larger window provides a more stable and reliable detection environment, reducing false alarms but potentially delaying the recognition of drifts or increasing the number of false negatives.

When the first detector in the ensemble detects a drift, the window is initiated, and all bits with drifts identified within this window are considered to be indicating the same drift. The ensemble uses a voting mechanism, where each detector contributes a vote when it signals a drift. The windowing process helps to aggregate these votes over the specified window size, ensuring that drifts identified by multiple detectors within the window are treated as a consensus on the same drift event. If a majority of detectors within the window agree on the presence of a drift, it is confirmed as a true drift.

Figure 3 shows an example where the window size is 3 bits. Detector 1 initializes the window, and any detector that signals a drift within this window interval is considered to be indicating the same drift point. The votes from these detectors are combined to reinforce the detection. Conversely, if a detector signals a drift outside of this window, it is considered to be indicating another drift point, and a new window and voting process would be initiated.

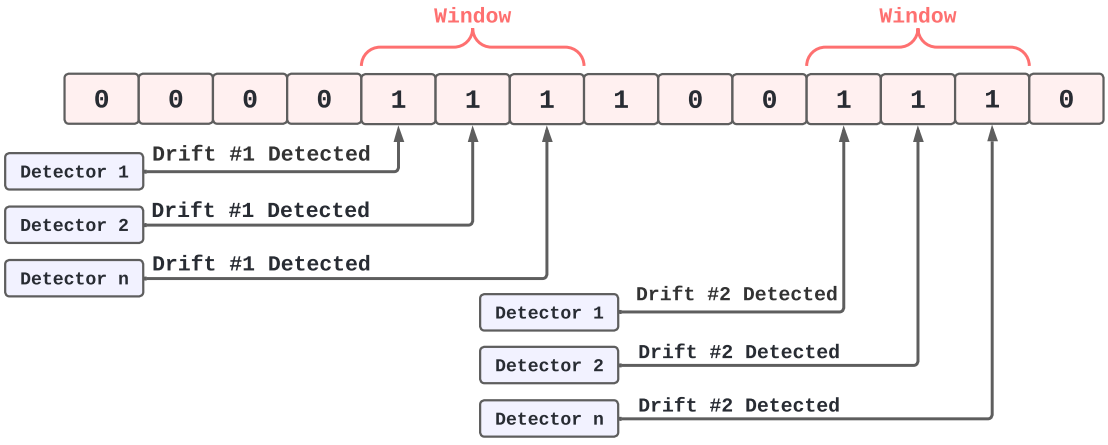

Figure 3 – Windowing mechanism overview.

4.2 Warm Start Phase

The Warm Start phase initializes the ensemble operation by tuning the hyperparameters of the drift detection algorithms (base learners). This offline process utilizes a

finite labeled data stream and requires a minimum of three base learners. This minimum number is essential to effectively implement the majority/minority voting system used in the ensemble. By tuning the algorithms based on this specific subset of data streams, we seek to ensure that the ensemble is robust and well-adapted for the subsequent online learning phase. The Warm Start phase consists of four steps, which are presented as follows:

1. **Hyperparameter Space Definition**: the search space for hyperparameters is defined to explore various configurations for the ensemble components. For both DDM and EDDM, the tuned hyperparameter is the threshold for triggering a warning (α). The value ranges are determined based on the specific characteristics and requirements of the problem at hand.

2. **Hyperparameter Optimization**: this step involves exploring the hyperparameter space to find the optimal settings for the ensemble. Various hyperparameter optimization techniques can be applied here, depending on the specific requirements and constraints of the problem. In our case, we employ a Random Search algorithm, which involves sampling a large number of hyperparameter combinations randomly and evaluating each set using a validation subset of the data streams. Random Search is chosen for its simplicity and effectiveness in finding good hyperparameter settings without the computational burden of exhaustive grid search, but another method for hyperparameter optimization and tuning could be used [34].

3. **Performance Evaluation**: each hyperparameter combination is assessed based on its performance in detecting concept drifts, measured through the F1-score metric.

4. **Selection of Hyperparameter Values**: the combination of hyperparameters that achieves the best performance across the data streams is selected. These parameters are then used to initialize DDM and EDDM instances for the subsequent online phase.

Algorithm 1 implements the Warm Start phase, formalizing the process of iteratively optimizing hyperparameters for a voting ensemble of drift detectors.

Algorithm 1 Warm Start Phase for Self-tuning Drift Ensemble (StDE)

Input:
- D: Initial finite data stream
- M: List of drift detectors
- H: Dictionary of hyperparameter ranges for each detector in M
- n_iter: Number of random hyperparameter configurations

Output:
- $best\_params$: Dictionary of best hyperparameters for each detector in M

begin $best\_params \leftarrow \emptyset$
$best\_score \leftarrow -\infty$
for $i \in [1, \ldots, n\_iter]$ **do**
  $current\_detectors \leftarrow \emptyset$
  for $m \in M$ **do**
    - Generate random hyperparameter configuration h_i for detector m from $H[m]$
    - Configure detector d_i of type m with hyperparameters h_i
    - Add d_i to $current\_detectors$
  end
  - Create hard voting ensemble E_i using $current\_detectors$
  - Evaluate E_i on subset D using F1 Score metric
  - $score_i \leftarrow$ Evaluation metric for E_i
  if $score_i > best\_score$ **then**
    $best\_score \leftarrow score_i$
    $best\_params \leftarrow \{h_i \mid m \in M\}$
  end
end
return $best\_params$

4.3 Online Phase

The Online phase is executed after the Warm Start phase. Its first step is to configure an odd number of detectors, DDM or EDDM, with the hyperparameter values obtained in the previous phase. After configuring the initial detector set, this phase moves on to the data stream monitoring, during which drifts are detected and the number of detectors is managed in real-time. This detector dynamic management relies on two main methods:

1. **Increasing Number of Detectors**: when the ensemble experiences a divided vote scenario, where a simple majority of detectors agree on the presence of a concept drift, this indicates potential uncertainty or the emergence of new patterns in the data stream. To address this and maintain an odd total number of instances, two

additional detectors are added to the ensemble. For our case, these new detectors are either DDM or EDDM, selected randomly. Each new detector is instantiated with hyperparameters perturbed by a Gaussian noise, centered around the optimal values defined in the Warm Start phase. This perturbation introduces variability, enhancing the ensemble's ability to capture and adapt to diverse drift patterns. By introducing controlled randomness into the hyperparameters of newly added drift detectors, the risk of the ensemble becoming overly specialized or biased towards the initial conditions set during the Warm Start phase is mitigated. Additionally, the mean-centered nature of the Gaussian distribution around optimal values ensures that while diversity is introduced, the detectors remain close to their optimal configurations.

2. **Decreasing Number of Detectors**: when there is an unanimity vote scenario, the detectors indicate strong confidence in the presence of a drift, which suggests a stable concept. In this case, the ensemble reduces its complexity by removing two randomly selected detectors. This reduction helps maintain computational efficiency and prevents over-fitting.

Algorithm 2 implements the Online phase to dynamically update the ensemble by changing the number of detectors based on data stream behavior.

Algorithm 2 Online Phase for Self-tuning Drift Ensemble (StDE)

Input:

- *Ensemble*: Set of initialized detectors $D_1, D_2, \ldots, D_n$ with optimal hyperparameters
- Streaming data: Incoming data stream
- k: Threshold for Minority Sequence Count
- m: Threshold for Unanimity Sequence Count

begin

1. **Set** MinorityThreshold $\leftarrow \frac{|\text{Ensemble}|}{2}$
2. **Set** UnanimityThreshold $\leftarrow |\text{Ensemble}|$
3. **Initialize** MinoritySequenceCount $\leftarrow 0$
4. **Initialize** UnanimitySequenceCount $\leftarrow 0$
5. **While** streaming data is available:
   a) Collect votes from all detectors in Ensemble if warning detected
   b) Count votes indicating drift ($driftCount$)
   c) **If** $driftCount <$ MinorityThreshold **then**
      i. MinoritySequenceCount $\leftarrow$ MinoritySequenceCount $+ 1$
      ii. **If** MinoritySequenceCount $= k$ **then**
         - Add two new random detectors $D_{\text{new1}}, D_{\text{new2}}$
         - Perturb hyperparameters of new detectors with Gaussian noise
         - Ensemble $\leftarrow$ Ensemble $\cup\, D_{\text{new1}}, D_{\text{new2}}$
         - Reset MinoritySequenceCount to 0
      iii. Reset UnanimitySequenceCount to 0
   d) **Else**
      i. **StDE detects the drift**
      ii. Reset MinoritySequenceCount to 0
      iii. **If** $driftCount =$ UnanimityThreshold **then**
         A. UnanimitySequenceCount $\leftarrow$ UnanimitySequenceCount $+ 1$
         B. **If** UnanimitySequenceCount $= m$ **then**
            - Remove two random detectors $D_{\text{remove1}}, D_{\text{remove2}}$
            - Ensemble $\leftarrow$ Ensemble $\setminus\, D_{\text{remove1}}, D_{\text{remove2}}$
            - Reset UnanimitySequenceCount to 0
   e) Update detectors in Ensemble with new data

end

5 EXPERIMENTAL EVALUATION

This section outlines the experimental framework used to evaluate our proposed methods and compare various drift detection algorithms. We begin by introducing the automatic stream generator, which was used for producing the synthetic data streams in both the benchmarking and ensemble tests. This generator ensured that the data streams were consistent and controlled across experiments. Next, we describe the evaluation metrics applied to assess the performance of the algorithms, including detection accuracy, detection time, and detection delay. After detailing the metrics, we present the results from the benchmarking phase, where we evaluate the performance of four drift detection algorithms—DDM, EDDM, HDDMA, and HDDMW—implemented using the River Framework version 0.19.0 [35]. These algorithms were assessed across various drift types.

Finally, we detail the ensemble tests, focusing on the performance of the Self-tuning Drift Ensemble (StDE) and its ability to adapt to varying drift scenarios. This structured approach provides a comprehensive view of how each method performs and their effectiveness in different contexts.

5.1 Automatic Generation of Streams

To conduct the experiments, we developed a synthetic data stream dataset generator capable of creating binary streams driven by multiple variables to simulate a wide range of scenarios. The tool, publicly available[1], enables the definition of variables such as drift type (abrupt, gradual, or incremental), data stream size, sample range, dataset size, percentage of maximum duration, and drift divisions or amplitude. The start and end positions of the drift are randomly generated for each sample of the data stream dataset.

Developed in Python 3.8.8 using the open-source Streamlit framework [36] version 1.12.2 , the tool provides an intuitive user interface that facilitates the selection of the necessary parameters for creating data streams as desired. Additionally, it allows for the generation of homogeneous data streams, with drifts of only one type, or heterogeneous data streams, with multiple different drift types.

It is important to clarify that different data streams were used for the validation of the benchmark algorithms and the StDE's validation. In the benchmarking phase, data streams with homogeneous characteristics were prioritized, focusing on generating specific types of drift, such as abrupt, gradual, and incremental, to facilitate a direct and controlled comparison between the baseline algorithms. On the other hand, for the validation of StDE, heterogeneous data streams were created, designed to simulate more complex

[1] Available on <https://github.com/gysakurai/datastream-synthetic>

and diverse scenarios. This distinction ensures that each approach is tested under conditions best suited to evaluate its performance, with the homogeneous streams providing a clear benchmark for comparison, and the heterogeneous streams reflecting the broader adaptability of the ensemble method.

All streams were generated with drifts characterized by pseudo-random properties. Both the initiation and termination of each drift were randomly determined within the stream, adhering to specific attribute intervals.

For streams with homogeneous drifts, the following parameters were established:

Table 2 – Parameters for Homogeneous Drifts

Drift Type	**Stream Size**	**Duration**	**Division**
Abrupt	100 to 10,000 bits	1 to 15% of drift size	5 to 15 divisions
Gradual	100 to 10,000 bits	1 to 5% of drift size	5 to 10 divisions
Incremental	100 to 10,000 bits	1 to 20% of drift size	N/A

In the case of streams with heterogeneous drifts, the mentioned characteristics and intervals remain applicable. However, a key distinction is the inherent randomness involved in the selection of drifts for each stream. This variability ensures that the drift characteristics are not only diverse but also reflective of real-world scenarios, where data distributions can change unpredictably.

We generated 1800 data streams — 900 homogeneous and 900 heterogeneous. Homogeneous streams each contain 300 samples of a single drift type, while heterogeneous streams mix all three drift types in varying sequences. Stream sizes and drift locations were randomly defined to introduce variability and ensure unbiased algorithm evaluation.

5.2 Evaluation Metrics

The drift detectors' performances were assessed according to four main criteria: detection performance, processing time, detection delay and memory cost to detect drifts. In this section, we provide further details on how the metrics for these criteria were calculated.

5.2.1 Detection Performance

F1 score was applied to evaluate the detection performance of all change detection algorithms. F1 score, as presented in Equations (5.1)–(5.3), is derived from two other metrics, namely precision and recall [37]. Precision measures the ability of the assessed algorithm to avoid false positives, while recall expresses its capacity to detect as many drifts as possible. By combining these two metrics, F1 score provides a broader picture of the algorithm's accuracy.

In this context, true positives (TP) correspond to the number of correctly detected drifts, whereas false positives (FP) represent instances where the algorithm incorrectly signals a drift that did not actually occur. Conversely, false negatives (FN) denote actual drifts that the algorithm failed to detect. Using these definitions, the precision, recall, and F1 score are computed as follows:

$$Precision = \frac{TP}{TP + FP} \tag{5.1}$$

$$Recall = \frac{TP}{TP + FN} \tag{5.2}$$

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{5.3}$$

5.2.2 Processing Time

The notion of processing time plays a critical role in the context of data stream drift detection since it directly impacts the ability of a detector to process incoming data streams in real-time. Applications requiring the timely detection of concept drifts need the use of fast and efficient detectors. The processing time for a data stream drift detector corresponds to the time required to process each incoming instance, and it is influenced by various factors, such as data complexity, data stream size, detector implementation specifics, and available computational resources. Given the trade-off between accuracy and processing time, careful consideration of both factors is vital in selecting and optimizing the data stream drift detectors. Therefore, an evaluation of the processing time of detectors is essential to ensure their ability to operate in real-time and fulfill the requirements of the given application. For our studies, the processing time of each algorithm was measured by assessing the amount of time it takes for the algorithm to process all the bits of each data stream.

5.2.3 Detection Delay

Detection delay is another metric related to time that must be assessed for drift detectors. This metric refers to the time interval between the occurrence of a concept drift and its detection by the detector. Drift detectors with low detection delays enable applications to make decisions about detected changes quickly, allowing them to meet real-time constraints. On the other hand, a high detection delay may result in missed opportunities, false alarms, or incorrect decisions, which can have serious consequences in critical applications. Therefore, the detection delay of data stream drift detectors must be evaluated in addition to their accuracy and processing time, to ensure that they can detect concept drifts in a timely and reliable manner. The detection delay depends on various

factors, such as the complexity of the data, the frequency and magnitude of the drifts, the performance of the detector and the decision threshold used. The trade-off between the detection delay and accuracy is an important consideration in the selection and optimization of data stream drift detectors and may depend on the specific requirements of the intended application.

In Figure 4, the calculation of the detection delay is illustrated as the difference between the exact onset point of each drift (true detection) and the point at which it was detected by the algorithm, as described in the work by Asghari et al. [38]. Notably, when detection occurs outside of the drift area, it is disregarded and excluded from any calculations related to detection delay metrics.

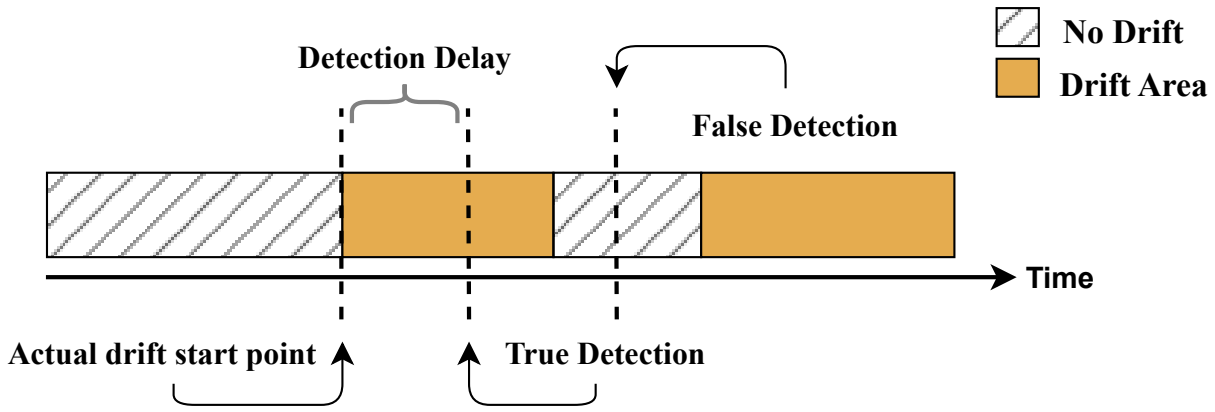

Figure 4 – Visualization of data drift over time: Detection Delay, True and False Detections, and Drift Area.

5.2.4 Memory Cost

The memory cost in drift detection algorithms refers to the amount of memory allocated and utilized during their execution. This metric helps assess the algorithm's efficiency and scalability, especially in resource-limited environments. It includes average memory usage, reflecting typical memory footprint. Efficient memory management is relevant for handling large data streams and adapting to changes without causing memory overflow or slowdowns, ensuring real-time performance and reliability. In our study, we use Python 3.10 and the tracemalloc library[2] to measure this memory cost, providing precise tracking of memory usage over time. We report memory usage in megabytes (MB) for comparisons.

[2] Available on <https://docs.python.org/3/library/tracemalloc.html>

5.3 Benchmarking Change Detector Algorithms

In this study, we conducted a benchmarking analysis of four drift detection algorithms—DDM, EDDM, HDDMA, and HDDMW—using a synthetic data stream designed to evaluate their performance in detecting various types of concept drifts: abrupt, gradual, and incremental. The primary goal of this research is to assess the effectiveness of these algorithms across different drift scenarios and to determine whether certain algorithms are better suited for specific types of drift. By evaluating their performance in terms of detection accuracy, detection time, and detection delay, our findings aim to provide valuable insights into which algorithms are most effective for each drift type, thereby guiding the selection of the most appropriate algorithm based on the characteristics of the drift encountered in practice.

To ensure a comprehensive evaluation, we employed consistent performance metrics across all detectors, including the F1 score, processing time, and detection delay. The synthetic data streams were randomly generated with three types of concept drifts: abrupt, gradual, and incremental. This rigorous comparison allows us to fairly assess and compare the accuracy and speed of each detection method.

5.3.1 Detection Performance

The initial results presented in this study focus on the detection of abrupt drifts. To thoroughly assess the performance of the drift detection algorithms, we conducted experiments by varying the number of drifts present within the data streams. Specifically, we generated multiple synthetic data streams with a homogeneous distribution, introducing different quantities of abrupt drifts ranging from 1 to 5 per stream. This approach allowed us to evaluate how the algorithms respond to varying levels of abrupt changes in the data, simulating scenarios that could occur in real-world applications.

Figure 5 displays the F1 score results across the different quantities of abrupt drifts. As shown in the figure, all algorithms achieved their maximum performance, with equivalent F1 scores when only one abrupt drift was introduced. However, as the number of drifts increased to 2, the HDDMW, HDDMA, and DDM algorithms consistently displayed reduced F1 scores, while the EDDM algorithm yielded a markedly inferior score. This trend of decreasing F1 scores continued as the number of drifts increased, highlighting the challenges these algorithms face in handling multiple abrupt drifts within a single data stream. Notably, the HDDMW algorithm maintained the most consistent average F1 score across different drift quantities, whereas the EDDM algorithm produced the lowest average score.

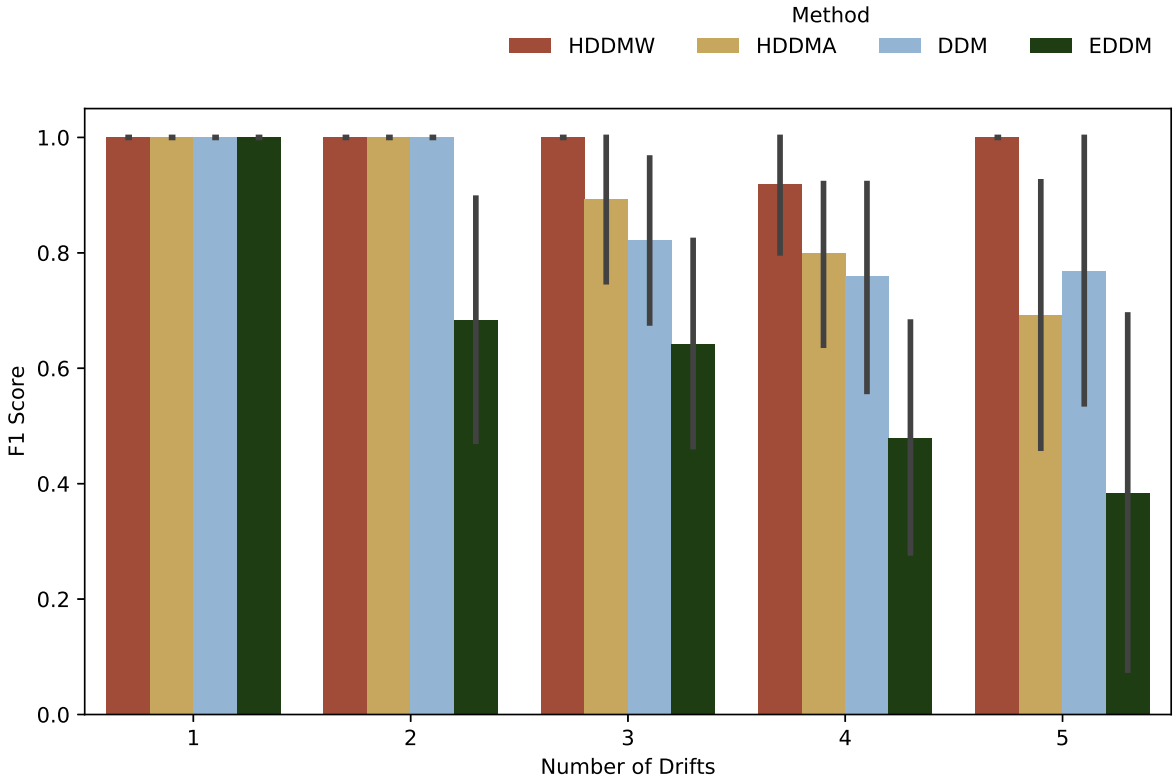

Figure 5 – F1 score of detection methods when dealing with abrupt drifts.

For incremental drifts, the experimental setup was similar, but with only one drift per stream, given the nature of incremental changes that gradually evolve over time. As shown in Figure 6, only the EDDM algorithm achieved a satisfactory result, with an F1 score close to the maximum possible. In contrast, HDDMW had the second-best performance, though significantly lower than EDDM, while HDDMA and DDM yielded much lower and almost negligible scores. This setup allowed us to assess the algorithms' ability to detect subtle, incremental changes, which are often more challenging to identify than abrupt shifts.

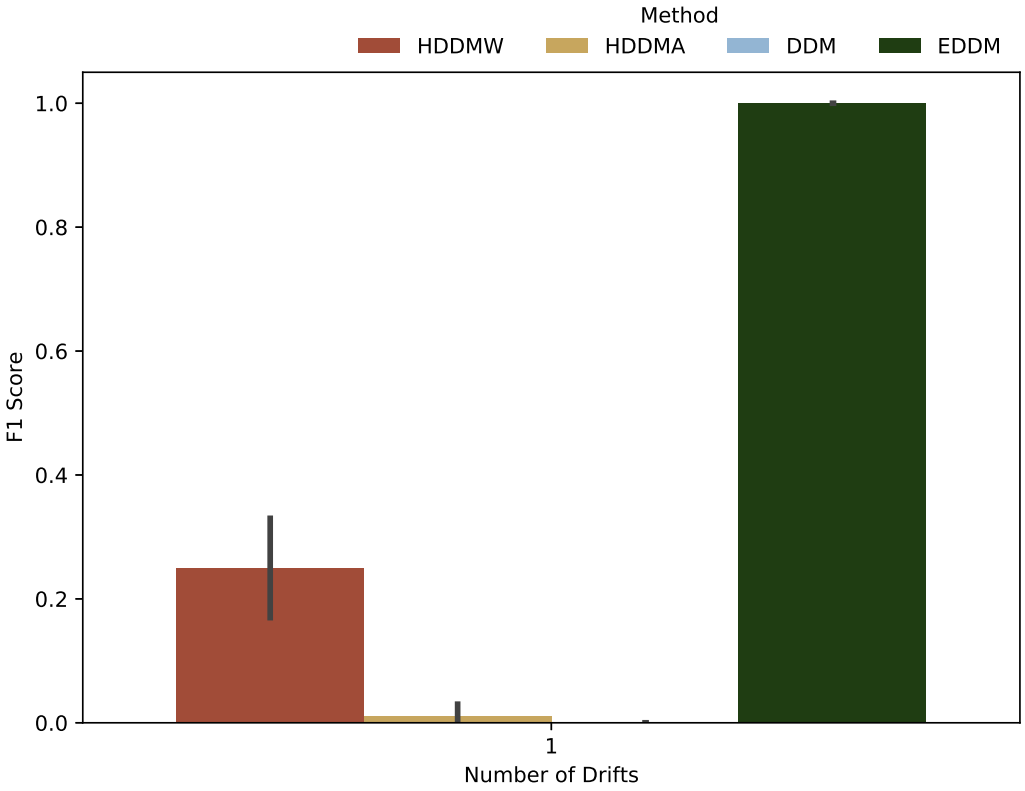

Figure 6 – F1 score of detection methods when dealing with incremental drifts.

The detection performance for gradual drifts, where the change occurs over a more extended period, is presented in Figure 7. For this scenario, we also varied the number of drifts per stream, similar to the abrupt drift experiments. When the number of drifts was 1 or 2, all algorithms showed identical performance, with F1 scores close to 1, indicating high detection accuracy. However, as the number of gradual drifts increased to 3 or 4, the HDDMW, DDM, and EDDM algorithms maintained their performance levels, while the HDDMA algorithm exhibited a decline in its average F1 score. When five drifts were introduced, all four algorithms achieved comparable performance levels, with F1 scores around 0.8.

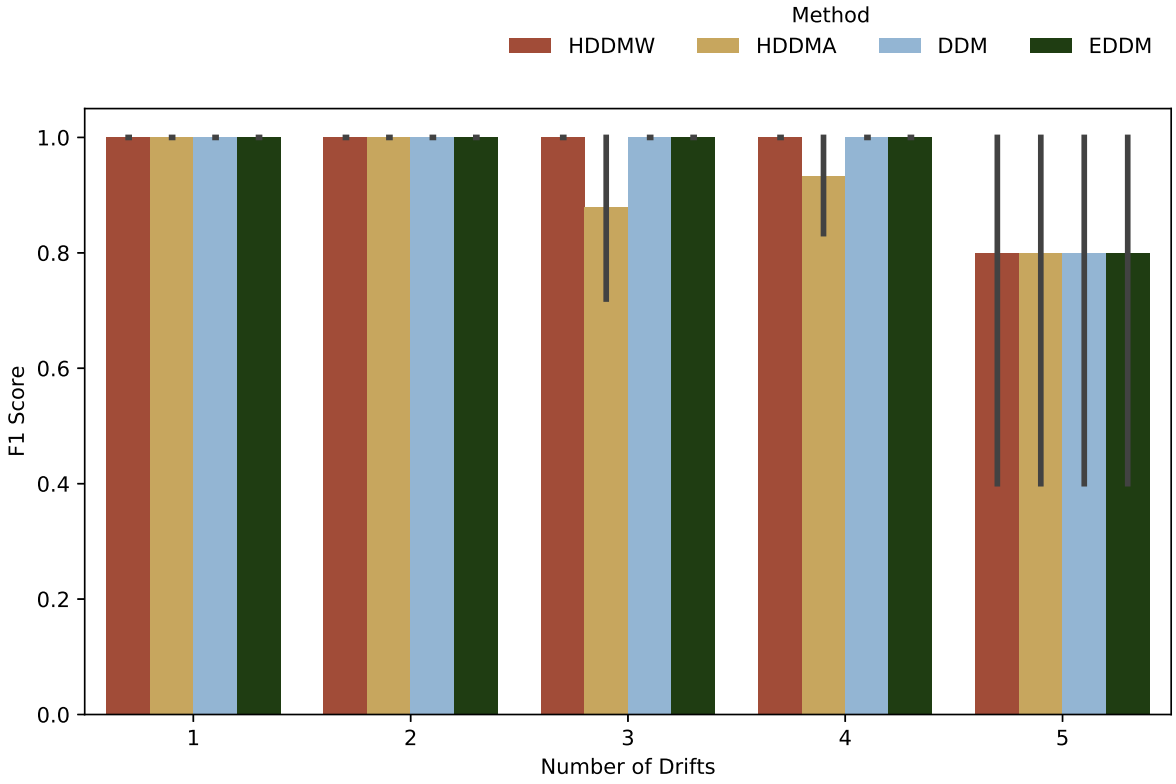

Figure 7 – F1 score of detection methods when dealing with gradual drifts.

Overall, the results demonstrate that while the HDDMW, DDM, and EDDM algorithms performed similarly across various scenarios, the HDDMA algorithm tended to perform less favorably, particularly as the complexity of the drift patterns increased. The experimental design, with varying drift quantities and types across homogeneous data streams, provides a comprehensive evaluation of each algorithm's robustness and adaptability in handling different drift scenarios.

We evaluate the drift detection performance in terms of F1 score based on a statistical analysis grounded on the non-parametric Friedman statistical test [39] to determine any significant differences among EDDM, DDM, HDDMW, and HDDMA with 300 streams. Following the Friedman test, we performed a Nemenyi post hoc test [40] with a significance level of 0.05. The null hypothesis posits that the drift detectors' performances are comparable based on the average F1 score per stream. If the null hypothesis is rejected, the Nemenyi post hoc test can be used. This test establishes that the performance of two detectors is significantly distinct if the average ranks differ by at least a critical difference (CD) value. Figure 8 shows the CD diagram for F1 score. Particularly, we assume that there are no significant differences between HDDMA and DDM. All other differences are significant. In other words, EDDM was statistically the most accurate method, followed by HDDMW. HDDMA and DDM are statistically similar.

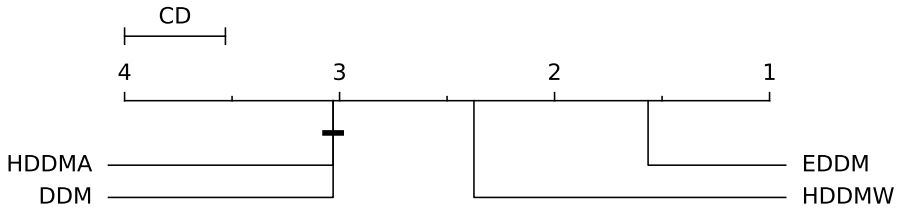

Figure 8 – Critical distance diagram using the CD of 0.469 based on the Nemenyi post hoc test considering the F1 score of four drift detection methods (EDDM, DDM, HDDMW and HDDMA) with 300 paired streams.

5.3.2 Processing Time

Regarding the algorithms' processing time performance, our experiments revealed that the DDM algorithm achieved a notably lower processing time (y axis) compared to the others, overall. The drift detection methods based on the Hoeffding algorithms exhibited similar behavior, albeit with a higher processing time. In contrast, the EDDM algorithm displayed a marked distance from the other algorithms, with the results varying considerably throughout the analysis of each sample (identified on the x axis), as shown in Figure 9.

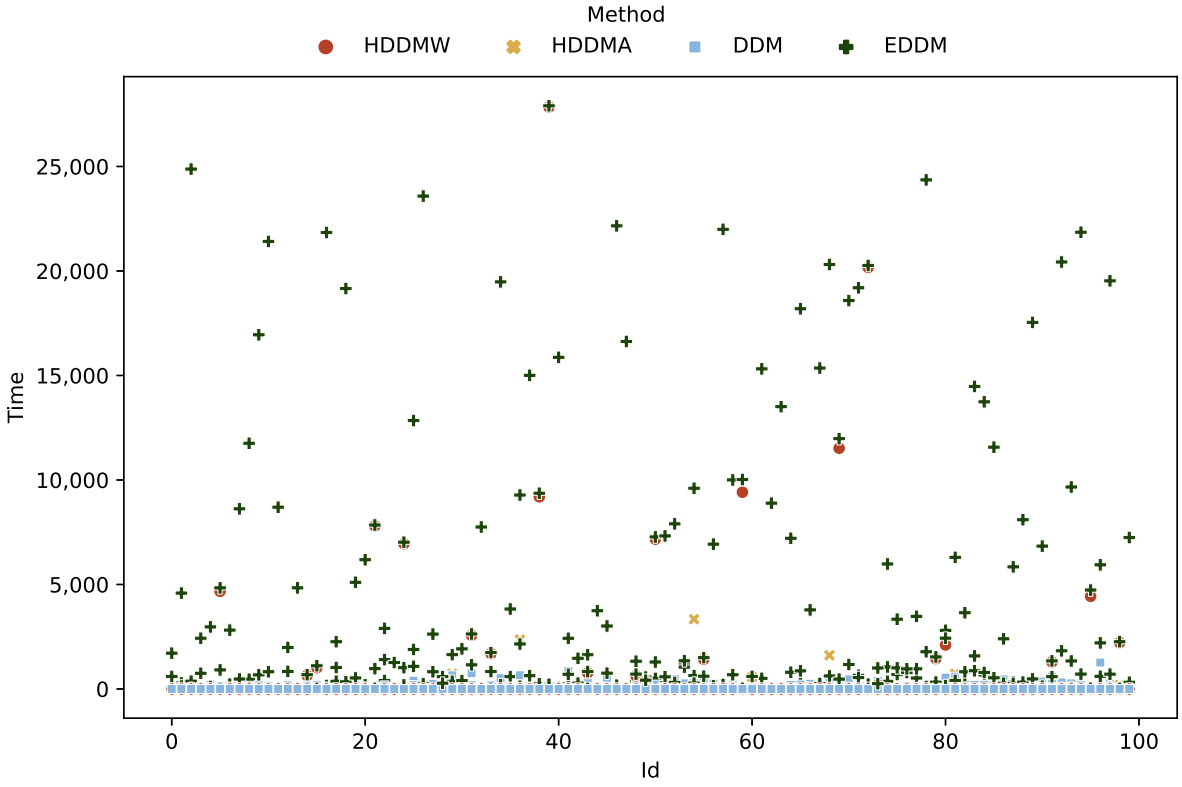

Figure 9 – Time of processing when dealing with different drifts.

We made the same evaluation process as when assessing the detection performance to evaluate the processing time using the non-parametric Friedman statistical test to identify any significant differences among EDDM, DDM, HDDMW, and HDDMA with 300 streams. Figure 10 presents the CD diagram for detection time. According to the statistical test, there are no statistically significant differences between only HDDMA and HDDMW. Thus, we can infer that these two detectors are the fastest in comparison to the others. When selecting a drift detector, both processing time and detection performance should be considered. HDDMW is a better option than HDDMA, DDM, and EDDM since it significantly reduces the detection time while maintaining adequate predictive performance.

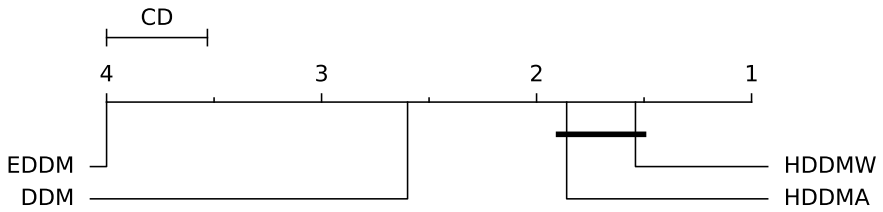

Figure 10 – Critical distance diagram using CD of 0.469 based on the Nemenyi post hoc test considering the time of four drift detection methods (EDDM, DDM, HDDMW, and HDDMA) with 300 paired streams.

5.3.3 Detection Delay

Figure 11 presents the results for detection delay in seconds in the logarithm scale. The HDDMW algorithm had the best performance for abrupt drifts, detecting drifts in a much shorter time than the others. HDDMW is followed by HDDMA, DDM, and EDDM, respectively, but with similar times among them. For gradual drifts, HDDMW showed again a significantly shorter time than the other algorithms, followed by HDDMA, DDM, and EDDM. For incremental drifts, the HDDMA algorithm had the best detection delay performance, followed by HDDMW and EDDM with markedly different results. In none of our experiment's samples, the DDM algorithm was able to detect an incremental drift, so there were no delay results.

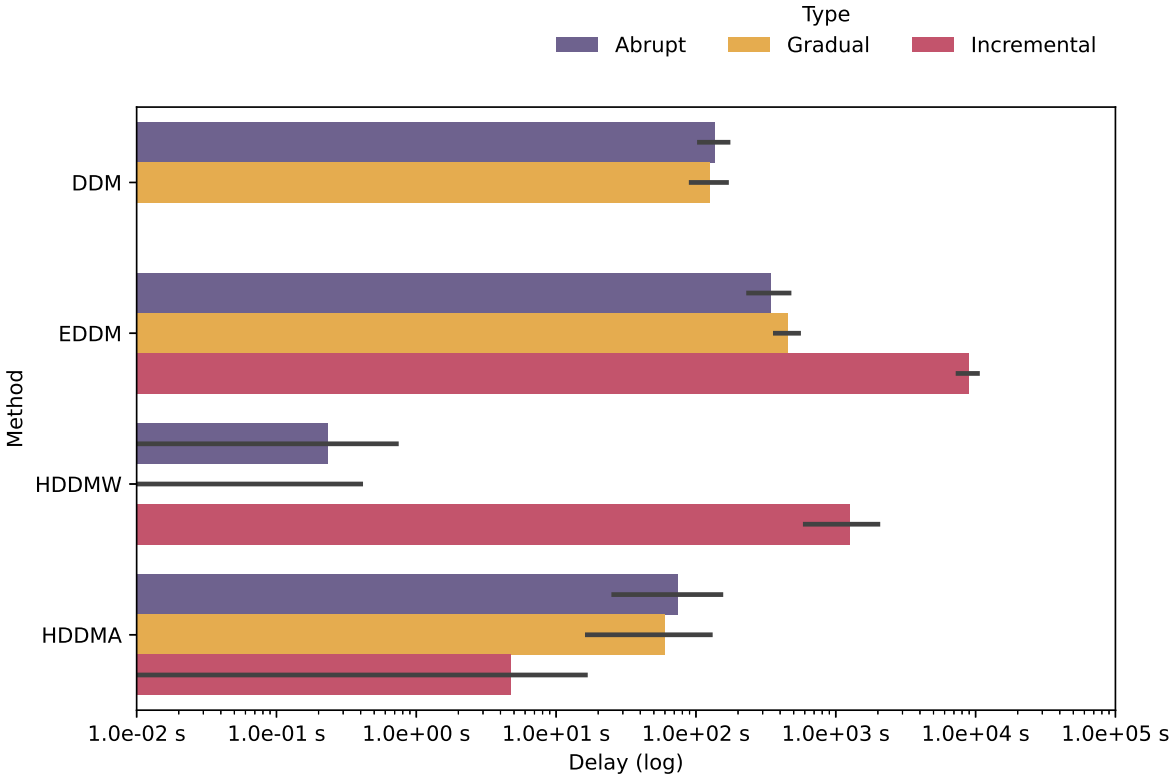

Figure 11 – Delay in logarithmic scale of seconds.

5.3.4 Memory Cost

Figure 12 presents the results for memory cost in Megabytes, shown on a logarithmic scale. For abrupt drifts, HDDMA proved to be the least memory-intensive. Gradual drifts were more costly to detect compared to other types of drift, with EDDM performing slightly better than the other algorithms. When dealing with incremental drifts, the results across all algorithms were quite similar, but once again, EDDM demonstrated superior performance compared to the other contenders.

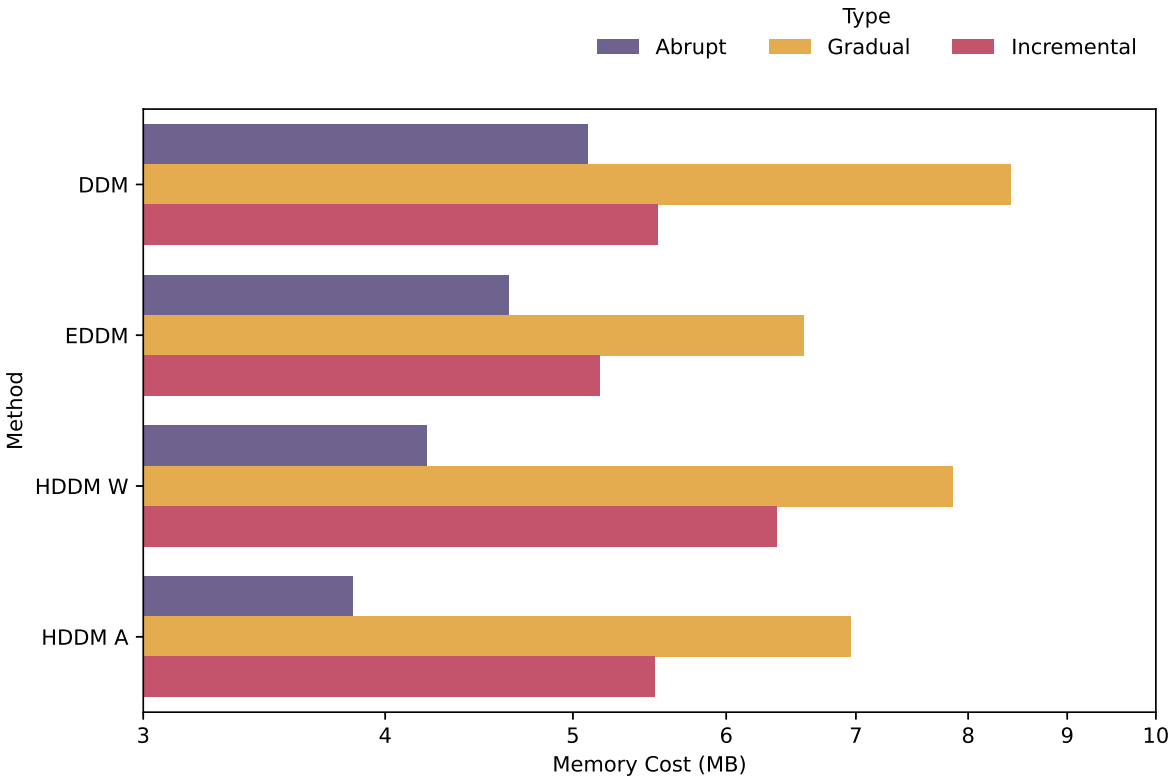

Figure 12 – Memory Cost logarithmic scale of Megabytes

5.3.5 Discussion

Incremental drifts proved to be the most difficult to simulate and validate. The creation of multiple samples with various random variables with our tool helped us find the best detector for this type of drift. Other works in the literature, such as those by Gonçalves Jr. et al. [26], Poenaru-Olaru et al. [25], and Barros and Santos [22] did not consider incremental drift in their analyses.

Our experiments have also shown that there is a significant trade-off among drift detection performance, detection time, and detection delay for the analyzed algorithms. Upon the analysis of the DDM algorithm, it demonstrated commendable proficiency in detecting abrupt and gradual drifts, as evidenced by its noteworthy average algorithm detection time. However, it was observed that the algorithm suffers from a relatively suboptimal detection delay, ranking as the second slowest algorithm in this regard. Notably, the algorithm's most pronounced limitation lies in its capacity to identify incremental drifts, which were unable to successfully detect during our experimental investigation.

In contrast to other algorithms, the EDDM algorithm excels in detecting incremental drifts, with a better F1 score performance than the other algorithms. However, it has a weakness in time-related criteria, such as a longer processing time and detection

delay. The algorithm encounters challenges in detecting abrupt drifts when the frequency of drifts increases in comparison to other algorithms. Additionally, EDDM has a significant limitation in terms of delay, generally across all types of drifts, where it exhibits a considerable lag in detecting drifts compared to other algorithms.

The HDDMW algorithm has demonstrated strong consistency in detecting abrupt drifts, even as their frequency increases. However, its detection time performance is less effective, ranking as the second slowest among the evaluated methods. Additionally, the algorithm faces challenges in detecting incremental drifts, as reflected in its reduced detection performance and increased delay. Conversely, its greatest strength lies in achieving the best detection delay for both abrupt and gradual drifts, surpassing all other algorithms in this aspect.

The HDDMA algorithm demonstrated slightly lower performance in handling gradual drifts when the frequency increases. However, it maintains a balanced processing time and has a better detection delay compared to the DDM and EDDM algorithms, only being outperformed by HDDMW.

Finally, our experiments have shown that choosing the best algorithm heavily depends on the objective and weaknesses we want to avoid, our prior knowledge of the data stream that will be subjected to drift detection analysis, and the type of drift. Throughout our studies, we were able to analyze the behavior of some of the most famous concept drift algorithms in detecting different types of drift, as well as their detection time and delay. Additionally, we were able to ensure replicability by varying several parameters in the construction of each type of drift using the tool developed for synthetic datasets.

Our findings have important implications for understanding data stream behavior. The ability to detect drifts in real-time allows for adjustments in strategies to better align with the evolving nature of these streams. However, it is crucial to recognize that the performance of drift detection algorithms can vary depending on the type of drift encountered. This variability can complicate the management of data streams that exhibit heterogeneous drifts, where different algorithms may perform better under specific conditions. Also, the results from this phase largely confirmed what we expected based on findings from the existing literature. Specifically, the strengths of certain algorithms, such as the superior performance of EDDM in gradual and incremental drifts, aligned with previously reported outcomes in related work.

Furthermore, the use of drift detection algorithms can enhance the overall accuracy and reliability of machine learning models in these settings. In future research endeavors, it would be advantageous to explore more sophisticated drift detection algorithms and techniques, including others methods and deep learning-based approaches. Nonetheless, it is important to acknowledge that while deep learning-based techniques may yield promising results in detecting concept drifts across multiple domains, their efficacy is frequently

constrained by the requirement for large volumes of labeled data and computational resources. Moreover, the black-box nature of these methods can impede the interpretation and understanding of their predictions, thereby restricting their utility in domains where interpretability is essential. Therefore, efforts to enhance the precision and efficiency of drift detection in both synthetic and real-world data streams should take these limitations into account when assessing and selecting the appropriate techniques. Additionally, the use of more diverse and larger datasets would help increase the quality of the findings, making them applicable to a broader range of applications and industries.

5.4 StDE's Evaluation

This subsection presents the results of the evaluation of our proposed Self-tuning Drift Ensemble (StDE) method. To thoroughly assess its performance, we compared StDE with multiple baseline methods to understand how it stands up to established techniques in various scenarios. One of the key advantages of StDE is its flexibility, as it allows the integration of any drift detection algorithm within the ensemble framework. In our evaluation, we employed DDM and EDDM as base learners within StDE. These algorithms were selected due to their simplicity, which facilitates better hyperparameter control, and because they share a similar nature, ensuring a more systematic assessment of our approach.

The comparative analysis includes:

- A Drift Detection Method (DDM) instance with the optimal hyperparameters identified during the Warm Start Phase.
- An Early Drift Detection Method (EDDM) instance, also optimized with hyperparameters determined during the Warm Start Phase.
- A modified version of our proposed ensemble method, referred to as Fixed-StDE. This variant maintains a fixed number of base learners throughout the stream, equivalent to the maximum number of detectors utilized by StDE during its online phase for the same data stream.
- The LE3D framework [29], a fixed ensemble of three base learners. It integrates ADWIN, KSWIN, and Page Hinkley algorithms for drift detection. This framework was selected for comparison due to its ensemble structure, which is similar to the one we are proposing, and its use of base learners that are well-established in the literature.

Additionally, the benchmarks were tested across both homogeneous and heterogeneous data streams to evaluate model performance under different data environments and

drift behaviors. For the testing of StDE, we employed the following parameters: Threshold for Minority Sequence Count = 1000, Threshold for Unanimity Sequence Count = 5, and Window Size = 5. These values were chosen based on preliminary tests that indicated they provided the best performance.

In the following sections, we will delve into the results and discuss how StDE and the other algorithms performed across these conditions.

5.4.1 Detection Performance

For detection performance in homogeneous data streams, StDE substantially outperforms the baselines and achieves the highest F1 score, indicating superior performance in handling drifts of the same type within a stream, as shown in Figure 13. StDE's F1-score exceeded Fixed-StDE's by 20 percentage points, which suggests that dynamically adjusting the number of detectors can be more effective than merely maximizing their quantity in this scenario. Additionally, StDE demonstrated significantly better performance compared to the LE3D, which failed to perform well in our validations.

When it comes to heterogeneous data streams, StDE also outperformed the baselines. DDM and Fixed-StDE also achieved satisfactory results. Conversely, EDDM produced significantly poor results, as shown in Figure 13. This outcome indicates that the balanced integration of different methods within the proposed StDE method, along with the hyperparameters tuning — making some detectors more sensitive and others less sensitive — results in superior detection performance even when multiple drift types are present in the same stream. By carefully adjusting these hyperparameters, we are able to enhance the overall performance, achieving more satisfactory results compared to the other methods.

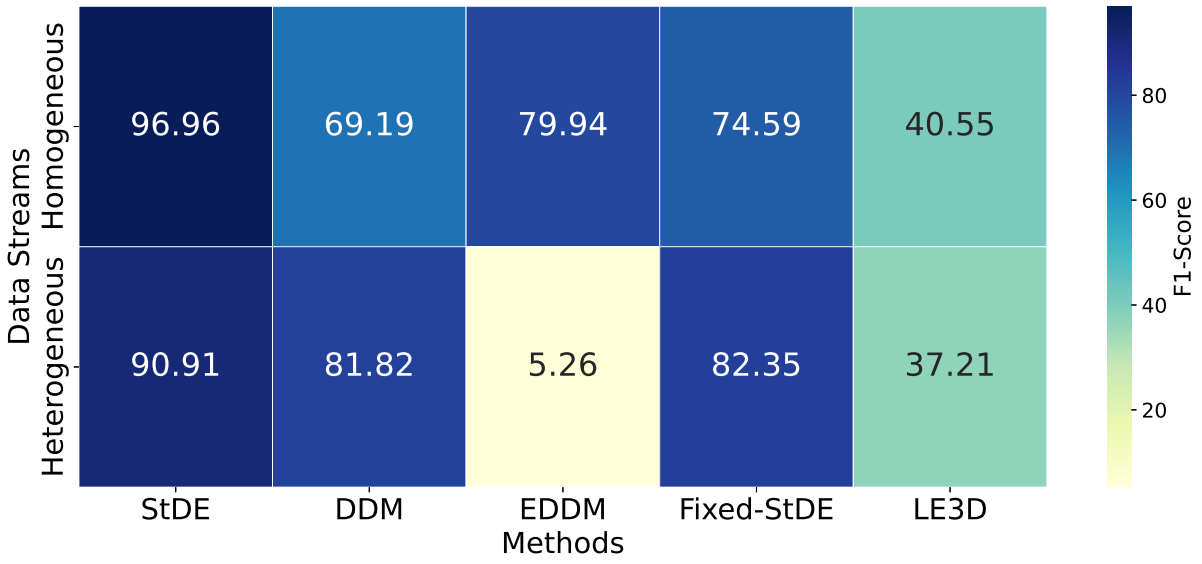

Figure 13 – Heat-map of F1 score in detection methods

5.4.2 Processing Time

In terms of processing time, StDE tends to exhibit significantly higher times compared to the baseline methods DDM and EDDM as shown in Figure 14. This is primarily due to the fact that StDE is an ensemble method that can aggregate multiple instances of the baseline detectors, resulting in a proportional scaling of the processing time. It is also important to highlight that, when using the Fixed-StDE approach, which limits the number of detectors to a fixed maximum, the processing time increases considerably compared to the more flexible StDE approach. Furthermore, compared to LE3D, StDE achieved a shorter processing time even when LE3D operated with a fixed number of detectors. This is likely due to the fact that LE3D is an ensemble method with a computationally more demanding base of detectors.

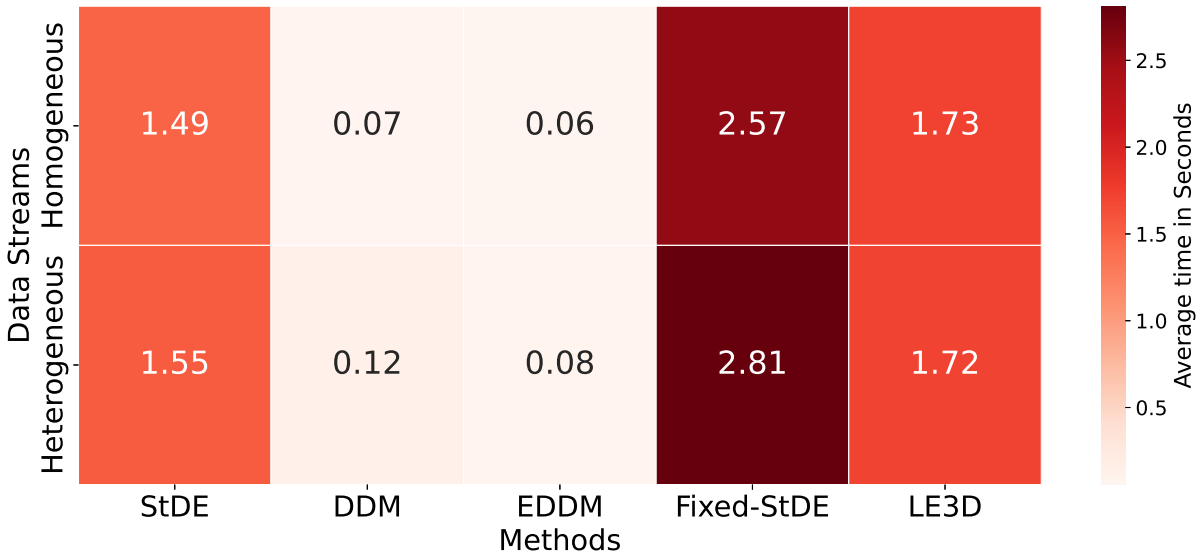

Figure 14 – Heat-map of Processing time in detection methods

5.4.3 Detection Delay

For detection delay in homogeneous data streams, StDE performed slightly better than DDM and Fixed-StDE, as shown in Figure 15. Moreover, StDE significantly outperformed LE3D in terms of delay. Conversely, EDDM yielded significantly worse results. For detection delay in heterogeneous data streams, StDE is also slightly better than DDM and Fixed-StDE, with StDE exhibiting a higher delay when compared to homogeneous streams, though. LE3D, however, achieved better results for heterogeneous delay, attaining a reasonably lower delay than StDE. EDDM presented a very rapid reaction, as observed in Figure 15, but performed poorly in terms of predictive capacity.

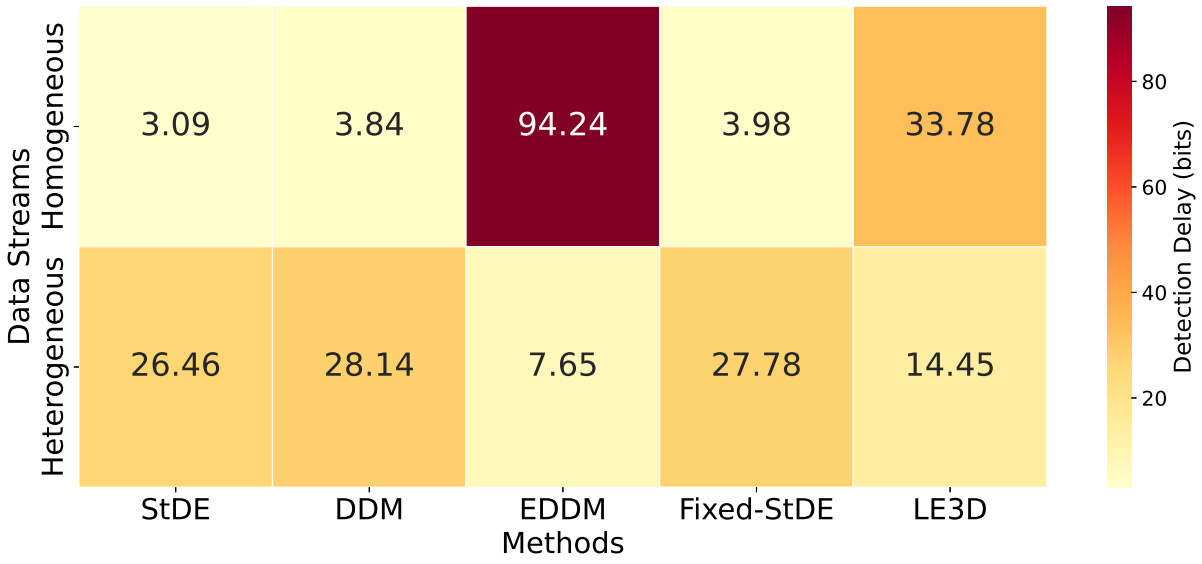

Figure 15 – Heat-map of delay in detection methods

5.4.4 Memory Cost

Figure 16 presents the results for this metric. As expected, StDE required more memory than the baselines based on DDM and EDDM. StDE utilizes a minimum of three base learner instances but only consumes up to twice the memory of a single base learner (DDM or EDDM), which can be deemed satisfactory. Another positive aspect here is that StDE's memory consumption is much lower than that of Fixed-StDE. This demonstrates that the dynamic adjustment of the number of base learners allowed for good predictive performance alongside a significant reduction in memory footprint.

Importantly, it should also be noted that StDE exhibited lower memory consumption compared to the LE3D framework. This can be attributed to the same reason for its superior performance in terms of processing time: the base detectors used in the LE3D ensemble demand more computational resources than those employed in StDE.

Although the memory costs observed in these comparisons represent a negligible amount for modern computers and personal devices, it is worth noting that these algorithms may be deployed in environments where memory availability is restricted or limited. In such cases, even small differences in memory consumption could have a significant impact.

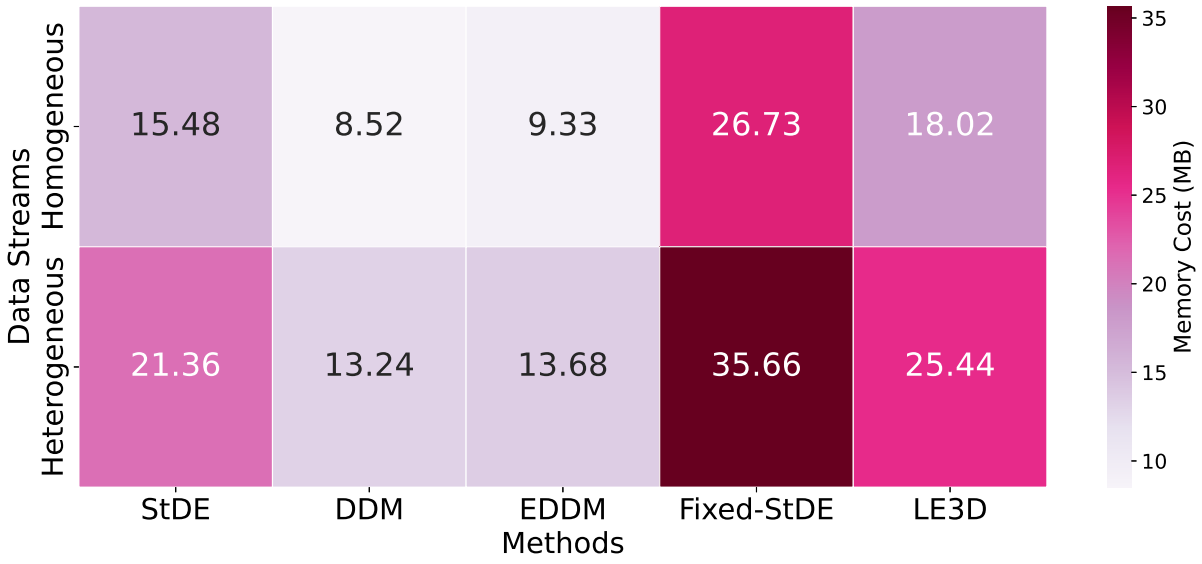

Figure 16 – Heat-map of Memory Cost in detection methods.

5.4.5 Number of Detectors Fluctuation

In this section, we provide an illustrative example of how StDE adapts to changes in the data stream. As presented in Figure 17, we use this specific case to demonstrate the internal workings of the ensemble and its ability to adjust to varying conditions. The blue line represents the number of detectors over time, while the red dashed lines indicate the points at which potential drifts were signaled by the detectors. The actual drifts in the data stream are highlighted by gray intervals.

Initially, the number of detectors is set at three. As potential drifts are detected, the number of detectors increases or decreases based on the voting results. For instance, at the point where 2 out of 5 detectors signal a potential drift and the threshold for minority votes is reached, the number of detectors increases. Conversely, when 9 out of 9 detectors vote unanimously and the threshold for unanimous votes is reached, the number of detectors decreases. This dynamic adjustment reflects the ensemble's adaptive response to changes in the data distribution, showcasing its capability to handle different drift scenarios effectively.

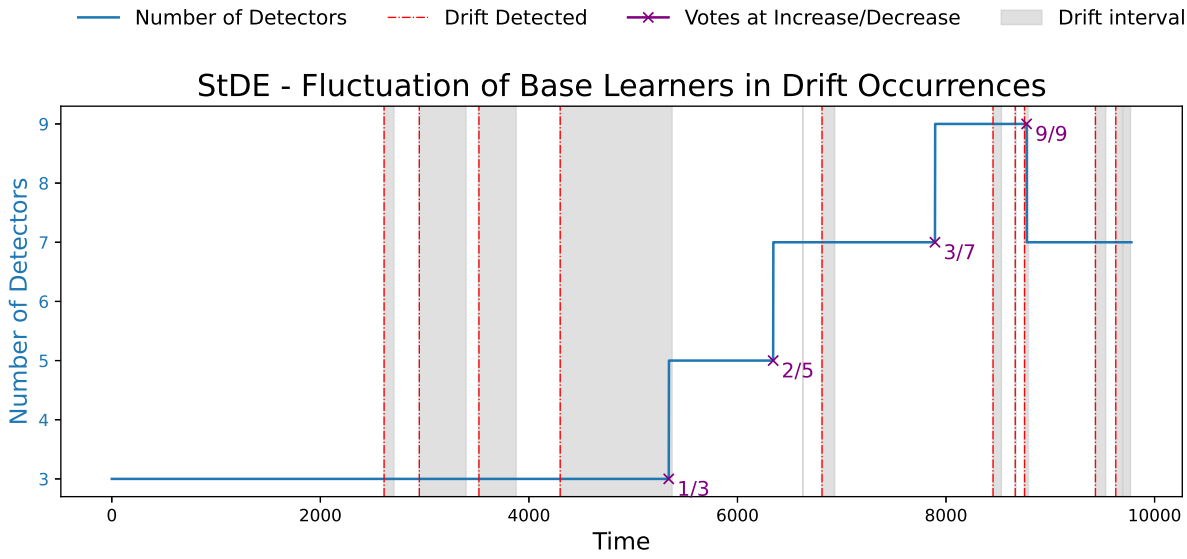

Figure 17 – Fluctuation in the number of detectors in response to the signaling of potential drifts by the detector

5.4.6 Discussion

StDE demonstrated robust predictive performance across different types of drifts — abrupt, gradual, and incremental. By dynamically adjusting its internal configuration to incorporate multiple drift detectors, StDE was able to effectively respond to a wide range of drift patterns. This adaptability allowed the ensemble to maintain high accuracy even in complex scenarios where single detectors might struggle. Notably, StDE achieved better predictive performance than the LE3D framework. The results indicate that StDE consistently achieved F1 scores comparable to or better than the individual algorithms it encompasses, highlighting the ensemble's ability to leverage the strengths of multiple detectors and providing a more comprehensive and reliable approach to drift detection.

A major strength of StDE lies in its dynamic adaptability to various drift types. Unlike static ensembles, StDE can adjust its internal configuration in real time, selecting and fine-tuning the most effective detectors for the specific drift scenario at hand. This flexibility is especially relevant in real-world applications, where data streams can change unpredictably. Additionally, the introduction of new instances of detection algorithms with perturbed hyperparameters during the online phase of StDE plays a crucial role in identifying diverse types of drift. Even small perturbations introduced via Gaussian noise significantly enhance the system's ability to detect more complex drifts, particularly in heterogeneous data streams.

Our experimental results highlight the advantages of this dynamic approach, as StDE consistently delivered high performance across all tested drift scenarios, swiftly

adapting to new patterns and reducing detection delays. The ability to incorporate controlled hyperparameter variations ensures that the ensemble remains sensitive to subtle distributional shifts while maintaining robustness against noise, further strengthening its effectiveness in dynamic environments.

In terms of computational efficiency, StDE was designed to balance accuracy with resource usage. The dynamic nature of the ensemble allows it to adjust the number of active detectors based on the characteristics of the data stream, which helps manage memory consumption. The experimental results show that while StDE requires more resources than individual detectors, the increase is justified by the significant gains in detection accuracy and reliability. Moreover, the adaptive configuration ensures that the ensemble does not unnecessarily inflate computational costs, as it can deactivate certain detectors when they are not needed, thereby optimizing performance based on the current drift situation. To provide a clearer understanding of the computational environment in which the proposed methods may be important, it is relevant to consider the hardware specifications of low-power embedded systems. For instance, the LE3D study uses the Raspberry Pi (RPi) Compute Module 3b+, which features a BCM2837B0 Cortex-A53 64-bit 1.2 GHz System-on-a-Chip (SoC) and 1 GB of RAM [29]. That helps contextualize the computational limitations and feasibility of implementing the proposed techniques in real-world scenarios.

The results were particularly surprising in terms of performance, processing time, and memory usage, especially when compared to the LE3D framework. StDE not only surpassed LE3D in predictive accuracy but also demonstrated superior efficiency in handling computational and memory demands, underscoring the benefits of its dynamic, adaptive configuration.

In conclusion, StDE offers a significant advantage over traditional, static drift detection methods, particularly in environments where data characteristics are dynamic and unpredictable. Additionally, our results demonstrate that StDE surpasses the LE3D method in terms of robustness and adaptability. Its ability to maintain high accuracy while optimizing resource usage makes it a compelling choice for real-time data stream analysis, confirming the worth of investing in a dynamic, adaptive ensemble configuration.

6 CONCLUSION

Drifts in data can be classified into different types, including abrupt, gradual, and incremental drifts, each presenting specific challenges for detection algorithms. Heterogeneous drifts increase the complexity of detection and adaptation for models. Understanding how detection algorithms perform in the presence of these diverse drift types is crucial for developing more robust and effective methods.

In conclusion, the study's results demonstrate significant variations in the performance of drift detection algorithms based on the type of drift. HDDMW exhibits superior performance for abrupt and gradual drifts, followed by HDDMA, DDM, and EDDM. Conversely, EDDM achieves the best performance for incremental drifts, with HDDMW ranking second, and HDDMA and DDM receiving notably lower scores. As the number of drifts increases, the performance of the algorithms deteriorates, although HDDMW maintains the most favorable detection performance, while EDDM exhibits the least favorable scores. DDM achieves the lowest detection time, followed by the drift detection methods based on Hoeffding algorithms, while EDDM shows significant variability in detection time.

In light of these findings, the Self-tuning Drift Ensemble (StDE) presents an advancement in concept drift detection for data streams. By dynamically adjusting the number and configuration of its base learners based on their voting behavior, StDE addresses the challenges posed by diverse and evolving data streams. Its dual-phase operation—initial Warm Start phase with offline data followed by real-time monitoring—ensures adaptability to various drift types.

The results demonstrate that StDE outperformed its individual base learners and also surpassed the LE3D framework ensemble proposed in the literature, particularly in scenarios involving complex drift patterns. This highlights the robustness and versatility of StDE, which consistently maintained high efficiency and accuracy across different drift scenarios. These findings emphasize the potential of StDE as a reliable and adaptive solution for real-time data stream analysis.

Future research should explore advanced drift detection algorithms, such as ensemble methods or deep learning-based approaches, and use more diverse and larger datasets to improve the quality of findings for various applications. Optimizing StDE's hyperparameters and integrating additional types of base learners could further enhance its performance in complex data environments.

REFERENCES

[1] LEMAIRE, V.; SALPERWYCK, C.; BONDU, A. A survey on supervised classification on data streams. *Lecture Notes in Business Information Processing*, 03 2015.

[2] CANO, A.; KRAWCZYK, B. Rose: robust online self-adjusting ensemble for continual learning on imbalanced drifting data streams. *Machine Learning*, Springer, v. 111, n. 7, p. 2561–2599, 2022.

[3] HAN, M. et al. A survey of active and passive concept drift handling methods. *Computational Intelligence*, Wiley Online Library, v. 38, n. 4, p. 1492–1535, 2022.

[4] POZZOLO, A. D. et al. Credit card fraud detection and concept-drift adaptation with delayed supervised information. In: *2015 International Joint Conference on Neural Networks (IJCNN)*. [S.l.: s.n.], 2015. p. 1–8.

[5] KOMORNICZAK, J.; ZYBLEWSKI, P.; KSIENIEWICZ, P. Statistical drift detection ensemble for batch processing of data streams. *Knowledge-Based Systems*, Elsevier, v. 252, p. 109380, 2022.

[6] MARTINS, V. E.; CANO, A.; JUNIOR, S. B. Meta-learning for dynamic tuning of active learning on stream classification. *Pattern Recognition*, Elsevier, v. 138, p. 109359, 2023.

[7] KORYCKI, L.; KRAWCZYK, B. Unsupervised drift detector ensembles for data stream mining. In: IEEE. *2019 IEEE international conference on data science and advanced analytics (DSAA)*. [S.l.], 2019. p. 317–325.

[8] PÉREZ, J. L. M.; BARROS, R. S.; SANTOS, S. G. Statistical tests ensemble drift detector. In: IEEE. *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*. [S.l.], 2020. p. 1021–1028.

[9] DU, L. et al. A selective detector ensemble for concept drift detection. *The Computer Journal*, Oxford University Press, v. 58, n. 3, p. 457–471, 2015.

[10] ABBASI, A. et al. Elstream: An ensemble learning approach for concept drift detection in dynamic social big data stream learning. *IEEE Access*, IEEE, v. 9, p. 66408–66419, 2021.

[11] IKONOMOVSKA, E.; LOSKOVSKA, S.; GJORGJEVIKJ, D. A survey of stream data mining. 09 2007.

[12] WARES, S.; ISAACS, J.; ELYAN, E. Data stream mining: methods and challenges for handling concept drift. *SN Applied Sciences*, v. 1, n. 11, p. 1412, Oct 2019. ISSN 2523-3971. Disponível em: <https://doi.org/10.1007/s42452-019-1433-0>.

[13] WEBB, G. I. et al. Characterizing concept drift. *Data Mining and Knowledge Discovery*, Springer, v. 30, n. 4, p. 964–994, 2016.

[14] MAHDI, O. A. et al. Fast reaction to sudden concept drift in the absence of class labels. *Applied Sciences*, MDPI, v. 10, n. 2, p. 606, 2020.

[15] GAMA, J. et al. A survey on concept drift adaptation. *ACM computing surveys (CSUR)*, ACM New York, NY, USA, v. 46, n. 4, p. 1–37, 2014.

[16] LU, J. et al. Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering*, IEEE, v. 31, n. 12, p. 2346–2363, 2018.

[17] GAMA, J.; CASTILLO, G. Learning with local drift detection. In: SPRINGER. *Advanced Data Mining and Applications: Second International Conference, ADMA 2006, Xi'an, China, August 14-16, 2006 Proceedings 2*. [S.l.], 2006. p. 42–55.

[18] BAENA-GARCIA, M. et al. Early drift detection method. In: CITESEER. *Fourth international workshop on knowledge discovery from data streams*. [S.l.], 2006. v. 6, p. 77–86.

[19] FRIAS-BLANCO, I. et al. Online and non-parametric drift detection methods based on hoeffding's bounds. *IEEE Transactions on Knowledge and Data Engineering*, IEEE, v. 27, n. 3, p. 810–823, 2014.

[20] GAMA, J.; MEDAS, P.; CASTILLO GLADYSAND RODRIGUES, P. Learning with drift detection. In: BAZZAN ANA L. C.AND LABIDI, S. (Ed.). *Advances in Artificial Intelligence – SBIA 2004*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004. p. 286–295. ISBN 978-3-540-28645-5.

[21] BIFET, A.; GAVALDà, R. Learning from time-changing data with adaptive windowing. In: . [S.l.: s.n.], 2007. v. 7.

[22] BARROS, R. S. M.; SANTOS, S. G. T. C. A large-scale comparison of concept drift detectors. *Information Sciences*, Elsevier, v. 451, p. 348–370, 2018.

[23] SANTOS, S. G.; BARROS, R. S.; JR, P. M. G. A differential evolution based method for tuning concept drift detectors in data streams. *Information Sciences*, Elsevier, v. 485, p. 376–393, 2019.

[24] BABÜROĞLU, E. S.; DURMUŞOĞLU, A.; DERELI, T. Novel hybrid pair recommendations based on a large-scale comparative study of concept drift detection. *Expert Systems with Applications*, Elsevier, v. 163, p. 113786, 2021.

[25] POENARU-OLARU, L. et al. Are concept drift detectors reliable alarming systems?–a comparative study. *arXiv preprint arXiv:2211.13098*, 2022.

[26] JR, P. M. G. et al. A comparative study on concept drift detectors. *Expert Systems with Applications*, Elsevier, v. 41, n. 18, p. 8144–8156, 2014.

[27] DECKERT, M. Batch weighted ensemble for mining data streams with concept drift. In: SPRINGER. *Foundations of Intelligent Systems: 19th International Symposium, ISMIS 2011, Warsaw, Poland, June 28-30, 2011. Proceedings 19*. [S.l.], 2011. p. 290–299.

[28] RAMANE, M.; GNANASEKAR, J. A novel hybrid concept drift detector ensemble for handling pure and hybrid drift in non-stationary data streams. *NeuroQuantology*, NeuroQuantology, v. 20, n. 8, p. 6170, 2022.

[29] MAVROMATIS, I. et al. Le3d: a lightweight ensemble framework of data drift detectors for resource-constrained devices. In: IEEE. *2023 IEEE 20th Consumer Communications & Networking Conference (CCNC)*. [S.l.], 2023. p. 611–619.

[30] KHAMASSI, I. et al. Ensemble classifiers for drift detection and monitoring in dynamical environments. In: *Annual Conference of the PHM Society*. [S.l.: s.n.], 2013. v. 5, n. 1.

[31] SAMANT, R. C. et al. A systematic ensemble approach for concept drift detector selection in data stream classifiers.

[32] ZHANG, S. et al. Sled: Semi-supervised locally-weighted ensemble detector. In: IEEE. *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. [S.l.], 2020. p. 1838–1841.

[33] MACIEL, B. I. F.; SANTOS, S. G. T. C.; BARROS, R. S. M. A lightweight concept drift detection ensemble. In: IEEE. *2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI)*. [S.l.], 2015. p. 1061–1068.

[34] SILVA, R. P. et al. Unsupervised tuning for drift detectors using change detector segmentation. *IEEE Access*, v. 12, p. 54256–54271, 2024.

[35] MONTIEL, J. et al. River: machine learning for streaming data in python. 2021.

[36] STREAMLIT. <https://streamlit.io/>. Accessed: 2024-05-01.

[37] DEVRIES, Z. et al. Using a national surgical database to predict complications following posterior lumbar surgery and comparing the area under the curve and f1-score for the assessment of prognostic capability. *The Spine Journal*, v. 21, n. 7, p. 1135–1142, 2021. ISSN 1529-9430. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1529943021000784>.

[38] ASGHARI, M. et al. Aggregate density-based concept drift identification for dynamic sensor data models. *Neural Computing and Applications*, v. 33, n. 8, p. 3267–3279, Apr 2021. ISSN 1433-3058. Disponível em: <https://doi.org/10.1007/s00521-020-05190-1>.

[39] FRIEDMAN, M. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, Taylor & Francis, v. 32, n. 200, p. 675–701, 1937. ISSN 01621459.

[40] NEMENYI, P. *Distribution-free Multiple Comparisons*. Tese (Doutorado) — Princeton University, 1963.

TRABALHOS PUBLICADOS PELO AUTOR

Trabalhos publicados pelo autor durante o programa.

Publicações principais do trabalho.

1. Guilherme Yukio Sakurai, Jessica Fernandes Lopes, Bruno Bogaz Zarpelão, Sylvio Barbon Junior, **Benchmarking Change Detector Algorithms from Different Concept Drift Perspectives**, Future Internet 2023, Abril/2024, 15, 169, https://doi.org/10.3390/fi15050169 (Fator de impacto (JCR) 2,8)

2. Guilherme Yukio Sakurai, Bruno Bogaz Zarpelão, Sylvio Barbon Junior, **A Self-Tuning ensemble approach for drift detection** , In: ENCONTRO NACIONAL DE INTELIGÊNCIA ARTIFICIAL E COMPUTACIONAL (ENIAC), 21., 2024, Belem/PA. Anais [...]. Porto Alegre: Sociedade Brasileira de Computação, 2024, Qualis CC 2020 B4.