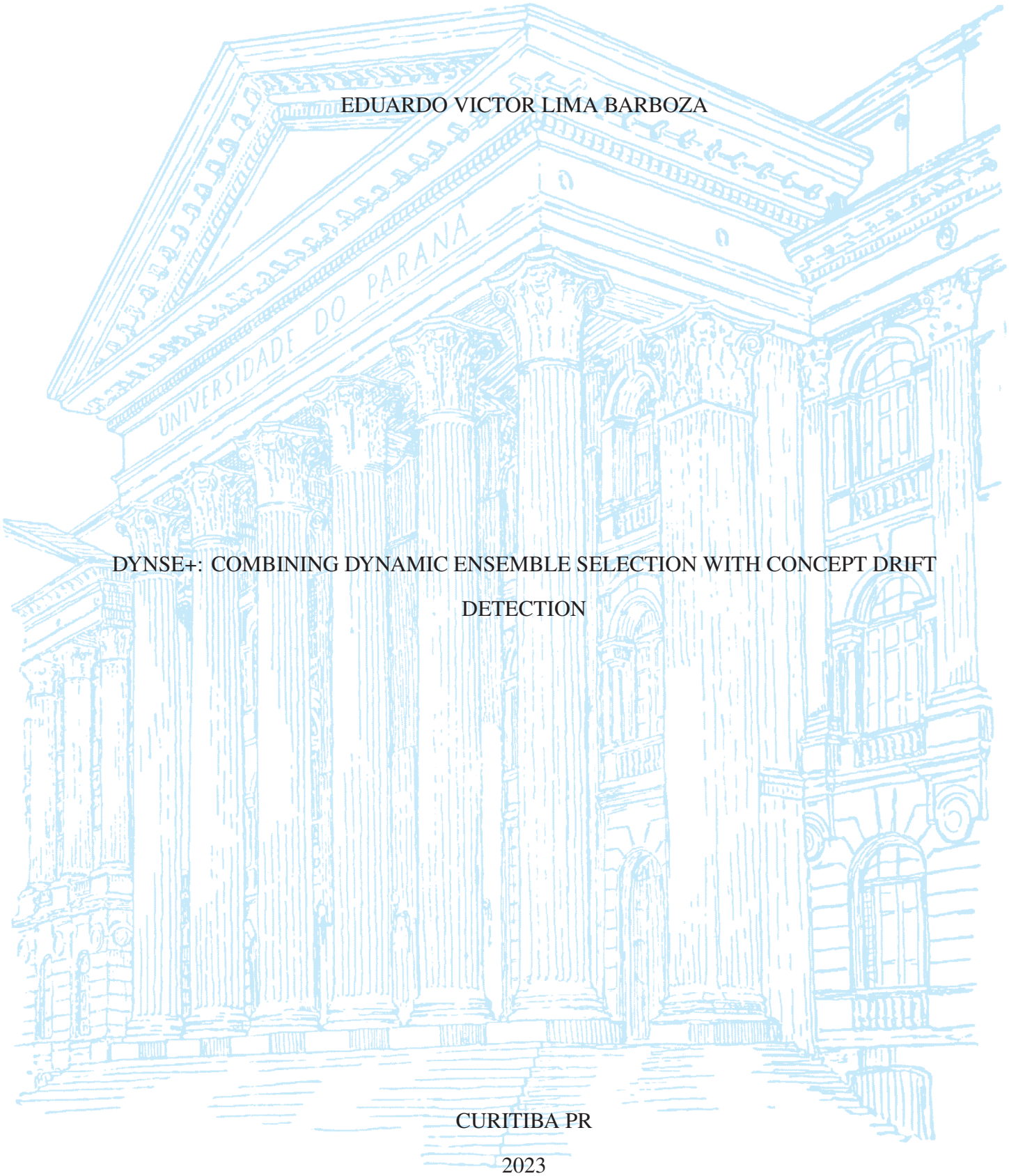
UNIVERSIDADE FEDERAL DO PARANÁ

EDUARDO VICTOR LIMA BARBOZA



DYNSE+: COMBINING DYNAMIC ENSEMBLE SELECTION WITH CONCEPT DRIFT DETECTION

CURITIBA PR

2023

EDUARDO VICTOR LIMA BARBOZA

DYNSE+: COMBINING DYNAMIC ENSEMBLE SELECTION WITH CONCEPT DRIFT DETECTION

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Informática no Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Paulo Ricardo Lisboa de Almeida.

Coorientador: Rafael Menelau Oliveira e Cruz.

CURITIBA PR

2023

DADOS INTERNACIONAIS DE CATALOGAÇÃO NA PUBLICAÇÃO (CIP)
UNIVERSIDADE FEDERAL DO PARANÁ
SISTEMA DE BIBLIOTECAS – BIBLIOTECA DE CIÊNCIA E TECNOLOGIA

Barboza, Eduardo Victor Lima
Dynse+: combining dynamic ensemble selection with concept drift detection . / Eduardo Victor Lima Barboza. – Curitiba, 2023.
1 recurso on-line : PDF.

Dissertação (Mestrado) - Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-Graduação em Informática.

Orientador: Paulo Ricardo Lisboa de Almeida
Coorientador: Rafael Menelau Oliveira e Cruz

1. Aprendizado do computador. 2. Framework (Arquivo de computador). I. Universidade Federal do Paraná. II. Programa de Pós-Graduação em Informática. III. Almeida, Paulo Ricardo Lisboa de. IV. Cruz, Rafael Menelau Oliveira e. V. Título.

Bibliotecário: Leticia Priscila Azevedo de Sousa CRB-9/2029

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação INFORMÁTICA da Universidade Federal do Paraná foram convocados para realizar a arguição da Dissertação de Mestrado de **EDUARDO VICTOR LIMA BARBOZA** intitulada: **DYNSE+: COMBINING DYNAMIC ENSEMBLE SELECTION WITH CONCEPT DRIFT DETECTION**, sob orientação do Prof. Dr. PAULO RICARDO LISBOA DE ALMEIDA, que após terem inquirido o aluno e realizada a avaliação do trabalho, são de parecer pela sua APROVAÇÃO no rito de defesa.

A outorga do título de mestre está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

CURITIBA, 15 de Fevereiro de 2024.

Assinatura Eletrônica
15/02/2024 15:22:02.0
PAULO RICARDO LISBOA DE ALMEIDA
Presidente da Banca Examinadora

Assinatura Eletrônica
15/02/2024 16:25:12.0
LUIZ EDUARDO SOARES DE OLIVEIRA
Avaliador Interno (UNIVERSIDADE FEDERAL DO PARANÁ)

Assinatura Eletrônica
18/02/2024 11:22:57.0
ALCEU DE SOUZA BRITTO JR
Avaliador Externo (PONTIFÍCIA UNIVERSIDADE CATÓLICA DO PARANÁ)

Assinatura Eletrônica
19/03/2024 16:49:20.0
RAFAEL MENELAU OLIVEIRA E CRUZ
Coorientador(a) (ÉCOLE DE TECHNOLOGIE SUPéRIEURE)

Para meus pais. Obrigado por todo o apoio.

ACKNOWLEDGEMENTS

Primeiramente, agradeço aos meus pais, Eduardo Ferreira Barboza e Edineide de Lima Barboza, que sempre me deram apoio em tudo o que eu quis fazer.

Obrigado aos meus orientadores, Professores Paulo Almeida e Rafael Cruz, que me mostraram o caminho para levar minha carreira acadêmica ao próximo nível. Aos membros da banca Luiz Oliveira e Alceu Britto, por aceitar fazer parte de minha banca de avaliação, avaliar meu trabalho, e pelas conversas que levaram a ideias de pesquisa.

Muitos fizeram parte da escrita deste trabalho, seja pelos momentos de descontração que recuperava minhas energias depois de horas de trabalho, ou pelos momentos de revelação depois de uma conversa. Como não é possível citar o nome de todos, estendo meus agradecimentos a quaisquer pessoas que fizeram parte de minha vida e da escrita deste trabalho.

RESUMO

Concept Drift é um problema comum quando lidamos com fluxos de dados. Como as distribuições de probabilidade dos dados mudam com o passar do tempo, é essencial que modelos de Aprendizado de Máquina consigam se adaptar. Na literatura, existem alguns métodos que tentam se adaptar a essas mudanças. Entre eles, o Dynse, que é baseado em seleção dinâmica de *ensemble*, é o foco deste trabalho. Ele mantém um pool de classificadores que é atualizado sempre que um novo lote de dados chega. Esses classificadores são selecionados baseado em sua performance em um *dataset* de validação para classificar uma instância. Duas mudanças foram propostas, que são fazer o Dynse capaz de fazer processamento online, e adicionar um detector de mudança, para auxiliar na adaptação para um novo conceito. Os resultados experimentais mostram que, após a adaptação para processamento online, que chamamos de ODynse, tivemos uma performance melhor do que o framework Dynse original em ambas mudanças de conceito real e virtual. Após a adição do detector de mudança, resultando no Dynse+, tivemos resultados melhores estatisticamente significativos do que o Dynse. O Dynse+ também foi o método melhor classificado entre 7 outros métodos do estado-da-arte, com uma diferença estatisticamente significativa para 4 deles. Também foram feitos testes considerando rótulos atrasados e parciais, para simular um cenário mais próximo do mundo real. O Dynse+ também foi o melhor comparado a outros métodos do estado da arte. Contudo, ele foi o método mais lento quando consideramos tempo de processamento.

Palavras-chave: Mudança de Conceito. Seleção Dinâmica. Machine Learning.

ABSTRACT

Concept drift is a common problem when we are dealing with data streams. As the probability distribution of data changes with time, it is crucial that Machine Learning models are able to adapt. In the literature, there are some methods that try to adapt to these changes. Between them, Dynse, which is based on Dynamic Ensemble Selection, is the focus of this work. It maintains a pool of classifiers that is updated whenever a new batch of data arrives. These classifiers are selected based on their performance in a validation dataset to classify an instance. Two changes were proposed, which are making Dynse able to perform online processing, and to add a drift detector to assist on the adaptation to a new concept. The experimental results show that, after the adaptation to online processing, which we call ODynse, we had a best performance to the original Dynse framework. After the addition of the drift detector, resulting on the Dynse+, we had a statistically significant best results than Dynse. Dynse+ also was the best ranked method among 7 other state-of-the-art methods, with a statistically significant difference to 4 of them. There were also performed tests considering delayed and partial labels, to simulate a scenario closer to the real world. Dynse+ was also the best one compared to other methods in the state of the art. However, it was the slowest method when we take into account the processing time.

Keywords: Concept Drift. Dynamic Selection. Machine Learning.

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

ACDDM Accurate Concept Drift Detection Method

ADWIN Adaptive Windowing

AnnSMOTE Synthetic Minority Oversampling Technique with Adaptive Nearest Neighbors

ARF Adaptive Random Forest

AUE Accuracy Updated Ensemble

AWE Accuracy-Weighted Ensemble

CVFDT Concept-adapting Very Fast Decision Tree

CWA Competing Windows Algorithm

DCS Dynamic Classifier Selection

DDCS Double Dynamic Classifier Selection

DDD Diversity for Dealing with Drifts

DDM Drift Detection Method

DES Dynamic Ensemble Selection

DES-ICD Dynamic Ensemble Selection for Imbalanced Data Streams with Concept Drift

DESW-ID Dynamic Ensemble Selection based on Window over Imbalanced Drift Data Stream

DS Dynamic Selection

DSEL Dynamic Selection Dataset

DWM Dynamic Weighted Majority

DynEd Dynamic Ensemble Diversification

ECDD EWMA for Concept Drift Detection

EDDM Early Drift Detection Method

EWMA Exponentially Weighted Moving Average

FHDDM Fast Hoeffding Drift Detection Method

FHDDMS Stacking Fast Hoeffding Drift Detection Method

FHDDMS$_{add}$ Additive FHDDMS

FPDD Fisher Proportions Drift Detector

FSDD Fisher Squared Drift Detector

FTDD Fisher Test Drift Detector

GDDM Group Drift Detection Method

GMM-VRD Gaussian Mixture Model for Dealing With Virtual and Real Concept Drifts

HAT Hoeffding Adaptive Tree

HDDM Hoeffding Drift Detection Method

HE-CDTL Hybrid Ensemble approach to concept drift-tolerate transfer learning

HOT Hoeffding Option Tree

HT Hoeffding Tree

KNORA K-nearest-oracles

LAE Local Accuracy Estimates

LD3 Label Dependency Drift Detector

MDDM McDiarmid Drift Detection Method

Meta-ADD Active Drift Detection based on Meta learning

ML Machine Learning

MOA Massive Online Analysis

OASW Optimized Adaptive and Sliding Windowing

OAUE Online Accuracy Updated Ensemble

OLA Overall Local Accuracy

PHT Page-Hinkley Test

RDDM Reactive Drift Detection Method

RoC Region of Competence

SEA Streaming Ensemble Algorithm

SEOA Selective Ensemble-based Online Adaptive Neural Network

SMOTE Synthetic Minimum Oversamples Technique

STEPD Statistical Test of Equal Proportions

STUDD Student-Teacher approach for Unsupervised Drift Detection

UDD Uncertainty Drift Detection

VFDT Very Fast Decision Tree

LIST OF SYMBOLS

C	Pool of Classifiers
C'	A subset of C
D	Maximum number of classifiers in C
F	Number of instances without drift detection before a new classifier is trained and added to C in Dynse+
M	Maximum size of DSEL
N	Minimum size of DSEL
s_i	Standard Deviation
p_i	Probability of Misclassification
k	Number of neighbors for k-nearest neighbors
s	Slack variable for KNORA-Eliminate
λ	Parameter of the Poisson distribution
α	Warning level threshold parameter for EDDM
β	Drift level threshold parameter for EDDM
$\hat{\mu}_W$	Mean of elements observed in the most recent window in ADWIN
μ_W	Mean of unknown elements in the most recent window in ADWIN

CONTENTS

1 INTRODUCTION

Nowadays we have data being generated at every moment, which are used in Machine Learning (ML) applications. In such applications, many works consider statically distributed data and do not take into account the possibility of data changing with time, which is known as concept drift (Lu et al., 2020). Concept drift impacts the performance of ML models, bringing the need for strategies of adaptation to maintain the performance (Wang et al., 2024).

An example of a changing concept is the people's opinion with time. We can cite the change in the opinion about vaccination after the COVID-19 pandemic. Müller and Salathé (2020) aims at this scenario and argues that such a crisis like the pandemic may have triggered a concept drift regarding people's opinions about vaccination. Yet in the 2020's pandemic scenario, we can cite the example where people began to wear masks. If a face-detection model was trained only on people without masks, it needs to learn the new concept of people wearing masks, which is not an anomaly anymore (Agate et al., 2022). This example is exposed in Figure 1.1.

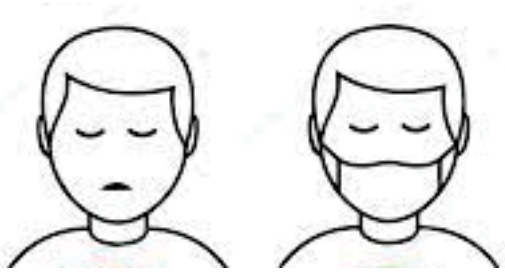


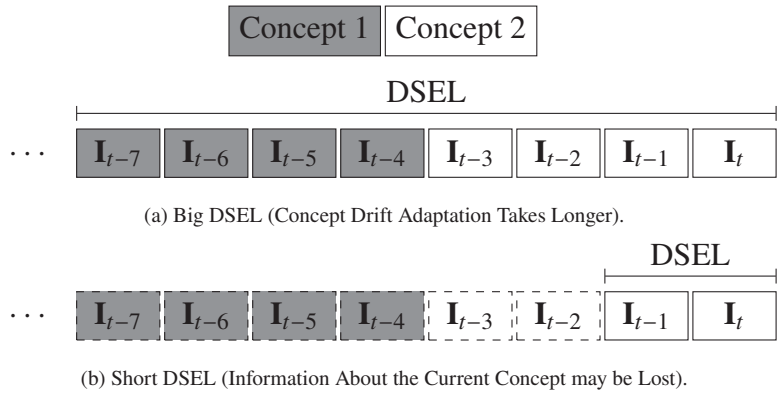
Figure 1.1: Man Wearing Mask vs Man Not Wearing Mask. Adapted from DepositPhotos[1].

In the cyber security context, Jordaney et al. (2017) shows how these changes impact classifying malware, as malware manufacturers are constantly changing their techniques. Thus, the ML model must be able to be on track with such changes to learn new concepts of malware. The same happens on detecting e-mails as spam (Kuncheva, 2004), which varies according to user preferences, and spam manufacturing methods are also constantly changing. The model needs to adapt as well.

The need of our ML models adapting to the expected changes occurring in real life shows the relevance on studying concept drift. In the literature, there are many ways to deal with it. For example, methods that aim at the error rate of the ML model (Gama et al., 2004; Baena-García et al., 2006), sliding windows (Kubát, 1989; Widmer and Kubat, 1992), ensembles (Oza, 2005; Bifet et al., 2010b) and Dynamic Selection (Cavalheiro et al., 2021; Jiao et al., 2022). In this work, we will address a Dynamic Selection-based approach called Dynse (Almeida et al., 2018). Dynamic Selection-based methods are those which will try to select the best classifier(s) depending on their performance on a Region of Competence (RoC) of the test instance, which is gathered from the Dynamic Selection Dataset (DSEL) through, for instance, the k-nearest neighbors of the test instance (Cruz et al., 2018).

Almeida et al. (2018) argue that any Dynamic Selection (DS) method can deal with concept drift when the time-dependence is considered in it. They do this by adding a new classifier to a pool C whenever a batch of a predefined number of instances is completed. The added classifier will then be selected based on the DS engine. However, a known limitation of

[1]Available at `https://depositphotos.com/vector-images/no-entry-without-face-mask.html`. Accessed on 08/05/2023.



(a) Big DSEL (Concept Drift Adaptation Takes Longer).

(b) Short DSEL (Information About the Current Concept may be Lost).

Figure 1.2: How the Size of the DSEL Impacts on Dynse's Adaptation to Concept Drift

the Dynse framework lies on the DSEL, which is the validation dataset that is used to extract the RoC to measure the competence of classifiers. Its size must be defined by the user. A short DSEL may not have enough information for an efficient DS. On the other hand, a larger DSEL may postpone the adaptation to a new concept. This is known as the stability-plasticity dilemma (Elwell and Polikar, 2011). Another point on the choice of the maximum size of the DSEL is that we need to know in advance whether we are dealing with *virtual* or *real concept drift* (see Section 2.2.1), as a larger DSEL is better for *virtual concept drift*, in which we may have a static concept where we know different regions of the feature space over time, and it does not affect the ideal decision boundary (Kolter and Maloof, 2007; Lu et al., 2020). A minor DSEL will help adapting to *real concept drift*, as most likely the decision boundary will have changed, and our classifier(s) must learn it (Lu et al., 2020; Gama et al., 2014). A scheme of how the size of the DSEL may impact on the Dynse's adaptation to concept drift is exposed in Figure 1.2.

In this work, we aim to soften the need to choose the right size of the DSEL, letting it grow larger in cases of stable concepts or *virtual concept drift*. This is possible by the addition of a concept drift detector. When a concept drift is detected, the DSEL shrinks to a minimum size to adapt to the new concept. To do so, first, the Dynse framework is adapted to be able to perform online processing of instances, i.e., update whenever a new data sample arrives, instead of batch-processing. Then the concept drift detector will be added to it.

Dynse accepts any DS method and base classifiers, provides flexibility and is extensible. In this work, we propose Dynse+, a novel framework which is an extension of the Dynse framework (Almeida et al., 2018) that will maintain the characteristics of flexibility and extensibility, and hope that it can help other authors to include a concept drift detector in their DS approaches for non-stationary streaming data and give insights about the pros and cons of doing so.

1.1 OBJECTIVES

In this work, we address some limitations of the Dynse framework (Almeida et al., 2018), which includes the batch processing and the limited size of the DSEL. By taking into account the need to continuously update ML models under stream scenarios, the ability of the resulting framework to perform online processing might be beneficial. Also, the ability to have the DSEL guided by drift detection shall make Dynse able to react to concept drift actively, enabling the possibility to have a larger estimation window in stable concepts and in cases of *virtual concept drift*, and a shorter when a concept drift takes place. That said, the objectives of this work are:

- To propose ODynse, a version of Dynse capable of performing online processing.

- To propose the adaptation of the DSEL through drift detection, resulting on the Dynse+ framework.
- To asses different drift detectors in the proposal.

1.2 HYPOTHESIS

Our hypothesis is that the usage of a trigger, i.e., a drift detector to help with the adaptation of the DSEL will lead to better results in scenarios with *real concept drift* and perform no worse than Dynse in scenarios with *virtual concept drift*. The authors of Dynse have proposed two different configurations for Dynse: one to deal with *real concept drift* and the other for *virtual concept drift*. The difference is that the DSEL for the *virtual concept drift* is larger. We argue that there is no need to have two different configurations anymore, as in cases of *virtual concept drift* the DSEL will keep growing, and when we have a *real concept drift* triggered by a drift detector, the DSEL will shrink to forget an old concept. As it is difficult to know in advance if we are dealing with *virtual* or *real concept drift*, to have only one setup that is able to deal with both is rather beneficial.

1.3 CONTRIBUTIONS

This work has the following contributions:

- Improvement on the Dynse framework for performing online processing.
- Improvement on the Dynse framework for being able to actively react to concept drift by automatically adjusting its DSEL.
- The availability of the resulting frameworks in a public repository.
- An evaluation of how good are common datasets in the literature for evaluating models from the concept drift perspective.
- An evaluation of various drift detectors in the literature.
- Comparison of various methods in the state of the art in various datasets.
- Paper published on Brazilian Symposium on Databases (SBBD) (Barboza and Almeida, 2022).
- Paper published on International Joint Conference on Neural Networks (IJCNN) 2023 (Barboza et al., 2023).

1.4 DOCUMENT STRUCTURE

The remainder of this work is structured as follows: in Section 2, there are presented key concepts regarding data streams, characterized concept drift and its types, presented the performance metrics used for comparison, explained what is a drift detector and how to evaluate them, there is explained what DS is, and the Dynse Framework is presented. In Section 3, some works in the literature of concept drift are presented. In Section 4 we describe Dynse+, the proposed framework of this work, the datasets utilized for the experimental evaluation, and the experiments guidelines. In Section 5 we report the results regarding Cramer's V autocorrelation test, naïve

methods, a hyperparameter analysis of Dynse+, and the comparison of the proposed framework to the state of the art. Finally, Section 6 concludes this work.

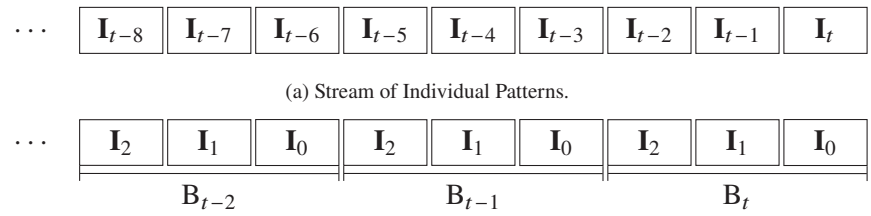
2 THEORETICAL FOUNDATION

2.1 DATA STREAMS

Data streams are characterized as potentially infinite flows of data, that may be presented in the form of individual patterns, also called online processing, or in batches, and may be evolving over time (Komorniczak and Ksieniewicz, 2023; Bifet et al., 2013). A data stream of n data points may be defined as a sequence $S = \{I_1, I_2, ..., I_{t-2}, I_{t-1}, I_t, \}$, where I_t is the instance that arrived at time t denoted by $I_t = (x_t, y_t)$, where x_t is the feature vector from I_t, and y_t the label for I_t.

Online processing algorithms process instances appearing one by one in time. In batches, also called blocks or chunks, the examples come in portions, generally of the same size, and the updating is done as these blocks of instances are available (Brzezinski and Stefanowski, 2014). Figure 2.1(a) presents a scheme of a stream of individual patterns, Figure 2.1(b) shows a stream of batches with 3 instances, where B_t is the current batch and B_{t-1} the previous.

The data stream literature relies on methods that deal with one of these two ways of processing data instances. Examples of works of online processing are (Domingos and Hulten, 2000; Brzezinski and Stefanowski, 2014; Oza, 2005; Bifet et al., 2010b; Gomes et al., 2017), and of batch processing (Brzeziński and Stefanowski, 2011; Wang et al., 2003; Kolter and Maloof, 2007; Almeida et al., 2018; Kozal et al., 2021).



(a) Stream of Individual Patterns.

(b) Stream of Batches.

Figure 2.1: Stream of Instances vs Stream of Batches.

2.2 CONCEPT DRIFT

In this section, the different types of concept drift regarding probabilistic sources and transition in time are explored. In data streams scenarios, as data arrives continuously, we may have some scenarios that we do not encounter in the classic batch-learning ML. The natural trend data has to change over time is best perceived in data streams. Changes may happen due to, for instance, weather, people's preferences, and politics. We can not neglect to adapt to changes in the data distribution, which is called concept drift.

Concept drift happens when input data and classes' relationship change with time (Gama et al., 2014). It brings the need to look for ways for the ML model to adapt to such changes. The formal definition of concept drift is as follows (Gama et al., 2014; Lu et al., 2020):

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

where y is the class of the instance, P_t is the (x, y) probability in time t, and x is a feature vector. Change happens in some time $t + \delta$, for any $\delta > 0$. $P_t(x, y)$ can be unraveled by following the Bayesian Decision Theory:

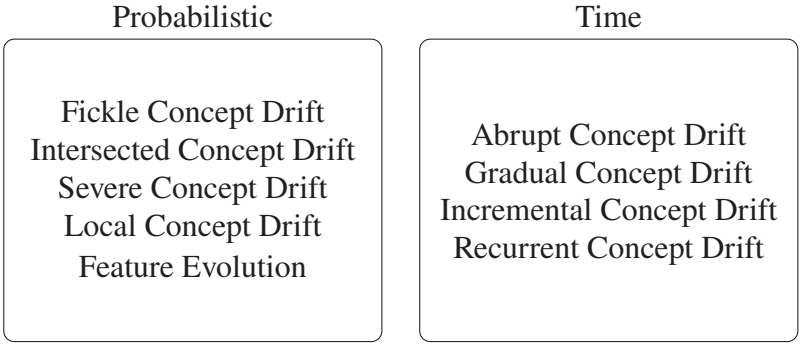


Figure 2.2: Types of Concept Drift From Different Sources.

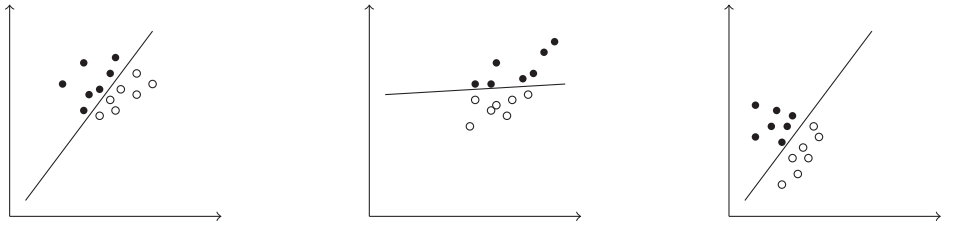
$$P_t(x, y) = P_t(y|x) \times P_t(X) = P_t(x|y) \times P_t(y) \tag{2.2}$$

$P_t(y|X)$ refers to the *a posteriori* probability distribution of the target labels, and $P_t(x)$ is the probability distribution of the input data. $P_t(y)$ refers to the *a priori* probability distribution of the target labels, and $P_t(x|y)$ refers to the probability density distribution conditioned to the class.

Concept drift might have different sources and different types. There is no consensus in the literature on the definition of different types of concept drift. Some authors split it into *real concept drift* and *virtual concept drift* (Gama et al., 2014). However, other authors defined other types (Bayram et al., 2022; Minku et al., 2010; Forman, 2006), as we explain in Section 2.2.1. We also have different types of concept drift based on how it happens on a timestamp (Lu et al., 2020; Gama et al., 2014; Minku et al., 2010), which we explore in Section 2.2.2. The division of the types of concept drift between probabilistic criterion and on how the data distribution changes with time is shown in Figure 2.2.

2.2.1 Types of Concept Drift – Probabilistic Source

In this section, we will see the terms *real concept drift* and *virtual concept drift*, as well as some subcategories found in the literature, which have their source in the probability distribution of data. The way *real* and *virtual concept drifts* may happen in a 2-dimensional space are exposed in Figure 2.3.



(a) Original Data.

(b) Real Concept Drift (Changes the Ideal Decision Boundary).

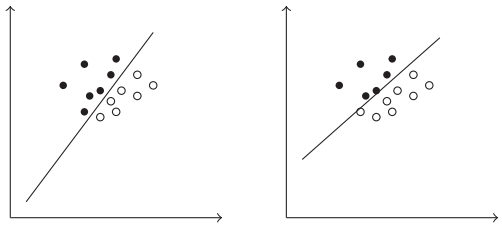
(c) Virtual Concept Drift (Does not Change the Ideal Decision Boundary).

Figure 2.3: Types of Concept Drift – Probabilistic Source.

1. Real Concept Drift it happens when $P_t(y|x) \neq P_{t+\delta}(y|x)$ (Gama et al., 2014). Other authors, like Lu et al. (2020) call it *actual concept drift*. *Real concept drift* changes the ideal decision boundaries of data, as we can see in Figure 2.3(b), and leads to a decrease in ML model's

accuracy. It can accompany or not a virtual concept drift (Lu et al., 2020). An example is the change of the weather on the outside environment (Almeida et al., 2015). Whether it is sunny, rainy, or cloudy may affect the decisions of a ML model.

The following types of concept drift were set as subcategories of *real concept drift* by Bayram et al. (2022). *Fickle concept drift*, in Figure 2.4 is characterized when an instance belongs to two different classes in different moments in the timestamp (Forman, 2006). It can be seen as a less severe concept drift, as defined by Minku et al. (2010), which defines severity as how many of the instances have changed classes in different moments of the timestamp. On the *fickle concept drift*, that happens only to one instance.

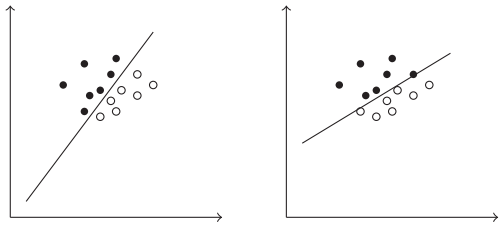


(a) Original Data.

(b) Fickle Concept Drift (One Instance belongs to two Different Classes at Different Moments in Time).

Figure 2.4: Fickle Concept Drift.

Yet talking about severity criterion to describe concept drift, Minku et al. (2010) divide concept drift between severe and intersected. The drift is considered intersected if a part of the input space has the same class in different concepts (Figure 2.5). If all of the classes of the input space change, i.e., if $\forall X : argmax(P_t(y|x)) \neq argmax(P_{t+\delta}(y|x))$, it is called *severe concept drift* (Figure 2.6).

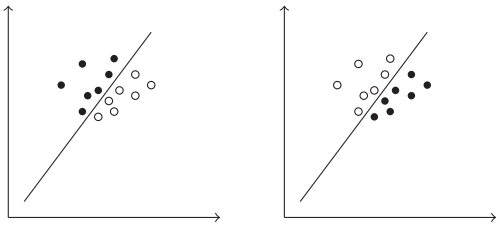


(a) Original Data.

(b) Intersected Concept Drift (Part of the Instances have Different Classes in Different Concepts).

Figure 2.5: Intersected Concept Drift.

2. Virtual Concept Drift, exposed in Figure 2.3(c), happens when $P_t(x) \neq P_{t+\delta}(x)$. $P_t(x)$ does not affect the target concept, so it does not affect the ideal decision boundaries (Lu et al., 2020; Bayram et al., 2022). Even though this type of concept drift does not affect the ideal decision boundary, we must still be careful with it, as we may have be losing useful information that could be used. It may happen when the old data that has already been used for training is still useful, but the new data supplies information that the model did not know before, like a new region in the feature space. Gama et al. (2014) cited as an example the change of the editor of a dwelling rental website. The writing style changes, but the houses are still relevant for the user.

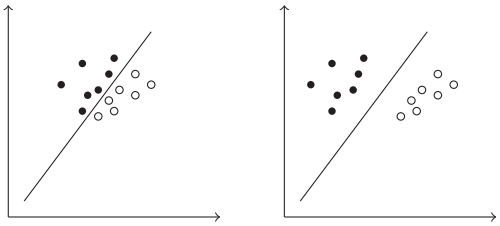


(a) Original Data.

(b) Severe Concept Drift (All of the Instances have Different Classes in Different Concepts).

Figure 2.6: Severe Concept Drift.

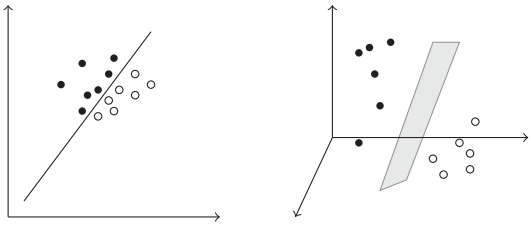
We also can find some subcategories of *virtual concept drift* in the literature. *Local concept drift* in Figure 2.7, for example, happens when the change takes place only in a region of the feature space, or only in one feature (Bayram et al., 2022). We also have *feature evolution*, which refers to the vanishing and/or the appearance of new features with time. In Figure 2.8 we show the example of a new feature emerging (Bayram et al., 2022; Masud et al., 2010).



(a) Original Data.

(b) Local Concept Drift (Change Takes Place Only in One Feature of the Input Space).

Figure 2.7: Local Concept Drift.



(a) Original Data.

(b) Feature Evolution (A New Feature Emerges).

Figure 2.8: Feature Evolution.

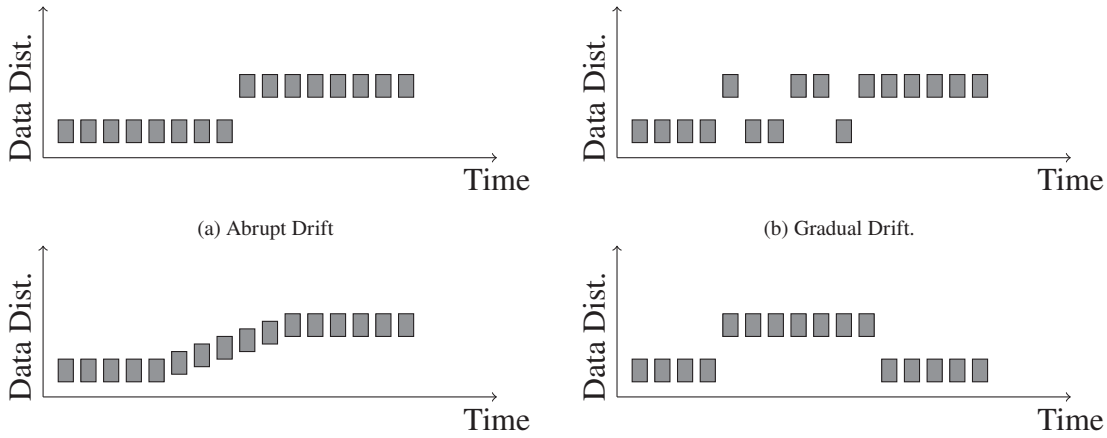
2.2.2 Types of Concept Drift – Transition in Time

We can also differ between some types of concept drift when we look at the way it happens in a timestamp. Like in the probabilistic source, authors have referred to different terms and approaches to define how concept drift happens in a timestamp. Minku et al. (2010) cited two types of concept drift according to speed: *abrupt* and *gradual*. *Abrupt concept drift* happens

when the change occurs in one step in time, or when there is no period of uncertainty between concepts.

On the other hand, a *gradual concept drift* might happen in two different ways. It might have a period of failure, like when a sensor starts to present a defect, where it will periodically send imprecise reads until it completely breaks. Minku et al. (2010) referred to this type of drift as *probabilistic gradual drift*. In this work, we will use the term by Lu et al. (2020): *gradual drift*. The other possibility is when there is some modification in every time step until the new concept stabilizes. Minku et al. (2010) called it *continuous gradual drift*, and Lu et al. (2020) used the term *incremental drift*, which we will also use when referring to this type of drift.

Concept drift may also present recurrence. *Recurrent drift* can have cyclic or non-cyclic behavior (Minku et al., 2010). As an example of cyclic behavior, we can cite the preferences of customers according to the seasons. For example, people might prefer buying a new coat in winter, instead of a tank top. In winter people also tend to turn their heaters on, which will influence energy consumption. When seasons change, an older concept might take place. The change of seasons is a cyclic recurrence that can be told as predictable, and seasons are an obvious influence on behavior. Random *recurrent drift* might happen as well, such as the change of the weather during the day, which can go from sunny to rainy and then go back to sunny. The types of concept drift by the way it happens in a timestamp are exposed in Figure 2.9.



(a) Abrupt Drift

(b) Gradual Drift.

(c) Incremental Drift.

(d) Recurrent Drift.

Figure 2.9: Types of Concept Drift – Transition in Time.

2.3 PERFORMANCE METRICS

Works related to concept drift tend to use different performance metrics. Such metrics tell us how well a model performs on a data stream. Based on the problem we are approaching, we must choose our metrics. Probably, the most known metric is Accuracy, which is the number of instances correctly classified divided by the total number of instances, as follows:

$$Accuracy = \frac{\text{\# Correct Predictions}}{\text{\# Instances}} \tag{2.3}$$

Many authors have used this metric (Oliveira et al., 2019; Krawczyk et al., 2018; Fischer et al., 2016). However, accuracy is not suitable for imbalanced class problems.

In Data Streams scenarios, it is common calculating the accuracy in a prequential manner, i.e., the Prequential Accuracy, which makes it able to evaluate the model's decision through a test-then-train manner, i.e., each individual instance is used to test the model before it is

used for training. With this, the accuracy can be incrementally updated, enabling the possibility of having a plot of accuracy over time (Gama et al., 2014). The prequential accuracy of the i-th instance in the stream can be calculated as:

$$PreqAccuracy_i = \frac{\sum_{j=i-N}^{i}(\hat{y}_j)}{N}, \tag{2.4}$$

where $\hat{y}_j = 1$ if the model predicts the instance j correctly, and 0 otherwise. The prequential accuracy is calculated on a sliding window of N instances.

2.4 CONCEPT DRIFT DETECTION

The literature of concept drift relies, on the majority, on two types of drift adaptation: methods that continuously update the system and are unaware of a concept drift, and those that try to track concept drift (Ditzler et al., 2015). The latter mostly utilize some statistics to keep track of changes and are constantly called as concept drift detectors (Page, 1954; Gama et al., 2004; Baena-García et al., 2006). Most drift detectors require access to the true label of the data in order to update such statistics, but in recent works the authors have been trying to overcome the need of labeled data, which may be limited in data streams (Baier et al., 2021; Gulcan and Can, 2023; Cerqueira et al., 2023). The idea behind using concept drift detectors is to update the models after the drift is detected in order to adapt to the new concept. Thus, if the concept is static, we can keep sending information to our model when available. If a concept drift has happened, one must adapt to the new concept.

2.4.1 Evaluating Drift Detectors

One of the proposals of this work is to add a drift detector to the Dynse framework. Thus, we must evaluate how a drift detector would behave along Dynse. We have three important measures to evaluate drift detectors: True Positive, False Positive, and False Negative (Pesaranghader and Viktor, 2016; Bifet and Gavaldà, 2007).

Firstly, we must define what is a True Positive (TP). We can say that a drift detector has correctly detected a drift if it triggers a change within the range $[t - \Delta, t + \Delta]$, which may be called the detection interval of True Positive. For reactive concept drift detectors, this range is within $[t, t + \Delta]$. A False Positive (FP) is given if a drift detector triggers a drift outside of the detection interval, and a False Negative (FN) if there is no drift detection inside the detection interval (Pesaranghader and Viktor, 2016). The authors recommend to use $\Delta = 250$ for abrupt concept drift, and $\Delta = 1000$ for gradual concept drift.

2.5 DYNAMIC SELECTION AND THE DYNSE FRAMEWORK

There are two types of ensembles in ML: static and dynamic (Cruz et al., 2017). Static ensembles are those in which the ensemble will be built during the training phase. Examples are the Bagging (Breiman, 1996) and Boosting (Freund, 1995). On the other hand, dynamic ensembles will try to select the best ensemble during the testing phase.

DS aims to estimate the competence of classifiers in a RoC in order to select the most competent classifier(s) to label a given instance (Cruz et al., 2018). The RoC is extracted from the DSEL through, for example, a k-nearest neighbors algorithm. The DSEL contains the data that is used to measure the competence of classifiers. For neighborhood-based DS, the most competent classifier(s) from C in the k instances nearest to the one to be classified will be chosen.

Techniques used to select a single classifier are called Dynamic Classifier Selection (DCS), such as Overall Local Accuracy (OLA) (Woods et al., 1997) and Local Accuracy Estimates (LAE) (Woods et al., 1997), and those that aim to select various classifiers to form an ensemble are called Dynamic Ensemble Selection (DES), such as K-nearest-oracles (KNORA) (Ko et al., 2008). These techniques maintain a pool of classifiers C, and when a new unlabeled instance arrives to be classified, the most competent classifier(s) is(are) chosen to label it. The selected classifier(s) is a subset of C, i.e., $C' \subseteq C$, where C' can be either a single classifier or various classifiers.

2.5.1 The Dynse Framework

On the Dynse Framework (Almeida et al., 2018), which is a core part of this work, whenever a new batch B_t of labeled instances arrives at time t, it trains a new classifier and adds it to the Pool of classifiers C. When a new unlabeled instance I_t comes for Dynse to classify it, the best classifiers in C are selected based on the DS method, creating a subset C', i.e., $C' = DS(DSEL, I_t, C)$, where the DSEL are the instances available for the DS method, I_t is the instance to be classified, and C is the pool of classifiers. The process of updating Dynse is exposed in Figure 2.10. See that the newest arrived batch B_t is being used to train a new classifier to be added to C, and when C reaches its maximum size, a classifier is pruned based on a Pruning Engine (e.g., age or accuracy-based)

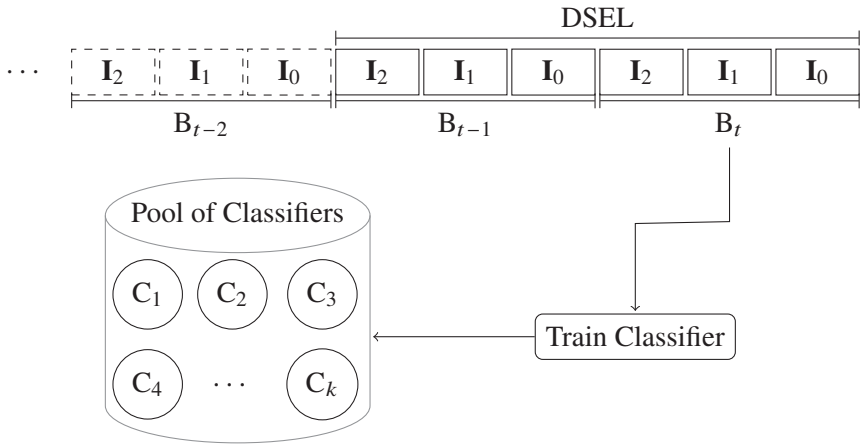


Figure 2.10: Dynse's Scheme for Updating.

If we have a DCS method, the best classifier will label I_t. If we have a DES method, the selected classifiers form an ensemble (e.g., by majority voting) to classify the new instance I_t. For classifying an unlabeled instance, the scheme is in Figure 2.11, where an unlabeled instance is given to the DS engine, or method, which selects the best classifiers in C to form an ensemble to classify that instance.

In Table 2.1, we have Dynse's parameters. The Base Classifier is the classifier that will be trained and added to C whenever a new batch B_t is completed. Train Size refers to the size of B_t or the size of a batch utilized to train new classifiers to be added to C. The size of DSEL is the number of instances in the window of instances that may be selected to the RoC. The authors of Dynse referred to it as the accuracy estimation window. The DS engine is responsible for selecting the classifiers in C according to their competence. The pruning Engine says how the classifiers will be dropped when C is full. The Number of Classifiers is the maximum size of C. Dynse was implemented on the Massive Online Analysis (MOA) framework (Bifet et al., 2010a).

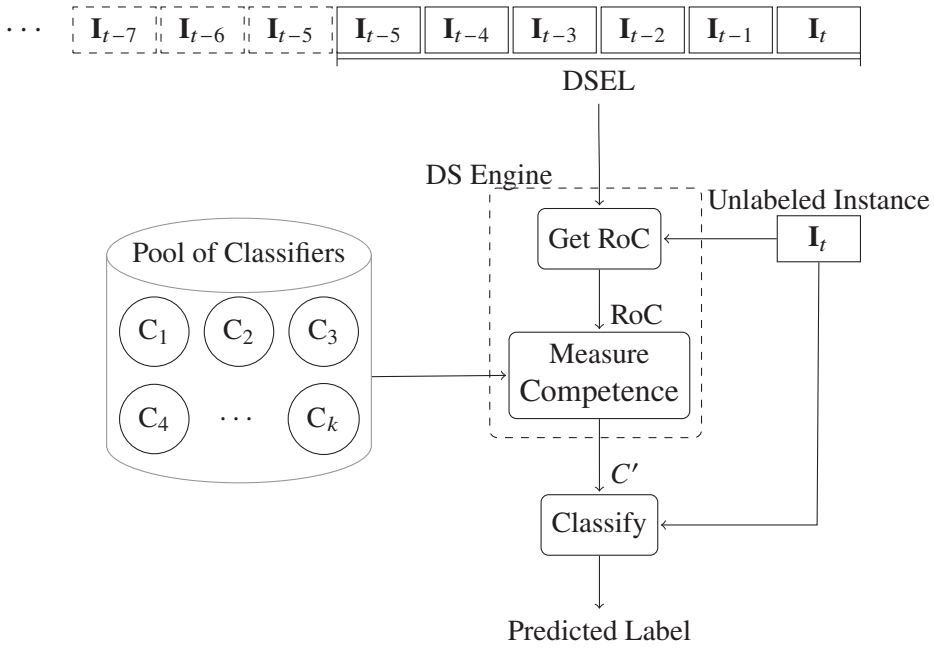


Figure 2.11: Dynse's Scheme for Classifying an Instance.

Table 2.1: Dynse's Parameters.

Parameter	Description
Base Classifier	Base classifier to be used that will compose C.
Training Size (B_t)	Batch size for training new classifiers
Size of DSEL	Size of the DSEL for estimating the accuracy of classifiers
DS Engine	Dynamic Selection Engine for selecting classifiers to form the ensemble of classifiers.
Pruning Engine	The method utilized to prune classifiers out of the pool of classifiers.
Number of Classifiers	Maximum number of classifiers in the pool of classifiers

3 RELATED WORK

In this section, we pass through some methods in the literature concerning data stream classification and concept drift. We will divide methods for dealing with concept drift between Passive, Active, Ensembles (Ditzler et al., 2015), and an additional Section for the DS methods for concept drift. In section 3.1, we discuss some passive methods in the literature, which update without considering whether a concept drift happened. In Section 3.2 are the active methods, which try to track concept drift. In Section 3.3 are the ensemble methods, which may be either passive or active, and in Section 3.4 are the DS methods for concept drift. For the familiar reader, at the end of each section there is a concept matrix summarizing the cited methods.

3.1 PASSIVE METHODS

Passive methods will continuously update themselves regardless a concept drift has happened. The Hoeffding Tree (HT), also called Very Fast Decision Tree (VFDT) (Domingos and Hulten, 2000), is an online learner designed to process high-speed data streams much used in the literature due to its ability to have earlier instances more important than the oldest ones. This is possible because of the Hoeffding's bound it uses for splitting the leaves (Hoeffding, 1963). Then we have Concept-adapting Very Fast Decision Tree (CVFDT) (Hulten et al., 2001), which tries to add the ability of drift adaptation to VFDT. It keeps a sliding window of examples, which makes it able to keep consistent, even when changes happen. They argue that when concepts change, old examples that used to pass the Hoeffding test will not do so anymore. When this happens, new examples start to be stored. When the new subtree with more recent examples becomes more accurate than the old one, it replaces it. One of its advantages is that it does not need to learn a new model. It just updates the statistics of new incoming examples, while decrements the counts of the oldest examples. Another extension of HT is the Hoeffding Option Tree (HOT), which contains additional option nodes in trees, enabling the application of several tests and the possibility of having multiple HT as separate paths.

Yet inside passive methods, there are window-based methods. They may consist of a fixed-size window, like in the FLORA system (Kubát, 1989), where the oldest one is forgotten when a new example arrives in the data stream. However, there is a problem with this because the user must define the window's length. Its consequence is that, if we have a short window length, the model might not have enough information for a reliable training. If the window is too large, otherwise, it might not adapt to a new concept. Taking into account this trade-off, Widmer and Kubat (1992) proposed FLORA2, which basically makes that window adaptive by means of heuristics. FLORA3 (Widmer and Kubat, 1993) has the ability to rescue old concepts in order to best adapt to reoccurring concepts. However, FLORA3 has problems on dealing with noisy data, as according to the authors, noise and concept drift have pretty much the same impact on the prediction error. Widmer (1994) address this issue and propose FLORA4, which is robust to noisy data and is, at the same time, flexible in adapting to concept drift.

The methods cited in this section rely on gradual forgetting by either discarding old examples (window-based), or by decreasing the weight that oldest instances have through the Hoeffding Bound. Other aim to perform statistical tests in order to know when to forget old information. Table 3.1 summarizes the passive methods mentioned.

Table 3.1: Concept Matrix of Passive Methods for Data Streams and Concept Drift.

Method	Source	Window-based	Hoeffding Bound	Statistical Control
FLORA	Kubát (1989)	X		
FLORA2	Widmer and Kubat (1992)	X		
FLORA3	Widmer and Kubat (1993)	X		
FLORA4	Widmer (1994)	X		
HT	Domingos and Hulten (2000)		X	
CVFDT	Hulten et al. (2001)		X	X
HOT	Pfahringer et al. (2007)		X	X

3.2 ACTIVE METHODS

Active methods are those that will react to a concept drift when it is detected. These methods may be based on monitoring some statistics, like the error rate of model prediction, or statistical difference between windows (Bayram et al., 2022). In this Section, we will divide the drift detectors between supervised, which need access to the real labels of the data, and unsupervised, which do not need it.

3.2.1 Supervised Drift Detection

Probably, the first work that considered data distribution changing with time was the Page-Hinkley Test (PHT) (Page, 1954). It aims to determine where a change took place by continuously applying statistical tests. When the statistical difference of errors is greater than a user-defined threshold λ, then it is said that a change has happened.

One of the most popular active methods is the Drift Detection Method (DDM) (Gama et al., 2004). This method consists of monitoring the error of ML model predictions. When this error rises above a certain level, a warning level is given. If the error keeps increasing until the called drift level, the ML model is substituted for a new one trained with the data that arrived between the warning level and drift level. For each instance i, the probability p_i of misclassifying is given by the standard deviation s_i calculated by $s_i = \sqrt{p_i(1-p_i)/i}$. The warning level is given when $p_t + s_t \geq p_{\min} + 2 \times s_{\min}$ and the drift level is given when $p_t + s_t \geq p_{\min} + 3 \times s_{\min}$.

DDM motivated Early Drift Detection Method (EDDM) (Baena-García et al., 2006), which is more suitable for slow and gradual changes, according to the authors. It takes into account the distance between two prediction errors $p_i^{'}$, measured by some distance function, and its standard deviation $s_i^{'}$. The maximum values of $p^{'}$ and $s^{'}$ are stored and are used to calculate both warning and drift levels. The warning level is given when $(p_i^{'} + 2 \times s_i^{'})/(p_{\max}^{'} + 2 \times s_{\max}^{'}) < \alpha$, and the drift level is given when $(p_i^{'} + 2 \times s_i^{'})/(p_{\max}^{'} + 2 \times s_{\max}^{'}) < \beta$. Authors defined the values of $\alpha = 0.95$ and $\beta = 0.90$ after some experimentation, but they can be defined by the user.

Reactive Drift Detection Method (RDDM) (Barros et al., 2017) tries to improve DDM by shortening the number of instances in large stable concepts, aiming to outline performance loss known to happen in DDM. The authors argue that, in large concepts, it might take a large number of instances to affect the prediction error, which may take DDM to delay drift detection. In RDDM, when a concept reaches a maximum number of instances, DDM calculates the thresholds by using only the most recent minimum instances. RDDM also will not detect drift in the warning level, as there would be a possibility of having a short number of instances. Authors also forced a DDM drift whenever the warning level became too large. They argue that, when a warning level stands for a long time, the chances that a concept drift already have happened are big.

Accurate Concept Drift Detection Method (ACDDM) aims the status of the prequential error by using Hoeffding's inequality and triggers concept drift based on the current error (Yan, 2020). EWMA for Concept Drift Detection (ECDD) (Ross et al., 2012) also aims to detect concept drift, and it is based on Exponentially Weighted Moving Average (EWMA) chart. Like other active methods, it monitors the classification error to get warning and drifting thresholds.

Hoeffding Drift Detection Method (HDDM) (Frías-Blanco et al., 2015) tracks concept drift through two different statistical tests, which give two different versions: HDDM-A, which uses A-test, and HDDM-W, with W-test. The A-test considers the possibility of Hoeffding's Inequality (Hoeffding, 1963) to detect significant changes in the moving average in streaming data. The W-test uses weighted moving averages in the statistical tests based on McDiarmid's Inequality (McDiarmid, 1989), which is a generalization of Hoeffding's Inequality for dependent random variables.

The Fast Hoeffding Drift Detection Method (FHDDM) algorithm considers that the classification accuracy over time must either increase or stay steady (Pesaranghader and Viktor, 2016). If it decreases, a concept drift is triggered. It uses a sliding window and Hoeffding's Inequality to compare the maximum probability of correct predictions to the current probability of correct predictions. Stacking Fast Hoeffding Drift Detection Method (FHDDMS) extends FHDDM by maintaining two windows of different sizes: a short and a long window. The rationale is that a short window is best to cope with sudden concept drift and a larger one for gradual concept drift (Pesaranghader et al., 2018b). In the same work, Additive FHDDMS (FHDDMS$_{add}$) is proposed, where the binary indicators of calculations are substituted by the summation of the 5 most recent bits for the short window, and the 20 most recent bits for the long window.

We also have Hoeffding Adaptive Tree (HAT) (Bifet and Gavaldà, 2009), which extends the idea from CVFDT, by using different estimators. There are HAT-INC, which uses a linear incremental estimator, HAT-EWMA, with an Exponential Weight Moving Average, and HAT-ADWIN, with Adaptive Windowing (ADWIN) as an estimator.

ADWIN (Bifet and Gavaldà, 2007) maintains a window W with the most recent instances, a mean $\hat{\mu}_W$ of elements observed in this window and another mean μ_W of unknown elements. In general, the idea behind ADWIN is as follows: when two sub-windows of W show different means, one can say that expected values will be different, and the oldest portion of the window, also called the tail, is forgotten.

The SeqDrift1 detector (Sakthithasan et al., 2013) creates blocks of data and compares the statistics between two batches B_1 and B_2 through a hypothesis test. If the hypothesis test is rejected, $B1$ and $B2$ are concatenated into B_{12}, and the hypothesis test is repeated when a new block of data B_3 is available. SeqDrift1 uses the Bernstein Bound, which the authors argue that it provides a tighter bound when compared to the Hoeffding bound, much used in the concept drift literature. In constrast, the SeqDrift2 detector (Pears et al., 2014) uses the same hypothesis testsing strategy from SeqDrift1, but implements the usage of reservoir sampling employs a tighter cut threshold.

Nishida and Yamauchi (2007) proposed Statistical Test of Equal Proportions (STEPD), which considers a recent accuracy and an old one. The rationale is that, for stable concepts, the accuracy of the new incoming examples will remain similar to the old ones. A decrease in accuracy on new incoming instances is an indicator of concept drift. This difference is checked by doing a statistical test and comparing its value to a percentile to get the P-value. If the P-value is less than a significance level, then a concept drift was detected.

Authors of STEPD state that the best approach was to use Fisher's Exact test. The reason why they did not use it is because of its high computational cost. However, de Lima Cabral and de Barros (2018) proposed ways to do the Fisher's Exact test more efficiently, by making

the size of the recent window equal to the size of the old window. This way, most of the factorial calculations in the Fisher's Exact test would be repeated. They simply stored the already calculated factorials in an array. They can also be used as intermediates for calculating factorials of bigger numbers. Then the author proposed three different methods: Fisher Proportions Drift Detector (FPDD), Fisher Squared Drift Detector (FSDD), and Fisher Test Drift Detector (FTDD).

FPDD works by using the Fisher's Exact test when the number of either correct or incorrect predictions in any window is small. FSDD also applies the Fisher's Exact test in the same scenario of FPDD. The difference is that it applies the chi-square statistical test for homogeneity proportions otherwise, instead of the test of equal proportions. FTDD detects concept drift only by using the Fisher's Exact test.

Oliveira et al. (2019) came with Gaussian Mixture Model for Dealing With Virtual and Real Concept Drifts (GMM-VRD), which uses ECDD as drift detector, and Gaussian Mixtures. The authors proposed two different approaches to deal with virtual and real concept drift separately. To deal with virtual concept drift, authors propose to do the maintenance of useful knowledge whenever a misclassification occurs. This is done by calculating the sample's location related to existing Gaussians. If there is any Gaussian near, the model is updated. On the contrary, a new Gaussian is created. However, this approach looks sensitive to outliers, as it will always update the model when misclassification occurs. In the real concept drift case, authors used ECDD. They argue that real concept drift will degrade the model more than virtual concept drift. By taking this point into account, they update its parameters of warning and drift levels.

The McDiarmid Drift Detection Method (MDDM) (Pesaranghader et al., 2018a) uses the McDiarmid's Inequality (McDiarmid, 1989) to detect concept drift. It has a sliding window of size n, where each element in the window receives a weight w_i, where $w_i < w_{i+1}$, i.e., earlier instances have a higher weight. MDDM has three different versions, in which each one employs different weighting functions. MDDMA uses the arithmetic weighting $w_i = 1 + (i-1)$, MDDMG the geometric weighting $w_i = r^{(i-1)}$, and MDDME the exponential $w_i = e^{\lambda(i-1)}$.

Yang and Shami (2021) proposed Optimized Adaptive and Sliding Windowing (OASW), which keeps two windows: one sliding window to detect drift, and an adaptive window, where incoming data is stored. It also monitors the drops in accuracy between an old and a new window in order to track warning and drifting alarms defined by thresholds. When a drift is detected, a new LightGBM model is created on the instances between warning and drift levels, and the hyperparameters of the drift detector are always updated by using Particle Swarm Optimization. Yu et al. (2023) address group concept drift, i.e., different data streams correlate with each other and propose Group Drift Detection Method (GDDM). It is also based on DDM, but it maintains three different windows: initial, sliding, and detecting windows. A distribution-free statistical test was developed by the authors, which is not susceptible to the underlying distribution of multiple data streams.

3.2.2 Unsupervised Drift Detection

Lazarescu et al. (2004) propose Competing Windows Algorithm (CWA), which uses three different windows of different sizes to track concept drift. The rationale is to have different perspectives on data. They try to track concepts by measuring the difference between two consecutive instances of the current concept, given by some distance function. If the difference is smaller than a threshold, then no drift has happened. If a change persists for p consecutive instances and $p \geq \frac{X}{2}$, where X is the window size, then the change is said to be persistent. One advantage of this method is that it does not need labeled data.

Label Dependency Drift Detector (LD3) is an unsupervised drift detection method that takes into account the temporal difference of labels to track concept drift (Gulcan and Can, 2023).

The idea was to use the correlation between the predicted labels, which the authors argued that if there is a change in the autocorrelation of the predicted labels over time, a concept drift might have happened.

Student-Teacher approach for Unsupervised Drift Detection (STUDD) is another unsupervised method, that utilizes two classifiers, one called the teacher, and the other called the student (Cerqueira et al., 2023). The idea is to train the teacher on an initial batch of the data, and then use it to label the same dataset used to train it. The resulting dataset labeled by the teacher is then used to train the student, with the objective of the student simulating the predictions of the teacher. The error between the predictions of both the teacher and the student is used in a drift detection method. The authors have used the PHT to this end.

Baier et al. (2021) uses neural network prediction uncertainty to track concept drift. The rationale is that the uncertainty can be considered as an indicator of the error rate. This way, there is no need for the true labels of data, which can be seen as an advantage over other drift detectors, which need true labels of data to track the error rate, which is often unfeasible in the real world. By combining the neural network model's uncertainty with ADWIN, authors came up with the Uncertainty Drift Detection (UDD) method.

A meta-learning approach for detecting concept drift was proposed by Yu et al. (2022). In their method, called Active Drift Detection based on Meta learning (Meta-ADD), meta-features are extracted based on the error rate of various concept drifts (they say that it is needed to know the types of concept drift in advance). After that, a meta-detector is built with a neural network that represents different concept drift classes. In the detection phase, the meta-detector is fine-tuned to adapt to the corresponding data stream. This method seems not proper when we have an unknown data stream, where we do not know the type or nature of the concept drift, as Meta-ADD needs to extract the meta-features from different concepts.

The methods cited above say that they do not need labeled data. Let us remember the *real concept drift* definition seen in Section 2.2.1: $P_t(y|X) \neq P_{t+\delta}(y|X)$ for any $\delta > 0$. So, it acts directly on the data label. It is difficult to track this type of drift without access to the real class label. The UDD (Baier et al., 2021) uses the uncertainty of neural networks prediction, which authors argue it can be seen as the error rate. If the existing neural network starts to be uncertain of its predictions, a concept drift might have taken place in their method. STUDD uses the difference in prediction of two different models, instead of the error rate itself. The CWA (Lazarescu et al., 2004) tracks the differences between windows of different sizes. They try to track concept drift by measuring the distance between consecutive instances. This approach looks more suitable in the cases of *virtual concept drift*. Meta-ADD uses a meta classifier with meta-features from different concepts, but it needs to know the concepts in advance. Unsupervised methods for dealing with drift try to overcome the issue of labeling in data streams, which tend to be costly (Krawczyk et al., 2018), as they do not need access to the real labels of the data.

In Table 3.2 there is a concept matrix summarizing the cited methods in this section. Notice that, recently, authors have been giving more attention for unsupervised drift detection methods, due to the labelling problem in data streams.

3.3 ENSEMBLE METHODS

Ensemble approaches are among the most popular in ML works. Some of the online ensemble algorithms adapt to either bagging (Breiman, 1996) or boosting (Freund, 1995). Ensemble techniques in ML aim to combine classifiers in order to increase efficiency and accuracy (Kittler et al., 1998).

Table 3.2: Concept Matrix of Active Methods for Concept Drift.

Method	Source	Error rate-based	Data distribution-based	Multiple Hypothesis test-based	Unsupervised
DDM	Gama et al. (2004)	X			
CWA	Lazarescu et al. (2004)		X		X
EDDM	Baena-García et al. (2006)	X			
ADWIN	Bifet and Gavaldà (2007)	X			
STEPD	Nishida and Yamauchi (2007)	X			
HAT	Bifet and Gavaldà (2009)	X			
ECDD	Ross et al. (2012)	X			
SeqDrift1	Sakthithasan et al. (2013)			X	
SeqDrift2	Pears et al. (2014)			X	
HDDM	Frías-Blanco et al. (2015)	X			
FHDDM	Pesaranghader and Viktor (2016)	X			
FHDDMS	Pesaranghader et al. (2018b)	X			
FHDDMS$_{add}$	Pesaranghader et al. (2018b)	X			
FPDD	de Lima Cabral and de Barros (2018)	X			
FSDD	de Lima Cabral and de Barros (2018)	X			
FTDD	de Lima Cabral and de Barros (2018)	X			
MDDM	Pesaranghader et al. (2018a)	X			
GMM-VRD	Oliveira et al. (2019)	X			
ACDDM	Yan (2020)	X			
OASW	Yang and Shami (2021)	X			
UDD	Baier et al. (2021)	X			X
Meta-ADD	Yu et al. (2022)	X			X
GDDM	Yu et al. (2023)			X	
LD3	Gulcan and Can (2023)		X		X
STUDD	Cerqueira et al. (2023)	X			X

Street and Kim (2001) proposed Streaming Ensemble Algorithm (SEA), a well-known ensemble method for data streams, where different classifiers are built in different blocks of the data and will form an ensemble of constant size (when new labeled instances arrive, a new classifier takes place of another one based on its performance). Accuracy-Weighted Ensemble (AWE) is similar to SEA, but it weights the classifiers based on its expected prediction error on testing samples by assuming that the class distribution of the most recent batch of training data is similar to the distribution of the current test data (Wang et al., 2003). They prove that by weighting the classifiers by their expected accuracy on test data, we can improve an ensemble's accuracy. The authors also used instance-based pruning with the objective of selecting a subset of classifiers in the ensemble. Accuracy Updated Ensemble (AUE) is an extension of AWE that updates classifiers based on the current distribution (Brzeziński and Stefanowski, 2011). Brzezinski and Stefanowski (2014) proposed the online version of AUE, called Online Accuracy Updated Ensemble (OAUE), which utilizes incremental learners in order to enable the online processing of instances, and updates the weight of classifiers whenever a new instance arrives.

Dynamic Weighted Majority (DWM) creates and removes classifiers from the ensemble based on the changes in the performance of the ensemble (Kolter and Maloof, 2007). If the ensemble makes a mistake, a new classifier is added to it, and if a single classifier makes a mistake, its weight is reduced. The added classifiers have an initial weight of one, which will be reduced if it start making mistakes. If a classifier reaches a weight minor than a predefined threshold, it is removed from the ensemble. Any online classifier can be used with DWM.

The online version of bagging is the Oza Bagging (Oza, 2005). It considers the incoming instances as a Poisson($\lambda = 1$). This way, each incoming instance has a chance of being selected to train a new model to substitute another one in the ensemble. We also have Leveraging Bagging (Bifet et al., 2010b), which tried to improve Oza Bagging by increasing resampling and implementing output detection codes. To increase resampling, authors increased the λ value in the Poisson distribution, which will increase the probability of an instance being chosen to train a

classifier. The idea behind adding output detection codes is to add randomization at the output of the ensemble.

Minku and Yao (2012) used ensemble diversity when addressing concept drift. By taking into account that high diversity is better for dealing with drifting scenarios and low diversity is better for stable concepts (Minku et al., 2010), the authors proposed Diversity for Dealing with Drifts (DDD). DDD maintains a low and a high diversity ensembles. Both ensembles are updated with incoming instances, but only the low-diversity ensemble is used for prediction. The ensemble with high diversity starts to be used for predictions if concept drift is detected, and then it starts to learn with low diversity. The diversity of learning of the ensembles is controlled by a parameter λ, which dictates how likely is that an instance will be used for training, such as in Oza Bagging (Oza, 2005).

The Learn++ (Polikar et al., 2001) family of algorithms has as a characteristic an incrementally trained ensemble of classifiers. Here, we refer to Learn++.NSE (Elwell and Polikar, 2011), which uses weighted majority voting, where the weights are dynamically updated based on the error of classifiers on new and old instances. Gomes et al. (2017) combined Online Bagging (Oza, 2005) with HT (Domingos and Hulten, 2000) and proposed Adaptive Random Forest (ARF). Drift detectors are used along with each HT. When a warning level is reached, a new tree starts to be trained, and when drift is detected, the tree that originated the warning is replaced. They use Online Bagging (Oza, 2005) with Poisson($\lambda = 6$), instead of Poisson($\lambda = 1$). Just like in the Leveraging Bagging, it will increase the probability of an instance being selected for training.

The Adaptive Block Chunk Size, by Kozal et al. (2021), in order to keep ensemble models updated, trains new classifiers on new blocks and adds them to the group of classifiers, while obsolete models are removed. The size of the data chunks is adapted when a concept drift is detected. The authors argue that this decreases the restoration time of a model after a concept drift is detected.

Selective Ensemble-based Online Adaptive Neural Network (SEOA) uses online shallow and deep neural networks in different moments in the timestamp (Guo et al., 2021b). The shallow neural network is used for classifying the data stream when there is a concept drift. In stable concepts, the deep neural network is used for classification. The rationale is that shallow neural networks have less weights, and because of that they are able to quickly learn a new concept. On the other hand, a deep neural network needs more training data to be able to have a good performance. In SEOA, the decision of using either a shallow or a deep neural network is made by using a fluctuation metric, which calculates the difference of the error rate between the current timestamp to the last five consecutive timestamps (batch or online). If the fluctuation is strong, i.e., the difference in the error rate is too high, the deep neural network is *frozen*, and the shallow neural network is used for classification. The deep neural network is used instead of the shallow one if the fluctuation is small. There is also the moderate fluctuation level, where poor performing classifiers are frozen.

Liu et al. (2021) propose CALMID, an active learning framework that deals with concept drift in multiclass imbalanced data streams. It has an ensemble of classifiers and uses ADWIN as a drift detector. The idea is to question the label given through a hybrid labeling strategy. If the label is questioned, the correct label is given by experts and is combined with a new training sample. A label sliding window is used to estimate the current imbalance status, and a given instance is weighted according to the class imbalance status. The new weighted sample is used to update the ensemble. When concept drift is detected, a new classifier is trained in the training sample, where instances are weighted according to their arrival time. The new model replaces the worst one in the ensemble.

Hybrid Ensemble approach to concept drift-tolerate transfer learning (HE-CDTL) uses class and domain-wise weighted ensemble in order to track concept drift in the transfer learning domain (Yang et al., 2022). The authors proposed an adaptive weighted correlation alignment to better promote the transfer of knowledge between different domains. In HE-CDTL, a new classifier is created in each arrived data chunk (or batch) and combines it with different classifiers with the class-wise ensemble strategy. In Table 3.3 there is a concept matrix of the cited ensemble methods for concept drift.

Table 3.3: Concept Matrix of Ensemble Methods for Concept Drift.

Method	Source	Online	Trigger
SEA	Street and Kim (2001)		
AWE	Wang et al. (2003)		
OzaBag	Oza (2005)	X	
LevBag	Bifet et al. (2010b)	X	
AUE	Brzeziński and Stefanowski (2011)		
Learn++.NSE	Elwell and Polikar (2011)		
DDD	Minku and Yao (2012)	X	X
OAUE	Brzezinski and Stefanowski (2014)	X	
ARF	Gomes et al. (2017)	X	X
Adaptive Block Chunk Size	Kozal et al. (2021)		X
SEOA	Guo et al. (2021b)	X	
CALMID	Liu et al. (2021)	X	X
HE-CDTL	Yang et al. (2022)		

3.4 DYNAMIC SELECTION FOR CONCEPT DRIFT

As explained in Section 2.5, DS methods differ from the classic ML methods in that, instead of having a classifier, or ensemble of classifiers defined at the training phase, they are defined at the testing phase. Some authors have adapted DS for concept drift detection.

The Dynse Framework (Almeida et al., 2018) has already been explored in Section 2.5. It starts from the assumption that any DS technique can be used to cope with concept drift if it considers the time dependence. This is done by continuously adding classifiers trained in the most recent Batch to a pool of classifiers C, in which the competence of the classifiers will be measured in the RoC of an unlabeled instance. The chosen classifiers according to a Classification Engine will be used to compose an ensemble to classify the instance.

Double Dynamic Classifier Selection (DDCS) (Cavalheiro et al., 2021) is an example of DES framework to deal with Data Streams. The idea is to use incremental classifiers, like VFDT, in order to be able to continuously update the members of the ensemble whenever a new labeled instance arrives. Authors argue that this makes older classifiers perform better on stable concepts while creating new classifiers on the most recent chunks that are more proper to the current concept. This approach looks a lot like an online version of Dynse, with some differences from the proposed one in this work.

Jiao et al. (2022) aims at imbalanced data streams with concept drift, proposing Synthetic Minority Oversampling Technique with Adaptive Nearest Neighbors (AnnSMOTE) with the objective of generating new samples from the minority class instance that follows a new concept. This is done by considering the degree of change in distribution between the current and the

previous batch of data. The AnnSMOTE is applied to the last arrived batch of data, which is also the DSEL in their resulting framework Dynamic Ensemble Selection for Imbalanced Data Streams with Concept Drift (DES-ICD).

Dynamic Ensemble Selection based on Window over Imbalanced Drift Data Stream (DESW-ID) also aims at the imbalanced data stream issue, but instead of using techniques such as Synthetic Minimum Oversamples Technique (SMOTE), it adopts a resampling strategy with different λ values for the different classes (Han et al., 2023). The selection is performed by using a reverse search algorithm to determine the best number of classifiers in the ensemble, and the classifiers are ranked and sorted by the error rate. The ADWIN is used as a drift detector, and when a concept drift is detected, the adaptive window that is used to rank and train the classifiers is shrunk. The DESW-ID only works with binary classification problems.

Abadifard et al. (2023) uses Maximal Marginal Relevance, a diversity-based ranking method to perform the dynamic selection of the classifiers. Their method is called Dynamic Ensemble Diversification (DynEd). The drift is monitored by the ADWIN drift detector, and when it triggers a concept drift, a new classifier is created and trained on the last instances from a sliding window W and added to the pool. Classifiers in the pool are updated with resampling, in which the λ value is constantly updated depending on the value of the diversity and on how the accuracy changes. The classifiers are splitted in two groups by using K-Means clustering before applying the DS method.

In Table 3.4 are exposed some key concepts of the DS methods for concept drift described here. There are some distinguishing characteristics between them. For instance, the DES-ICD and DESW-ID aim at the imbalanced class problem in data streams, but they are not able to deal with multiclass classification problems directly. The DDCS, as Dynse, accepts different DS techniques, making it flexible, but does not has a drift detector and uses resampling for training. The DynEd also uses resampling for training and the ADWIN as a drift detector. The classifiers are sorted based on their accuracy to a window W and a diversity-based DS is performed. Its difference to the proposed Dynse+ is on the training technique and that our proposed framework can receive any DS method.

The proposed Dynse+ most differs on the aspect that it can accept any DS method and any drift detector. Thus, there is the possibility to choose the best ones according to the scenario, as different drift detectors may behave differently depending on the problem we are dealing with and the type of concept drift (Sakurai et al., 2023). Furthermore, we use a different training technique, that does not use a resampling strategy. The rationale is that we will have classifiers trained in different regions of the feature space, which will follow the availability with time.

Table 3.4: Concept Matrix of DS Methods for Concept Drift.

Method	Online	Trigger	Imb. Class	Multiclass
Dynse				X
DDCS	X			X
DES-ICD	X	X	X	
DynEd	X	X		X
DESW-ID	X	X	X	
Dynse+	X	X		X

4 METHODOLOGY

In this section, we describe how the Dynse framework was adapted to perform online processing, and how a drift detector was added, resulting in the proposed framework Dynse+. Further, we supply information about the datasets used in the experiments, and the experimental protocol for comparing the proposal with the state of the art.

4.1 PROPOSED METHOD

In this work, we propose Dynse+, an extension of the Dynse framework described in Section 2.5. We start by making Dynse able to perform online processing, as explained in Section 4.1.1. Incoming instances are used to maintain a DSEL that is used to estimate the competence of the classifiers in C through DS and to update a classifier that we call C_k. The size of the DSEL must be defined by the user, which can cause some problems when we take into account the stability-plasticity dilemma (Elwell and Polikar, 2011). A short window causes the classifiers to not have enough information to perform well. On the other hand, a large window may postpone the drift adaptation, if it has happened. This problem is mitigated by adding a trigger, i.e., a drift detector to Dynse, as explained in Section 4.1.2. We call the resulting framework with online processing and the addition of a drift detector as Dynse+Trigger, or simply Dynse+.

Some other DS methods for concept drift rely on a resampling strategy for updating the classifier in C. Dynse, on the other hand, maintains classifiers trained at different moments in time. Whenever a new batch of size b arrives, a new classifier is trained in it and added to C. The rationale is that, this way, we can have classifiers trained in different regions of the feature space. We can think of two characteristics of this approach. Firstly, we may have various classifiers trained in old concepts, which may be more useful in cases of *recurrent concept drift*, as when an old concept takes place, we already would have the classifiers trained in it. Secondly, the pool of classifiers C will need more time to create diversity, as the classifiers are added one by one by following the availability of the instances. The advantage of this approach is that we will have classifiers trained on different regions of the feature space, which will follow the availability with time. Thus, the DS method would selected the best classifiers for a specific region.

4.1.1 Adapting to Online Processing

The first proposal is to make Dynse able to update after receiving a single instance (online processing), i.e., there is no need to wait for a batch B_t to be completed. This is done by using an incremental learner as a base classifier, like the Hoeffding Tree (HT). The idea is to train each classifier in C on a maximum number of instances. When the incremental learner has received its maximum corresponding instances, its training is stopped and, when the next instance arrives, a new classifier is added to the pool and starts to be trained. This way, we have the benefits of a pool with old classifiers, while having a new classifier being continuously updated to the newest incoming instances. The scheme of this version of Dynse that can update with a single instance is in Figure 4.1. The classifier C_k refers to the last classifier added to C. Remind that after a predefined number of instances, the training of C_k is ended and a new classifier starts to be trained. If C reaches its maximum size when this happens, a classifier is dropped from C according to the pruning engine.

The incremental training proposed here differs from other methods that use resampling strategies, such as OzaBag (Oza, 2005; Gomes et al., 2017) on the fact that we use every available labeled instance to train only the last classifier added to C, instead of having a chance of training each one of them. The idea of doing so is to have classifiers trained on different regions of the feature space as they are available with time. Latter, the DS method will be in charge to select the best classifiers to compose the ensemble based on the RoC of the test instance. In this work, we will refer to the online version of Dynse as ODynse.

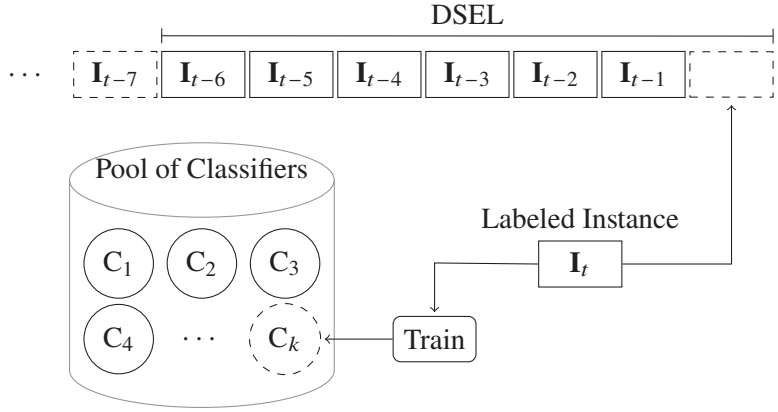


Figure 4.1: Online Dynse's Scheme.

4.1.2 Adding a Drift Detector

On Dynse, the DSEL is rather important for an efficient DS. However, the possibility of a concept drift brings the need to be conservative on how big it can be. If the DSEL grows too big, it is more likely to have instances from an old concept, which would be used to rank the classifiers. However, a short DSEL would not have enough information for an efficient selection of the classifiers, and useful data could be lost. We try to overcome this issue by adding a drift detector to Dynse. We can let the DSEL grow bigger when the concept is stable, and shrink it only when a concept drift is detected. Thus, we can have a more diverse DSEL in stable concepts, which might improve the efficiency of the DS. This approach may help in both *real and virtual concept drifts*, as in *virtual concept drifts* a larger DSEL will be better, as we can have instances in different regions of the feature space, but on the same concept, while the drift detectors, which most rely on the tracking of the error rate, will trigger whether a *real concept drift* has happened.

The proposed Dynse+ framework works as follows: an online classifier C_k is trained incrementally. Incoming instances are used to train C_k and are added to the DSEL, which grows until a predefined maximum size, and when it reaches its maximum size, the oldest instance is dropped whenever a new one arrives. The DSEL has a maximum size due to memory constraints, which we must be attentive to in streaming scenarios (Lu et al., 2020).

When a concept drift is detected, the DSEL is shrunk to N instances (another possibility is to use warning and drifting levels of drift detectors), and a new classifier is started to be trained and added to C. Thus, the drift detector assists in adapting the DSEL to a new concept. The rationale is that, when a drift detector does not trigger a drift, the concept is stable and the DSEL can grow. When concept drift is detected, otherwise, we must discard some instances in the DSEL to adapt to the new concept.

In Figure 4.2 there is a scheme of Dynse+. Notice that, when the drift detector triggers a concept drift, the DSEL is shrunk to N instances, as shown in Figure 4.3 considering $N = 3$, and

a new classifier starts to be trained. If that does not happen, we simply use the labeled instance I_t to train C_k.

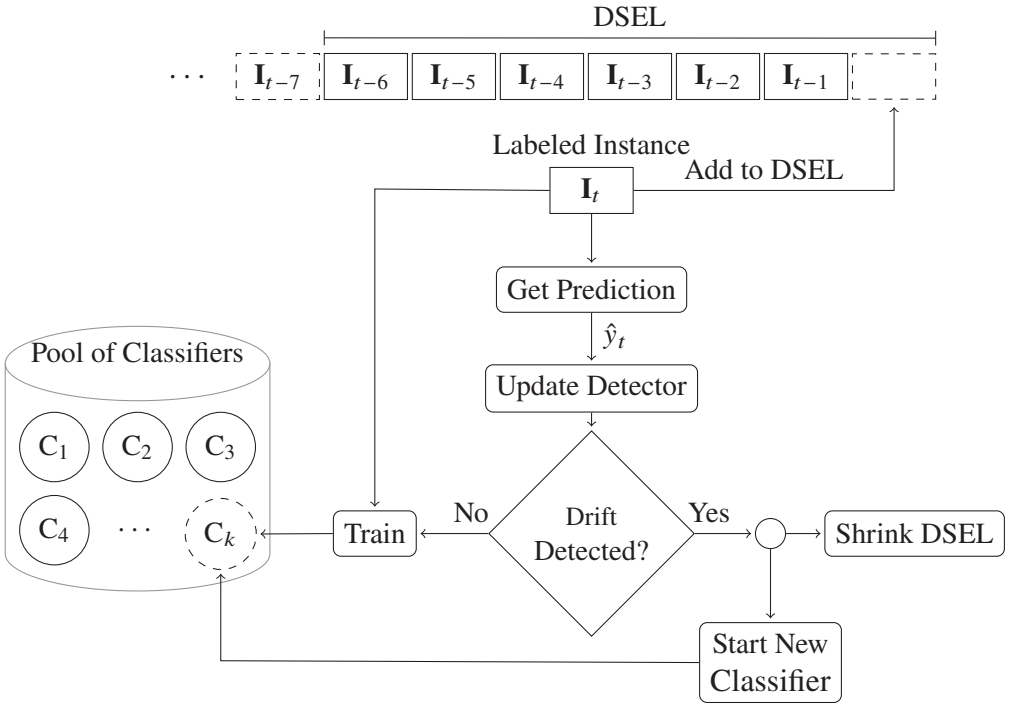


Figure 4.2: Dynse+'s Scheme When Concept Drift is Detected.

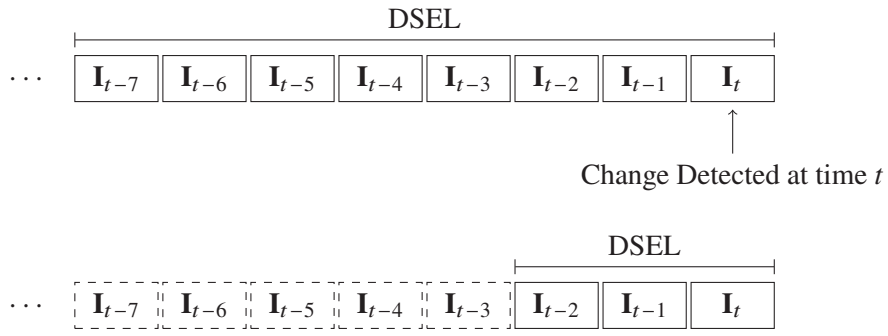


Figure 4.3: Dynse+'s Scheme of Adapting the DSEL When Concept Drift is Detected Considering N=3.

4.1.3 Classification

Just like in the original Dynse framework, when a new unlabeled instance arrives, the best classifiers according to the DS engine are selected to compose an ensemble to classify the newly arrived instance. For convenience, we show again the scheme of Dynse for classifying an unlabeled instance in Figure 4.4. The difference is that the competence of C_k will be measured in this phase.

The core part for the classification in the Dynse+ framework is the DS engine, which chooses the ensemble to classify an instance. The resulting ensemble C' is a subset of C, which follows $C' = DS(C, DSEL, I_t)$. I_t is an unlabeled instance that arrived at time t, and the DS engine can be any DES or DCS method. Notice that the DSEL is mandatory for an efficient DS. In this work, we use the KNORA-Eliminate (Ko et al., 2008) version proposed by Almeida

et al. (2018), which carries a slack variable s. We start with k neighbors, and C' will be the classifiers that can classify all of the k nearest instances to I_t with 100% accuracy. If no classifier satisfies this, the process is repeated to $k - s$ until there are no neighbours left. If no classifier can label the RoC resulting from any of the possible neighborhoods, then $C' = C$, i.e., the resulting ensemble is composed by all of the classifiers in C. The parameters of the proposed Dynse+ are presented in Table 4.1, and its pseudocode in Algorithm 1. We must state that it considers neighborhood-based DS methods, but it can be easily adapted to other types of DS (Guo et al., 2021a; Davtalab et al., 2024).

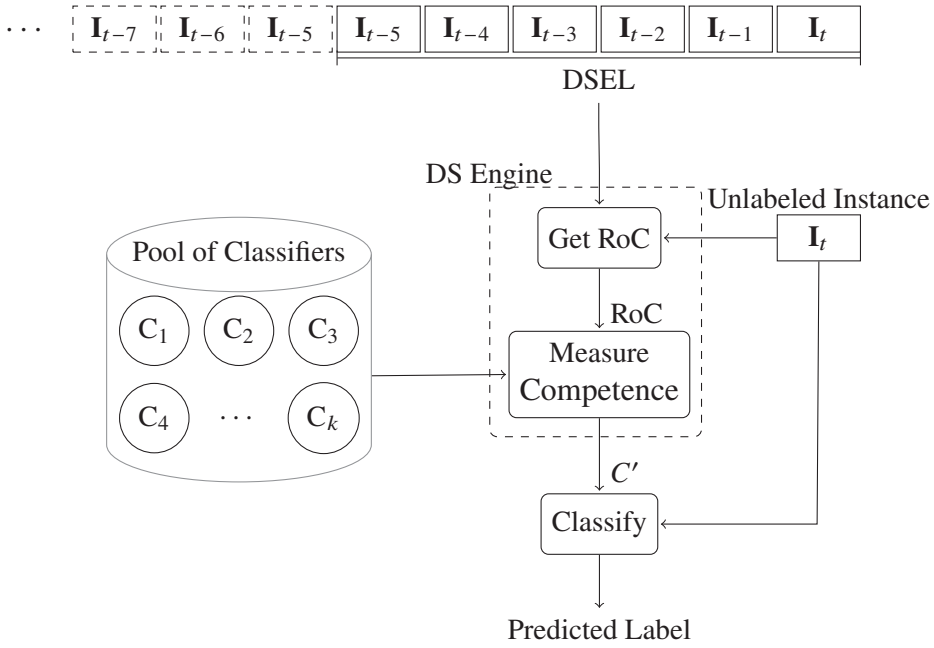


Figure 4.4: Dynse's Scheme for Classifying an Instance.

Table 4.1: Dynse+'s Parameters.

Parameter	Description
Base Classifier (BC)	Base classifier to be used that will compose C.
Maximum Size of DSEL (M)	Maximum Size of the DSEL
Minimum Size of DSEL (N)	Size that the DSEL will be shrunk to after drift detection
DS Engine (DS)	Dynamic Selection Engine for selecting classifiers to form the ensemble of classifiers.
Pruning Engine (PE)	The method utilized to prune classifiers out of the pool of classifiers.
Number of Classifiers (D)	Maximum number of classifiers in the pool of classifiers
Change Detector (CD)	Concept Drift Detector

When we receive a labeled instance, it is added to the DSEL, in step 5. If the DSEL had its maximum size before the arrival of the new instance, the oldest instance is removed from

```
input : Stream of Instances {I_1, I_2, ..., I_t},
        Maximum Pool Size (D),
        Maximum Size of DSEL (M),
        Minimum Size of DSEL (N),
        Dynamic Selection Method (DS),
        Pruning Engine (PE),
        Base Classifier (BC),
DSEL ← ∅
C ← ∅
foreach Instance I ∈ Stream do
    if I is labeled then
        DSEL ← DSEL ∪ I
        if |DSEL| > M then
            removeOldestInstance(DSEL)
        if C is ∅ then
            C_k ← startNewClassifier(BC)
            C ← C ∪ C_k
        updateDriftDetector(I)
        if driftIsDetected() then
            shrinkDSEL()
            C_k ← startNewClassifier(BC)
            C ← PE(C, DSEL, C_{k-1}, D)
            C ← C ∪ C_k
        if lastClassifierIsTrainedOnMaximumInstances() then
            C_k ← startNewClassifier(BC)
            C ← PE(C, DSEL, C_{k-1}, D)
            C ← C ∪ C_k
        trainLastClassifier(I)
        updateCompetences(C, I)
    else
        RoC← kNN(DSEL, I)
        measureCompetence(C_k, RoC)
        E_I ← DS(RoC, C)
        I_class ← classify(I, E_I)
        makeAvailable(I_class)
```

Algorithm 1: The Dynse+ algorithm

the DSEL in step 7. In step 9, a new classifier is started if there is no classifier in C, i.e., the training has just started. The drift detector is updated in step 11, and if concept drift is detected, the DSEL is shrunk in step 13, a new classifier will start to be trained in step 14, and we prune the classifier according to the pruning engine in step 15 if C was full. If C_k was trained on a maximum number of instances, its training is stopped and a new incremental classifier starts to be trained in step 18, which becomes the new C_k. Then, the pruning is done in step 19, if C is full. Finally, C_k is trained on the newly arrived labeled instance in step 21 and the competences of the classifiers in C are measured to I_t, except for the C_k, which competences are updated on the testing phase. If the new arriving instance is not labeled, we get its RoC in step 24, measure the competence of C_k on it in step 25, and build the ensemble according to the DS engine in step 26, which classifies the instance in step 27. Its label then becomes available for the user for further evaluation in step 28.

4.2 EXPERIMENTAL PROTOCOL

In this section, we describe the datasets used for the tests, give information such as the number of features, classes, instances, and type of concept drift, if known. We also present the experimental protocol for the tests, which were followed with the objective of comparing the proposal to other methods in the state of the art.

4.2.1 Datasets

Literature divides concept drift datasets into two: Synthetic datasets and real-world datasets. Some authors have generated Synthetic datasets to have better control over how data distribution changes (Schlimmer and Granger, 1986; Domingos and Hulten, 2000; Gama et al., 2004), so they are able to evaluate how a given method behaves in a type of concept drift. Real-world datasets give us more challenges as most real-world environments are unpredictable, and most of the times we are not sure on how many drifts have happened, and its magnitude or severity (Lu et al., 2020; Minku et al., 2010).

In this section, we present some datasets encountered in the concept drift literature, and these datasets were used in the experiments of Dynse+. For the familiar reader, the datasets are listed in Table 4.2.

Table 4.2: Datasets.

Dataset	Reference	Drift Types	# Inst.	# Feats	# Classes
Synthetic					
STAGGER	Schlimmer and Granger (1986)	Abrupt; Recurrent	Custom	3	2
SEA Concepts	Street and Kim (2001)	Abrupt; Recurrent	Custom	3	2
RandomRBF	Bifet et al. (2009)	Abrupt; Gradual; Incremental	Custom	Custom	Custom
LED	Breiman et al. (1988)	Abrupt; Gradual	Custom	24	10
Hyperplane	Hulten et al. (2001)	Incremental; Gradual	Custom	10	2
Random Tree	Domingos and Hulten (2000)	Abrupt; Recurrent	Custom	Custom	Custom
Sine	Gama et al. (2004)	Abrupt; Recurrent	Custom	2	2
Agrawal	Agrawal et al. (1993)	Abrupt	Custom	9	2
Real-World					
Forest Covertype	Blackard (1998)	Unknown	581,012	54	7
Electricity	Harries et al. (1999)	Unknown	45,312	8	2
Airlines	Bifet et al. (2010a)	Unknown	539,384	7	2
NOAA	Ditzler and Polikar (2013)	Unknown	18,159	8	2
Outdoor	Losing et al. (2015)	Unknown	4,000	21	40
Ozone	Zhang et al. (2008)	Unknown	2,536	72	2
Insects	Souza et al. (2020)	Abrupt; Gradual; Incremental; Recurrent	905,145	33	6
Gas Sensor	Vergara (2012)	Unknown	13,910	128	6
Adult	Becker and Kohavi (1996)	Unknown	48,842	14	2
Yeast	Nakai (1996)	Unknown	1,484	8	10
Nursery	Rajkovic (1997)	Unknown	12,960	8	5
Letters	Slate (1991)	Virtual	20,000	16	26
Digits	Alpaydin and Kaynak (1998)	Virtual	5,620	64	10
Pen Digits	Alpaydin and Kaynak (1998)	Virtual	10,992	16	10
Dry Bean	Koklu and Özkan (2020)	Virtual	13,611	16	7
Rice	Cinar and Koklu (2019)	Virtual	3,810	7	2

4.2.1.1 Synthetic Datasets

Synthetic datasets bring ease to control the environment. We know when a concept drift happens, and the type of concept drift. This way, we are able to understand how one method behaves in known scenarios, and makes the evaluation of drift detectors easier, as we know exactly the points where the concept drifts.

STAGGER: it has three features: size $\in$ {small, medium, large}, color $\in$ {red, blue, green} and shape $\in$ {circle, square, triangle} (Schlimmer and Granger, 1986). The concepts are divided into three, which the positive class is given if:

1. size = small $\wedge$ color = red.
2. color = green $\vee$ shape = circle.
3. size = (medium $\vee$ large).

Concepts are changed abruptly in the data stream. For the tests made in this work, 30,000 instances of the STAGGER dataset were generated, with the concept changing every 10,000 instances. There was also generated a version of the STAGGER dataset with a *recurrent concept drift*, where 40,000 instances were generated and the concept drifts every 2,000 instances.

SEA Concepts: The SEA Concepts dataset (Street and Kim, 2001) also has concepts that change abruptly. It has three features f_1, f_2, and f_3 that vary in the interval $[0, 10]$. Their concepts say that a given instance belongs to the positive class if:

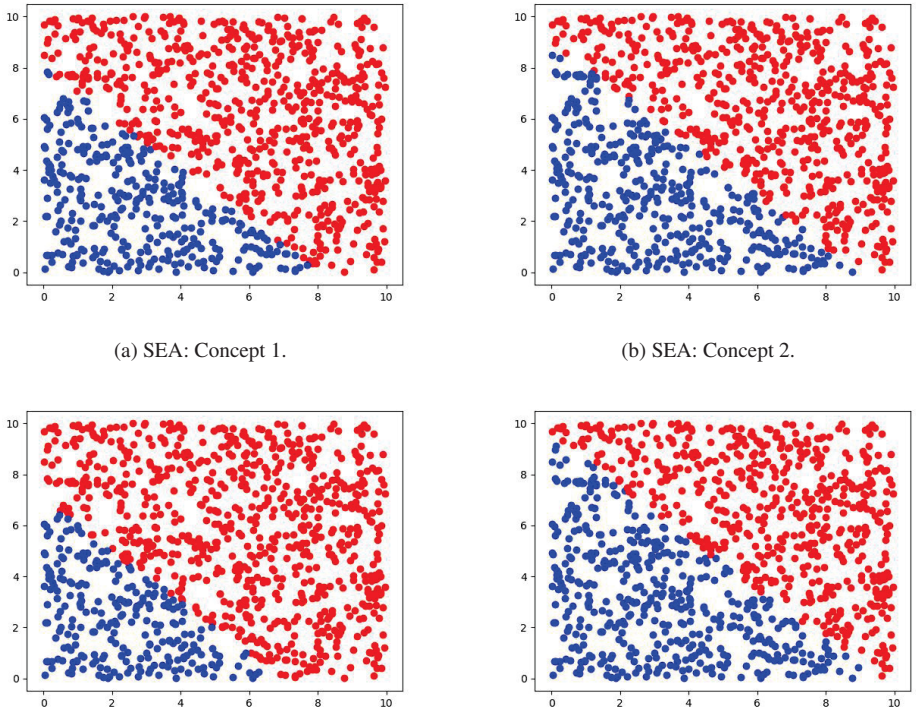
1. $f_1 + f_2 \leq 8$.
2. $f_1 + f_2 \leq 9$.
3. $f_1 + f_2 \leq 7$.
4. $f_1 + f_2 \leq 9.5$.

Notice that f_3 is noise. Figure 4.5 shows how the decision boundaries of the SEA Concepts dataset change between the different concepts. We have generated two different versions of SEA: The first one with 40,000 instances with the concept drifting abruptly every 10,000 instances, and the other with *recurrent concept drift*, which we called Sea-Rec, where we also generated 40,000 instances, but the concept drifted every 2,000 instances in a cycle.

LED: The LED dataset (Breiman et al., 1988) is available at the UCI Repository (Dua and Graff, 2017). In this version, we have 24 boolean features, of which only 7 of them are relevant, and we have 10 different classes. It has a noise of 10%, and concept drift is introduced by changing the relevant attributes. In this work, we have generated 20,000 instances on this dataset, with a gradual drift starting on the 10,000th instance, with a width of 500 instances. Thus, we have instances from two concepts being sent to our model between the 10,000th instance to the 10,500th instance. After that, the second concept will have taken place completely. The concepts differ between themselves by which are the significant features.

RandomRBF: The Random Radial Basis Function, or RandomRBF (Bifet et al., 2009), generates a fixed number of random centroids where each one has a random position, a single standard deviation, a class label, and a weight. New instances are generated by randomly selecting a center. Centers with higher weights have more chances to be chosen. The instance class is given by the centroid chosen, and drift happens by moving the centroids at a constant speed, defined by a parameter. We have 20,000 instances generated, and the concept drifts incrementally throughout the whole stream.

Hyperplane: The Hyperplane dataset is generated in a d-dimensional set that satisfies $\sum_{i=1}^{d} w_i x_i = w_0$. For a start, a reference hyperplane is generated with some weight w, except for w_0. Let $s = \sum_{i=1}^{d} w_i x_i$. If $|s| \leq 0.1 \times w_0$, the example is labeled as positive. If $|s| \leq 0.2 \times w_0$ it is labeled as negative. The feature x_i ranges in the interval $[0, 1]$. For this dataset, 20,000 instances



(a) SEA: Concept 1.

(b) SEA: Concept 2.

(c) SEA: Concept 3.

(d) SEA: Concept 4.

Figure 4.5: SEA Concepts' Decision Boundaries.

were generated, with a gradual concept drift happening in the 10,000th instance with a width of 500 instances (Hulten et al., 2001).

Sine: The Sine dataset has two relevant attributes varying in the range $[0, 1]$. It has four different concepts. In the first, the positive class is given for the labels below the curve of $y = sin(x)$. The second concept is the contrary of the first. In the third concept, the classification function is positive if $y < 0.5 + 0.3 \times sin(3\pi x)$, and it is reversed for the fourth concept (Gama et al., 2004). Two versions of this dataset were generated in the experiments, just like in the SEA dataset. In the first one, 40,000 instances are generated with the concept drifting every 10,000 instances. In the second version, we have a recurrent concept drift, where 40,000 instances are generated, and the concept drifts every 2,000 instances.

Agrawal: It generates nine features, of which six are numeric and three categorical. There are 10 different classification functions that are drifted with time. Such functions put the instances in one of two different groups, A or B, which were listed by Agrawal et al. (1993). We have generated 40,000 instances with the concept drifting abruptly every 4,000 instances.

4.2.1.2 Real-world Datasets

Real-world datasets are different from synthetic datasets, as we usually do not know the nature of concept drift, or how it happens. On these scenarios, we usually do not know the real starting and ending points a concept drift took place, and we may have various types of concept drift happening (Lu et al., 2020).

Forest Covertype: The Forest Covertype dataset, available in the UCI repository (Dua and Graff, 2017), contains cartographic variables in 30 x 30-meter cells in forests located in the Roosevelt National Forest, where there had minimal disturbance caused by human action. It

includes 10 continuous features and 44 binary features (0, 1). It has 581,012 instances, and the objective is to predict one between 7 different cover types.

NOAA: The NOAA dataset (Ditzler and Polikar, 2013), also known as the Nebraska Weather dataset, contains climate data in the Offurt Air Force Base in Nebraska in an interval of over 50 years, which may give us cyclic and long-term climate changes. Features include weather information such as temperature, visibility, pressure, and wind speed. The main task is to predict if it has rained that day.

Insects: In this dataset, the authors took into account that the behavior of different types of insects may change when characteristics of the environment, such as temperature and humidity, change. There are five different types of insects giving five different classes in this dataset. The temperature changes in different manners, creating different datasets based on how the change in the temperature was performed. We have Abrupt(A), Gradual(G), Incremental(I), and Recurrent(R) concept drifts. The authors made available versions with balanced(B) and imbalanced(I) classes (Souza et al., 2020). Information about each version of the Insects dataset is in Table 4.3

Dataset	# Instances
Insects-AB	52,848
Insects-AI	355,275
Insects-GB	24,150
Insects-GI	143,323
Insects-IB	57,018
Insects-II	452,044
Insects-IAB	79,986
Insects-IAI	452,044
Insects-IRB	79,986
Insects-IRI	452,044

Table 4.3: Instances of Each Version in the Insects Dataset.

Yeast: The Yeast dataset (Nakai, 1996) has 1,484 instances with 8 features containing different measures regarding the domain of proteins, and 10 classes. The objective is to predict where the proteins are localized.

Adult: It has 48,842 instances with 14 features including the education level, marital status, occupation, race, etc. It is a binary classification problem, where the objective is to predict if a person makes over 50K a year (Becker and Kohavi, 1996).

Ozone: A binary classification problem with 2,536 instances and 72 features which contains information such as temperature and wind speed. The objective is to differentiate between two different ground ozone levels (Zhang et al., 2008).

Gas Sensor: 13,910 instances and 128 features, gathered by using 16 chemical sensors. The objective is to predict one between 6 gases at various levels of concentrations (Vergara, 2012).

Nursery: It has 12,960 instances and 8 features, which include parent's occupation, number of children, housing conditions, etc. This dataset was developed to rank candidates for nursery school. The objective is to rank applications for nursery school into 5 different classes (Rajkovic, 1997).

Outdoor: It contains data of 40 outdoor objects such as dogs, shoes, basketball, etc., in sunny and cloudy environments. It has 21 features and 4,000 instances. The objective is to predict which is the outdoor object (Losing et al., 2015).

Letters: In the Letters dataset (Slate, 1991), the objective is to differentiate between the 26 capital letters in the alphabet in black-and-white pixel displays from 20 different fonts, which were converted to 16 numerical attributes. Data was randomly distorted to generate 20,000 instances in total.

Digits: Consists of 5,620 instances of handwritten digits from 43 different people in an 8x8 matrix in the range $[1, 16]$. The objective is to predict one digit between 0 and 9 (10 classes) (Alpaydin and Kaynak, 1998).

Pen Digits: 10,992 instances from 44 different writers. The digits were captured in a sensitive tablet. The dataset contains 16 features in the range $[1, 100]$ and the objective, just like in the Digits dataset, is to predict one digit between 0 and 9 (Alpaydin and Kaynak, 1998).

Dry Bean: It contains 13,611 instances containing 16 features informing shape, type, and structure. The objective is to predict one between 7 different dry beans (Koklu and Özkan, 2020).

Rice: It has 3,810 instances, with 7 features that supply information such as area, perimeter, extent, etc. The objective is to predict one between two different species of grains of rice (Cinar and Koklu, 2019).

4.2.2 Experiments Guidelines

The main purpose of the tests in this work is to compare Dynse+ with ODynse, the original Dynse, and other state-of-the-art methods. As explained in Chapter 4, Dynse+ updates incrementally and aims to improve the quality of the DSEL through the track of concept drift.

Before comparing Dynse+ to the other methods, there was performed an analysis of its hyperparameters to choose the best ones. Each of the hyperparameters were assessed individually, and those that achieved the highest accuracy have composed a default version to be compared to different works in the literature. To do this analysis, the chosen real-world datasets were the NOAA, Nursery, and Electricity. The Electricity, even though it is not good for comparing methods from the concept drift perspective, as we explain in Section 5.1.1, was used for the hyperparameter analysis.

The synthetic datasets were used to evaluate the effectiveness of different drift detectors as well, in addition to the classification accuracy. The chosen synthetic datasets were the SEA Concepts and the STAGGER. The measures for evaluating drift detectors were explained in Section 2.4.1. The value for the Δ parameter of detection interval was set to 250 (Pesaranghader and Viktor, 2016). The experiments were done in a test-then-train manner, and there are reported both the overall accuracy and the prequential accuracy with a sliding window of 1000 instances, as with it we can get a plot of accuracy over time. For some of the experiments, we have reported the average processing time as well. After the hyperparameter tuning, the resulting Dynse+ framework is compared to ODynse, Dynse, and other methods from the literature by considering the same metrics of overall accuracy and prequential accuracy.

Tests considering delayed and partial labels were performed as well, where we have considered a delay of 50 instances, and only 1 every 2 labels are available for training, i.e., 50% of the datasets are labeled. The rationale is that, in real-world environments, the access to the true labels of the data is limited, and might arrive with a delay (Gomes et al., 2017; Cerqueira et al., 2023).

The datasets that were utilized for the hyperparameter tuning are in Table 4.4, and for comparing Dynse+ to the state of the art are in Table 4.5. These datasets have passed through

a validation step involving a Cramer's V autocorrelation test and tests comparing methods for dealing with concept drift to naïve methods. To the static datasets (Letters, Digits, PenDigits, DryBean, and Rice), there was induced a *virtual concept drift* as done by Almeida et al. (2018), where, on the first step, a random instance is chosen from the dataset. Then, the 199 nearest neighbors to it are sent first on the stream, making 200 instances at each step. This is repeated until the dataset is empty. We have chosen 200 instances per step because it is also the initial training size that we have used.

Table 4.4: Datasets Used for Hyperparameter Tuning.

Dataset	Source
Electricity	Harries et al. (1999)
NOAA	Ditzler and Polikar (2013)
Nursery	Rajkovic (1997)
SEA	Street and Kim (2001)
STAGGER	Schlimmer and Granger (1986)

Table 4.5: Datasets Used for the Tests Comparing Dynse+ to Other Methods.

Dataset	Source
Airlines	Bifet et al. (2010b)
Ozone	Zhang et al. (2008)
Adult	Becker and Kohavi (1996)
Insects-AB	Souza et al. (2020)
Insects-AI	Souza et al. (2020)
Insects-GB	Souza et al. (2020)
Insects-GI	Souza et al. (2020)
Insects-IB	Souza et al. (2020)
Insects-II	Souza et al. (2020)
Gas Sensor	Vergara (2012)
Sine	Gama et al. (2004)
Sine-Rec	Gama et al. (2004)
SEA-Rec	Street and Kim (2001)
STAGGER-Rec	Schlimmer and Granger (1986)
Agrawal	Agrawal et al. (1993)
Hyperplane	Hulten et al. (2001)
LED	Breiman et al. (1988)
RandomRBF	Bifet and Gavaldà (2009)
Letters	Slate (1991)
Digits	Alpaydin and Kaynak (1998)
Pen Digits	Alpaydin and Kaynak (1998)
Dry Bean	Koklu and Özkan (2020)
Rice	Cinar and Koklu (2019)

Further, the Friedman-Nemenyi test described by Demšar (2006) was performed on the results comparing Dynse+ to Dynse, ODynse, and other methods in the literature. The equation

used for calculating the Critical Difference is the same as presented by Demšar (2006), shown in Equation 4.1. All of the tests were run 10 times.

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}} \tag{4.1}$$

5 EXPERIMENTS AND RESULTS

5.1 PRELIMINARY TESTS

In this Section, there were performed various tests to eliminate datasets that do not seem to have concept drift. To do so, a Cramer's V autocorrelation test is performed. The objective is to check how likely is that a class at the time t is the same as in $t - k$, where k is the lag. The lagged Cramer's V test was explained by Weiss (2018).

In addition to that, we have compared some naïve methods, such as a classifier that predicts the instance I_t to the same class as the instance I_{t-1}, as well as a HT classifier with a random trigger, that triggers a concept drift with $p\%$ chance. These methods were compared to a HT classifier along with the DDM detector. The rationale is that, if the naïve methods can get better results than methods that were built to cope with concept drift, either that dataset is not good for evaluating drifting scenarios, or the used method is not able to do so. These test are a reproduction of the ones made by Almeida et al. (2020).

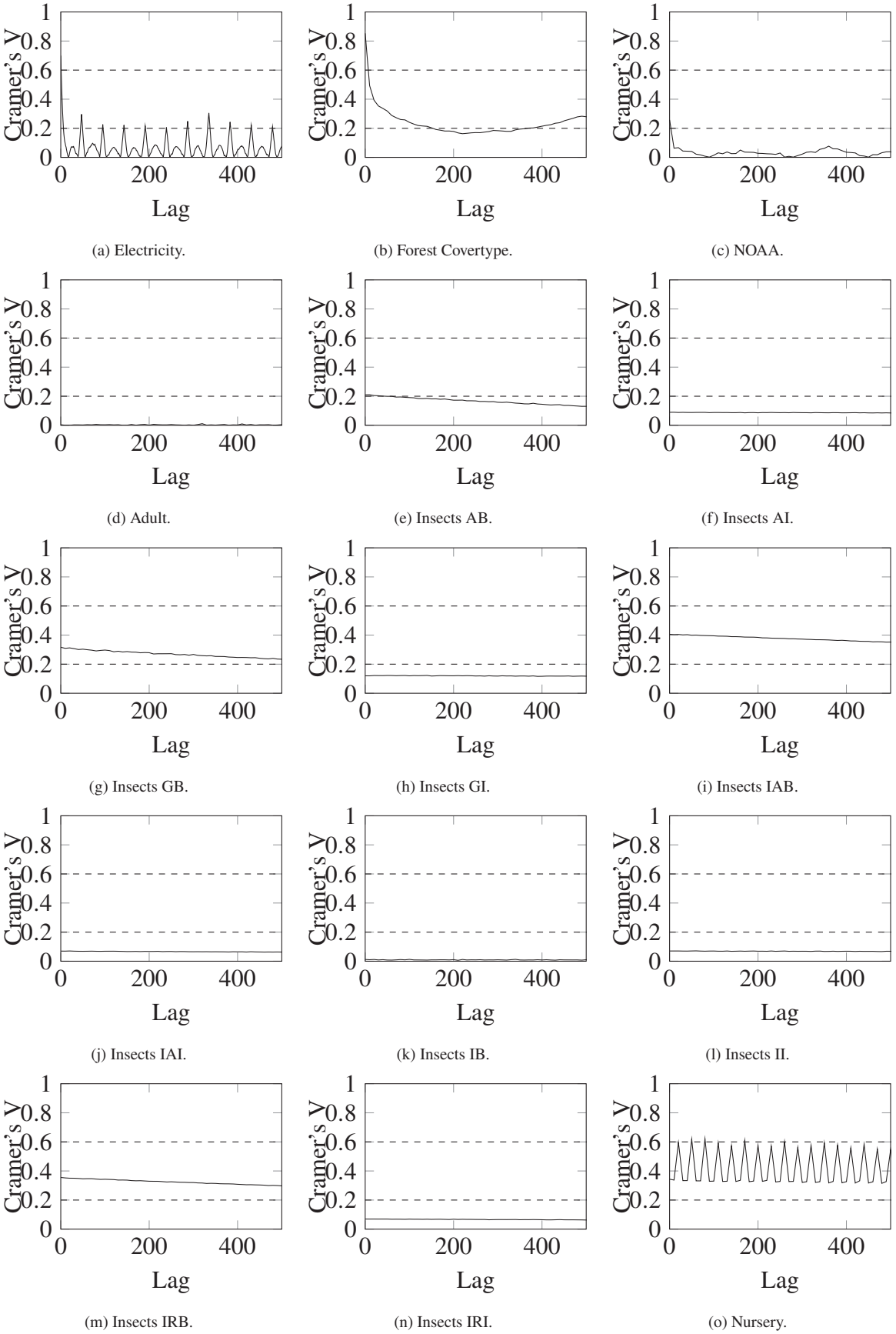
5.1.1 Cramer's V Autocorrelation Test

In this Section is performed a Cramer's V autocorrelation test on the datasets presented in Table 4.2. The lagged version of Cramer's V is used. The objective here is to check the correlation level of the labels of the instances k lag far from each other. A high autocorrelation shows that we may have some time dependence on the datasets, i.e., we would be able to label an instance by simply looking to the past. The graphs of the Cramer's V autocorrelation test are exposed in Figure 5.1. Rea and Parker (1992) describes strengths of autocorrelation for the Cramer's V test, which are exposed in Table 5.1. To simplify, in the graphs we draw horizontal lines that divide the results between a weak correlation, when $v < 0.2$, a moderate correlation when $0.2 \leq v < 0.6$, and a strong correlation when $v \geq 0.6$. Static datasets, such as Digits, Letters, and Dry Bean, on which *virtual concept drift* is induced, are excluded from these tests.

Table 5.1: Strength of Association of Different Intervals in the Cramer's V Test.

Cramer's V Interval	Strength of Association
$[0, 0.1)$	Negligible
$[0.1, 0.2)$	Weak
$[0.2, 0.4)$	Moderate
$[0.4, 0.6)$	Relatively Strong
$[0.6, 0.8)$	Strong
$[0.8, 1.0]$	Very Strong

We must state that some time dependency is expected in Data Streams, especially on pricing forecast, which is the case of the Electricity dataset, which may explain the high value of v for the lag $k = 1$. That makes sense because in this case we are trying to predict if a price will go up or down, and this type of scenario usually follows the inertia. We can also see a peak in autocorrelation for the Electricity dataset every after 48 lags, i.e., every after 24 hours, as shown in Figure 5.1(a). It is also expected that there are specific moments in the day that we may have prices going up, at rush hours, to cover the high demand, or going down to cover a



(a) Electricity. (b) Forest Covertype. (c) NOAA.

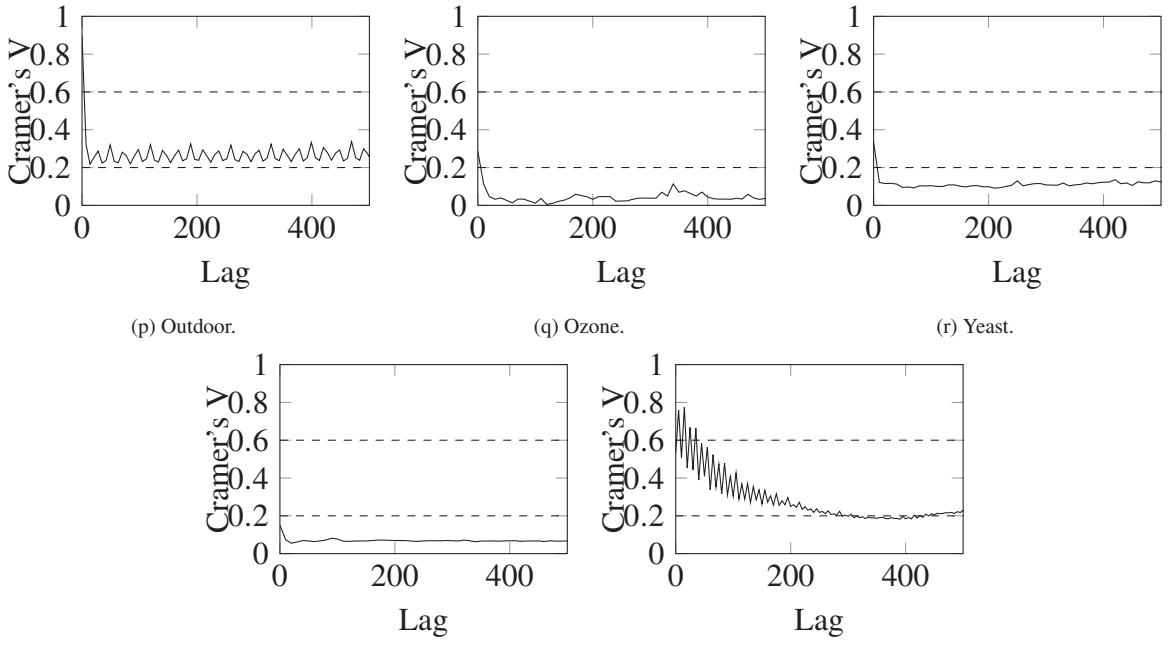
(d) Adult. (e) Insects AB. (f) Insects AI.

(g) Insects GB. (h) Insects GI. (i) Insects IAB.

(j) Insects IAI. (k) Insects IB. (l) Insects II.

(m) Insects IRB. (n) Insects IRI. (o) Nursery.

Figure 5.1: Lagged Cramer's V on Datasets.



(p) Outdoor. (q) Ozone. (r) Yeast.

(s) Airlines. (t) Gas Sensor.

Figure 5.1: Lagged Cramer's V on Datasets (continued).

low demand. The Forest Covertype dataset also had a strong autocorrelation for $k = 1$, and a moderate autocorrelation for most values of k.

We can also cite weather forecasting, like the NOAA dataset, in Figure 5.1(c). We can have an algorithm that predicts if tomorrow is going to rain based on if it has rained today. If our ML model cannot surpass this algorithm, then there is no reason for using it. The NOAA dataset had a weak correlation for every tested value of k, thus this is good evidence that there is not much correlation between the classes over time. The same can be said about the Adult dataset in Figure 5.1(d).

Some of the Insects datasets, such as AB (Figure 5.1(e), GB (Figure 5.1(g), IAB (Figure 5.1(i), and IRB (Figure 5.1(m) had moderate correlation in some or all values of k tested. The other datasets from the Insects benchmark had a weak correlation for all k values. The Nursery dataset (Figure 5.1(o) had a moderate correlation most of the time, with some peaks on the edge of the strong level of correlation. The outdoor dataset (Figure 5.1(p)) presented a moderate correlation. The Ozone (Figure 5.1(q)), Yeast (Figure 5.1(r)), and Airlines (Figure 5.1(s)) presented a weak correlation between the labels, and the Gas Sensor (Figure 5.1(t)) had a strong correlation for some early values of k and a moderate correlation in most lags.

We see that many of the datasets used in the literature present some autocorrelation between the labels. That by itself does not concludes that a dataset should not be used for comparing methods from the concept drift perspective, but a high autocorrelation is a strong indicative to be attentive to. To complete this analysis, there were performed tests considering naïve methods in Section 5.1.2.

5.1.2 Naïve Methods

In this Section, tests were performed to understand how naïve methods behave on the datasets in Table 4.2, except the ones with induced *virtual concept drift*. The objective of these tests, in combination with the autocorrelation test performed in Section 5.1.1, is to help us deciding if the datasets are proper for evaluating models from the concept drift perspective. A HT classifier

along with the DDM detector is compared to naïve classifiers, as done by Almeida et al. (2020) and Bifet et al. (2013). These methods include a naïve classifier, which labels the instance at time t with the same label of the instance at time $t-1$, and a HT classifier along with a random trigger, which randomly triggers a concept drift with p% chance. If a method made for dealing with concept drift is not able to surpass the accuracy of these naïve methods, either that method is not proper to do so, or the dataset is not proper to be used for evaluating methods for dealing with concept drift (Almeida et al., 2020). Results are presented in Table 5.2.

Table 5.2: Accuracies of HT+DDM and Naïve Methods.

Dataset	HT+DDM	HT+RT ($p = 1$)	Naïve Classifier
Electricity	84.15	83.27	**85.33**
Forest	84.94	85,60	**95.06**
NOAA	71.35	**71.49**	68.03
Insects AB	61.46	**61.49**	28.98
Insects AI	**64.02**	62.23	29.16
Insects GB	**68.03**	66.01	36.59
Insects GI	**65.10**	61.97	30.16
Insects IAB	65.02	**65.66**	42.38
Insects IAI	59.67	**60.42**	28.12
Insects IB	**55.48**	47.17	16.08
Insects II	**67.16**	60.46	28.20
Insects IRB	60.71	**66.76**	40.45
Insects IRI	59.02	**60.49**	28.18
Insects OOC	**49.98**	44.24	13.06
Gas Sensor	79.83	**87.32**	59.20
Airlines	**65.33**	62,01	58.05
Ozone	93.27	**93.59**	91.55
Adult	**84.37**	76.45	63.44
Yeast	46.57	49.63	**51.65**
Outdoor	59.26	57.25	**90.25**
Nursery	**91.40**	81.56	23.93

Starting with the Electricity dataset, we can see that the Naïve classifier performed better than HT with DDM and with the random trigger. The same happened on the Forest dataset. This is evidence that there is a high dependence of the current class on the last one, as we saw in Section 5.1.1. The random trigger got a better accuracy than DDM on the NOAA dataset, but the Naïve classifier did not perform better than any of them.

On the Insects benchmark, the Naïve Classifier was not better in any of the scenarios, and on most of them the HT+DDM classifier was the best one, but the HT with a random trigger got some of the best accuracies, even though in most of them there was not a huge difference. The Insects where the HT+RT was the best accurate were the ones with *recurrent concept drift* (Insects IRB and IRI) and with an *incremental and abrupt concept drift* (Insects IAB and IAI), which are discarded, as we already have six versions of the Insects dataset as well. The Yeast and Outdoor datasets were discarded, as the Naïve methods had a significantly better accuracy. We will maintain the Gas Sensor, as the naïve classifier did not surpass any of the methods.

5.2 HYPERPARAMETER ANALYSIS

In this Section we analyze the effects of different hyperparameters in Dynse+. The hyperparameters to be tuned are 1) The F parameter, which denotes how long Dynse+ can stay without start training a new classifier; 2) the drift detector that will support on adapting the DSEL; 3) The Maximum size of the DSEL (M); 4) The minimum size of the DSEL (N).

We have an initial set of parameters, exposed in Table 5.3, which were tuned throughout the experiments. The maximum size of the DSEL (M) is set to 4000 instances, and its minimum size (N) is 200 instances. The drift detector, for a start, is the DDM. The DS method utilized is the KNORA-Eliminate (Ko et al., 2008) version proposed by Almeida et al. (2018), with $k = 9$ and the slack variable $s = 2$. The maximum size of C is to set to $D = 75$. Regarding the distance function, in an early paper of us, we have come to the conclusion that the Canberra Distance is the best one for streaming scenarios. However, in this work we use the Euclidean Distance, as it is the most used in the literature

Table 5.3: Initial Set of Hyperparameters

Parameter	Initial Value
F	200
Drift Detector	DDM
M	4000
N	200

5.2.1 Frequency of Training New Classifiers

Firstly, let us analyze the impact of the F parameter on the accuracy. The F parameter dictates how many instances without drift detection we can have without a new classifier being added to the pool C. After F instances without drift detection or a new classifier being trained, a new classifier will start to be trained even though we did not have a drift detection. If F is too high, we may have less diversity in the pool, as the classifiers are more likely to know instances from different regions in the feature space. For ensembles, it is preferred to have various classifiers trained in different regions of the feature space (Kittler et al., 1998). On the other hand, a low F may lead to underfitted models, even though the diversity would be higher.

The average accuracy for the selected datasets for each F value is presented in Table 5.4. As we can see, the F with the greatest accuracy was $F = 200$. Thus, it is the choice for the next experiments. We must state that this parameter does not affect the maximum or the minimum size of the DSEL.

Table 5.4: Accuracy (%) of Different F values on Dynse+DDM.

Dataset	50	100	200	500	1000
NOAA	**77.64**	77.50	77.23	77.26	77.04
Nursery	**93.67**	92.96	93.43	93.01	92.69
Electricity	82.96	83.43	**84.60**	83.82	84.12
SEA	86.92	**87.14**	87.12	87.04	**87.14**
STAGGER	**99.33**	99.29	99.28	99.29	99.30
Average	88.10	88.06	**88.33**	88.08	88.06

5.2.2 Concept Drift Detectors

Next, we perform the analysis on the drift detectors. The drift detectors considered for these tests are reported in Table 5.5. For a reminder, in addition to the average accuracy, we have used the measurements of True Positive (TP), False Positive (FP), and False Negative (FN) on the synthetic datasets, as explained in Section 2.4.1. As we are using the SEA and STAGGER datasets with abrupt concept drifts, the acceptable interval of TP Δ is set to 250. In addition to these measures, we also evaluate the average delay for detection of TPs of the detectors, if they had any. The delay is gathered at the moment the concept drift is detected, if it was a TP. Thus, we can understand if we may have instances from the old concept after the DSEL is shrunk. Remind that the minimum size of the DSEL, at the moment, is set to 200 instances – this parameter is tuned in Section 5.2.4.

The analysis of the drift detectors of the synthetic datasets were done separately to the real-world datasets, as in the latest we do not know for sure whether or where concept drift has happened. In practice, we want drift detectors that can track the drift at the right moments. According to Komorniczak and Ksieniewicz (2023), valuable detectors have a low number of FPs, while are still able to detect the moments of change.

Table 5.5: Tested Drift Detectors.

Detector	Reference
PHT	(Page, 1954)
DDM	(Gama et al., 2004)
EDDM	(Baena-García et al., 2006)
ADWIN	(Bifet and Gavaldà, 2007)
STEPD	(Nishida and Yamauchi, 2007)
ECDD	(Ross et al., 2012)
SeqDrift1	(Sakthithasan et al., 2013)
SeqDrift2	(Pears et al., 2014)
HDDMA	(Frías-Blanco et al., 2015)
HDDMW	(Frías-Blanco et al., 2015)
FHDDM	(Pesaranghader and Viktor, 2016)
FHDDMS	(Pesaranghader and Viktor, 2016)
RDDM	(Barros et al., 2017)
FHDDMS_{add}	(Pesaranghader et al., 2018b)
MDDMA	(Pesaranghader et al., 2018a)
MDDME	(Pesaranghader et al., 2018a)
MDDMG	(Pesaranghader et al., 2018a)

5.2.2.1 Synthetic Datasets

The results of the tests of different drift detectors for the synthetic datasets are in Table 5.6. We can see that the detectors with the lowest Delay to TP got the lowest accuracies (DDM, EDDM). The reason for that is because N is set to 200 instances. Thus, after a concept drift, these detectors got to detect drift early, and we still had instances from the old concept on the DSEL after it has shrank, on which the classifiers still would be ranked. The low accuracy of these detectors also reflect the low number of TPs. The ADWIN detector had the best accuracy, but the difference was not significant compared to some of the triggers.

Table 5.6: Average Accuracy and Average Delay to TP of Drift Detectors on Synthetic Datasets.

Detector	Av. Accuracy	Av. TP	Av. FP	Av. FN	Av. Delay to TP
DDM	93.21	1	0.4	1.5	31
EDDM	93.20	1	1.7	1.5	38
RDDM	93.64	1.35	2.2	1.2	108
STEPD	93.65	2	252.2	0.5	65
ADWIN	**93.70**	1.65	0.8	0.9	97
ECDD	93.20	2.15	56	0.4	27
PHT	93.46	1	0.9	1.5	166
HDDMA	93.53	1.4	0.7	1.1	103
HDDMW	93.29	1.1	0.1	1.4	62
FHDDM	93.55	1.45	0.3	1.1	97
FHDDMS	93.56	1.45	0.3	1.1	81
FHDDMS$_{add}$	93.48	1.45	0.3	1.1	91
MDDMA	93.61	1.4	0.4	1.1	112
MDDME	93.57	1.4	0.5	1.1	91
MDDMG	93.57	1.55	0.5	1	94
SeqDrift1	93.39	1.1	0.5	1.4	204
SeqDrift2	93.67	1.85	0.3	0.7	204

5.2.2.2 Real-world Datasets

Different from the synthetic datasets, we do not know the drifting points on the real-world datasets. Thus, we cannot use the measurements of TP, FP, and FN. To compare the drift detectors on the real-world datasets, we have used the classification accuracy and the average number of detections. These results are in Table 5.7. On the real-world datasets, the EDDM was the most accurate one, followed by the STEPD. The overall accuracy considering both real-world and synthetic datasets is in Table 5.8. We see that the STEPD had the best overall accuracy among the detectors, so it is the drift detector chosen for the next experiments.

5.2.3 Maximum Size of DSEL (M)

In this Section, we evaluate the impact of the maximum size of the DSEL by taking into account the accuracy and the time to process each dataset. A larger DSEL may give us better performance in cases of *virtual concept drift* and stable concepts. However, as we have more data for calculating the neighborhood, we hypothesize that the processing time will also be higher. The results regarding the accuracy and processing time on different values of M are in Table 5.9.

Notice that, different from the expected, a higher M led to a minor processing time. The reason behind that lies on the drift detector. When the DSEL is bigger, we are likelier to have instances from an old concept in it. Thus, if a concept drift has happened, it will trigger the change and shrink the DSEL. So, we can say that a larger DSEL will make the drift detector more likely to trigger a concept drift. That is the reason why $M = 6000$ got a lower processing time overall. On the Nursery dataset, we could not see much difference in processing time after $M = 3000$, as it is a dataset with fewer instances than the others. The DSEL never reached its maximum size.

However, in the cases where our drift detector does not trigger a concept drift, the DSEL maintains its size on the maximum, then depending on the dimensionality of our data, the processing time may increase substantially, since we have a distance-based DS (KNORAE). We consider that we have enough memory for $M = 4000$, and that we can deal with the processing time it may give us if it reaches the maximum size.

Table 5.7: Average Accuracy of Drift Detectors on Real-world Datasets.

Detector	Av. Accuracy	Av # of Detections
DDM	85.12	7
EDDM	**85.54**	35
RDDM	85.35	21
STEPD	85.49	35
ADWIN	85.17	4
ECDD	85.38	141
PHT	84.48	1
HDDMA	85.39	6
HDDMW	84.52	5
FHDDM	85.09	5
FHDDMS	85.13	5
FHDDMS$_{add}$	85.09	4
MDDMA	85.06	5
MDDME	85.13	5
MDDMG	85.17	5
SeqDrift1	84.60	1
SeqDrift2	85.07	3

Table 5.8: Average Accuracy of Detectors on Real-world and Synthetic Datasets.

Detector	Av. Accuracy
DDM	89.16
EDDM	89.37
RDDM	89.49
STEPD	**89.57**
ADWIN	89.44
ECDD	89.29
PHT	88.97
HDDMA	89.46
HDDMW	88.90
FHDDM	89.32
FHDDMS	89.34
FHDDMS$_{add}$	89.28
MDDMA	89.33
MDDME	89.35
MDDMG	89.37
SeqDrift1	88.99
SeqDrift2	89.37

5.2.4 Minimum Size of DSEL (N)

In this Section, we analyze how the minimum size of the DSEL, which we denote by N, influences the accuracy of Dynse+. The N is the value that the DSEL is shrunk to when a drift is detected. Thus, after a drift is detected, the instances in the DSEL are discarded until there are only N instances. The results are in Table 5.10. We see that $N = 100$ got the best overall accuracy

Table 5.9: Average Accuracies and Execution Time on Different Values of M.

M	1000		2000		3000		4000		5000		6000	
Dataset	Acc	Time	Acc	Time	Acc	Time	Acc	Time	Acc	Time	Acc	Time
NOAA	76.51	219.28	**76.80**	97.83	76.73	83.05	76.75	79.89	**76.80**	**65.46**	76.76	71.99
Nursery	92.95	51.21	**93.12**	29.04	93.09	25.50	92.93	23.72	93.09	**21.77**	93.06	23.87
Electricity	**87.07**	708.26	87.01	99.61	86.93	80.43	87.01	83.22	87.05	**75.00**	87.01	79.00
SEA	87.69	6584.47	87.76	4169.29	87.76	2453.00	**87.89**	801.93	87.84	1128.88	**87.89**	**763.75**
STAGGER	**99.39**	**9.15**	**99.39**	9.26	**99.39**	9.91	99.35	10.31	99.36	9.56	99.35	9.95
Average	88.72	1514.47	88.82	881.01	88.78	530.38	88.79	199.81	**88.83**	260.13	88.81	**189.71**

most of the times, and it also got the best average accuracy. Thus, we use $N = 100$ for the remaining experiments. In Table 5.11 is the final set of hyperparameters that compose the default configuration of Dynse+ to compare it to the other methods.

Table 5.10: Average Accuracies on Different Values of N.

Dataset	100	200	300	400	500
NOAA	76.55	76.20	76.99	**77.12**	77.03
Nursery	92.88	**93.11**	93.02	93.03	92.84
Electricity	**87.32**	87.12	86.82	86.50	86.28
SEA	**87.94**	87.77	87.62	87.71	87.69
STAGGER	**99.65**	99.36	99.11	98.82	98.49
Average	**88.87**	88.82	88.71	88.64	88.47

Table 5.11: Final Set of Hyperparameters

Parameter	Final Value
F	200
Drift Detector	STEPD
M	4000
N	100

5.3 THE IMPACT OF THE DRIFT DETECTOR

In this Section, we analyze the impact of the addition of the drift detector by comparing Dynse+ to the online version of the Dynse framework and the batch version proposed by Almeida et al. (2018). For the Dynse Framework, we have used the same set of parameters proposed by Almeida et al. (2018) for both *real* and *virtual concept drifts*. Notice that there are two different configurations for Dynse and ODynse: one for *real concept drift* and other for *virtual concept drift*, while Dynse+ only have one.

The Results are in Table 5.12. We can see that, most of the time, Dynse+ got better accuracy than Dynse and ODynse. The exceptions were the Airlines, Adults, and Gas Sensor datasets, and on four out of the five datasets with an induced *virtual concept drift*. We argue that the reason for Dynse and ODynse getting better accuracy on the datasets with *virtual concept drift* was because its maximum size of the DSEL was bigger (6400 instances, while Dynse+ had 4000 instances).

However, we have used two different configurations for Dynse and ODynse, for *real* and *virtual concept drifts*. The reason for that is because, when we have a *virtual concept drift*, the ideal decision boundary has not changed, so the more information we have, the best (larger DSEL). On the contrary, with *real concept drift*, the ideal decision boundary changes, and we must adapt to the new concept. However, in most of the environments, we do not know which kind of concept drift we are dealing with, so we would not know which configuration to use. We do not have this problem with Dynse+, as it adapts only when a concept drift is triggered.

By looking at the graphs of the prequential accuracy in Figure 5.2, we can see that Dynse+ has a faster adaptation to concept drift in some of the Insects dataset (we can notice some points where the recovery of accuracy after a drop was quicker), and also on the synthetic datasets. Dynse seems to struggle on some datasets as well, getting a lower accuracy in many moments in the timestamp.

In the SEA dataset, in Figure 5.2(k), Dynse+ got a faster adaptation to concept drift, as well as in the STAGGER-Rec (Figure 5.2(l)), Sine (Figure 5.2(m)), Sine-Rec (Figure 5.2(n)), and Agrawal (Figure 5.2(q)) datasets. On the datasets with gradual concept drift (Hyperplane and LED), we can notice a better prequential accuracy from Dynse+ over time as well. Dynse+ also got better prequential accuracy throughout the stream on the RandomRBF dataset in Figure 5.2(r). On the datasets with *virtual concept drift*, ODynse seems to be the best one most of the time.

Thus, we can say that the adaptation of the Dynse framework to perform online processing led to better accuracy overall, and the addition of a drift detector led to a faster adaptation to concept drift when compared to Dynse and ODynse.

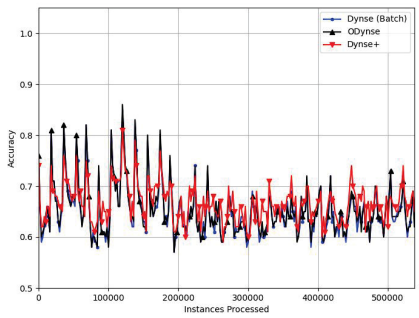
Remembering the hypothesis set at the beginning of this work, it was told that the adaptation of the DSEL would lead to better results in scenarios with *real concept drift*, and perform no worse in scenarios with *virtual concept drift*. We can say that both statements were accomplished, as Dynse+ had best accuracy overall on *real concept drift*, and got a competitive accuracy to ODynse on *virtual concept drift* without the need of having two configurations for the different types of *concept drift*.

5.3.1 Statistical Analysis

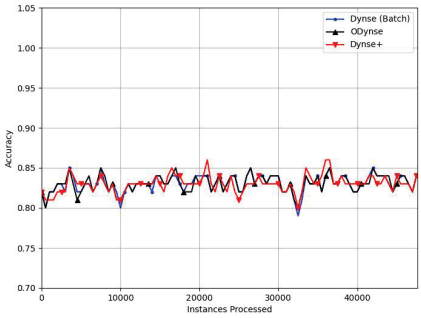
We have performed a Friedman-Nemenyi test to check how significant are the differences regarding accuracy. We have used $k = 3$, since we are comparing 3 different classifiers, and $N = 23$, since we have 23 datasets. We have considered $\alpha = 0.05$ and used the values supplied in Demšar's paper for 3 classifiers, i.e., $q_{0.05} = 2.343$.

On the Friedman test, we had $p = 0.000144$. Thus, we reject the Friedman test's null hypothesis that there is no significant difference between the classifiers. The Nemenyi post-hoc Critical Difference diagram is shown in Figure 5.3. We follow the guideline presented by Demšar (2006), where the best ranked methods are more on the right of the diagram.

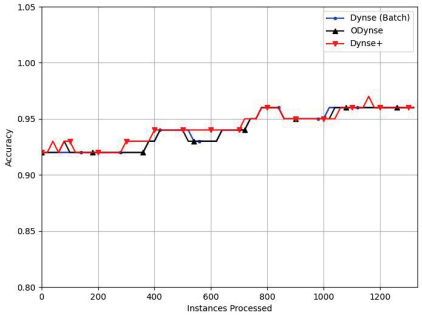
By these results, we can conclude that ODynse and Dynse+ got better results than Dynse. Dynse+ got better accuracy on 15 out of the 18 tested real-world and synthetic datasets with *real concept drift*, but only on 1 out of 5 datasets with an induced *virtual concept drift*. The Friedman-Nemenyi test show us that Dynse+ had a significant better accuracy than Dynse. Remind that we have used two different configurations for Dynse and ODynse, but only one for Dynse+. Thus, we got better results on *real concept drift* and a competitive performance on *virtual concept drift* without the need to have two different configurations. For the remaining tests that are performed to compare to the state of the art, in Section 5.4, we report only the results of Dynse+.



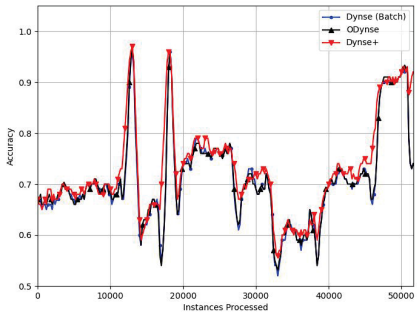
(a) Airlines.



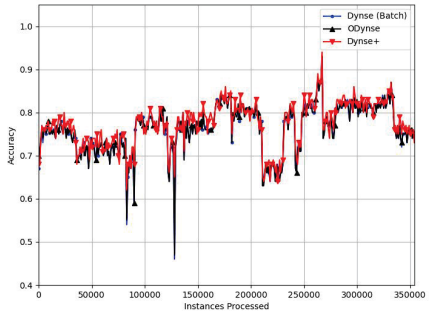
(b) Adult.



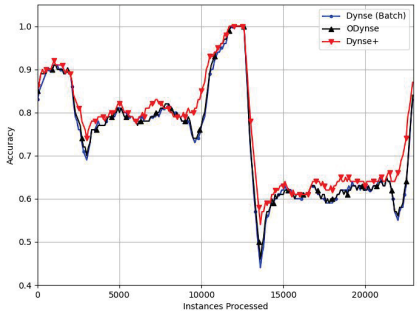
(c) Ozone.



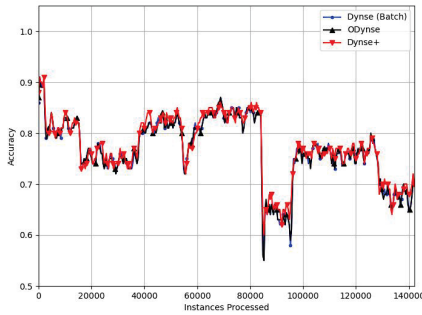
(d) Insects-AB.



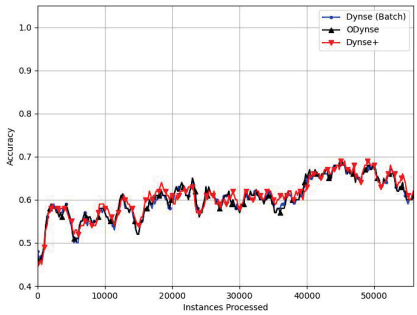
(e) Insects-AI.



(f) Insects-GB.

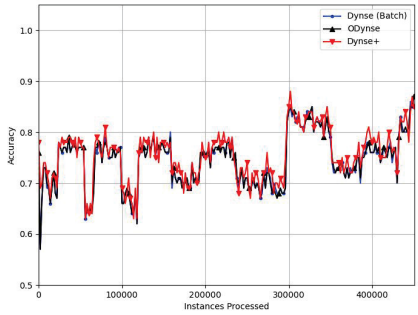


(g) Insects-GI.

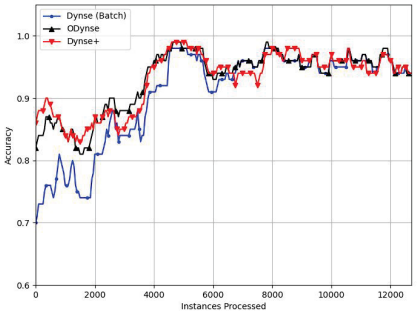


(h) Insects-IB.

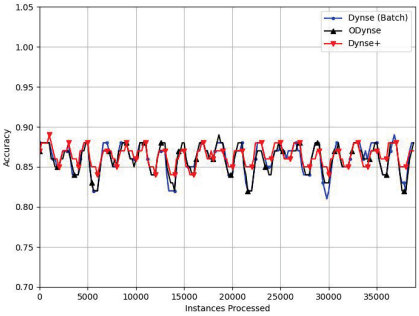
Figure 5.2: Prequential Accuracies of Dynse, Online Dynse, and Dynse+.



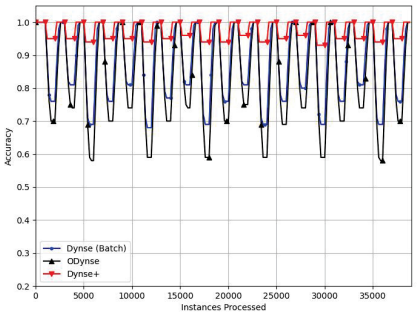
(i) Insects-II.



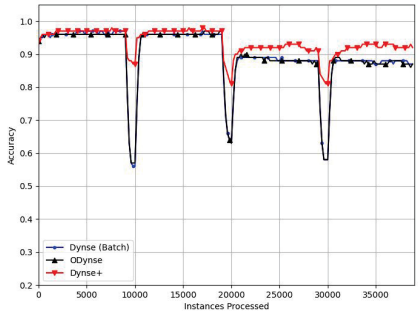
(j) Gas Sensor.



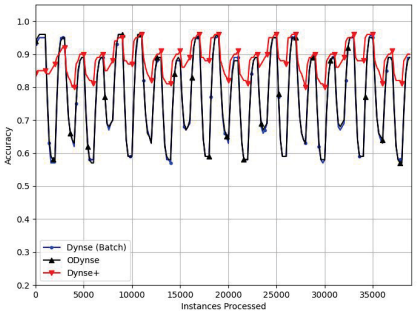
(k) SEA Recurrent.



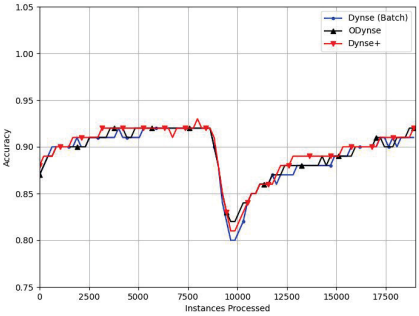
(l) STAGGER Recurrent.



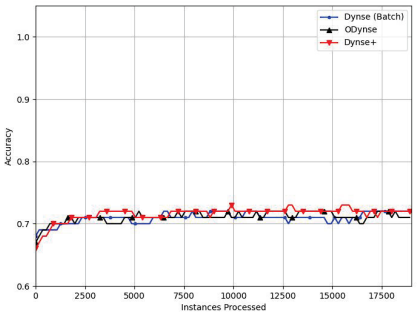
(m) Sine.



(n) Sine Recurrent.

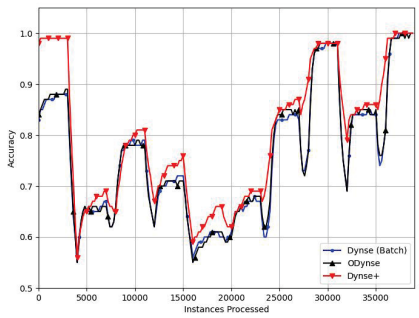


(o) Hyperplane.

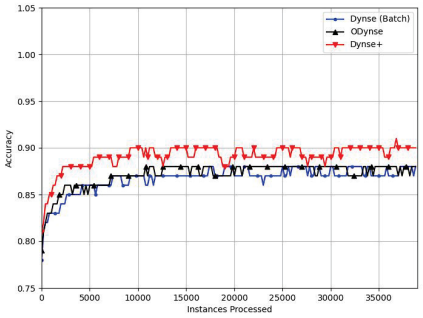


(p) LED.

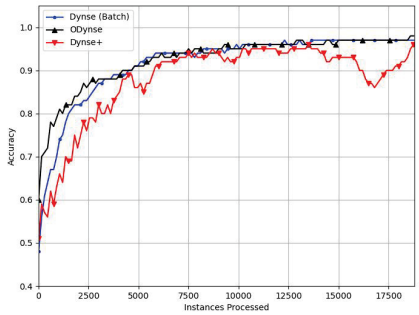
Figure 5.2: Prequential Accuracies of Dynse, Online Dynse, and Dynse+ (Continued).



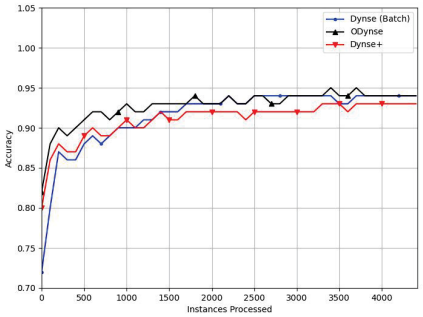
(q) Agrawal.



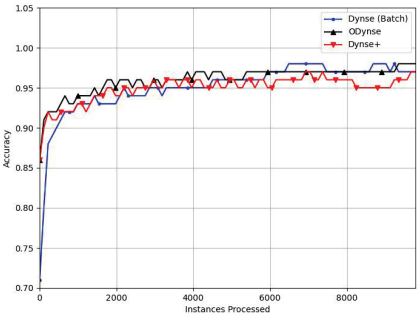
(r) Random RBF.



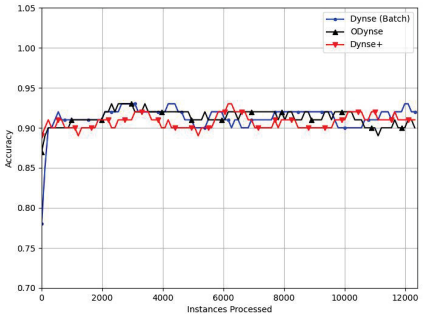
(s) Letters.



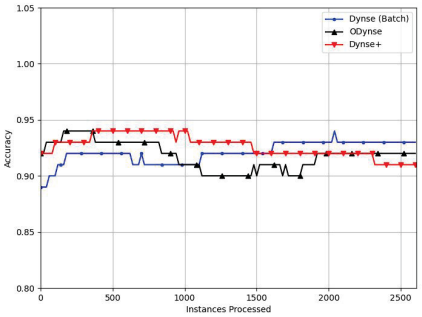
(t) Digits.



(u) Pen Digits.



(v) Dry Bean.



(w) Rice.

Figure 5.2: Prequential Accuracies of Dynse, Online Dynse, and Dynse+ (Continued).

Table 5.12: Accuracies Comparing Dynse, ODynse, and Dynse+.

Dataset	Dynse*	ODynse*	Dynse+
Real Concept Drift			
Airlines	65.13	**65.26**	65.11
Ozone	94.04	94.09	**94.22**
Adults	**83.07**	**83.07**	83.04
Insects-AB	70.93	71.06	**73.02**
Insects-AI	76.30	76.33	**77.24**
Insects-GB	74.09	74.41	**76.39**
Insects-GI	76.69	76.69	**77.40**
Insects-IB	60.23	60.23	**60.57**
Insects-II	74.62	74.65	**75.60**
Gas Sensor	89.98	**92.72**	92.71
Sine	89.78	89.65	**93.67**
Sine-Rec	77.34	77.91	**89.07**
SEA-Rec	86.08	86.06	**86.48**
STAGGER-Rec	88.23	84.61	**97.45**
Agrawal	76.42	76.62	**80.25**
Hyperplane	89.42	89.22	**89.70**
LED	70.88	71.04	**71.41**
RandomRBF	86.59	87.01	**88.99**
Average	79.43	79.48	**81.80**
Virtual Concept Drift			
Letters	90.35	**91.64**	88.16
Digits	89.97	**91.42**	90.67
Pen Digits	93.67	**95.60**	94.27
Dry Bean	90.33	**91.19**	90.73
Rice	91.15	91.93	**92.19**
Average	90.98	**92.36**	91.09
Total Average	81.94	82.28	**83.82**

*Different configurations for different types of *concept drift*.

5.4 TESTS COMPARING WITH THE STATE OF THE ART

In this Section, the proposed Dynse+ was compared to other methods in the literature. We have used the default configuration available in the MOA framework (Bifet et al., 2010a) for such methods. Results are exposed in Table 5.13, and graphs for the prequential accuracy in Figure 5.4. For visualization purposes, we show the graphs only for Dynse+, OAUE, LevBag, and ARF, as they got the best accuracies.

These results indicate us that Dynse+ gets a better accuracy in some of the real-world datasets and synthetic datasets with *real concept drift*. It got better accuracy on the Insects-AB and Ozone datasets. OAUE was the best one on the Airlines, the OzaBag got the best accuracy on the Adult dataset, and the ARF got on most of the Insects datasets. The ARF had the best average accuracy on the datasets with *real concept drift*, with Dynse+ in the second place. By looking at the synthetic datasets, Dynse+ got the best accuracy on the SEA concepts dataset with

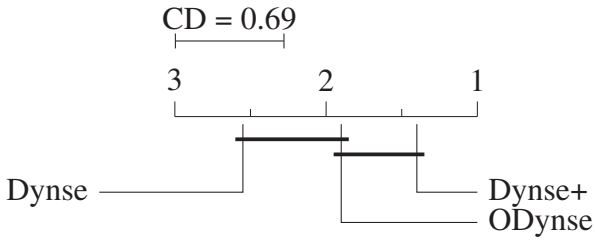


Figure 5.3: CD Diagram of the Nemenyi Test Comparing Dynse, ODynse, and Dynse+.

a *recurrent concept drift*, and on the Hyperplane dataset with a *gradual concept drift*. Despite that, it did not have accuracy far from the best performing state-of-the-art methods.

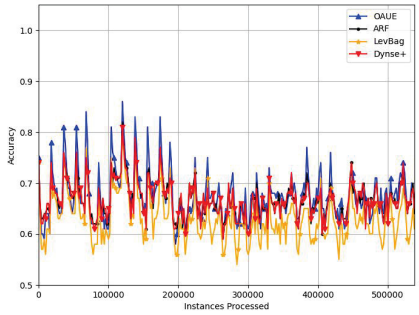
Getting into the datasets with *virtual concept drift*, the Dynse+ got the best accuracy on most of the datasets, with a high difference in some of them. On the Letters dataset, for example, it got a difference of over 20 percentage points to the OAUE, which was the best one among the other state-of-the-art methods. The only dataset with *virtual concept drift* that Dynse+ did not get the best accuracy was the Rice dataset, in which ARF got the best accuracy. Dynse+ got the best average accuracy among the datasets with an induced *virtual concept drift*, and also the best average accuracy considering both *real* and *virtual concept drifts*.

By looking on the graphs of the prequential accuracy in Figure 5.4, Dynse+ got a competitive accuracy to ARF on most of the datasets, while LevBag and OAUE had more drops in accuracy. We notice that on the Airlines dataset in Figure 5.4(a), OAUE seemed to have a higher accuracy in most of the timestamp. This also happened on the Adult dataset in Figure 5.4(b), where LevBag also got good performance. On the Ozone dataset, Dynse+ and ARF seemed to have the same accuracy in many moments in the timestamp.

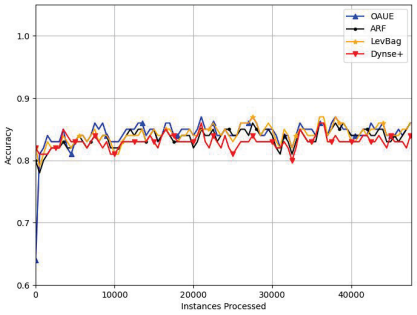
On the Insects-AB dataset in Figure 5.4(d), the OAUE dataset had a minor accuracy and presented drops throughout the timestamp. The same we can say about LevBag, but the drops in accuracy were not as big as OAUE. Dynse+ and ARF had similar drops and recovery on the timestamp. On the Insects-AI, in Figure 5.4(e), ARF had the best accuracy on most of the timestamp, and that was repeated for the remaining Insects datasets in Figures 5.4(f), 5.4(g), 5.4(h), and 5.4(i).

Getting into the synthetic datasets, Dynse+ seems to have a better recovery on the SEA-Rec dataset in Figure 5.4(k) than ARF, LevBag, and OAUE. On the STAGGER-Rec dataset, in Figure 5.4(l), the OAUE had higher drops in accuracy after concept drift, while Dynse+, LevBag, and ARF got a better recovery, although the ARF seems to have a quicker recovery on most of the concept drifts. LevBag and OAUE had big drops in accuracy on the Sine and Sine-Rec datasets in Figures 5.4(m) and 5.4(n), where ARF seems to have a higher accuracy when compared to Dynse+ at some points. On the Hyperplane dataset in Figure 5.4(o), Dynse+ had the best accuracy over time, and got a good recovery after the gradual concept drift. On the LED dataset in Figure 5.4(p), Dynse+ was the worst-performing, while LevBag and OAUE had best accuracy at the beginning of the timestamp, and a competitive accuracy related to ARF, even though LevBag presented some drops throughout the timestamp. On the Agrawal dataset, in Figure 5.4(q), OAUE had the best accuracy on most of the timestamp, being able to have the best recovery on most concept drifts. On the RandomRBF in Figure 5.4(r), ARF had the best accuracy over time.

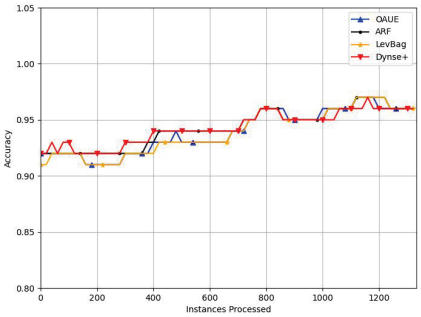
On the datasets with an induced *virtual concept drift*, we saw that Dynse+ had the best overall accuracy in Table 5.13. On the Letters dataset in Figure 5.4(s), it had a dominant accuracy on most of the timestamp – it only had a minor accuracy compared to LevBag and OAUE on the first instances. On the Digits, Pen Digits, and Dry Bean datasets in Figures 5.4(t), 5.4(u),



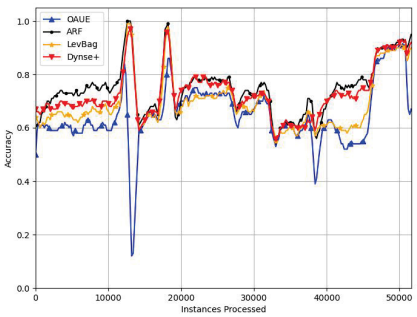
(a) Airlines.



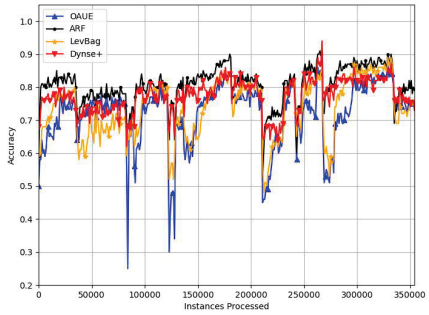
(b) Adult.



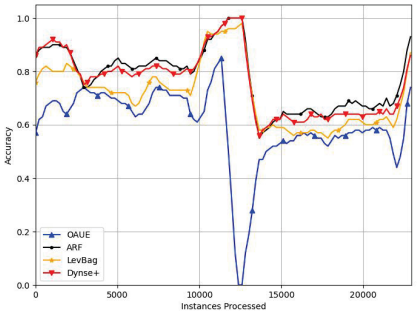
(c) Ozone.



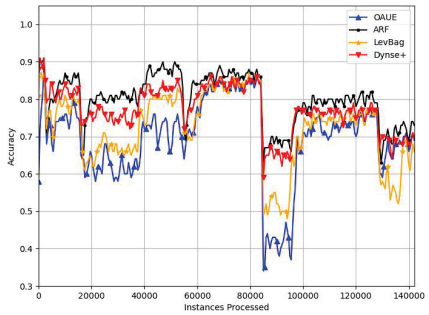
(d) Insects-AB.



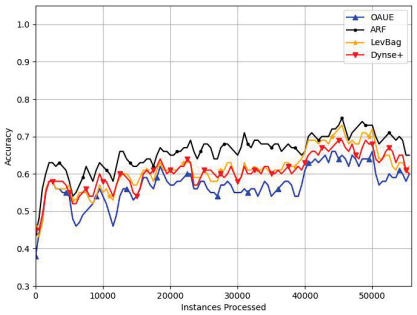
(e) Insects-AI.



(f) Insects-GB.

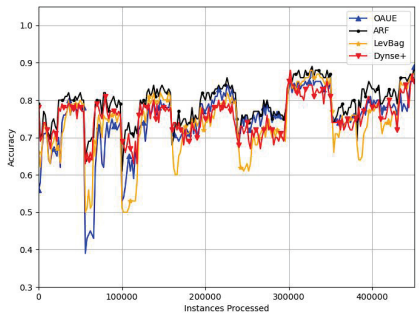


(g) Insects-GI.

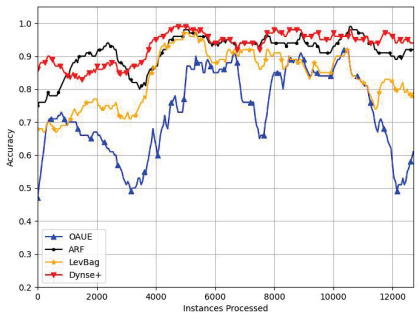


(h) Insects-IB.

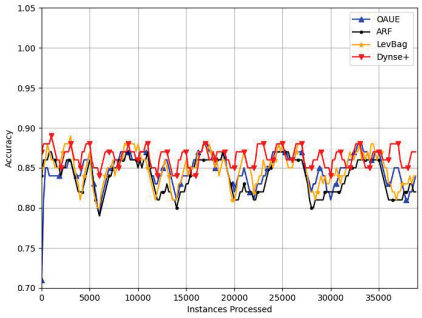
Figure 5.4: Prequential Accuracies of Dynse+, ARF, LevBag, and OAUE.



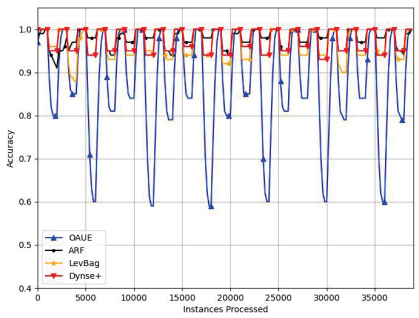
(i) Insects-II.



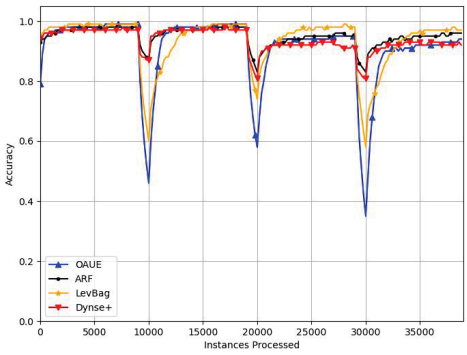
(j) Gas Sensor.



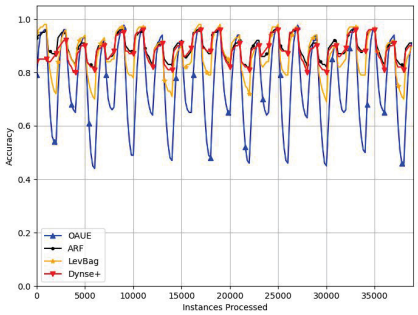
(k) SEA Recurrent.



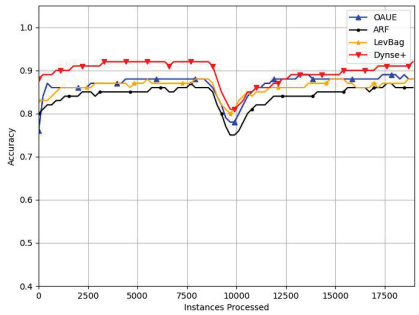
(l) STAGGER Recurrent.



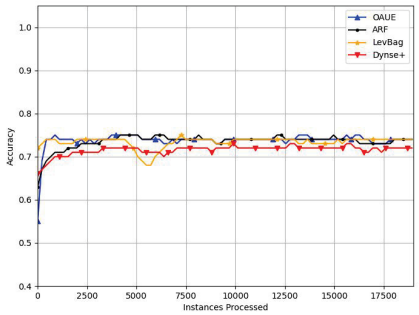
(m) Sine.



(n) Sine Recurrent.

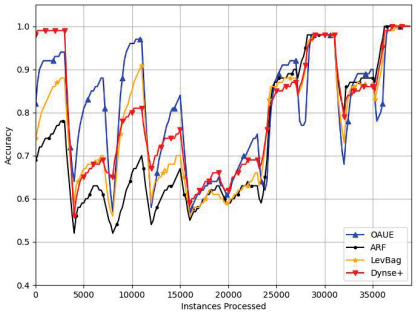


(o) Hyperplane.

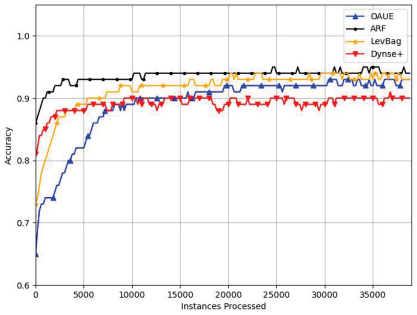


(p) LED.

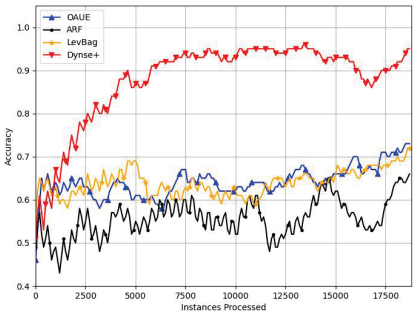
Figure 5.4: Prequential Accuracies of Dynse+, ARF, LevBag, and OAUE (Continued).



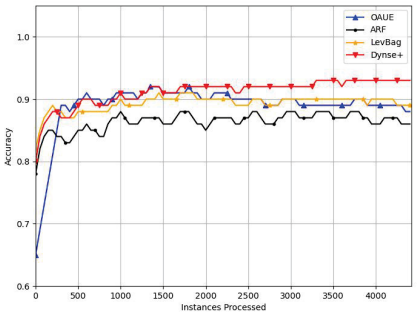
(q) Agrawal.



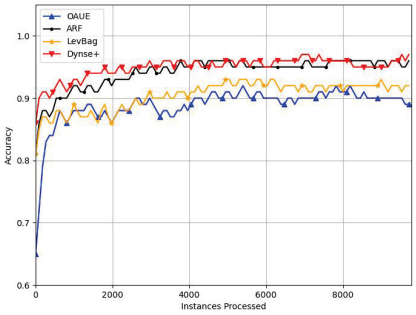
(r) Random RBF.



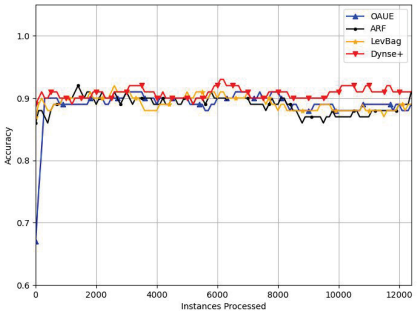
(s) Letters.



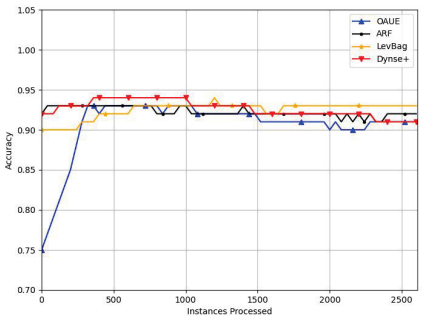
(t) Digits.



(u) Pen Digits.



(v) Dry Bean.



(w) Rice.

Figure 5.4: Prequential Accuracies of Dynse+, ARF, LevBag, and OAUE (Continued).

Table 5.13: Results of Dynse+ and the State of the Art on Real-world datasets.

Dataset	Dynse+	OzaBag	LevBag	OAUE	AUE	AWE	ARF	Learn++.NSE
Real Concept Drift								
Airlines	65.11	64.92	63.13	**67.51**	66.66	61.87	66.66	62.34
Ozone	**94.22**	93.96	93.89	93.96	93.96	75.28	94.16	78.66
Adult	83.00	**84.67**	84.25	84.16	84.10	82.84	83.70	81.60
Insects-AB	**73.01**	57.36	69.09	64.64	63.35	61.12	71.80	61.67
Insects-AI	77.24	68.94	73.22	71.90	70.72	67.06	**80.58**	63.93
Insects-GB	76.39	62.07	71.79	59.69	58.63	61.34	**77.97**	61.77
Insects-GI	77.40	64.94	72.47	69.43	68.95	69.76	**80.07**	65.27
Insects-IB	60.57	54.48	61.32	56.92	59.91	57.23	**65.71**	56.37
Insects-II	75.60	73.00	73.83	74.86	73.35	66.69	**79.27**	62.57
Gas Sensor	**92.71**	55.46	82.60	71.93	58.61	47.64	90.77	51.30
Sine	93.67	73.19	93.20	90.90	89.93	89.19	**94.95**	86.35
Sine-Rec	89.07	60.60	87.01	75.95	68.56	77.40	**89.53**	78.29
SEA-Rec	**86.48**	83.94	84.81	84.48	83.87	84.58	83.96	84.77
STAGGER-Rec	97.45	71.85	96.81	88.00	80.69	88.81	**98.01**	55.38
Agrawal	80.25	64.67	77.68	**81.37**	79.42	76.02	74.15	75.41
Hyperplane	**89.70**	85.76	85.99	86.49	86.14	89.20	84.08	86.21
LED	71.41	**73.77**	73.57	73.04	72.98	72.98	73.17	67.79
RandomRBF	88.99	85.32	91.63	88.23	88.47	72.07	**93.58**	70.25
Average	81.80	71.05	79.79	76.86	74.91	72.28	**82.34**	69.44
Virtual Concept Drift								
Letters	**87.94**	62.23	63.83	65.54	63.99	60.20	56.15	42.04
Digits	**90.67**	87.92	88.27	85.39	83.10	83.57	84.80	71.51
Pen Digits	**94.27**	85.93	90.37	86.90	84.73	83.11	93.74	78.41
Dry Bean	**90.73**	89.29	89.20	87.82	87.61	88.01	88.58	85.50
Rice	92.26	91.67	91.59	88.57	88.16	87.42	**92.34**	89.32
Average	**91.09**	83.41	84.65	82.17	81.52	80.46	83.12	73.36
Total Average	**83.82**	73.74	80.85	78.01	76.34	74.06	82.51	70.29

and 5.4(v), respectively, Dynse+ still had a best accuracy on most of the timestamp, but it was not such a significant difference as on the Letters dataset. Remember that the Rice dataset was the only dataset with an induced *virtual concept drift* that Dynse+ did not have the best overall accuracy, but we see that it got a similar accuracy over time compared to ARF.

5.4.1 Statistical Analysis

The Friedman-Nemenyi test is also performed for comparing the proposed Dynse+ with the state-of-the-art methods. Here, we have used $k = 8$, since we have 8 classifiers. We also have used $\alpha = 0.05$, and $q_{0.05} = 3.031$, by following the Table provided by Demšar (2006).

The p-value for the Friedman test was $p = 2.032 \times 10^{-11}$, Thus, we reject the null hypothesis from Friedman's test that there is no significant difference between the methods. The diagram for the Nemenyi test is in Figure 5.5.

We see that Dynse+ was the best ranked method, with a significant difference to 4 out of 7 methods: Learn++.NSE, AWE, OzaBag, and AUE. The difference was not statistically significant to ARF, LevBag, and OAUE. Thus, the combination of dynamic selection and drift detection seems to provide greater accuracy on both *real* and *virtual concept drifts*. The possibility of saving useful information and to drop outdated knowledge led to best results when comparing to the state of the art.

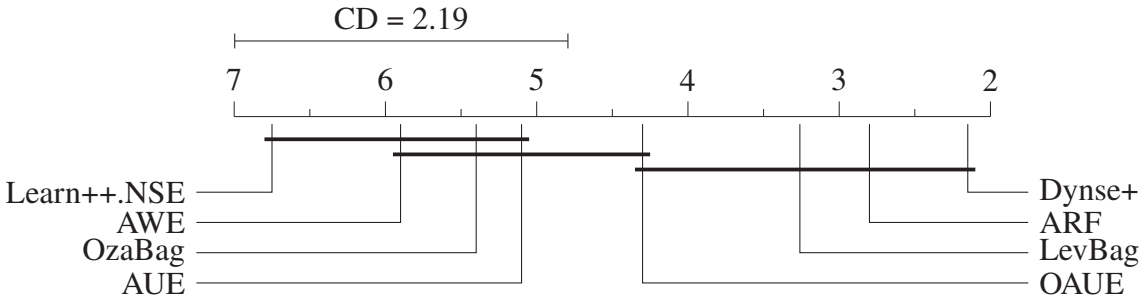


Figure 5.5: CD Diagram of the Nemenyi Test Comparing State-of-the-art methods.

5.5 TESTS WITH DELAYED AND PARTIAL LABELS

In this Section, a comparison of Dynse+ to the state of the art considering delayed and partial labels is performed. The results regarding accuracy are presented in Table 5.14. Here, ARF got the best overall accuracy on *real concept drift* followed by Dynse+, and Dynse+ got the best accuracy on *virtual concept drift*, followed by LevBag. Considering both types of concept drift, Dynse+ had the best overall accuracy once more, and ARF got the second best accuracy. ARF was dominant on the Insects dataset in these tests as well, while Dynse+ had the best accuracies in some of the datasets, but with a competitive accuracy when comparing to most of the best results.

Table 5.14: Accuracies of Dynse+ and the State of the Art With Delayed and Partial Labels.

Dataset	Dynse+	OzaBag	LevBag	OAUE	AUE	AWE	ARF	Learn++.NSE
Real Concept Drift								
Airlines	64.60	64.06	61.97	**65.86**	65.46	61.60	65.60	61.90
Ozone	93.92	**93.96**	93.16	**93.96**	**93.96**	71.55	93.84	73.95
Adults	83.01	**83.96**	83.73	83.55	83.34	82.23	83.31	81.61
Insects-AB	71.38	54.86	66.43	62.43	58.87	58.19	**71.43**	60.78
Insects-AI	77.05	66.12	70.10	67.95	67.13	66.15	**79.57**	63.93
Insects-GB	73.21	56.92	68.43	60.15	58.40	59.16	**74.23**	60.33
Insects-GI	77.15	63.20	70.51	66.34	66.40	68.49	**78.74**	65.01
Insects-IB	59.50	49.54	59.52	55.40	55.34	55.86	**64.53**	55.66
Insects-II	75.31	69.77	72.05	71.88	70.16	64.91	**78.42**	62.45
Gas Sensor	**65.52**	58.38	62.09	55.57	51.15	46.88	61.26	39.40
Sine	92.26	61.11	90.13	86.20	83.54	85.75	**92.89**	84.58
Sine-Rec	**83.35**	56.67	79.29	66.26	55.20	62.39	82.76	66.89
SEA-Rec	**84.67**	83.84	84.15	83.38	82.68	83.02	83.70	83.50
STAGGER-Rec	93.36	70.62	88.93	80.59	68.71	75.79	**95.15**	53.37
Agrawal	76.47	64.43	73.98	**76.76**	73.49	70.60	70.96	73.47
Hyperplane	**88.86**	86.38	85.01	85.17	84.91	86.13	83.30	85.94
LED	70.84	**73.91**	73.38	71.81	71.64	71.90	72.60	67.48
RandomRBF	88.66	80.65	89.39	85.41	84.42	71.91	**92.86**	70.11
Average	78.84	68.80	76.24	73.26	70.82	69.03	**79.18**	67.24
Virtual Concept Drift								
Letters	59.92	59.55	58.28	58.64	57.50	**60.38**	46.32	41.93
Digits	83.22	84.45	**85.45**	79.42	76.01	75.69	82.00	73.24
Pen Digits	**90.16**	81.55	85.73	81.32	79.75	80.19	89.40	77.55
Dry Bean	**89.72**	88.78	88.38	85.76	85.36	85.49	87.99	85.73
Rice	91.24	90.52	91.53	83.22	83.22	81.74	**91.82**	88.13
Average	**82.85**	80.97	81.87	77.67	76.37	76.70	79.51	73.32
Total Average	**79.71**	71.44	77.46	74.22	72.03	70.70	79.25	68.56

Regarding processing time in Table 5.15, Dynse+ was the slowest method. We can notice that the processing time of Dynse+ tends to be higher when we have more instances and high-dimensional data, which makes sense when we remember the need to calculate the distances between data points. Overall, OzaBag was the method with the lowest processing time in average.

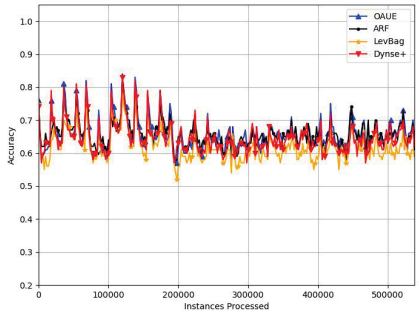
Table 5.15: Processing Time of Dynse+ and the State of the Art.

Dataset	Dynse+	OzaBag	LevBag	OAUE	AUE	AWE	ARF	Learn++.NSE
Airlines	38288.04	**33.40**	329.09	198.05	163.33	195.86	2487.29	284.88
Ozone	2.99	0.25	1.09	0.23	0.19	0.52	14.26	**0.17**
Adults	1731.32	**1.68**	4.99	3.08	2.67	2.68	99.38	1.93
Insects-AB	164.00	**9.57**	17.54	11.96	11.71	17.55	136.72	13.12
Insects-AI	67634.57	**33.46**	72.34	65.14	68.60	138.48	874.87	440.13
Insects-GB	49.79	**3.56**	5.66	4.43	6.97	6.83	65.78	5.86
Insects-GI	9679.73	**16.61**	29.87	26.59	40.99	55.98	351.85	88.95
Insects-IB	1019.94	**11.63**	19.54	16.42	13.03	20.25	165.79	16.54
Insects-II	97976.81	**56.17**	85.71	73.56	80.19	177.63	1103.37	716.62
Gas Sensor	27.08	10.16	16.66	11.13	12.18	8.88	150.69	**8.45**
Sine	215.70	**0.77**	1.65	1.22	1.23	1.72	33.59	0.82
Sine-Rec	31.08	**0.49**	1.53	0.91	0.91	1.48	42.67	0.74
SEA-Rec	503.47	0.63	1.48	0.88	0.87	1.34	52.51	**0.70**
STAGGER-Rec	8.55	**0.43**	0.88	0.66	0.62	0.90	8.73	0.47
Agrawal	61.90	1.41	4.61	2.21	1.92	2.14	90.74	**1.03**
Hyperplane	77.25	1.49	3.11	2.32	2.07	2.70	60.76	**1.18**
LED	196.68	**2.67**	8.00	3.50	4.64	4.92	48.21	5.34
RandomRBF	1121.31	1.61	3.94	2.56	2.07	2.70	60.76	**1.18**
Letters	21.02	**4.86**	8.84	6.11	6.51	7.74	91.70	7.98
Digits	13.61	2.02	3.71	1.08	1.09	1.62	38.49	**1.04**
Pen Digits	16.64	**1.24**	2.95	1.61	1.79	2.09	29.95	1.66
Dry Bean	19.76	**1.10**	2.67	1.67	1.78	2.01	27.09	1.72
Rice	1.96	0.11	0.28	0.09	0.09	0.22	3.01	**0.05**
Average	9515.79	**8.45**	26.96	18.82	18.34	28.35	260.49	69.43

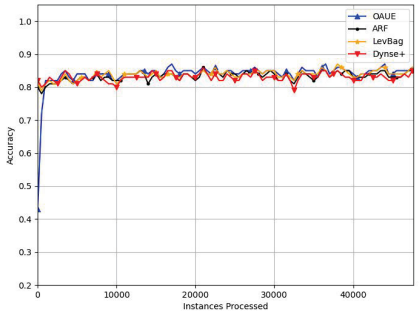
The prequential accuracies of Dynse+, LevBag, OAUE and ARF are shown in Figure 5.6. The OAUE, once more, seems to struggle on adapting to concept drift on most of the datasets, as well as the LevBag. Dynse+ and ARF again seem to have a quick adaptation in most scenarios. ARF appears as the best on the Insects datasets (Figures 5.6(d), 5.6(h), 5.6(f), 5.6(g), 5.6(h), and 5.6(i)). Mostly, the results on the prequential accuracy for the delayed tests were repeated to the test-then-train configuration, with a quick adaptation of Dynse+ and ARF to concept drift, and some drops in accuracy concerning OAUE and LevBag. OAUE still got the best accuracy on the Airlines and Agrawal datasets. The biggest difference was on the Letters dataset, where Dynse+ had a lower accuracy over time when compared to the test-then-train.

5.5.1 Statistical Analysis

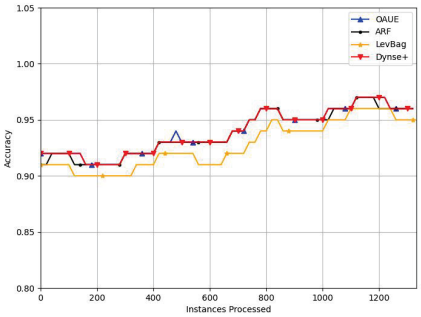
The p-value for the Friedman test on the results considering Dynse+ and state-of-the-art methods was $p = 8.52 \times 10^{-12}$, so the null hypothesis is rejected. The CD diagram for the Nemenyi test is in Figure 5.7. Again, Dynse+ comes as the best ranked method, followed by ARF and LevBag. Just like on the test-then-train configuration, there was no statistically significant difference to ARF, LevBag and OAUE. OzaBag also have surpassed AUE here when comparing to the test-then-train. So we can say that, providing less information about the scenarios (partial and delayed labeling), Dynse+ keeps performing best than other methods in terms of accuracy.



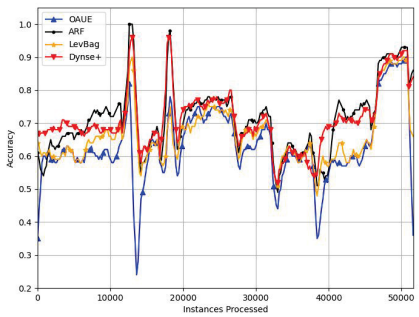
(a) Airlines.



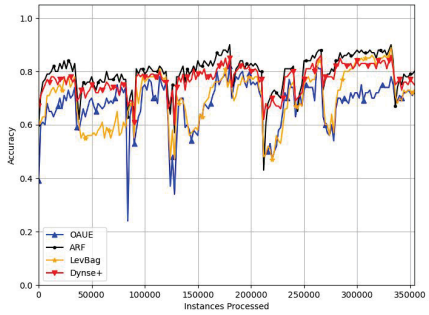
(b) Adult.



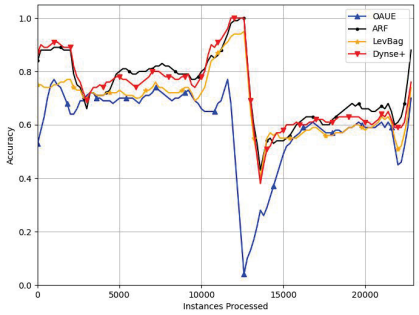
(c) Ozone.



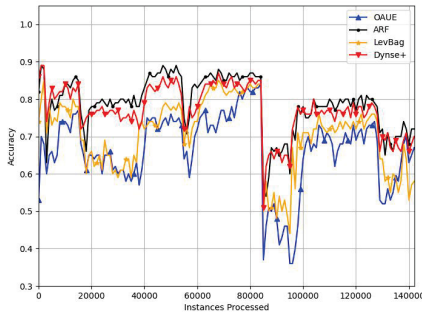
(d) Insects-AB.



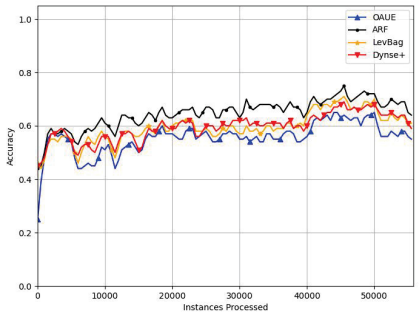
(e) Insects-AI.



(f) Insects-GB.

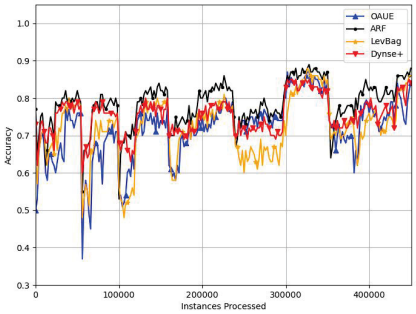


(g) Insects-GI.

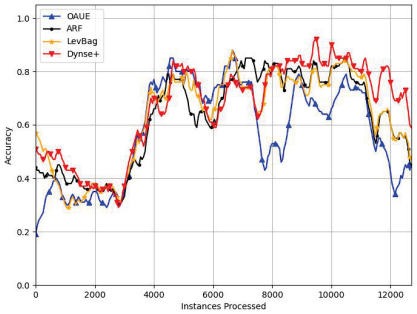


(h) Insects-IB.

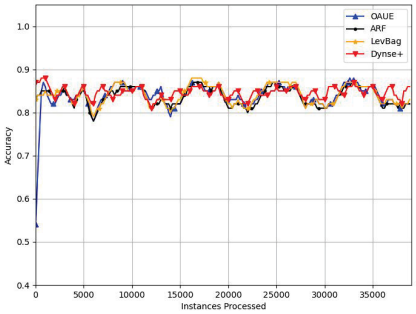
Figure 5.6: Prequential Accuracies of Dynse+, ARF, LevBag, and OAUE With Delayed and Partial Labels.



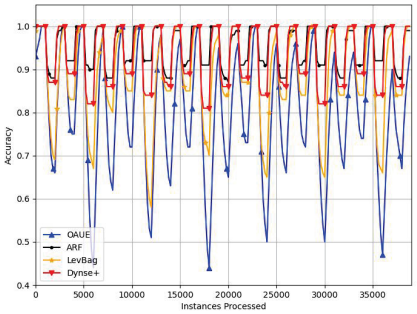
(i) Insects-II.



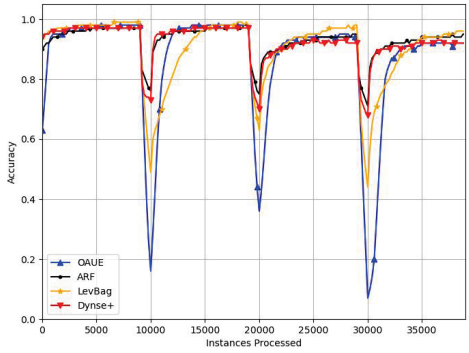
(j) Gas Sensor.



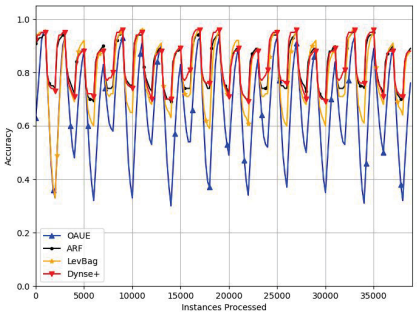
(k) SEA Recurrent.



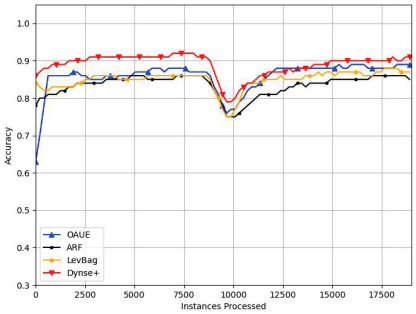
(l) STAGGER Recurrent.



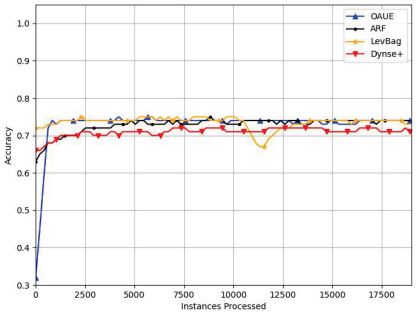
(m) Sine.



(n) Sine Recurrent.

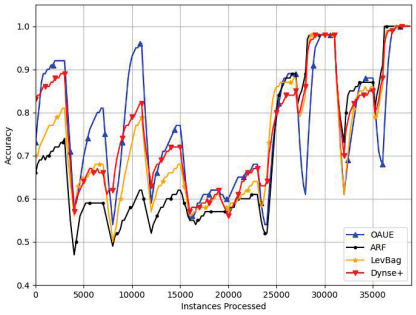


(o) Hyperplane.

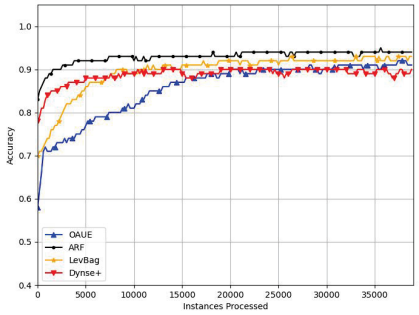


(p) LED.

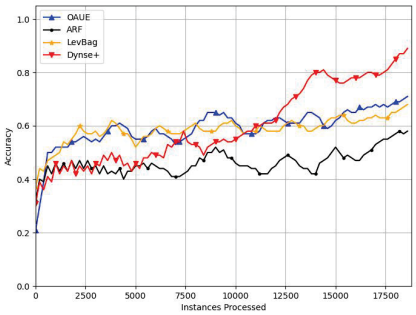
Figure 5.6: Prequential Accuracies of Dynse+, ARF, LevBag, and OAUE With Delayed and Partial Labels (Continued).



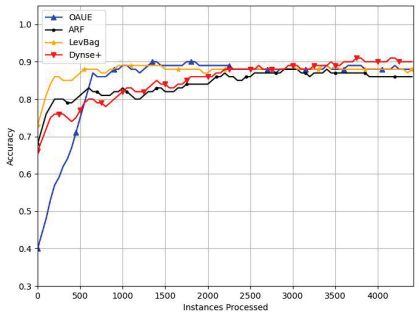
(q) Agrawal.



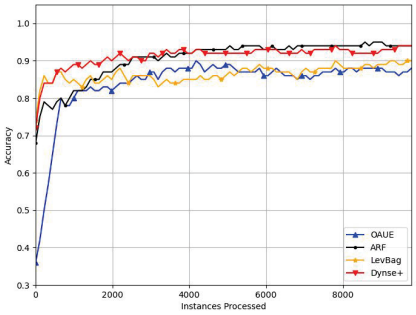
(r) Random RBF.



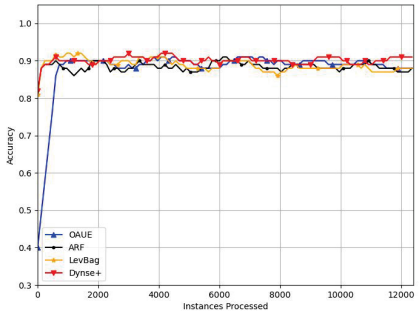
(s) Letters.



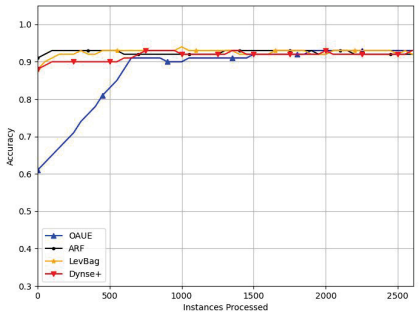
(t) Digits.



(u) Pen Digits.



(v) Dry Bean.



(w) Rice.

Figure 5.6: Prequential Accuracies of Dynse+, ARF, LevBag, and OAUE With Delayed and Partial Labels (Continued).

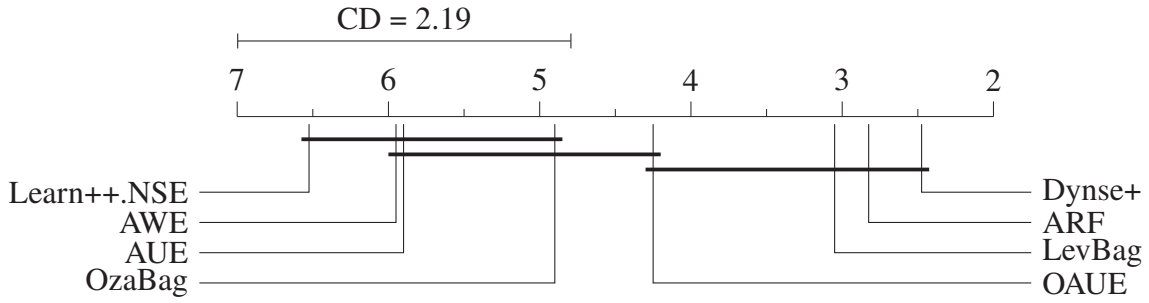


Figure 5.7: CD Diagram of the Nemenyi Test Comparing Dynse+ and the State Of The Art With Delayed and Partial Labels.

5.6 DISCUSSION

In this Section, we have performed a Cramer's V autocorrelation test on the datasets and compared different approaches for classifying data streams. Further, a hyperparameter analysis on the parameters of Dynse+ was performed, giving us a default configuration to be compared to other methods.

Focusing on the results of the proposed framework Dynse+, we had an improvement in accuracy comparing to Dynse and ODynse regarding *real concept drift*, and a competitive accuracy to ODynse on *virtual concept drift*. The Friedman-Nemenyi test showed us that Dynse+ was the best placed when comparing with the two versions of Dynse, and to 7 other methods in the literature. We got a quicker adaptation to concept drift, and kept a good performance on most of the datasets with *virtual concept drift*. We can say the same on the tests with delayed and partial labels, where we try to be closer to a real-world environment, in which the labels of the data might arrive with some delay, and will not be fully available. Dynse+ was the best placed as well.

The results indicate that the addition of a drift detector to the Dynse framework, in addition to the adaptation to online processing, led to best results overall. The concept drift detector helps on the maintenance of the information in the DSEL, while the ability to perform online processing helps keeping the system up-to-date, as every incoming instance is used for training an incremental classifier. Furthermore, we can conclude that there is no need of prior knowledge on the type of concept drift anymore, an issue that the original Dynse has. The adaptation of the DSEL now is in charge of the drift detector, which now is able to grow larger on stable concepts, and shrink when a concept drift is triggered in order to forget an old concept.

The biggest drawback of the proposed framework is the processing time, as we use a neighborhood-based DS method, with which we need to get the neighborhood of the test instance to perform classification. As we have a drift detector, we must calculate the neighborhood on both training and testing phases in order to update the drift detector, leading to a higher processing time. To overcome this, we can consider kNN optimization algorithms, or different approaches for DS, such as cluster-based (Guo et al., 2021a).

6 CONCLUSION

In this work, we have proposed Dynse+, which uses triggers, i.e., drift detectors, to keep track of the current concept. That opens space for the DSEL to grow larger, as the drift detector tracks whether a concept drift has happened. It auxiliates on the adaptation of Dynse+ to a new concept. In addition to that, Dynse was modified to be able to perform online processing, which we called ODynse.

The results showed us that ODynse had a better performance than the batch version of Dynse on both *real* and *virtual* concept drifts, and Dynse+ was the best ranked between Dynse and ODynse according to the Friedman-Nemenyi test, but with a statistically significant difference only to Dynse.

Dynse+ was also the best ranked method when compared to seven methods in the state of the art according to the Nemenyi test. It had a statistically significant difference to 4 methods. The advantages of Dynse+ is that it is able to track concept drift and has a faster adaptation when compared to Dynse and ODynse, and maintains a good performance on *virtual concept drift* when compared to other methods in the literature. Remember that Dynse and ODynse have two configurations for two different types of concept drift, while Dynse+ has only one. Thus, there is no need to know the type of concept drift in advance.

However, the more instances we have, and for high-dimensional data, the processing time may be bigger, as we have a distance-based DS method. For future works, we intend to test different DS methods that do not involve distance calculations to improve the processing time of Dynse+. Some examples in the literature are cluster-based methods (Guo et al., 2021a) and methods based on Fuzzy Hyperbox (Davtalab et al., 2024).

Furthermore, the Dynse+ framework, just like Dynse, is flexible and extensible, and is able to receive more mechanisms to help on the adaptation to concept drift. Examples are instance selection when pruning the DSEL, such as instance hardness measures, and different measures to prune the classifiers from C, like diversity.

REFERENCES

Abadifard, S., Bakhshi, S., Gheibuni, S., and Can, F. (2023). Dyned: Dynamic ensemble diversification in data stream classification. In *CIKM '23: Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, CIKM '23, New York, NY, USA. Association for Computing Machinery.

Agate, V., Drago, S., Ferraro, P., and Re, G. L. (2022). Anomaly detection for reoccurring concept drift in smart environments. In *2022 18th International Conference on Mobility, Sensing and Networking (MSN)*, pages 113–120.

Agrawal, R., Imielinski, T., and Swami, A. (1993). Database mining: a performance perspective. *IEEE Transactions on Knowledge and Data Engineering*, 5(6):914–925.

Almeida, P. R., Oliveira, L. S., Britto, A. S., and Sabourin, R. (2018). Adapting dynamic classifier selection for concept drift. *Expert Systems with Applications*, 104:67–85.

Almeida, P. R., Oliveira, L. S., Britto, A. S., Silva, E. J., and Koerich, A. L. (2015). Pklot – a robust dataset for parking lot classification. *Expert Systems with Applications*, 42(11):4937–4949.

Almeida, P. R. L., Oliveira, L. S., Souza Britto, A. d., and Paul Barddal, J. (2020). Naïve approaches to deal with concept drifts. In *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 1052–1059.

Alpaydin, E. and Kaynak, C. (1998). Optical Recognition of Handwritten Digits. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C50P49.

Baena-García, M., Campo-Ávila, J., Fidalgo-Merino, R., Bifet, A., Gavald, R., and Morales-Bueno, R. (2006). Early drift detection method. *Proc. 4th Int. Workshop Knowledge Discovery from Data Streams*.

Baier, L., Schlör, T., Schöffer, J., and Kühl, N. (2021). Detecting concept drift with neural network model uncertainty. In *Hawaii International Conference on System Sciences*.

Barboza, E. and Almeida, P. (2022). Challenges on classifying data streams with concept drift. In *Anais Estendidos do XXXVII Simpósio Brasileiro de Bancos de Dados*, pages 126–132, Porto Alegre, RS, Brasil. SBC.

Barboza, E. V. L., de Almeida, P. R. L., de Souza Britto, A., and Cruz, R. M. O. (2023). Distance functions and normalization under stream scenarios. In *2023 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8.

Barros, R. S., Cabral, D. R., Gonçalves, P. M., and Santos, S. G. (2017). Rddm: Reactive drift detection method. *Expert Systems with Applications*, 90:344–355.

Bayram, F., Ahmed, B. S., and Kassler, A. (2022). From concept drift to model degradation: An overview on performance-aware drift detectors. *Knowledge-Based Systems*, 245:108632.

Becker, B. and Kohavi, R. (1996). Adult. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5XW20.

Bifet, A. and Gavaldà, R. (2007). Learning from time-changing data with adaptive windowing. In *SDM*.

Bifet, A. and Gavaldà, R. (2009). Adaptive learning from evolving data streams. In Adams, N. M., Robardet, C., Siebes, A., and Boulicaut, J.-F., editors, *Advances in Intelligent Data Analysis VIII*, pages 249–260, Berlin, Heidelberg. Springer Berlin Heidelberg.

Bifet, A., Holmes, G., Kirkby, R., and Pfahringer, B. (2010a). MOA: massive online analysis. *J. Mach. Learn. Res.*, 11:1601–1604.

Bifet, A., Holmes, G., and Pfahringer, B. (2010b). Leveraging bagging for evolving data streams. In Balcázar, J. L., Bonchi, F., Gionis, A., and Sebag, M., editors, *Machine Learning and Knowledge Discovery in Databases*, pages 135–150, Berlin, Heidelberg. Springer Berlin Heidelberg.

Bifet, A., Holmes, G., Pfahringer, B., Kirkby, R., and Gavaldà, R. (2009). New ensemble methods for evolving data streams. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Bifet, A., Read, J., Žliobaitė, I., Pfahringer, B., and Holmes, G. (2013). Pitfalls in benchmarking data stream classification and how to avoid them. In Blockeel, H., Kersting, K., Nijssen, S., and Železný, F., editors, *Machine Learning and Knowledge Discovery in Databases*, pages 465–479, Berlin, Heidelberg. Springer Berlin Heidelberg.

Blackard, J. (1998). Covertype. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C50K5N.

Breiman, L., F., J.H., Olshen, R., and Stone, C. (1988). LED Display Domain. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5FG61.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, 24(2):123–140.

Brzeziński, D. and Stefanowski, J. (2011). Accuracy updated ensemble for data streams with concept drift. In Corchado, E., Kurzyński, M., and Woźniak, M., editors, *Hybrid Artificial Intelligent Systems*, pages 155–163, Berlin, Heidelberg. Springer Berlin Heidelberg.

Brzezinski, D. and Stefanowski, J. (2014). Combining block-based and online methods in learning ensembles from concept drifting data streams. *Information Sciences*, 265:50–67.

Cavalheiro, L. P., De Souza Britto, A., Barddal, J. P., and Heutte, L. (2021). Dynamically selected ensemble for data stream classification. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7.

Cerqueira, V., Gomes, H. M., Bifet, A., and Torgo, L. (2023). Studd: a student–teacher method for unsupervised concept drift detection. *Machine Learning*, 112(11):4351–4378.

Cinar, I. and Koklu, M. (2019). Classification of rice varieties using artificial intelligence methods. *International Journal of Intelligent Systems and Applications in Engineering*.

Cruz, R. M., Sabourin, R., and Cavalcanti, G. D. (2018). Dynamic classifier selection: Recent advances and perspectives. *Information Fusion*, 41:195–216.

Cruz, R. M. O., Zakane, H. H., Sabourin, R., and Cavalcanti, G. D. C. (2017). Dynamic ensemble selection VS k-NN: Why and when dynamic selection obtains higher classification performance? In *2017 Seventh International Conference on Image Processing Theory, Tools and Applications (IPTA)*. IEEE.

Davtalab, R., Cruz, R. M., and Sabourin, R. (2024). A scalable dynamic ensemble selection using fuzzy hyperboxes. *Information Fusion*, 102:102036.

de Lima Cabral, D. R. and de Barros, R. S. M. (2018). Concept drift detection based on fisher's exact test. *Information Sciences*, 442-443:220–234.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7(1):1–30.

Ditzler, G. and Polikar, R. (2013). Incremental learning of concept drift from streaming imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 25(10):2283–2301.

Ditzler, G., Roveri, M., Alippi, C., and Polikar, R. (2015). Learning in nonstationary environments: A survey. *IEEE Computational Intelligence Magazine*, 10(4):12–25.

Domingos, P. and Hulten, G. (2000). Mining high-speed data streams. *Association for Computing Machinery*, page 71–80.

Dua, D. and Graff, C. (2017). UCI machine learning repository.

Elwell, R. and Polikar, R. (2011). Incremental learning of concept drift in nonstationary environments. *IEEE Transactions on Neural Networks*, 22(10):1517–1531.

Fischer, L., Hammer, B., and Wersing, H. (2016). Online metric learning for an adaptation to confidence drift. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 748–755.

Forman, G. (2006). Tackling concept drift by temporal inductive transfer. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '06, page 252–259, New York, NY, USA. Association for Computing Machinery.

Freund, Y. (1995). Boosting a weak learning algorithm by majority. *Information and Computation*, 121(2):256–285.

Frías-Blanco, I., Campo-Ávila, J. d., Ramos-Jiménez, G., Morales-Bueno, R., Ortiz-Díaz, A., and Caballero-Mota, Y. (2015). Online and non-parametric drift detection methods based on hoeffding's bounds. *IEEE Transactions on Knowledge and Data Engineering*, 27(3):810–823.

Gama, J., Medas, P., Castillo, G., and Rodrigues, P. (2004). Learning with drift detection. In Bazzan, A. L. C. and Labidi, S., editors, *Advances in Artificial Intelligence – SBIA 2004*, pages 286–295, Berlin, Heidelberg. Springer Berlin Heidelberg.

Gama, J. a., Žliobaitundefined, I., Bifet, A., Pechenizkiy, M., and Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM Comput. Surv.*, 46(4).

Gomes, H. M., Bifet, A., Read, J., Barddal, J. P., Enembreck, F., Pfahringer, B., Holmes, G., and Abdessalem, T. (2017). Adaptive random forests for evolving data stream classification. *Machine Learning*, 106:1–27.

Gulcan, E. B. and Can, F. (2023). Unsupervised concept drift detection for multi-label data streams. *Artificial Intelligence Review*, 56(3):2401–2434.

Guo, C., Liu, M., and Lu, M. (2021a). A dynamic ensemble learning algorithm based on k-means for icu mortality prediction. *Applied Soft Computing*, 103:107166.

Guo, H., Zhang, S., and Wang, W. (2021b). Selective ensemble-based online adaptive deep neural networks for streaming data with concept drift. *Neural Networks*, 142:437–456.

Han, M., Zhang, X., Chen, Z., Wu, H., and Li, M. (2023). Dynamic ensemble selection classification algorithm based on window over imbalanced drift data stream. *Knowledge and Information Systems*, 65(3):1105–1128.

Harries, M., Wales, N. S., et al. (1999). Splice-2 comparative evaluation: Electricity pricing. *University of New South Wales, School of Computer Science and Engineering*.

Hoeffding, W. (1963). Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30.

Hulten, G., Spencer, L., and Domingos, P. (2001). Mining time-changing data streams. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, page 97–106, New York, NY, USA. Association for Computing Machinery.

Jiao, B., Guo, Y., Gong, D., and Chen, Q. (2022). Dynamic ensemble selection for imbalanced data streams with concept drift. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–14.

Jordaney, R., Sharad, K., Dash, S. K., Wang, Z., Papini, D., Nouretdinov, I., and Cavallaro, L. (2017). Transcend: Detecting concept drift in malware classification models. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 625–642, Vancouver, BC. USENIX Association.

Kittler, J., Hatef, M., Duin, R., and Matas, J. (1998). On combining classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(3):226–239.

Ko, A. H., Sabourin, R., and Britto, Jr., A. S. (2008). From dynamic classifier selection to dynamic ensemble selection. *Pattern Recognition*, 41(5):1718–1731.

Koklu, M. and Özkan, I. A. (2020). Multiclass classification of dry beans using computer vision and machine learning techniques. *Comput. Electron. Agric.*, 174:105507.

Kolter, J. Z. and Maloof, M. A. (2007). Dynamic weighted majority: An ensemble method for drifting concepts. *Journal of Machine Learning Research*, 8(91):2755–2790.

Komorniczak, J. and Ksieniewicz, P. (2023). Complexity-based drift detection for nonstationary data streams. *Neurocomputing*, 552:126554.

Kozal, J., Guzy, F., and Woźniak, M. (2021). Employing chunk size adaptation to overcome concept drift.

Krawczyk, B., Pfahringer, B., and Woźniak, M. (2018). Combining active learning with concept drift detection for data stream mining. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 2239–2244.

Kubát, M. (1989). Floating approximation in time-varying knowledge bases. *Pattern Recognition Letters*, 10(4):223–227.

Kuncheva, L. I. (2004). Classifier ensembles for changing environments. In Roli, F., Kittler, J., and Windeatt, T., editors, *Multiple Classifier Systems*, pages 1–15, Berlin, Heidelberg. Springer Berlin Heidelberg.

Lazarescu, M. M., Venkatesh, S., and Bui, H. H. (2004). Using multiple windows to track concept drift. *Intell. Data Anal.*, 8:29–59.

Liu, W., Zhang, H., Ding, Z., Liu, Q., and Zhu, C. (2021). A comprehensive active learning method for multiclass imbalanced data streams with concept drift. *Knowledge-Based Systems*, 215:106778.

Losing, V., Hammer, B., and Wersing, H. (2015). Interactive online learning for obstacle classification on a mobile robot. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8.

Lu, J., Liu, A., Song, Y., and Zhang, G. (2020). Data-driven decision support under concept drift in streamed big data. *Complex & Intelligent Systems 6*, pages 157–163.

Masud, M. M., Chen, Q., Khan, L., Aggarwal, C., Gao, J., Han, J., and Thuraisingham, B. (2010). Addressing concept-evolution in concept-drifting data streams. In *2010 IEEE International Conference on Data Mining*, pages 929–934.

McDiarmid, C. (1989). *On the method of bounded differences*, page 148–188. London Mathematical Society Lecture Note Series. Cambridge University Press.

Minku, L. L., White, A. P., and Yao, X. (2010). The impact of diversity on online ensemble learning in the presence of concept drift. *IEEE Transactions on Knowledge and Data Engineering*, 22:730–742.

Minku, L. L. and Yao, X. (2012). Ddd: A new ensemble approach for dealing with concept drift. *IEEE Transactions on Knowledge and Data Engineering*, 24(4):619–633.

Müller, M. and Salathé, M. (2020). Addressing machine learning concept drift reveals declining vaccine sentiment during the covid-19 pandemic.

Nakai, K. (1996). Yeast. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5KG68.

Nishida, K. and Yamauchi, K. (2007). Detecting concept drift using statistical testing. In Corruble, V., Takeda, M., and Suzuki, E., editors, *Discovery Science*, pages 264–269, Berlin, Heidelberg. Springer Berlin Heidelberg.

Oliveira, G. H. F. M., Minku, L. L., and Oliveira, A. L. I. (2019). Gmm-vrd: A gaussian mixture model for dealing with virtual and real concept drifts. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8.

Oza, N. (2005). Online bagging and boosting. In *2005 IEEE International Conference on Systems, Man and Cybernetics*, volume 3, pages 2340–2345 Vol. 3.

Page, E. S. (1954). Continuous inspection schemes. *Biometrika*, 41(1/2):100–115.

Pears, R., Sakthithasan, S., and Koh, Y. S. (2014). Detecting concept change in dynamic data streams. *Machine Learning*, 97(3):259–293.

Pesaranghader, A., Viktor, H., and Paquet, E. (2018a). Mcdiarmid drift detection methods for evolving data streams.

Pesaranghader, A., Viktor, H., and Paquet, E. (2018b). Reservoir of diverse adaptive learners and stacking fast hoeffding drift detection methods for evolving data streams. *Machine Learning*, 107(11):1711–1743.

Pesaranghader, A. and Viktor, H. L. (2016). Fast hoeffding drift detection method for evolving data streams. In Frasconi, P., Landwehr, N., Manco, G., and Vreeken, J., editors, *Machine Learning and Knowledge Discovery in Databases*, pages 96–111, Cham. Springer International Publishing.

Pfahringer, B., Holmes, G., and Kirkby, R. (2007). New options for hoeffding trees. In Orgun, M. A. and Thornton, J., editors, *AI 2007: Advances in Artificial Intelligence*, pages 90–99, Berlin, Heidelberg. Springer Berlin Heidelberg.

Polikar, R., Upda, L., Upda, S., and Honavar, V. (2001). Learn++: an incremental learning algorithm for supervised neural networks. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 31(4):497–508.

Rajkovic, V. (1997). Nursery. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5P88W.

Rea, L. and Parker, R. (1992). *Designing and Conducting Survey Research: A Comprehensive Guide*. A Joint publication of the Jossey-Bass social and behavioral sciences series, the Jossey-Bass public administration series, and the Jossey-Bass management series. Jossey-Bass Publishers.

Ross, G. J., Adams, N. M., Tasoulis, D. K., and Hand, D. J. (2012). Exponentially weighted moving average charts for detecting concept drift. *Pattern Recognition Letters*, 33(2):191–198.

Sakthithasan, S., Pears, R., and Koh, Y. S. (2013). One pass concept change detection for data streams. In Pei, J., Tseng, V. S., Cao, L., Motoda, H., and Xu, G., editors, *Advances in Knowledge Discovery and Data Mining*, pages 461–472, Berlin, Heidelberg. Springer Berlin Heidelberg.

Sakurai, G. Y., Lopes, J. F., Zarpelão, B. B., and Barbon Junior, S. (2023). Benchmarking change detector algorithms from different concept drift perspectives. *Future Internet*, 15(5).

Schlimmer, J. C. and Granger, R. H. (1986). Incremental learning from noisy data. *Mach. Learn.*, 1(3):317–354.

Slate, D. (1991). Letter Recognition. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5ZP40.

Souza, V. M. A., dos Reis, D. M., Maletzke, A. G., and Batista, G. E. A. P. A. (2020). Challenges in benchmarking stream learning algorithms with real-world data. *Data Mining and Knowledge Discovery*, 34(6):1805–1858.

Street, W. N. and Kim, Y. (2001). A streaming ensemble algorithm (sea) for large-scale classification. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, page 377–382, New York, NY, USA. Association for Computing Machinery.

Vergara, A. (2012). Gas Sensor Array Drift Dataset. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5RP6W.

Wang, H., Fan, W., Yu, P. S., and Han, J. (2003). Mining concept-drifting data streams using ensemble classifiers. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, page 226–235, New York, NY, USA. Association for Computing Machinery.

Wang, P., Yu, H., Jin, N., Davies, D., and Woo, W. L. (2024). Quadcdd: A quadruple-based approach for understanding concept drift in data streams. *Expert Systems with Applications*, 238:122114.

Weiss, C. H. (2018). *Analyzing Categorical Time Series*, chapter 6, pages 121–132. John Wiley & Sons, Ltd.

Widmer, G. (1994). Combining robustness and flexibility in learning drifting concepts. In *European Conference on Artificial Intelligence*.

Widmer, G. and Kubat, M. (1992). Learning flexible concepts from streams of examples: Flora2. In *Proceedings of the 10th European Conference on Artificial Intelligence*, ECAI '92, page 463–467, USA. John Wiley & Sons, Inc.

Widmer, G. and Kubat, M. (1993). Effective learning in dynamic environments by explicit context tracking. In Brazdil, P. B., editor, *Machine Learning: ECML-93*, pages 227–243, Berlin, Heidelberg. Springer Berlin Heidelberg.

Woods, K., Kegelmeyer, W., and Bowyer, K. (1997). Combination of multiple classifiers using local accuracy estimates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):405–410.

Yan, M. M. W. (2020). Accurate detecting concept drift in evolving data streams. *ICT Express*, 6(4):332–338.

Yang, C., Cheung, Y.-M., Ding, J., and Tan, K. C. (2022). Concept drift-tolerant transfer learning in dynamic environments. *IEEE Transactions on Neural Networks and Learning Systems*, 33(8):3857–3871.

Yang, L. and Shami, A. (2021). A lightweight concept drift detection and adaptation framework for IoT data streams. *IEEE Internet of Things Magazine*, 4(2):96–101.

Yu, H., Liu, W., Lu, J., Wen, Y., Luo, X., and Zhang, G. (2023). Detecting group concept drift from multiple data streams. *Pattern Recognition*, 134:109113.

Yu, H., Zhang, Q., Liu, T., Lu, J., Wen, Y., and Zhang, G. (2022). Meta-add: A meta-learning based pre-trained model for concept drift active detection. *Information Sciences*, 608:996–1009.

Zhang, K., Fan, W., and Yuan, X. (2008). Ozone Level Detection. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5NG6W.