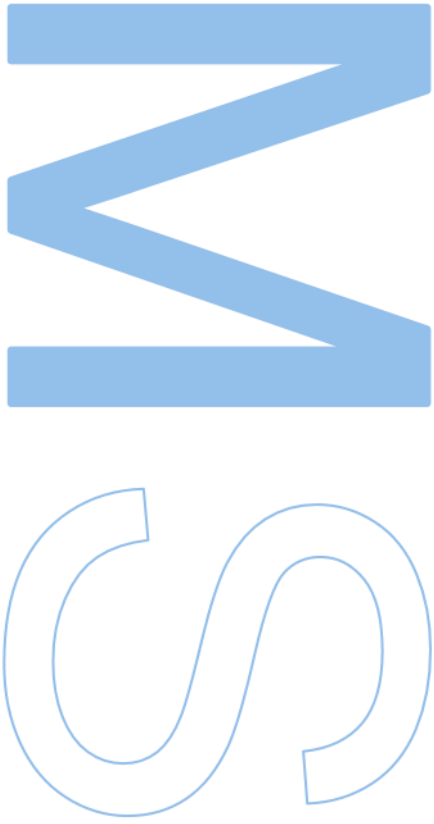
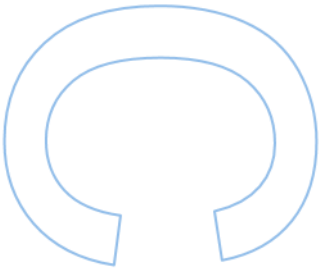
Imbalanced MultiClass Classification with Concept Drift



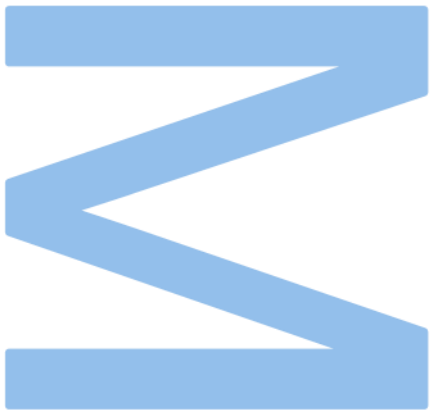
José Gabriel Moreira Pinto
Mestrado de Ciência de Dados
Departamento de Ciência dos Computadores
2023

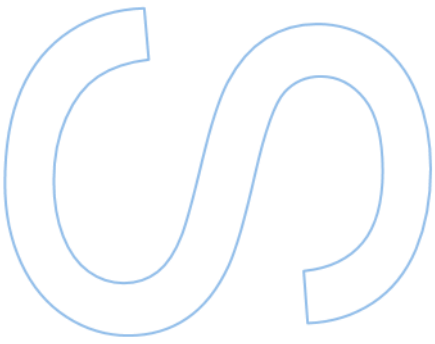
Orientador
Nuno Guimarães, Professor Auxiliar Convidado, Faculdade de Ciências da Universidade do Porto

Coorientador
Álvaro Figueira, Professor Auxiliar, Faculdade de Ciências da Universidade do Porto




U.PORTO
FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO





Abstract

As our digital universe grows, more and more data is constantly being generated at a fast rate. Consequently, to make use of these large quantities of data, machine learning models must be able to process large quantities of streaming data due to the large memory costs required to store data in a persistent way. However, since it comes in the form of streams, this data is susceptible to various issues that can hinder these models. For example, data can have an imbalanced presence in the representation of the possible classes, making the models biased towards the more represented ones. Furthermore, because this data is constant, changes in the properties related to classes can also change over time, denoted as concept drift, requiring models to adapt to this. Alongside all these, the classes associated with data may not arrive at the same point in time as the data itself, since it may take some time to validate the information.

Although it is a relevant problem in the domain of Machine Learning, there is still a lack of works related to the classification of streaming data in a multiclass setting since most only try to tackle these in a binary class setting. The goal of this dissertation is to help with all the aforementioned problems through the exploration of the consequences caused by these problems and proposing a way to tackle these problems more broadly through more specific models. To do this, we built a framework for the evaluation of Machine Learning models in a setting where data can be imbalanced, contain concept drifts, and necessitate extra time to validate the data. Furthermore, we also proposed an approach that makes use of two methods to generalize the classification task. With these, we were able to better understand the causes for performance drops in multiclass classification tasks where data may contain the problems mentioned. Alongside this, the proposed method also proved to be competitive, while showing potential for improvement due to limitations in its implementation.

Resumo

Com o crescimento do universo digital, dados são cada vez mais gerados a um ritmo alto. Consequentemente, para fazer uso destas quantidades altas dados, os modelos de Machine Learning precisam de ser capazes de processar quantidades elevadas de dados em *stream* devido ao custo alto necessário para guardá-los de forma persistent. No entanto, como os dados vem em forma de *stream*, estes são suscetíveis a vários problems que podem afetar os modelos. Por exemplo, dados podem ter uma presença desbalanceada na representação das possíveis classes, tornando os modelos enviasados para as mais representadas. Para além disso, uma vez que estes dados são constantes, as propriedades das classes podem mudar ao longo do tempo, chamado de concept drift, que requer que os modelos adaptem-se a estas mudanças. Juntamente com estes, as classes associadas aos dados podem não chegar ao mesmo tempo que os dados em si, uma vez que pode ser necessário algum tempo para validar os dados.

Apesar de ser um problema relevante no domínio de Machine Learning, ainda há uma falta de trabalhos relacionados com a classificação de dados em *stream* em cenários de multiclass porque a maioria o problema em cenários binários. O objetivo desta dissertação é ajudar nos problemas mencionados através da exploração das consequências causadas por estes e a através da proposta de um método para combater estes métodos de uma forma mais geral usando modelos específicos. Para isto, nós construímos uma *framework* para a avaliação de modelos de Machine Learning num cenário em que os dados podem ser desbalanceados, conter concept drift, e ainda necessitar tempo extra para os validar. Além disto, também propomos uma abordagem que usa dois métodos para generalizar a tarefa de classificação. Com estes, nós fomos capazes de entender melhor as causas para as quedas na performance em classificação multiclass onde os dados podem conter os problemas mencionados. Por fim, também reparámos que o método proposto provou ser competitive enquanto, ao mesmo tempo, mostrou potencial para melhoria devido a limitações na sua implementação.

Acknowledgements

I would like to acknowledge and thank my supervisor Prof. Dr. Nuno Guimarães and my co-supervisor Prof. Dr. Álvaro Figueira for helping, supporting, and motivating me during all the work of this dissertation. I would also like to thank all my family and friends for their continuous support.

Contents

List of Tables

List of Figures

Listings

Acronyms

AUC Area Under Curve

CD Concept Drift

EWAUC Equally Weighted Area Under Curve

IMC Imbalanced Multiclass Classification

IR Imbalance Ratio

MOOB Multiclass Oversampling-based Online Bagging

MUOB Multiclass Undersampling-based Online Bagging

PMAUC Prequential Multiclass Area Under Curve

WAUC Weighted Area Under Curve

Chapter 1

Introduction

Communication technologies are constantly evolving. One of these technologies, the Internet, allows people from around the globe to communicate with each other with ease. Nowadays, as it has improved to be accessible and fast, it has become a very used tool by many such that many services have expanded their businesses to make use of this technology. Alongside these, new services also emerged, some of which have become widely used, one of these kinds of services being social media. As services become more and more accessible through the Internet, many feel the need to use this technology. This has led us to be ever increasingly dependent on the Internet. The use of this technology made data collection easier. However, as it has grown immensely in usage, the rate at which data is generated has grown exponentially. This makes it harder to analyze data manually and due to this, the usage of machine learning increased. Despite that, machines still have a hard time fitting all the generated data into memory. Alongside this, the data generation is constant like a stream, forcing machines to make use of the data as soon as it comes.

1.1 Motivation

As data is constantly generated in large quantities, there is an ever-increasing need for machine learning technologies to analyze and understand this data. However, data may not be consistent over time as it can have its properties change. These changes can lead machine learning algorithms to become confused and in turn, lower their performance over time. To improve the performance of machine learning models in these situations, more and more systems are being implemented in a way that allows them to adapt, resulting in more robust and lasting systems. Alongside the need for ever-adapting machine learning models, some algorithms are better at understanding data with specific properties. This results in a need for more specialized systems for different data characteristics and with the growth in the data generation rate, even the more infrequent properties have an increasing need to be representative in these systems. One of these properties is Multiclass (data with more than two descriptive classes). This property, however, is not as prevalent as binary classification because many of the machine learning problems can be

categorized as simple binary problems. In other words, they need to identify one out of two possibilities. However, in some cases, it is not as easy to translate a problem into a binary one. These problems can range from different themes and sources in real-life situations, from simple board games to more complex tasks such as the detection of harmful gas using sensor data. Data can also have another issue that can affect the performance of machine learning algorithms. This issue is the possibility of data having an imbalanced class representation. In other words, data can have class labels that appear more frequently than others. This is known as class imbalance and it can lead machine learning systems to become biased towards understanding the more frequent classes at the cost of being less capable of understanding the least frequent ones. This issue is not necessarily a problem if our system only needs to identify the most frequent classes. However, when the system needs to be robust across all classes, the performance achieved in the least frequent ones will be lower. Furthermore, in data streams, there is a general assumption that the data is constantly been generated, which, as stated before, means we need to use it as it comes to avoid memory and storage issues. However, not all data comes with the class identified preemptively, which leads to the inability to use it when it arrives. Some systems solve this by trying to predict the class using the attributes of the data and using this prediction to fill in the class. However, if the properties of the data change the system might not be able to identify the correct class, leading the system to learn with wrong assumptions. This can be corrected once the data is validated through extra validation steps which can be time-consuming. Given these limitations, it is necessary to understand how the systems perform under different circumstances. Some systems are already capable of having decent performance even with some of the issues mentioned. However, these tend to be a *blackbox*, which means that is hard to explain the predicted classification. This further accentuates the necessity of understanding these issues beforehand.

1.2 Objectives and Research Questions

In order to tackle some of the problems mentioned in the previous section, we elaborated the following Research Questions:

1. How relevant is the impact of the delay in data validation in Multiclass Imbalanced data in State-of-the-Art learning systems?
2. What causes the most relevant impact on performance for Multiclass Imbalanced State-of-the-Art machine learning systems when faced with data validation delay?
3. Are generalized approaches suitable to deal with Multiclass Imbalanced data when there is a data validation delay?

To answer these questions, we searched and used the state-of-the-art Multiclass systems. In addition, we complement these with datasets from different sources to diversify the data properties (imbalanced and Multiclass). Some of these datasets were artificially generated to

reproduce some properties, while others were extracted from real-world examples. Making use of the systems mentioned, we developed a system that simulates delays on data validation to compare how the models are affected by this limitation when compared with models trained without delay. The comparison of the performance of these systems with and without delay, further allows us to evaluate the impact delay can have on the performance. In turn, this provides us with conclusions pertaining to the first two questions. Concerning the last question, we propose a system that makes use of State-of-the-Art metrics to select which of the State-of-Art Multiclass systems is the best whenever both new data and/or human-verified data arrives. These contributions allow the development of a more generalized system, able to adapt to new data characteristics on the fly.

1.3 Real World Contributions

The work of this dissertation contributes towards the implementation of more robust self-adapting Multiclass machine learning systems to deal with delays in data validation. The proposed work can also help to improve the performance of systems in real-world scenarios where such conditions apply. For example, scenarios where class verification is delayed which can contain gaps in the information at a certain time interval and systems that can have its data properties shift over time. One of these scenarios is present in the detection of gases through the usage of sensors, where sensor deterioration can cause a change in the properties of the data without necessarily rendering the sensor useless as long as the system adapts to this change. Alongside this, in the case of sensor failure, the system should be able to adapt on its own once the sensor is replaced. Another example can be seen in a simple Connect 4 game which illustrates the potential application of this system to other games as long as they have more than 2 possible states (win, loss, and draw in the case of Connect 4). Even the identification of forest cover types through the usage of pixel features can be described as a Multiclass problem that contains changes in properties over time. In general, the work of this dissertation contributes towards the improvement of current State-of-the-Art systems which have several different possible applications in real-world scenarios.

1.4 Organization

This document contains five chapters, which are described below.

Introduction 1 introduces the problems, the motivation for our work, the research questions, and a brief overview of what was done.

State of the Art 2 describes necessary core concepts and provides a systematic review of related works, describing what has already been achieved in this research topic and also the tools used in this work.

Proposed Solution 3 describes the approaches and methodologies used in a detailed way to

facilitate future related works.

Tests and Results 4 provides all the results obtained for this work and describes all the test scenarios used.

Conclusion 5 Summarizes the motivations and main contributions of this work. Alongside this, it also describes the limitations of the work and the direction for future research paths.

Chapter 2

State of the Art

This chapter introduces the core concepts used and discussed throughout this work. We begin by explaining the literature review criteria we defined for this research. Then, we define important concepts in the context of this work. Finally, we describe the state of the art related to these concepts, highlighting the most important contributions found in the literature.

2.1 Literature Review Method

To understand the literature, we decided to search the literature following a Systematic Literature Review Methodology. To do this, we first defined the search criteria to determine which studies were relevant to this work. These criteria were used as our inclusion criteria and to exclude works that are not relevant we excluded any that did not meet these as well. The search criteria defined were the following:

- Does the work introduce one of the relevant core concepts or aggregate information related to it?

If a work meets this criteria, we decided to include it as it provides a way to introduce the concepts. However, if a work does not meet this criteria, it does not necessarily mean it is irrelevant. Due to this, we defined some additional conditions to help dictate the relevance of a work when it does not meet the first criteria:

- Is the work from 2018 or more recent?

This condition was set primarily to limit the works researched to be of more relevancy as they had a smaller lifespan.

- Does the work introduce innovative approaches?

With this condition, we further limit the works to be of more relevancy, ignoring potential works that test more on already introduced technology.

- Does the work provide means which allow us to replicate its methods and results?

This criterion was set to reduce the number of works for which details on how the approaches were set up were limited. In addition, it was also set to limit the number of works by disregarding those with limited information regarding the datasets. Providing only works for which replication of the results was unfeasible.

Using this set of conditions, we established a way to minimize the number of works to explore without reducing their relevancy. Furthermore, this also allows for the introduction of some core concepts and the most relevant and recent advances concerning these. Next, we define the search terms we explored when researching. The selected terms were the following:

- Data Stream
- Concept Drift
- Multiclass
- Class Imbalance
- Data Validation

Using these search terms and the criteria defined, we extracted the relevant information by first introducing what each search term represents and then by describing the advances of each of them individually. It is also important to denote that these search criteria were searched simultaneously. For this, we used Google Scholar and ACM Digital Library to extract works that fit the criteria. In total, we denoted around 40 works as relevant.

2.2 Core Concept Introduction

Each of the search terms is a core concept that is relevant to the comprehension of this work. This section will introduce each of the core concepts.

2.2.1 Data Streams

Data Streams is the term used to describe data that is constantly being generated at a high rate. Furthermore, this factor makes it unreasonable to store and instead requires an analysis in real-time. This analysis can lead to the extraction of important knowledge and allow for important decisions as the data is generated. Therefore, Data Streams are an important source of knowledge to be explored [3, 10, 11, 23].

2.2.2 Concept Drift

Concept Drift is the change of statistical properties of the data which occurs in time-related data and real-time data (Data Streams). This change can reduce the efficacy of supervised machine learning models and lead systems towards making decisions based on old data which, due to the occurrence of drifts, are no longer relevant [13, 19, 52]. To reduce the effects caused by Concept Drift (CD), machine learning algorithms can be adjusted with three components:

1. **Concept Drift Detection**: A component used to alert the system whether or not a CD occurs.
2. **Drift Understanding**: A component with the goal of detecting when exactly a CD occurs, what caused it, and where exactly it occurred.
3. **Concept Drift Adaptation**: A component that dictates how the system will react to the existence of a drift in order to learn the new concept [30].

Furthermore, Concept Drifts can be distinguished into four types which can also be seen in Figure 2.1 (retrieved from [25]) and are the following:

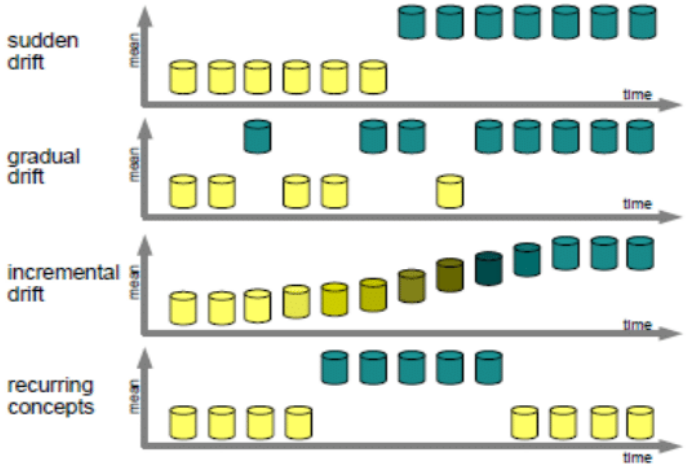


Figure 2.1: Types of Concept Drift.

1. **Sudden Drift**: A sudden change in the statistical properties of data, which remains consistent after the drift occurs.
2. **Gradual Drift**: Gradual Drift is the appearance of batches of data pertaining to a new concept with increasing sizes over time until this new concept eventually becomes the more prominent one.
3. **Incremental Drift** This drift describes a slow but steady change in the statistical properties of the data causing a slow change in the concept over time until the data properties stabilize on a new concept.

4. **Recurring Concepts**:

   Recurring Concepts are the reappearance/recurrence of CDs which leads to concepts previously observed in the data. [19, 30, 38]

2.2.3 Multiclass

Multiclass refers to a subgroup of classification problems in which there are more than two possible classes in a classification task. In these problems, machine learning models are tasked to learn and predict the target class value correctly. However, unlike binary cases, the models are required to better understand the distinction between each of the classes as with more potential labels, the harder it becomes for the system to predict the correct one. Furthermore, unlike Binary Classification, Multiclass may not be well evaluated using accuracy and other traditional metrics as these tend to be biased towards the Majority classes. Thus, metrics adapted for multiple class scenarios need to be used like Balanced Accuracy which is the mean of the Recalls of each class. In turn, this metric weighs all classes equally which allows us to find potential predictive problems toward under-represented classes [14].

2.2.4 Imbalanced Data

Imbalanced Data is a term that describes data for which the class distribution is skewed. This means, that one class (or multiple) in the data is insufficiently represented. These underrepresented classes are denoted as Minor or Minority classes and the others are denoted as Major or Majority classes. Furthermore, this discrepancy in class representation can lead machine learning systems to produce less favorable results as they will be less capable of discerning Minority classes which are often the classes of more interest. [1, 8, 29, 34]

2.2.5 Data Validation

Due to the fast generation of data in streams, data may not necessarily arrive fully prepared to be used. Thus an extra validation of data is necessary, for example through an outside unbiased annotator/expert. However, in most works, data quality is assumed to be present. [5] This in turn, can result in a delay where for some time part of the data can be unknown or not ready to be used as it has not been validated yet. This wait for the data to be usable is known as Data Delay.

2.3 Advances in the Literature

In this section, we will cover the literature related to the domain of this work.

2.3.1 Data Streams

Following the criteria set by our Literature Review Methodology, we noticed that most works for this concept revolve around three themes:

Field Specific Works This theme is denoted by works that try to make use of data stream methods in real-world environments like proposing a healthcare architecture for efficient healthcare data sharing [39] and more theoretical fields like detecting Multivariate Anomaly Detection for Time Series data [26].

General Surveys This theme mostly tries to summarize the current methods used to tackle Data Streams and where future work is still necessary. One of these focuses mostly on the advances regarding big data service architecture [12] while another targets data analysis in a large-scale wireless network[32]. Another survey focuses mostly on anomaly detection for Internet of Things [9]. Finally, the last survey analyzed focuses on exploring the State of the Art and identifying current and future trends of the area [42]

Online Learning The last theme tackled in this section has been one of the most relevant in the Data Stream area. Online learning revolves around machine learning methods that try to tackle tasks by learning from a sequence of data instances one at a time, avoiding having to learn from an entire training data set at once. Furthermore, this area has become more prominent as online learning algorithms are often easy to understand, simple to implement and, alongside this, also help tackle an area in urgent need, Big Data Streams. [18] We can further see some implementations that try to tackle this problem through the use of neural networks [28, 41] and some that try to tackle some real-world problems using online learners. [33]

2.3.2 Concept Drift

To tackle CD, several works tried to make their own approach regarding its detection while others tried to understand its impact on current systems. One work proposed an approach that makes use of entropy as a CD detector alongside an evolving ensemble of classifiers. This method proved to improve the performance in the presence of Sudden and Gradual CDs [31]. Another work proposed an unsupervised method using a discriminative classifier over a sliding window which they named D3 (Discriminative Drift Detector). This method tries to discern the presence of CD by comparing two sets of observations through the estimation of their distributions and changes between these. In addition, this method further proved to improve performance when compared with other widely used CD detectors (ADWIN, DDM, EDDM) [51]. Other works focused on exploring the impact caused by CD in general [20, 38] and on performance-aware drift detectors [4]. Finally, the study conducted in [2] tried to make use of Dynamic Ensemble

Selection which proved to be better than baselines. Although these works present important contributions to CD methods, they only explore binary class scenarios.

2.3.3 Imbalanced Data

The work in [47] explored resampling-based ensembles, OOB (Oversampling Online Bagging), and UOB (Undersampling Online Bagging) as a way to deal with Imbalanced Data Streams. In the same work, these methods were used together in ensembles using an adaptive weight adjustment. These approaches provided competitive results, with the weighted ensembles proving to produce the best results. Furthermore, to tackle the problem of imbalanced data, several methods were proposed over the years, with the more prominent being SMOTE (a synthetic minority oversampling technique)[7], ADASYN (an adaptive synthetic sampling approach)[17] and ANN (Artificial Neural Network)[48]. These and other methods were evaluated in the study by Kaur et al..

2.3.4 Imbalanced Data with Concept Drift

The authors in [44] tried to tackle the problem of imbalanced data with concept drift with the development of a CD detector named DDM-OCI. This CD detector measures the reduction of the recall in the Minority class to detect CDs. This method was shown to be faster at responding to new concepts when compared with the method it is based on, DDM. However, as the authors note, the method assumes that the system knows the minority class and it proved to need a reduction in the impact of false alarms detected. Other work proposed the usage of a Classifier Selection method alongside chunk-based ensemble learning. This method proved very effective compared to other classifier selection methods, especially at high levels of Imbalance Ratio (IR).[49]. A different study proposed the usage of both Data Processing and Dynamic Ensemble Selection methods together and proved that this combination is significantly better than the approaches that the individual approaches [50]. Concerning the same topic, another study proposed the usage of Dynamic Ensemble Selection which makes use of several resampling mechanisms to increase class presence of minority classes. This method improved the recognition rate of Minority classes [15]. The work presented in [35] proposed the usage of a deep learning framework that makes use of an Adadelta optimizer-based deep neural network, an adaptive synthetic technique to handle class imbalance, and ADWIN to detect CD. This method named CIDD-ADODNN proved to be the best when compared with several other baseline methods. The study in [21] proposed a method using Dynamic Ensemble Selection which made use of an improved SMOTE technique to produce minority instances and constantly calculates the competence of its Dynamic Ensemble models to only use the best ones at all times. This method produced solid results when compared with other methods. In general, the methods proposed to tackle this problem include the proposal of new CD detectors, new methods to balance the data for training, and new models.

2.3.5 Metrics in Imbalanced Multiclass with Concept Drift

To better evaluate the performance of systems faced with Imbalanced Multiclass data with CD, a series of new metrics were proposed. These new metrics try to adapt the Area Under Curve (Area Under Curve (AUC)) measure to a scenario where data contains multiple classes with imbalanced representation. One of the metrics is Prequential Multiclass Area Under Curve (PMAUC) (Prequential Multiclass AUC) which is a combination of two other metrics PAUC (Prequential AUC). This has proved to be effective at indicating CD in Binary Classification [6]. Another metric is MAUC (Multiclass AUC) which simply averages the AUC scores between each class pair. [16]. Two other metrics were also proposed: Weighted Area Under Curve (WAUC) (Weighted AUC) and Equally Weighted Area Under Curve (EWAUC) (Equally Weighted AUC). Both of these extend a weighted AUC from another work [36, 37] for data stream learning. The only difference between WAUC and EWAUC is that WAUC weighs the AUC value over class frequency while EWAUC uses the same weight for all AUC values for each class. Furthermore, using these metrics it was also shown that using a Page-Hinkley Test (PHT-test) as a CD detector, PMAUC and EWAUC reflect well the impact of CDs and EWAUC alongside G-Mean provided a better True Detection Rate. These AUC-based metrics when used as detectors, in general, proved to bring a significant classification improvement. [43]
Aside from newly introduced metrics, the authors in [24] tried to tackle this problem through the usage of a trainable drift detector, RBM-IM which is based on the Restricted Boltzmann Machine, a neural network with a skew-sensitivities loss function. This CD detector proved to be effective at different levels of IR while having the advantage of adapting to the current data stream.

2.3.6 Models in Imbalanced Multiclass with Concept Drift

In the remaining works analyzed, models were implemented to address the problem. One of these works proposed two models: Multiclass Oversampling-based Online Bagging(Multiclass Oversampling-based Online Bagging (MOOB)) and Multiclass Undersampling-based Online Bagging (Multiclass Undersampling-based Online Bagging (MUOB)). These models adapt OOB and UOB from the work in [47] to a multiclass scenario. These in turn proved to be competitive with other methods. [46] Another study, reviewed the usage of boosting methods to tackle the problem, which showed they were not suitable for big datasets with high levels of IR. [40] The last work that met our literature review criteria, proposed an active learning method that made use of a uncertainty strategy based on an asymmetric margin threshold matrix. This leads the proposed method to assign higher training weights to new data and Minority class samples. Furthermore, this method proved to be more efficient when compared with other active learning algorithms. [27]

From all the works extracted, we can see that many different approaches produce competitive results. However, most of these infer data validation which may not be true for all systems. Furthermore, works mostly try to tackle individual problems with specific methods. Due to

these, this work will touch on the topics researched while introducing methods to induce data validation delay and propose a more generalized approach toward generalized data.

Chapter 3

Proposed Solution

This chapter describes the methodology used in the development of an experimental workflow to answer the research questions previously established in this dissertation.

Our objective with the development of this framework was to create an environment where testing Imbalanced Multiclass Classification (Imbalanced Multiclass Classification (IMC)) systems could be easily adjusted to simulate different properties through a simple change of settings. Alongside this, the framework was also developed to allow for an easy implementation of new models and easy change of its underlying components. To do this, we implemented an initial framework that followed other State-of-the-Art works, primarily the studies where Area Under Curve (AUC)-based metrics were tested [43]. Once this initial framework was implemented, we complemented it with additional modules to try to answer the defined Research Questions. Thus, to answer the first two research questions (listed below),

1. How relevant is the impact of the delay in data validation in Multiclass Imbalanced data in State-of-the-Art learning systems?
2. What causes the most relevant impact on performance for Multiclass Imbalanced State-of-the-Art machine learning systems when faced with data validation delay?

we decided to modify the original framework so that it could simulate human verification delay. For the last Research Question:

3. Are generalized approaches suitable to deal with Multiclass Imbalanced data when there is a data validation delay?

, we implemented a method that makes use of multiple models to create a more generalized approach. The implementation of the initial framework and our approaches are further described below.

3.1 Initial framework

Firstly, we decided to implement an initial framework with the objective of training models for the task of IMC problems with CD. This framework is based on the work in [43]. Furthermore, we chose two State-of-the-Art models which produced very competitive performance when compared with more complex models, Multiclass Oversampling-based Online Bagging (MOOB) and Multiclass Undersampling-based Online Bagging (MUOB) [45]. Furthermore, it is important to emphasize some functions, some of which are very important as they help the system tackle IMC issues and others that help us understand the data. These functions are described in the subsections below.

3.1.1 Class Percentage Update - UpdateClassPerc

This function has the purpose of updating the percentage value of the presence of each class up to a given data index. This is a very important task as MOOB and MUOB use the class percentage to calculate how much they should oversample or undersample respectively. Furthermore, it does not calculate the exact class percentage and instead makes use of a size decay factor variable (for our work it used the same value used by the similar framework of 0.9 and was defined as a variable that is received by this function). This function uses this variable as a way to define how much the class percentage goes up or down at each time step depending on whether a class is the one present in the last data entry or not. The function makes use of the four arguments:

- **class percentage matrix**: Contains the class percentage of each class at all moments
- **the class of the last data instance**
- **the total number of data instances received**
- **size decay factor**

Furthermore, it also makes use of a global variable that stores the number of different classes. Using these variables, the function checks if it is the very first data instance, and if it is, it changes the class percentage value for the received class only to 1. If it is not the first data point, the function goes through each class and updates their respective class percentage based on one of two possible scenarios:

- **Class is not the same as the last received one**: The class percentage at data index i is set to the value of the class percentage at data index (i-1) multiplied by the size decay factor.
- **Class is the same as the last received one**

The class percentage at data index i is set to the value of the class percentage at data index (i-1) multiplied by the size decay factor and then it is added to the value of the difference of 1 minus the size decay factor.

This function does not return anything as it only updates a global variable. Furthermore, it was implemented assuming numerically assigned classes where each class is given a number starting from 0. The pseudo-code for this function can be further seen in Algorithm 1.

Algorithm 1 UpdateClassPerc

```
1:  procedure UpdateClassPerc(currentIndexClass, currentIndexNum, sizeDecayFactor)
2:      if currentIndexNum > 1 then
3:          i ← 0.
4:  loop:
5:          value ← 0.
6:          if i = currentIndexClass then
7:              value ← classPercentage[currentIndexNum][currentIndexClass − 1]×
8:  sizeDecayFactor + (1 − sizeDecayFactor).
9:          else
10:             value ← classPercentage[currentIndexNum][currentIndexClass − 1]×
11: sizeDecayFactor.
12:         classPercentage[currentIndexNum][currentIndexClass] ← value
13:         i ← i + 1.
14:         if i < totalNumberClasses then
15:             goto loop.
16:     else
17:         classPercentage[currentIndexNum][currentIndexClass] ← 1.
```

3.1.2 Decay Recall Update - UpdateRecall

This function just like the last one has the sole purpose of updating another global variable, in this case pertaining to the current recall value of each class. This variable is used in another function, *Imbalance Status Detection*, which is further detailed in Section 3.1.3. To update this global variable, this function receives three arguments, the class of the current data instance, whether the system predicted the class correctly or not, and the recall decay factor (which we used in the same way as the size decay factor and also with the same static value of 0.9). Alongside these, it also makes use of a global variable that stores the total number of instances of each individual class. Using all of these variables, the function can do three things depending on different scenarios:

- **The system predicted correctly and it is the first data instance of a class**: In this situation, the function simply attributes the value of 1 to the recall for this class.

- **The system predicted correctly and it is not the first data instance of a class**: When it is not the first data instance and it is predicted correctly, the recall value of the class is multiplied by the recall decay factor and then it is added the difference of 1 minus the recall decay factor.
- **The system predicted incorrectly and it is not the first data instance of a class**: In this case, the function simply multiplies the recall value of the class by the recall decay factor.

The pseudo-code for this function can be seen in Algorithm 2.

Algorithm 2 updateRecall

```
1: procedure UpdateRecall(currentIndexClass, isCorrect, recallDecayFactor)
2:     value ← 0.
3:     if numberOfClassInstances[currentIndexClass] > 1 and isCorrect = True then
4:         value ← 1.
5:     else if isCorrect = True then
6:         value ← currentClassRecall[currentIndexClass] × sizeDecayFactor+
7: (1 − sizeDecayFactor).
8:     else if isCorrect = False then
9:         value ← currentClassRecall[currentIndexClass] × sizeDecayFactor.
10:    currentClassRecall[currentIndexClass] ← value
```

3.1.3 Imbalance Status Detection - ImbStatus

This function has the objective of defining which classes appear more often in the data (majority class) and which ones appear less (minority class). Alongside this, it can also denote a class as being neither of them as it is too frequent to be classified as a minority class and too infrequent to be a majority class when compared with others. Furthermore, this function also updates a global variable which represents whether the data so far is balanced or imbalanced. To achieve this, it makes use of the global variables updated in both Class Percentage Update (3.1.1) and Decay Recall Update (3.1.2), the global variable which contains the number of instances pertaining to each class and the one which contains the number of different classes. This function also receives three arguments, the total number of data samples and two factor values which are predefined and used for the calculations on this function (in our work we used the static values as used in the similar work, 0.4 for the first value and 0.3 for the second one) and are the following:

- **First factor value**: Describes the minimum difference in class representation percentage between 2 classes for the system to be able to classify one of them as a majority class and the other as a minority class
- **Second factor value**: Describes the minimum difference in class recall between 2 classes for the system to be able to classify one of them as a majority and the other as minority

Furthermore, this function updates constantly three more global variables which will contain a list of all the classes which are minority, majority, or neither of them. First, the function clears these three lists. Then, it checks if the difference between each pair of classes is bigger than the first factor variable. In addition, it also checks if the difference between their current class recalls is bigger than the second factor variable. If both of these conditions are met, the class with bigger values is added to the majority class list if it is not present in it yet. The same is done for the other class but adding it to the minority class if it is not considered one yet. This is all done by ignoring classes that do not contain instances yet. Once this is done, the function proceeds to check the presence of each class in the minority and majority lists. If a class is present in both of them, it is removed from the majority list, being considered only a minority class. Furthermore, if a class does not appear in either list, it is added to the normal class list as it is portrayed as neither majority nor minority. After all of these, the function checks if the minority and majority lists are empty. If they are the global variable which represents whether the data is balanced or not is updated to say that the data is balanced. Otherwise, this variable is updated to represent an imbalance in the data. The pseudo-code for this function is presented in Algorithm 3.

3.1.4 Reset Imbalance Status - resetImbStatus

This function only changes the balance state variable to True and empties the list which contains the number of instances pertaining to each class. This function is important for further data understanding as will be seen in 3.1.6. This function can be seen in more detail in Algorithm 4.

3.1.5 Reset Model - resetModel

The purpose of this function is to reset a model by retraining it using only the data instances it receives. To do this, it receives both a model and the data instances to train it with. With these, it first calls a function that resets the received model to an untrained state and then proceeds to train it using the instances until there are no more instances. Due to the easy description of this function, we decided to not provide additional pseudo-code.

3.1.6 Drift Detector - driftDetector

The last function has the purpose of updating the variables that the drift detector uses and resetting the models and multiple variables in case a drift is detected. The function receives as arguments the model being used, the drift detector (which in our case is a Page-Hinkley Test drift detector as we decided to use a similar one to the work we based the initial framework on), the current data instance, the value obtained of a AUC-based metric on the last data point and the index of the last data entry. Based on these arguments, the function updates a global variable that contains a AUC-based metric over the last window of data. This procedure starts by calling a function that makes use of the AUC-based metric on the last data instance. This

Algorithm 3 ImbStatus

```
1: procedure IMBSTATUS(delta1, delta2, currentIndexNum)
2:     minorityClasses ← ∅.
3:     majorityClasses ← ∅.
4:     normalClasses ← ∅.
5:     i ← 0.
6: loop:
7:     if not numberOfClassInstances[i] = 0 then
8:         j ← i + 1.
9: loop2:
10:        if (classPercentage[currentIndexNum][j] - classPercentage[currentIndexNum][i] >
11: delta1) and (currentClassRecall[j] - currentClassRecall[i] > delta2) then
12:            if not majorityClasses ∩ j then
13:                majorityClasses ← majorityClasses + {j}.
14:            if not minorityClasses ∩ i then
15:                minorityClasses ← minorityClasses + {i}.
16:            j = j + 1.
17:            if j < totalNumberClasses then
18:                goto loop2.
19:    i = i + 1.
20:    if i < totalNumberClasses - 1 then
21:        goto loop.
22:    i ← 0.
23: loop3:
24:    if not numberOfClassInstances[i] = 0 then
25:        if (majorityClasses ∩ i) and (minorityClasses ∩ i) then
26:            majorityClasses ← majorityClasses − {i}.
27:        if ( not majorityClasses ∩ i) and (not minorityClasses ∩ i) then
28:            normalClasses ← normalClasses + {i}.
29:    i = i + 1.
30:    if i < totalNumberClasses then
31:        goto loop3.
32:    dataBalance ← False
33:    if minorityClasses = ∅ and majorityClasses = ∅ then
34:        dataBalance ← True
```

Algorithm 4 resetImbStatus

```
1: procedure RESETIMBSTATUS
2:     dataBalance ← True
3:     i ← 0.
4: loop:
5:     numberOfClassInstances[i] ← 0.
6:     i ← i + 1.
7:     if i < totalNumberClasses then
8:         goto loop.
```

is done to check if the detector has observed a performance drop big enough to trigger just a warning or if the drop was large enough to cause a drift detection. Depending on the outcome of this function, the driftDetector function will execute different commands:

- **Only a warning was triggered**: The function will store the data instance for later training in case of drift detection, preparing for the worst.
- **No warning was triggered nor a drift is detected**: The function will clear all instances saved in case of a potential drift, as there is no threat currently due to no triggered warning.
- **CD is detected**: Firstly, the function will call *resetImbStatus* as it will start training the model from scratch and so it will ignore all previous instances until the model is further trained. Then, it calls *resetModel*, resetting it to fit to a new concept with the most recent data points only. Following this, the variable relating to the AUC-based metric over the last window is reset as it will be necessary to recalculate it from scratch again from this point onwards. Finally, the function adds the data index to a global list when the drift occurred and increments 1 to a global variable which stores the total number of drifts that were detected.

After going through one of these options, the function finalizes its execution by returning whether a warning was triggered, a drift was detected, or neither through a value of 1,2 or 0 respectively. The pseudo-code for this function is presented in Algorithm 5.

These functions are the foundations of the systems proposed. The remaining components of the framework have the sole purpose of simulating a training moment on the fly using a data stream. This is done by training until there are no more training samples. Along with this training, the system also tries to predict the class of a data instance before using it to train. This is done to assess its performance. To do this before every single test run the framework initializes some variables which are used in its composing functions:

- currentIndexNum
- classPercentage

Algorithm 5 driftDetector

```
1: function DRIFTDETECTOR(model,detector, currentInstance, metricValue, currentIndex)
2:     driftDetected ← DETECT(detector, metricValue)
3:     if driftDetected = 1 then
4:         storedInstances ← storedInstances + {currentInstance}
5:     else if driftDetected = 2 then
6:         RESETIMBSTATUS
7:         RESETMODEL(model, storedInstances)
8:         windowMetric ← 0
9:         driftLocation[numDrifts] ← currentIndex
10:        numDrifts ← numDrifts + 1
11:    else
12:        storedInstances ← ∅
13:    return driftDetected
```

- numberOfClassInstances
- currentClassRecall
- sizeDecayFactor
- recallDecayFactor
- delta1
- delta2
- driftLocation
- numDrifts
- storedInstances

After this, the framework initializes the drift detector and one of the two possible models (MOOB or MUOB). Furthermore, this initialization makes use of the first data instance and a seed number which will determine the randomness of the models. The framework also makes use of another variable which is defined by the user before each run and contains the window size for our data. Once all these steps have been performed the framework starts to execute a run. During a run, the system always starts by trying to predict the class of the new piece of data. Then, the framework increases the value of two variables by one: current index number (currentIndexNum) and the number of instances of each class that have been seen by the system (numberOfClassInstances), increasing this latter variable only on the index which contains the currently seen class.

With these first steps concluded, the framework updates the class percentage variable (classPercentage) by calling the function UpdateClassPerc and then proceeds to train the model

using the new data entry and the updated class percentages for oversampling/undersampling (depending on the chosen model). Once the training is concluded. The framework then checks if the prediction was correct and updates both the class recalls and the imbalance status of the data by calling updateRecall and ImbStatus functions respectively. However, these are not necessary for the general work of the framework as they contain only further information about the data. Following these steps, the performance metric selected pertaining to the last window of data is calculated. Finally, if the current data index is bigger than the window size, the framework checks for CD by calling the driftDetector function and then proceeds to repeat the whole process until there are no more data instances. This initial framework can be further seen in Algorithm 6, it is important to note that the classIndex variable simply represents the index position in the data instance where the class is described. Furthermore, its process after the first data instance initialization can be seen in Figure 3.1.

Algorithm 6 Initial Framework

```
1: procedure iniFrame
2:     model ← initializeModel(chosenModel, data[0], seed)
3:     detector ← initializePHTdetector()
4:     i ← 1.
5: loop:
6:     currentInstance ← data[i]
7:     prediction ← modelPrediction(model, currentInstance)
8:     currentIndexNum ← i
9:     currentIndexClass ← currentInstance[classIndex]
10:    numberOfClassInstances[currentIndexClass] ← numberOfClassInstances[currentIndexClass]+
11: 1
12:    updateClassPerc(currentIndexClass, currentIndexNum, sizeDecayFactor)
13:    trainModel(model, currentInstance, currentIndexNum)
14:    isCorrect ← False
15:    if prediction = currentIndexClass then
16:        isCorrect ← True
17:    updateRecall(currentIndexClass, isCorrect, recallDecayFactor)
18:    imbStatus(delta1, delta2, currentIndexNum)
19:    metricValue ← calculatePerformance(selectedMetric)
20:    if i > windowSize then
21:        driftDetector(model,detector, currentInstance, metricValue, currentIndexNum)
22:    i ← i + 1.
23:    if i < totalNumberOfDataInstances then
24:        goto loop.
```

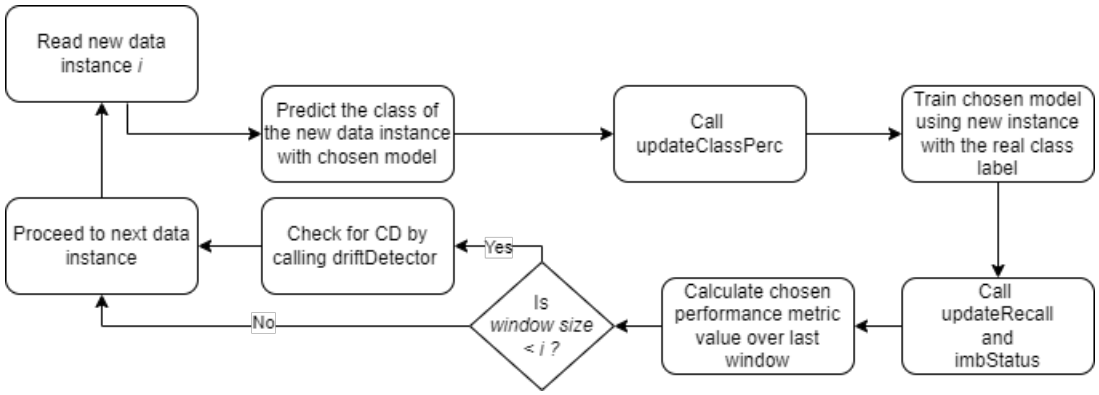


Figure 3.1: Training after the starting window of data by the initial framework.

3.2 Our approaches

As the initial framework had the single purpose of training a set of baseline models, we decided to adjust it as it could not help us answer the defined research questions on its own. To do this, we first adjusted the framework so it could simulate potential delays on the ground-truth labels for each data instance (3.2.1). Furthermore, to try to answer the last research question we also had to implement our own generalized approach using the delayed framework, which tries to use both implemented models as much as possible (3.2.2).

3.2.1 Delayed framework

The delayed framework has the sole objective of providing an environment where testing different delays could be easily done using already implemented models. To do this, the system needs to simulate a constant delay in the computation of the performance metrics due to the delay in the ground-truth labels. To simulate this, the window size is used as the delay value (resulting in a static delay in each test). Consequently, the system was adapted to be a self-supervised learning system, meaning it preemptively predicts the class value of the data instance and then uses it to train the model. However, MOOB and MUOB make use of the class representation percentage to oversample and undersample respectively. As the system does not know how much this value should be as it some misses real class values, it should in turn use the class percentage value that it knows which is defined by the delay (in our case, the window size). Following this, the framework will always make use of the class percentage of moment i - *window size*, where i represents the index of the last piece of data received. Alongside this, the CD detector also makes use of the defined performance metric at the same moment as the class percentage. Furthermore, we implemented it so the first window-sized data batch of instances is trained into the system as if there were no delays. This should in turn simulate an initial training batch before the system starts its tests. Keeping these in mind, we concluded there were two potential additions to the initial framework that would result in the ability to simulate the delay:

- **Calculating the performance of the system and updating the class percentages of data index *i* - *window size* at data instance *i***: This method saves up data instances and their predictions pertaining to the interval [*i* - *window size*, *i*] at all moments. It makes use of the data instance and its prediction from instance (*i* - *window size*) whenever a new data instance arrives, to calculate the new value for the performance metric and to update the class percentages. This results in the necessity for the system to save both instances and predictions.
- **Calculating the performance of the system and updating the class percentages of data index *i* at data instance *i***: This method results in a similar performance calculation and class percentage update as the initial framework. However, as is, it would not simulate a delay. To do so, the system only makes use of the class percentage and performance metric at moment *i* - *window size*. This, in turn, requires the system to save at least the class percentages and the performance metrics from moments in the interval [*i* - *window size*, *i*].

Following the latter method, we also made it so the framework would use the information of moment *window size* until data index *(window size* $\times$ *2)*, as this should be the latest information the system knows until that point.

With all of these in mind, we made it so the framework makes use of a variable to decide whether it should use a model with the delayed framework or the initial one. Furthermore, two functions had to be implemented in a new way which made use of delayed information only. These were the training functions pertaining to MOOB and MUOB which were changed to also receive a delayed class percentage instead of making use of the global class percentage variable. Following all these, when in use, the delayed framework first trains the model using the first window-sized data batch. Following this, at every data instance until time window (*window size* $\times$ *2*), the framework predicts the class label and uses it to train the model with the class percentages from moment *window size*, while always saving the class percentages and performance metrics of moment *i* as if there was no delay. Once data instance *window size* $\times$ *+ 1* arrives, the system keeps training with predictions but using class percentages and performance metric value from index *i* - *window size*, while saving the respective class percentages and performance metric value of moment *i*. As some functions did not impact this training, we decided to not change them (ImbStatus and updateRecall). The resulting code can be further seen in Algorithm 7. Furthermore, its process after the first data instance initialization is presented in Figure 3.2.

3.2.2 Hybrid approach

To try to answer the last research question:

3. Are generalized approaches suitable to deal with Multiclass Imbalanced data when there is a data validation delay?

Algorithm 7 Delayed Framework

```
1: procedure DELAYFRAME
2:     model ← INITIALIZEMODEL(chosenModel, data[0], seed)
3:     detector ← INITIALIZEPHTDETECTOR()
4:     allClassPercentages ← ∅.
5:     performanceResults ← ∅.
6:     i ← 1.
7: loop:
8:     currentInstance ← data[i]
9:     prediction ← MODELPREDICTION(model, currentInstance)
10:    currentIndexNum ← i
11:    currentIndexClass ← currentInstance[classIndex]
12:    numberOfClassInstances[currentIndexClass] ← numberOfClassInstances[currentIndexClass]
13: +1
14:    UPDATECLASSPERC(currentIndexClass, currentIndexNum, sizeDecayFactor)
15:    allClassPercentages ← allClassPercentages + {classPercentages}
16:    delayedIndex ← i − windowSize
17:    trainingInstance ← prediction
18:    if i < windowSize then
19:        delayedIndex ← i
20:        trainingInstance ← currentInstace
21:    else if i < (windowSize × 2) then
22:        delayedIndex ← windowSize
23:    TRAINMODELDELAYED(model, trainingInstance, currentIndexNum, allClassPercentages[delayedIndex])
24:    isCorrect ← False
25:    if prediction = currentIndexClass then
26:        isCorrect ← True
27:    UPDATERECALL(currentIndexClass, isCorrect, recallDecayFactor)
28:    IMBSTATUS(delta1, delta2, currentIndexNum)
29:    metricValue ← CALCULATEPERFORMANCE(selectedMetric)
30:    performanceResults ← performanceResults + {metricValue}
31:    if i > windowSize then
32:        DRIFTDETECTOR(model,detector, currentInstance, performanceResults[delayedIndex], currentIndexNum)
33:    i ← i + 1.
34:    if i < totalNumberOfDataInstances then
35:        goto loop.
```

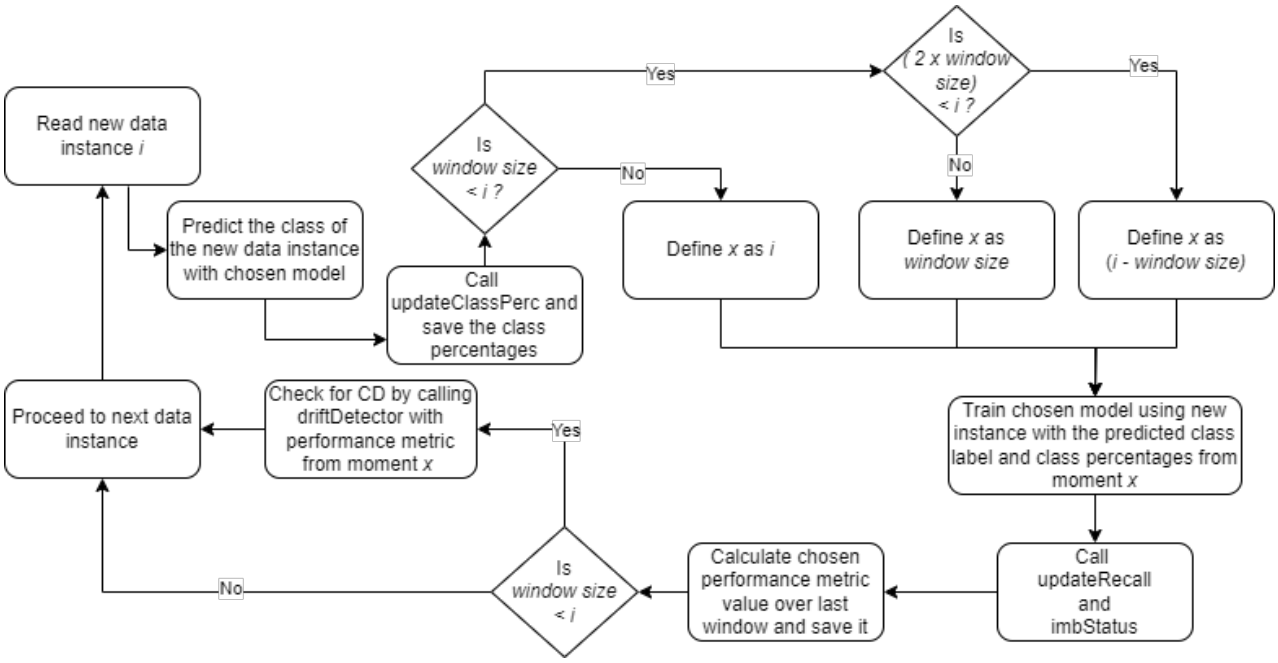


Figure 3.2: Training after the starting window of data by the delayed framework.

we tried to implement an approach that makes use of the simulated delay produced in the delayed framework while adjusting it to make use of both models. This approach tries to generalize the problem by training both MUOB and MOOB and choosing which it sees as the best whenever a new data instance arrives. This, in turn, should make the system adapt whenever the data is easier to predict using either an oversampling or an undersampling-based model as these can have varying results mostly influenced by the class imbalance. To determine the fitness of a model, this approach makes use of the same global variables as the delayed framework (the class representation percentage variable, and the performance metric at each data instance variable). However, as this method has the objective of using the best of two models at each given moment, it features a global performance metric variable for each model, saving the performances at each data point. First, just like the delayed framework, during the first window-sized data batch, both models are trained with the respective real class labels as if there was no delay (just to achieve the starting points of the models). While this training occurs, both models also produce their predictions of the class labels which are used to calculate the performance metric value for each model. Once this initial training moment is finished, the approach starts making use of the *delayedIndex* variable. This variable saves the last data index for which the system knows the class representation percentages (this variable was defined in the delayed framework). Using this variable, whenever new data arrives, the method will first predict using both models, save their performance for moment i, and then compare the performance of both models at the moment defined by the aforementioned variable. With this comparison, the method then decides on the best model. Once this decision is made, the prediction pertaining to the chosen model is used to train both models using delayed class percentages. It is important to denote that this approach made a slight adaptation to the saved delayed class percentages for each moment. Instead of making use of the class percentages of moment i - *window size*, our approach only updates the

class percentages (after the initial training) whenever a full window-sized data batch is processed. However, it still keeps the same delay of a full *window size* at this moment, updating the class percentages whenever $i \mod \textit{window size} = 0$ and saving the class percentages from the moment i - *window size* whenever this is true. Lastly, the only missing difference of this method compared to the delayed models is that it makes use of two Concept Drift (CD) detectors, one for each model. In other words, it calls the drift detector function for each model individually following the same guidelines as the ones set by the delayed framework. This should make each model reset individually without affecting the other if a CD is detected. The resulting code can be further seen in Algorithm 8. The steps taken during the training process of this approach after its initial window of training are also described in image 3.3.

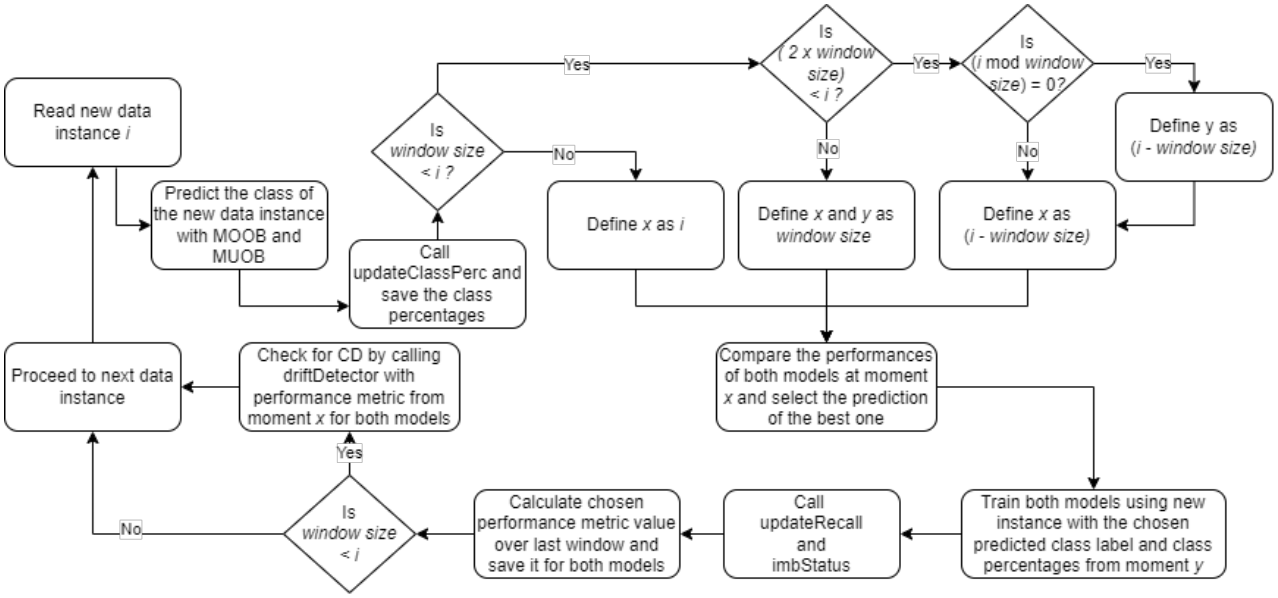


Figure 3.3: Training after the starting window of data by the Hybrid Approach.

Algorithm 8 Hybrid Approach

```
1: procedure HYBRID
2:     model1 ← INITIALIZEMODEL(MUOB, data[0], seed)
3:     model2 ← INITIALIZEMODEL(MOOB, data[0], seed)
4:     detector1 ← INITIALIZEPHTDETECTOR()
5:     detector2 ← INITIALIZEPHTDETECTOR()
6:     allClassPercentages ← ∅.
7:     performanceResults1 ← ∅.
8:     performanceResults2 ← ∅.
9:     i ← 1.
10:    classPercIndex ← 1
11: loop:
12:    currentInstance ← data[i]
13:    prediction1 ← MODELPREDICTION(model1, currentInstance)
14:    prediction2 ← MODELPREDICTION(model2, currentInstance)
15:    currentIndexNum ← i
16:    currentIndexClass ← currentInstance[classIndex]
17:    numberOfClassInstances[currentIndexClass] ← numberOfClassInstances[currentIndexClass]
18: +1
19:    UPDATECLASSPERC(currentIndexClass, currentIndexNum, sizeDecayFactor)
20:    allClassPercentages ← allClassPercentages + {classPercentages}
21:    delayedIndex ← i − windowSize
22:    if i < windowSize then
23:        delayedIndex ← i
24:        classPercIndex ← i
25:        trainingInstance ← currentInstace
26:    else if i < (windowSize × 2) then
27:        delayedIndex ← windowSize
28:        classPercIndex ← windowSize
29:    else if i mod windowSize = 0 then
30:        classPercIndex ← i − windowSize
31:    if delayedIndex ≥ window size then
32:        prediction ← prediction2
33:        if performanceResults1[delayedIndex] > performanceResults2[delayedIndex] then
34:            prediction ← prediction1
35:    trainingInstance ← prediction
36:    TRAINMODELDELAYED(model1, trainingInstance, currentIndexNum, allClassPercentages[classPercIndex])
37:    TRAINMODELDELAYED(model2, trainingInstance, currentIndexNum, allClassPercentages[classPercIndex])
38:    isCorrect ← False
```

39: **if** *prediction* = *currentIndexClass* **then**
40: *isCorrect* ← *True*
41: updateRecall(*currentIndexClass*, *isCorrect*, *recallDecayFactor*)
42: imbStatus(*delta1*, *delta2*, *currentIndexNum*)
43: *metricValue* ← calculateModelPerformance(*selectedMetric,model1*)
44: *performanceResults1* ← *performanceResults1* + {*metricValue*}
45: *metricValue* ← calculateModelPerformance(*selectedMetric,model2*)
46: *performanceResults2* ← *performanceResults2* + {*metricValue*}
47: **if** *i* > *windowSize* **then**
48: driftDetector(*model1*,*detector*, *trainingInstance*, *performanceResults*[*delayedIndex*], *currentIndexNum*)
49: driftDetector(*model2*,*detector*, *trainingInstance*, *performanceResults*[*delayedIndex*], *currentIndexNum*)
50: *i* ← *i* + 1.
51: **if** *i* < *totalNumberOfDataInstances* **then**
52: **goto** *loop*

Chapter 4

Tests and Results

This chapter contains the details of how our tests were set up using the framework and system described in Chapter 3.2, what was implemented in the framework to allow for testing, which datasets were used, and the respective results of each test.

4.1 Additional implementations

The frameworks implemented, using the global variable that saves the performance at each time step were already capable of keeping track of an individual test. However, as we further wanted to test multiple datasets, with different window sizes and different models, we made some additional implementations on the existing framework. First, we made it so the global variable containing the window size could contain multiple different window sizes. The same was done for the chosen model variable, allowing the system to test all possible pairs of window size and chosen model. However, we made it so the hybrid approach would always run using the framework from 3.2.2. Alongside all these, the system would run each test with each setting a user-defined number of times. This was done to better understand each of the models in each setting as Multiclass Oversampling-based Online Bagging (MOOB) and Multiclass Undersampling-based Online Bagging (MUOB) have some randomness associated with them. Furthermore, to make the tests consistent we made it so the number of tests defined by the user is used as the seed number for the tested models, ranging from 1 in the first run to the user-defined number. Using all of these, the system then saves the mean performance metric value overall runs in a file, and in another file, the system saves the mean performance metric value at each time step for each test set. This means the system in total saves two files for each test setting.

4.2 Datasets

To test the approaches proposed in Chapter 3.2, we used multiple datasets with different properties. These datasets can be divided into two categories: real-world and synthetic.

4.2.1 Real-world Datasets

4.2.1.1 Connect-4

This dataset contains 67557 board states of Connect 4 split into 42 attributes. Each of these attributes describes a position on the board and whether it is taken by the first player, the second player, or if it is still blank. Alongside this, each entry is described by one of three classes: Win, Loss, and Draw

All of these three classes represent the outcome for the first player. This dataset features a very significant class imbalance as shown in the table below:

Table 4.1: Class distribution for Connect-4 Dataset

Win	Draw	Loss
44473	6449	16635

This dataset contains concept drifts pertaining to less frequent board states.

4.2.1.2 Gas Sensor Array Drift Dataset at Different Concentrations

This dataset contains 13910 measurements of 16 sensors exposed to multiple gases at different concentration levels over 36 months. This is split into 129 attributes. One of these attributes represents the gas concentration level while the other 128 represent 8 measurements recorded by each of the different sensors. Each entry of this dataset represents one of 6 classes which are numbered from 1 to 6 and which represent the following gases:

1. Ethanol
2. Ethylene
3. Ammonia
4. Acetaldehyde
5. Acetone
6. Toluene

However, as this dataset was used in the same work that explored the AUC-based metrics [43], we decided to use the same batches that were used for that work. Given this, the first two batches of data were used which contain 5289 examples. Furthermore, these two batches contain entries from specific months from the whole gas sensor data collection. Chosen batches contain the instances collected in the first 4 months and months 8 through 10. The data distribution during these months can be seen below:

Table 4.2: Class distribution for the Gas Sensor Dataset

Month	Ethanol	Ethylene	Ammonia	Acetaldehyde	Acetone	Toluene
1	76	0	0	88	84	0
2	7	30	70	10	6	74
3	0	0	7	140	70	0
4	0	4	0	170	82	5
8	0	0	0	20	0	0
9	0	0	0	4	11	0
10	100	105	525	0	1	0

From the table above, we can understand that this dataset contains imbalanced classes. However, we can also notice that the class imbalance is not fixed over time. Another thing to note is that this dataset contains concept drift as well which can be seen through the changes of gas representation during the elapse of the months.

4.2.1.3 Covertype

This dataset contains 581012 instances divided into 54 attributes which describe certain features such as elevation, aspect, slope, hill shade, soil type, and more. These attributes describe an instance that pertains to one of 7 different forest covertypes:

1. Spruce/Fir
2. Lodgepole Pine
3. Ponderosa Pine
4. Cottonwood/Willow
5. Aspen
6. Douglas-fir
7. Krummholz

The representation of each of these classes can be seen in the table below:

Table 4.3: Class distribution for the Forest Covertype Dataset

Spruce/Fir	Lodgepole Pine	Ponderosa Pine	Cottonwood/Willow
211840	283301	35754	2747

Aspen	Douglas-fir	Krummholz
9493	17367	20510

From the table, it is possible to understand that in this particular dataset, there is a very large imbalance ratio among the classes. And also contains CDs related to some of the types being described in different ways over time.

4.2.2 Generated Datasets

To further test our framework we made use of generated datasets as they can provide us with specific different data properties.

For these datasets, we decided to follow the data generation process used in the AUC-based metrics work [43]. This process generates datasets of 5000 entries comprised of 2 numeric attributes and 1 class. Each of these class entries is generated following a multivariate Gaussian distribution and can be classified as C different classes. However, to keep the data imbalanced half of the C are chosen as minority class and fewer instances are generated following an Imbalance Ratio (IR) I. This Imbalance Ratio is defined by how many instances of a specific majority class there are for each one instance of a specific minority class. This means an IR of 1:9 can be seen when comparing any two classes where one is majority and one is minority and the majority class contains 9 times more instances than the minority one. Furthermore, a severity variable S is used as well to determine how impactful a Concept Drift (CD) is on the generated data. This severity refers to how much a class changes after a drift in which a small severity value would cause small changes to a single class while a 100% severity would make all classes completely change their data properties. This can also make a class have the same data properties another class had prior to the CD. The generated data also contains 1 single CD at the halfway point of the data, which for our work is at 2500.

The data generated was comprised of 12 classes but followed different IR I and different Severity values S.

For datasets following an IR of 3:7, we used three different severity values: 0.3, 0.5, and 1.

For datasets following an IR of 1:9, we used three different severity values: 0.1, 0.5, and 1.

4.3 Settings

To test the data thoroughly we made use of three different window sizes as this could provide us with more information about how different window sizes\delays can impact the models. These three different window sizes: 50, 100, and 200.

Since the datasets have different levels of imbalance and because we want to test the models on how well they perform across all classes in the data, we decided to use Equally Weighted Area Under Curve (EWAUC) as the main performance test metric. Not only that, we also used it as the metric the CD detector uses to check for potential CDs using the drops in its values. This means a CD is detected if the performance of the system drops significantly.

For the tests, since the models have some randomness in their core, we decided to go with a number of runs which should be large enough to provide accurate results. Furthermore, as the randomness of the models follows Gaussian Distribution, we decided to use 50 as this number should be enough.

For the tests, we used the methods defined in 3.1 as a baseline and compared them with the proposed methods defined in 3.2 using all datasets.

4.4 Results

The results pertaining to all the tests following the described settings were divided by window size. This was done to make it easier to compare the individual performance of each model in order to understand the potential impact that delay can cause in performance. The results are presented as the average EWAUC metric over the 50 runs along their respective standard deviation in brackets. Furthermore, to make the results of each window size more comparable, the EWAUC represents the performance over the last 200 data instances of each dataset. All result tables follow the abbreviations for the dataset names:

- 3:7 Sev 0.5 - artificial dataset with IR 3:7 with CD severity of 0.5
- 3:7 Sev 1 - artificial dataset with IR 3:7 with CD severity of 1
- 3:7 Sev 0.3 - artificial dataset with IR 3:7 with CD severity of 0.3
- 1:9 Sev 0.5 - artificial dataset with IR 1:9 with CD severity of 0.5
- 1:9 Sev 1 - artificial dataset with IR 1:9 with CD severity of 1
- 1:9 Sev 0.1 - artificial dataset with IR 1:9 with CD severity of 0.1
- Gas - Gas Sensor Array Drift Dataset at Different Concentrations Dataset
- Con4 - Connect 4 Dataset
- CovType - Forest Covertype Dataset

4.4.1 Average Results

The first window size, 50, achieved the results shown in Table 4.4.

The results using a window size of 50 show that the models proposed can achieve similar results in many of the datasets. Not only this, but in many datasets the proposed models with delay can many times produce slightly better results than their counterparts with no delay. This can be seen when comparing MOOB and delayed MOOB on all of the artificial datasets with CD severity lower than 1. However, it falls behind when faced with the real-world datasets in general

Table 4.4: Results obtained on the last data batch using a window size of 50

Dataset	MOOB	MUOB	delayed MOOB	delayed MUOB	Hybrid
3:7 Sev 0.5	58.04(3.71)	60.45(2.92)	59.99(2.27)	61.21(0.69)	57.85(1.97)
3:7 Sev 1	57.47(3.62)	58.19(2.58)	56.42(1.71)	58.12(1.96)	63.03(1.06)
3:7 Sev 0.3	53.81(3.29)	62.27(2.65)	65.54(2.88)	63.05(0.28)	64.86(1.28)
1:9 Sev 0.5	62.46(2.84)	65.82(1.6)	66.56(3.31)	66.31(1.62)	66.72(2.09)
1:9 Sev 1	67.0(3.01)	69.46(1.55)	63.89(2.01)	67.65(1.44)	68.57(1.47)
1:9 Sev 0.1	61.35(3.27)	67.65(1.68)	63.66(2.93)	69.81(1.87)	67.04(1.02)
Gas	99.97(0.12)	99.37(0.73)	97.68(0.0)	97.67(0.04)	100.0(0.0)
Con4	100.0(0.0)	100.0(0.0)	100.0(0.0)	100.0(0.0)	100.0(0.0)
CovType	99.69(0.12)	97.72(0.52)	97.98(0.87)	98.39(0.16)	100.0(0.0)

and when faced with high severity levels. The delayed MUOB also produced similar results when compared with its non-delayed counterpart, producing similar or better results whenever the severity is lower than 1. However, unlike the delayed MOOB model, delayed MUOB was able to perform better than the standard MUOB in the Forest Cover Type dataset. When comparing the general results produced with this window size, we can see that the proposed delayed methods can produce competitive results compared to the non-delayed models. However, we can also conclude that for a window size of 50, performance issues caused by the delay can be attributed to severe CD occurrence. The hybrid approach we proposed performed competitively when compared with the other two delayed models, producing similar results in most datasets. Furthermore, the hybrid approach proved to be better at high-severity datasets producing competitive results, surpassing in some cases the non-delayed models. This can be seen on the artificial datasets with Severities of 1 where the proposed approach outperformed all models in 5% when IR was 3:7 and having only a 1% difference with MUOB when IR was 1:9. However, in this last dataset, it still outperformed delayed MUOB by close to 1%, which makes it the best delayed method for this dataset.

The next window size used was 100 and produced the results in Table 4.5.

For the results of this window size, we can see once more comparable results across all models. Alongside this, delayed MOOB, similar to the previous experiment, when compared to MOOB, outperformed in all artificial datasets with a Severity lower than 1. Delayed MUOB still produced competitive results. However, it did not follow the same pattern as seen with a window size of 50, producing better results than its counterpart on the artificial dataset with IR 3:7 and severity of 1. Not only this, Delayed MUOB still proved worse when dealing with Severities of 0.5 and higher when dealing with an IR of 1:9. Just like the last window size, we can also see that in most of the datasets, a delayed model performed better or as well as non-delayed ones. However, with the window size increase, we can also see that the delayed models failed to outperform in 3 of the artificial datasets compared to the 2 from window size 50. In the results for window size 100, we can observe that the hybrid approach can produce competitive results.

Table 4.5: Results obtained on the last data batch using a window size of 100

Dataset	MOOB	MUOB	delayed MOOB	delayed MUOB	Hybrid
3:7 Sev 0.5	50.69(3.94)	52.51(3.15)	50.83(4.05)	54.48(1.47)	51.56(1.81)
3:7 Sev 1	52.55(3.6)	50.2(2.45)	51.52(3.5)	51.65(2.16)	54.74(1.78)
3:7 Sev 0.3	48.58(3.55)	56.18(2.66)	52.87(1.14)	56.9(1.15)	57.32(1.35)
1:9 Sev 0.5	57.37(2.89)	62.78(1.54)	61.31(3.72)	61.83(2.42)	63.03(1.68)
1:9 Sev 1	62.61(2.4)	67.54(1.15)	62.52(0.31)	62.67(1.15)	61.7(1.83)
1:9 Sev 0.1	54.42(3.84)	61.67(1.6)	56.9(4.06)	62.83(2.7)	62.17(1.35)
Gas	100.0(0.0)	100.0(0.0)	100.0(0.0)	100.0(0.0)	100.0(0.0)
Con4	100.0(0.0)	100.0(0.0)	100.0(0.0)	100.0(0.0)	100.0(0.0)
CovType	100.0(0.0)	100.0(0.0)	100.0(0.0)	100.0(0.0)	100.0(0.0)

These results further show that the hybrid approach outperforms non-delayed models in 4 of the 6 artificial datasets. However, its performance dropped when dealing with IR 1:9 and Severity of 1 to the point of being the worst of the tested models (even if only slightly when compared to delayed ones). Also, it is important to note that on the dataset with IR 1:9 and Severity of 0.1, the hybrid approach outperformed the non-delayed models however it was outperformed by the delayed MUOB as well. This leads us to conclude that with this window size, this approach is more robust than on the last window size. Nonetheless, we can also see that the hybrid approach will not consistently outperform the models it uses.

Lastly, we tested the models using a window size of 200 which obtained the results in Table 4.6.

Table 4.6: Results obtained on the last data batch using a window size of 200

Dataset	MOOB	MUOB	delayed MOOB	delayed MUOB	Hybrid
3:7 Sev 0.5	50.4(3.67)	49.66(2.62)	47.92(5.41)	53.07(2.15)	53.3(1.69)
3:7 Sev 1	50.28(3.89)	47.88(2.33)	51.13(3.78)	49.63(2.58)	53.89(1.14)
3:7 Sev 0.3	47.15(3.06)	52.31(2.32)	50.09(1.32)	52.01(1.57)	55.41(1.56)
1:9 Sev 0.5	55.01(2.84)	61.47(1.4)	58.93(4.17)	59.89(2.55)	55.61(2.28)
1:9 Sev 1	55.29(3.84)	53.99(1.27)	51.72(2.31)	53.18(1.73)	55.88(3.37)
1:9 Sev 0.1	54.02(4.05)	58.69(1.45)	55.73(5.26)	61.64(2.02)	56.68(1.61)
Gas	100.0(0.0)	100.0(0.0)	100.0(0.0)	100.0(0.0)	90.13(0.0)
Con4	100.0(0.0)	100.0(0.0)	100.0(0.0)	100.0(0.0)	100.0(0.0)
CovType	100.0(0.0)	100.0(0.0)	100.0(0.0)	100.0(0.0)	95.24(1.15)

The results produced with this window size further show that the delayed models produce competitive results. However, we still do see a slight decrease in performance in some datasets. The delayed MOOB was able to outperform its non-delayed counter-part in most of the datasets, lacking behind in performance on the artificial dataset with IR of 3:7 and Severity of 0.5 and when dealing with IR of 1:9 and Severy of 1. Delayed MUOB on the other hand produced worse

results compared to the original MUOB on half of the artificial datasets. This can be seen when dealing with IR 3:7 with a Severity of 0.5 and IR 1:9 with a Severity of 0.5 or higher. The hybrid approach outperformed or produced very close results when compared with the delayed models in four artificial datasets while being able to produce better results than the non-delayed counterparts in these datasets as well. However, it ended up being the only model that achieved worse results in real-world datasets. This shows us that the hybrid approach can obtain results close to the other two standalone models and even outperform them in some datasets. However, it is not the best in all scenarios.

4.4.2 Results over window sizes

Following this, we plotted the performances of each model as the window size increased to allow for easier comparison of performance along different window sizes. To analyze the results we decided to look at the results taking into account all datasets at once. However, as the real-world datasets mostly produced very high results that skewed the average results, we decided to also look at the performances taking only into account the synthetic datasets.

The first model we plotted was MOOB which can be seen in 4.1.

Figure 4.1: MOOB results for each dataset as window size increased.

MOOB started at window size 50 with a EWAUC score of 73.31% on average across all datasets, 60.02% if we do not account for the real-world datasets. As the window size increased

we can see its average score dropped to 68.02% with real-world datasets, and 52.02% without them. Producing an average performance drop of 5.29% without real-world datasets and 8% with them. Not only this, but we can also see that MOOB ended up dropping below 50% performance when dealing with IR 3:7 and Severity of 0.3.

Following MOOB, we plotted the performance evolution of MUOB which is shown in 4.2

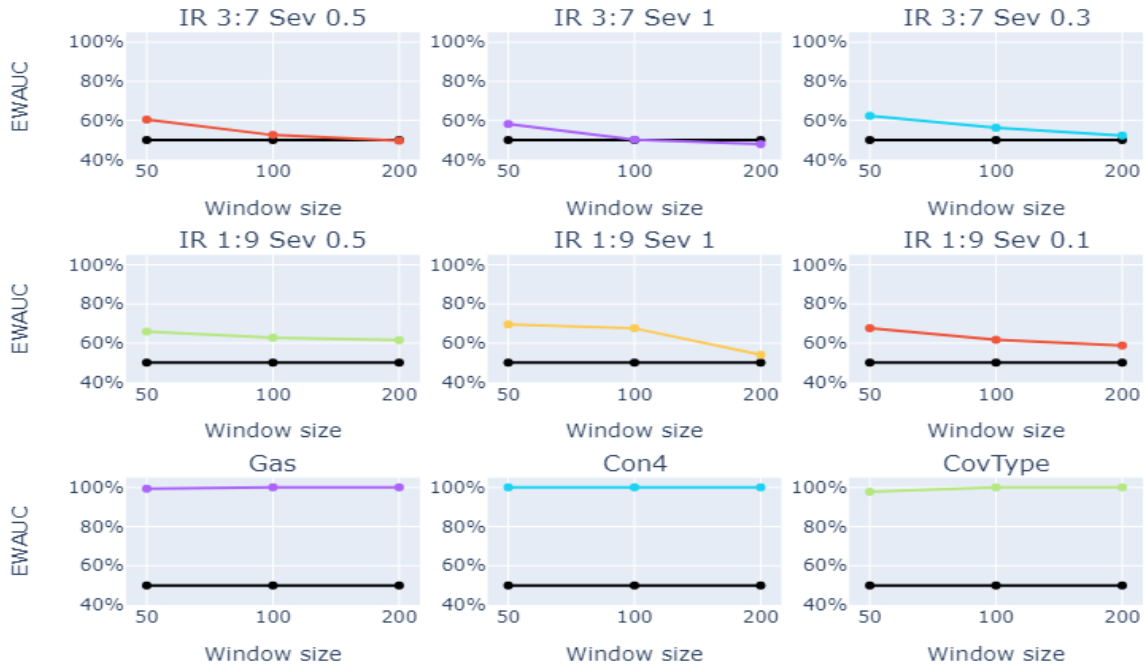


Figure 4.2: MUOB results for each dataset as window size increased.

MUOB at window size 50 ended up producing a better average EWAUC score when compared to MOOB. MUOB produced a EWAUC score of 75.66% and 63.97% when not taking into account real world datasets. This however also ended up dropping as the window size increased, going down to 69.33% with real-world datasets at window size 200 and 54% without them. Producing an average performance drop of 6.33% with real-world datasets and of 9.97% without taking them into account.

Then we plotted both of the delayed counterparts in the same order as seen in 4.3 and 4.4.

Delayed MOOB at window size 50 produced better results than its non-delayed counterpart, having a EWAUC score of 74.64% when taking into account real-world datasets and 62.68% when not taking them into account. We can also see that as the window size increased, its performance dropped more than its counterpart producing an average drop of 10.09% without taking into account real-world datasets and 6.25% with them. Alongside this, delayed MOOB at window size 200 still outperformed non-delayed MOOB on average but only by around 0.3%. However, as we see its rapid decrease with window size we can assume that at higher window sizes this can further drop and potentially fall behind the non-delayed counterpart.

Delayed MUOB, unlike Delayed MOOB, was able to outperform on average its counterparts at both starting and ending window sizes. It produced an average EWAUC of 75.8% and 64.36%

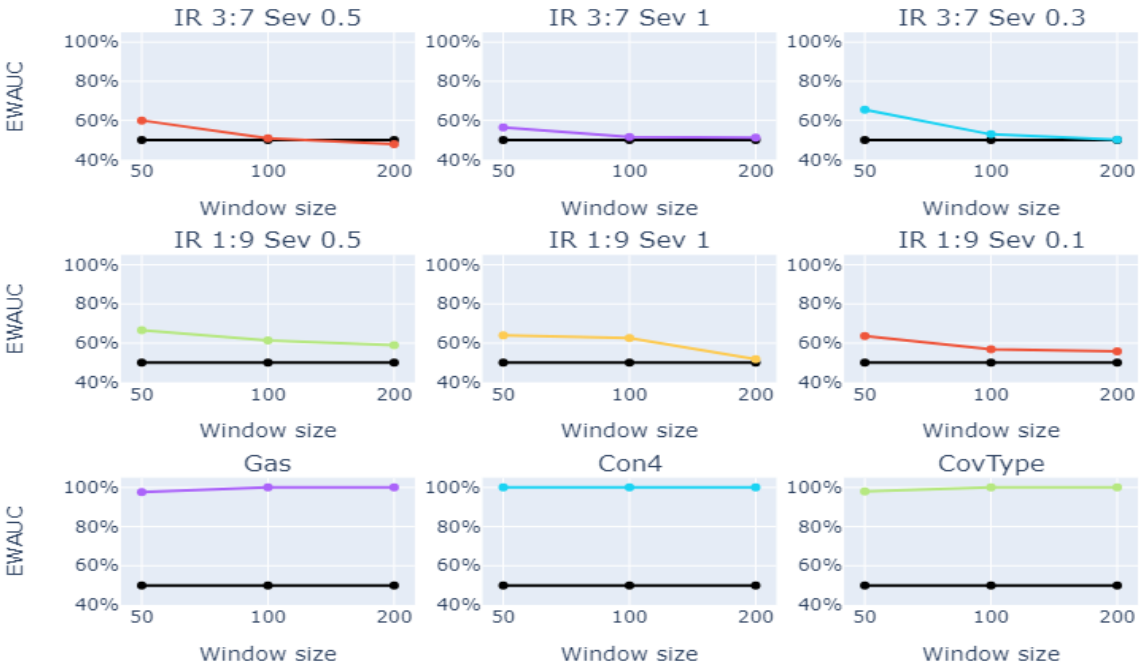


Figure 4.3: Delayed MOOB results for each dataset as window size increased.

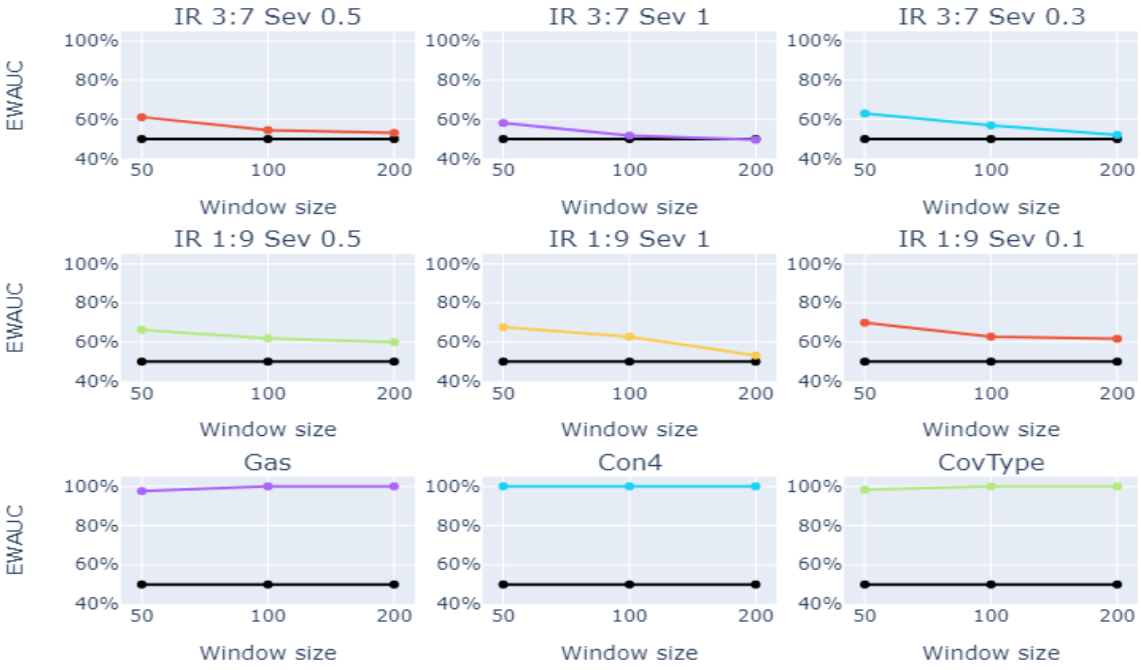


Figure 4.4: Delayed MUOB results for each dataset as window size increased.

when not taking into account real-world datasets at window size 50, which is 0.2-0.4% better than its counterpart. Its average drop was also lower than its counterpart having an average drop of 5.86%-9.46% depending on whether we take into account the real-world datasets or not. Furthermore, it also produced a performance drop of around 0.4% lower on average when compared with its non-delayed MUOB. This drop alongside higher starting scores can show us that MUOB should still have competitive results in delayed scenarios. It is also important to note that this model just like its counterpart produced results lower than 50% when dealing with IR 3:7 and Severity of 1. However, it did still produce a better result which was closer to the 50% mark.

The last performance evolution we plotted was the Hybrid approach which can be seen in 4.5.

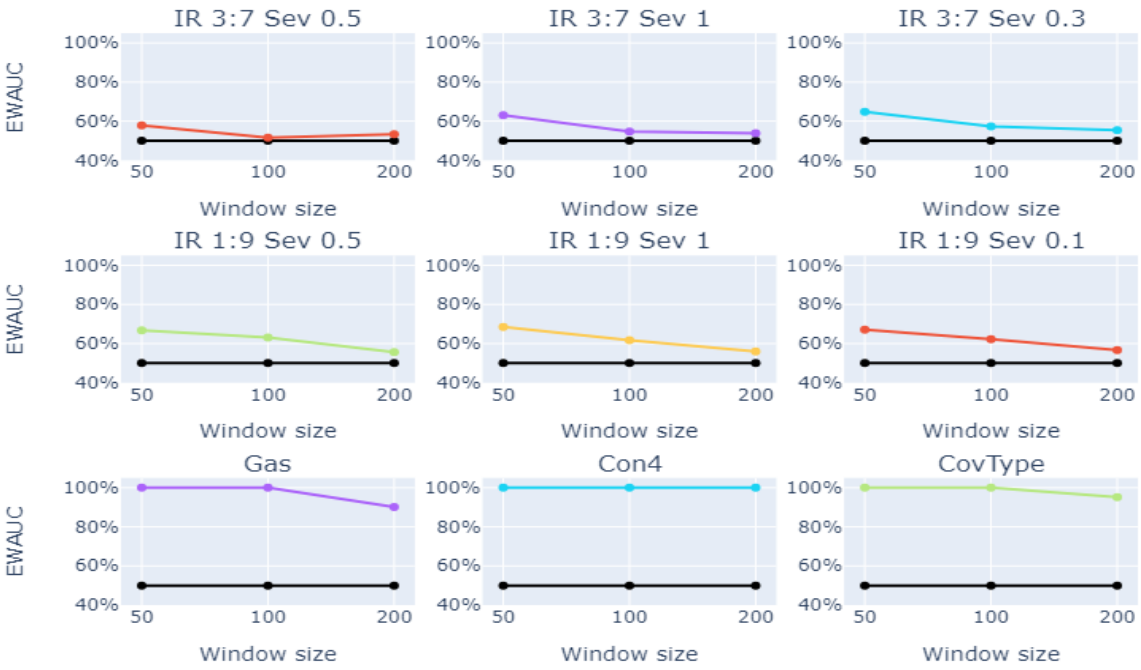


Figure 4.5: Hybrid approach results for each dataset as window size increased.

The hybrid approach in general proved to be better at window size 50, outperforming all other models. It produced an average EWAUC of 76.45% and 64.48%, the latter being without taking into account real-world datasets. This is an increase of at least 0.65% when compared with the other and an increase of at least 0.3% when not taking into account real-world datasets. However, as the window size increased we saw a total average drop in performance of 7.99% when taking into account real-world data compared to all other models. Nonetheless, on the artificial datasets, the hybrid method produced a drop competitive with the Delayed MUOB model. Not only this, it also produced a smaller performance drop than both non-delayed models. We can conclude that the Hybrid approach is well fit for situations with small delays, but is not

as consistent with higher delays as seen with the real-world data.

After all these tests we can conclude that a general approach is only suitable for smaller window sizes. However, this data is still not enough to answer the research question about the impact of delay. To further try to answer this we decided to plot the performance evolution during the runs. This allows us to understand where CDs were potentially detected and the score at these spots. This should allow us to see the difference when both delayed and non-delayed drift detection occurs. The plots we used focus only on the largest window size since this should provide us with a better overview of how severe the delay impact can be. Furthermore, we grouped the delayed models with their non-delayed counterparts for an easier comparison.

4.4.3 Grouped Results

The first plot we produced provides us an overview of MOOB and Delayed MOOB and can be seen in 4.6.

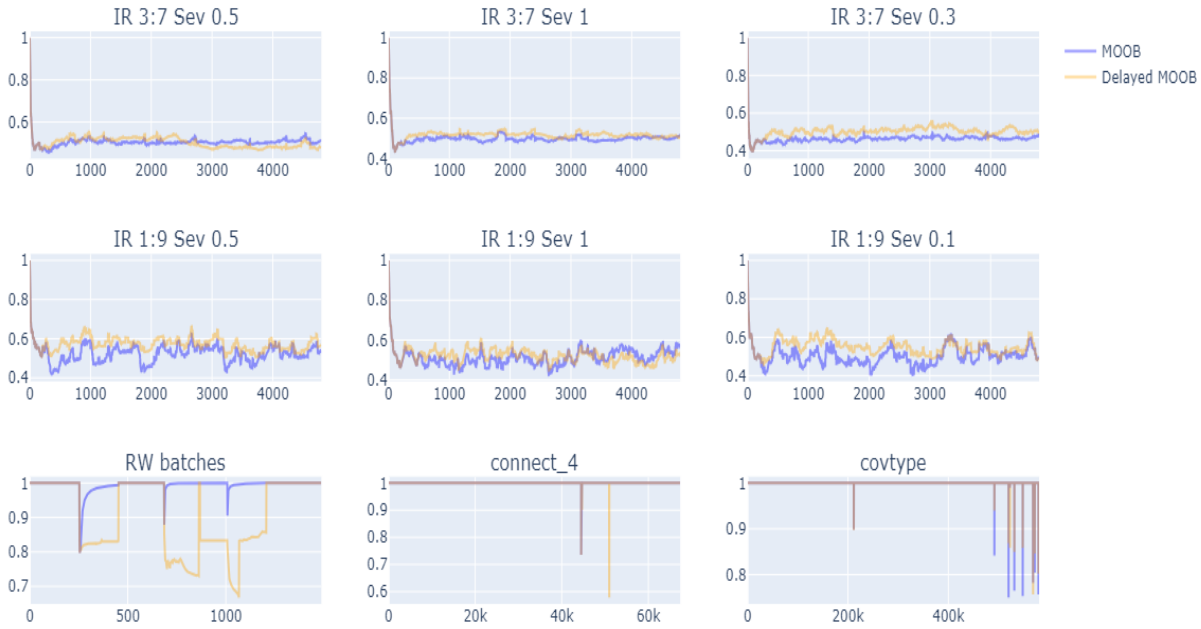


Figure 4.6: Average EWAUC score at each data instance for MOOB and Delayed MOOB with a window size of 200.

From these plots, we can see that in general, both models did not detect many noticeable CDs on the artificial datasets. However, the real-world datasets show several drops in performance which are followed by increases. These are the data instances where the performance has dropped enough to trigger a CD detection, resetting the model. Furthermore, we can see on the Gas

dataset that MOOB is able to re-adapt to the new concept quickly while delayed MOOB takes longer to respond to the drift, consequently struggling to re-adapt to the new concept. Resulting in a drop in performance and causing another CD detection. On the other two real-world datasets, we can see that there were several moments of drift detection. However, as these have a large number of data instances it is not discernible in 4.6 how each model reacted to the drift. To be able to further analyze it we decided to produce new plots which provide us with the evolution over those cases. These plots can be seen in 4.7.

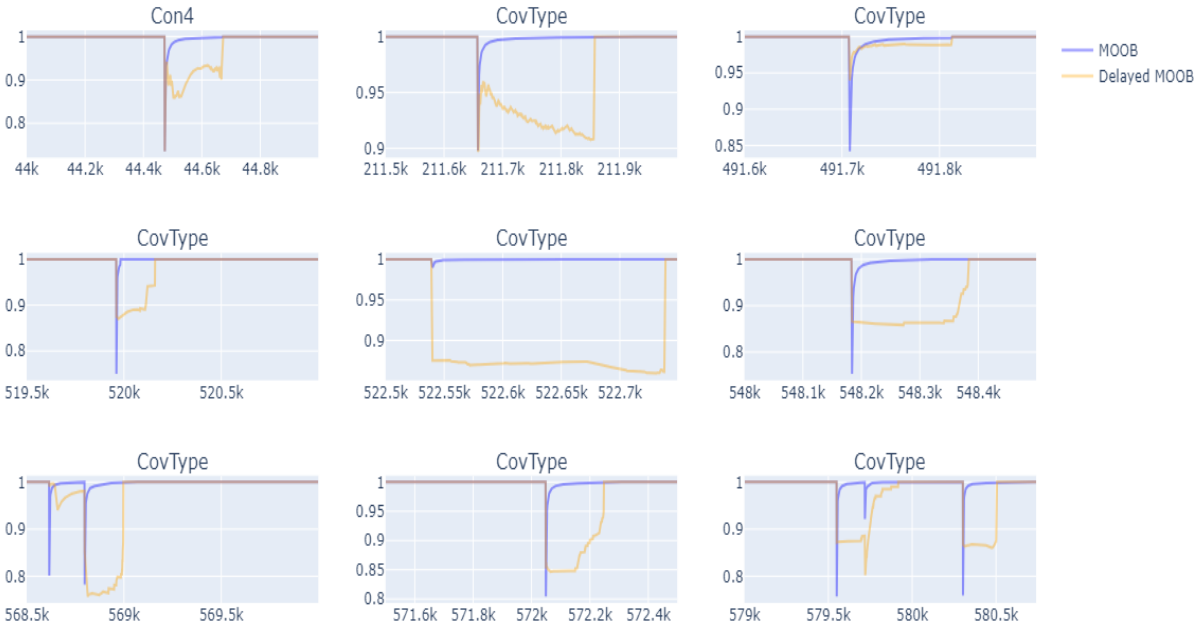


Figure 4.7: Average EWAUC score at each data instance for MOOB and Delayed MOOB with a window size of 200 at CD spots for real-world datasets.

From these new plots, we can see that the models follow the same pattern of having a longer delay in CD detection causing them to have lower performance until the delay has passed. Alongside this, we can also see that in some moments, the delayed MOOB is still dealing with a CD when another one occurs, causing the model to perform worse.

Following the MOOB comparison, we plotted the performance evolution of both MUOB and delayed MUOB. The evolution can be seen in 4.8. In addition, we also decided to plot the CD locations for the respective models on the Connect 4 and Forest Cover Type datasets to further analyze the evolution at those moments. The plots for these moments are in Figure 4.9.

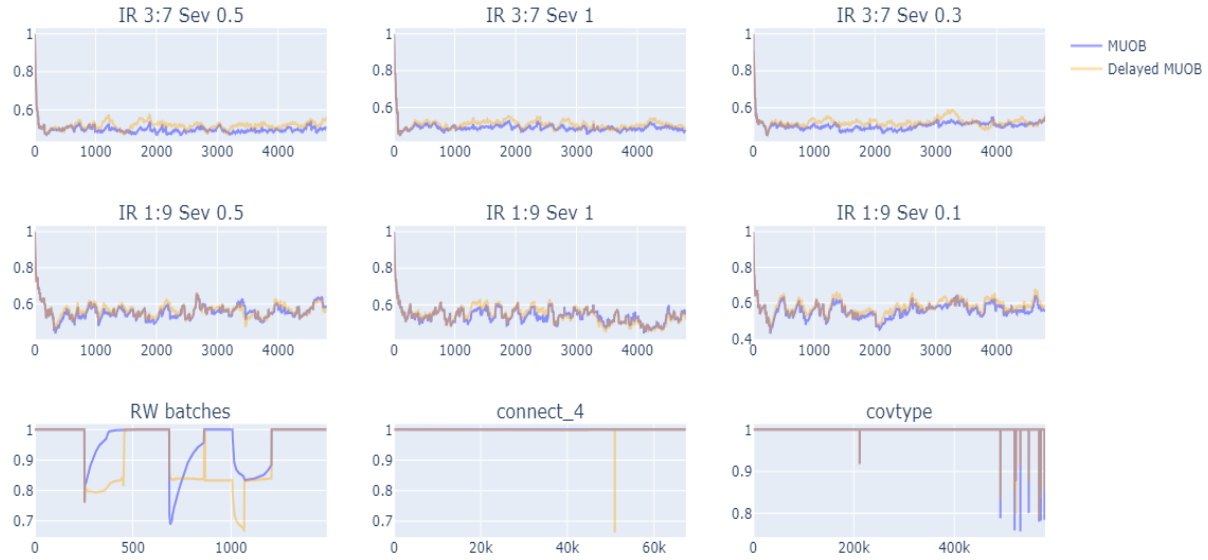


Figure 4.8: Average EWAUC score at each data instance for MUOB and Delayed MUOB with a window size of 200.

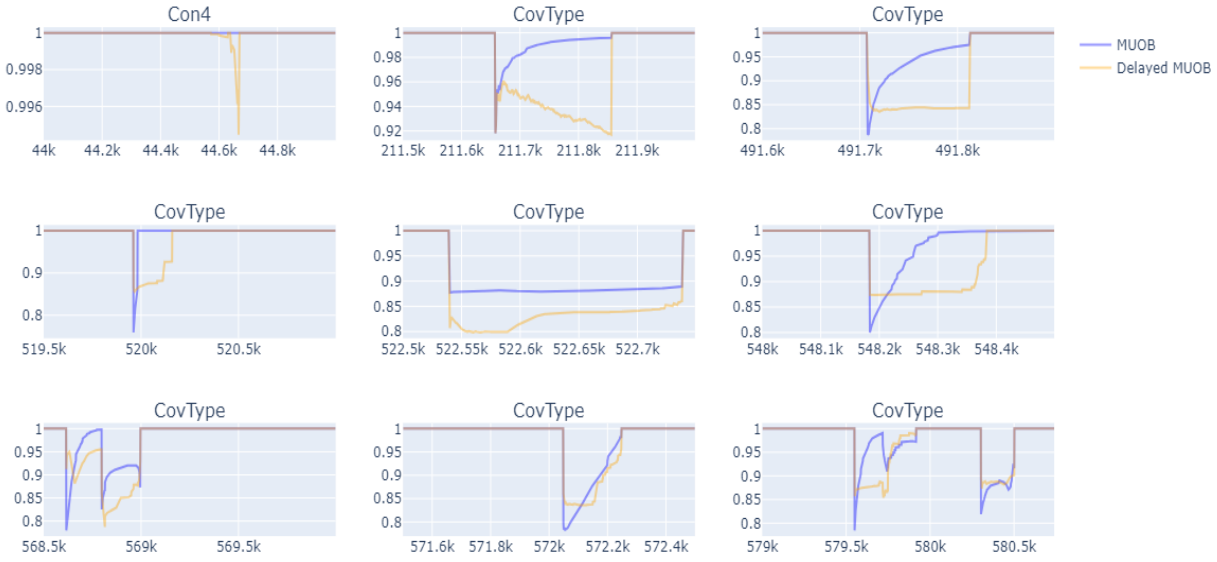


Figure 4.9: Average EWAUC score at each data instance for MUOB and Delayed MUOB with a window size of 200 at CD spots for real-world datasets.

When comparing MUOB and Delayed MUOB we can see the same problem to adjust to the

CD. Furthermore, MUOB showed, just like MOOB, a clear adaptation as soon as a EWAUC score drop occurs. Delayed MUOB also proved to have a harder time adapting to new concepts. This can be seen as in some moments several consecutive performance drops occurred before its performance stabilized.

Through this comparison between the delayed models and their non-delayed counterparts, we can see that the delay in data validation impacts the performance since it affects the moment of drift detection on a system using a performance metric-based CD detector. This impact mostly results in the system producing a lower performance output until the system detects the CD and adapts to the new concept. It is important to note however that the general performance of a system is still able to be on par with its non-delayed counterpart as long as the system is not faced with several consecutive CDs. This in turn leads us to be able to answer the first two research questions as follows:

1. How relevant is the impact of the delay in data validation in Multiclass Imbalanced data in State-of-the-Art learning systems?

Data validation delay provides a performance drop in Multiclass Imbalanced data in State-of-the-Art machine learning systems which can be severe.

2. What causes the most relevant impact on performance for Multiclass Imbalanced State-of-the-Art machine learning systems when faced with data validation delay?

The main issue that proved to cause the biggest drops in performance was the slow detection of CDs due to the delay.

Having these answered, we plot the performance evolution of the delayed models and the hybrid approach to compare them. Furthermore, with this comparison, we want to try to answer the third research question and understand how good this approach is compared to the other delayed models. For this comparison, we also plotted the CD locations for the real-world datasets. The plots can be seen in 4.10 and 4.11 respectively.

As we can see from the plots, the hybrid approach ends up producing results between both the delayed models and their counterparts. This can be seen by the gradual performance evolution the hybrid approach goes through after missing detection of CD (because it is delayed). When eventually detected, the models once again stabilize. However, as our approach still takes into account the EWAUC of the moments before the CD to choose the best model, in some moments it ends up being slower at adapting. This, in turn, shows us that the hybrid approach is not very good at generalizing around CDs as its performance is not consistent on all of the tested ones. In turn, this allows us to partially answer the third research question:

3. Are generalized approaches suitable to deal with Multiclass Imbalanced data when there is a data validation delay?

Figure 4.10: Average EWAUC score at each data instance for Delayed MOOB, Delayed MUOB, and Hybrid with a window size of 200.

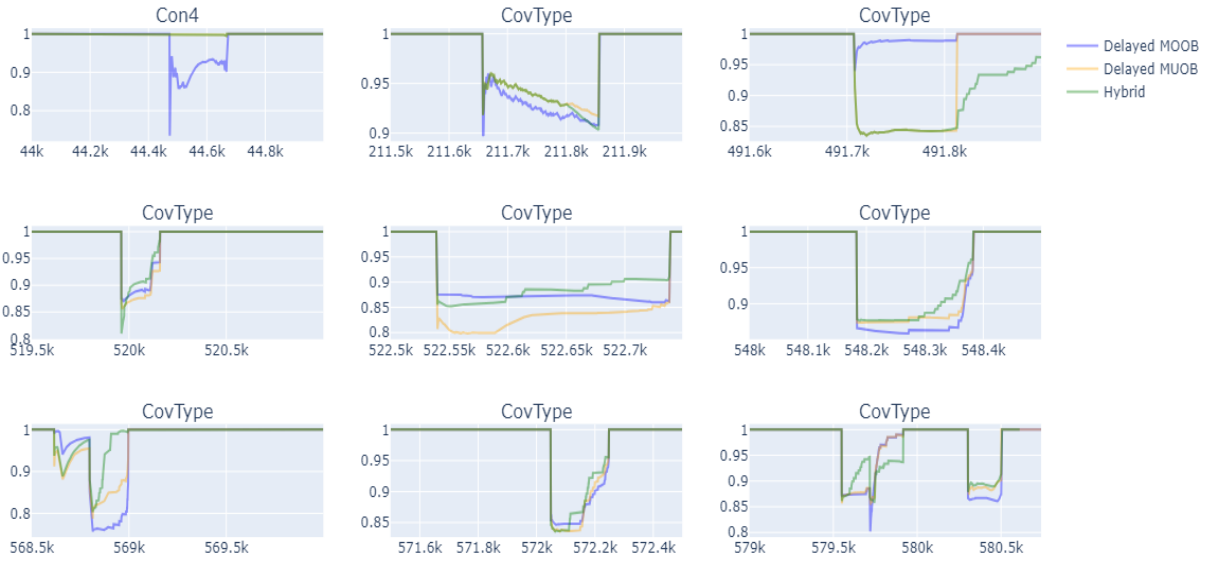


Figure 4.11: Average EWAUC score at each data instance for Delayed MOOB, Delayed MUOB, and Hybrid with a window size of 200 at CD spots for real-world datasets.

Generalized approaches can provide better performance when adapting toward newly detected concepts. However, depending on how it is implemented it may not be suitable for all CDs.

Chapter 5

Conclusions

This chapter covers the conclusions obtained from our work. These conclusions, first, provide a summary of the work done. Then, they focus on answering our research questions as much as possible. Furthermore, this chapter will also cover the limitations of the work done and what can be explored in future work to deal with these shortcomings.

5.1 Work done

In this work, we explored the State of the Art regarding Multiclass Classification where data is imbalanced and contains Concept Drift (CD). To do this, we followed a Systematic Literature Review Methodology which allowed us to extract the most relevant works. From these works, we were able to notice two important things:

- Works regarding CD and/or imbalanced data mostly explored Binary Classification scenarios
- Works mostly infer that the data is already validated and do not take into account the potential necessity of extra validation time for some systems

Due to these, we established research questions and worked with them in mind. For this, we developed a framework that allowed us to test Multiclass Classification models in imbalanced data with CD. Furthermore, we also made it so the framework was capable of testing these models with a simulated data validation delay time. Then, we proposed a method that made use of the models from the previous framework with delay. This method constantly evaluates both methods with the known information and makes use of the one it sees as best. Using these, we were able to extract conclusions and answer partially the research questions established. This, in turn, should result in the providence of extra knowledge related to problems of this nature for future works. Alongside this, it should also provide a new proposed method that proved to have potential.

Related to this work, a poster was made which touched on the approach we had at an earlier

stage. This work can also be accessed through the link:

https://github.com/nrguimaraes/MultiClassCD

5.2 Research Questions Answers

In section 1 we defined the 3 research questions, these were:

1. How relevant is the impact of the delay in data validation in Multiclass Imbalanced data in State-of-the-Art machine learning systems?
2. What causes the most relevant impact on performance for Multiclass Imbalanced State-of-the-Art machine learning systems when faced with data validation delay?
3. Are generalized approaches suitable to deal with Multiclass Imbalanced data when there is a data validation delay?

Regarding the first Research Question and the results we obtained, we can conclude that the impact of the delay is severe enough to lower the performance of our systems. However, this does not mean a system with delay will perform worse as the performance drops seen were mainly momentary. Furthermore, as the system we tested used only a performance drop-based CD detector, it is unclear if this occurs in other types of CD detectors.

Concerning the second Research Question, multiple window sizes were tested and they produced generally similar results when comparing delayed and non-delayed models. Therefore, we can conclude that window size was not the cause behind the largest performance drops due to delay. However, as we plotted the performance over every single run, we were able to notice drops caused by CD. In these spots, we were able to understand that the delayed methods had lower performance until the window size delay was over since the system was able to detect the CD after that period. This helps us conclude that for systems with performance-based CD detectors, the main cause of performance drop is the delay itself.

The generalized approach we explored proved to be able to slowly adapt while the baseline approach fell behind in performance in moments of CD. However, as the system checks CD for both models individually this can cause the system to have a slower re-adaptation once the window size delay is concluded. Nonetheless, the system did prove to be more stable in most cases. With all of this, we can conclude that these generalized approaches are suitable for Imbalanced Multiclass Classification (IMC) problems when compared with the delayed models. The generalized approach had some limitations, some of which proved to cause some performance drops. The limitation that we were able to determine as a cause for some drops was the fact the system did not reset both models in case a CD occurred. Instead, it would reset only the model for which CD was detected. This, in turn, made the fact that this approach checks for the best performance over the last known window a problem. This is mainly because if a CD is detected,

the reset model may have the best performance (which should be inferred as it was trained with the latest known window of data). However, because the approach checks the performance it may still choose the other model for a while, decreasing the potential performance boost from the reset model. Another limitation to note is the fact the approach only made use of two models, meaning that more can still be added to try to further improve its performance. Lastly, the methods explored only tried to use performance drop-based CD detectors, so we can not be sure of how the systems will fare using others.

5.3 Future Work

As most of our research questions were only partially answered, future work would revolve around that. To answer these more thoroughly, the system we implemented would need to be adapted to be able to integrate different CD detectors. This helps us understand the problems caused by the delay in data verification. Furthermore, the generalized approach we implemented should be improved to be able to adapt to CDs faster, to at least keep the same performance as the delayed models after a CD. Finally, more models should be explored with our system and different delays should be further explored using both hybrid and non-hybrid approaches to further validate the conclusions of the current work.

Bibliography

[1] Aida Ali, Anca Ralescu, Siti Mariyam Shamsuddin, and Anca L Ralescu. Classification with class imbalance problem: A review. *Classification Int. J. Advance Soft Compu. Appl*, 5, 2013. ISSN: 2074-2827.

[2] Régis Antônio, Saraiva Albuquerque, Albert França, Josuá Costa, Eulanda Miranda, Dos Santos, Robert Sabourinécole, and Rafael Giusti. A decision-based dynamic ensemble selection method for concept drift. doi:10.1109/ICTAI.2019.00158.

[3] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems, 2002.

[4] Firas Bayram, Bestoun S. Ahmed, and Andreas Kassler. From concept drift to model degradation: An overview on performance-aware drift detectors. *Knowledge-Based Systems*, 245:108632, 6 2022. ISSN: 0950-7051. doi:10.1016/J.KNOSYS.2022.108632.

[5] Eric Breck, Neoklis Polyzotis, Sudip Roy, Steven Euijong Whang, and Martin Zinkevich. Data validation for machine learning, 2019.

[6] Dariusz Brzezinski and Jerzy Stefanowski. Prequential auc for classifier evaluation and drift detection in evolving data streams.

[7] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, jun 2002. doi:10.1613/jair.953.

[8] Nitesh V. Chawla, Nathalie Japkowicz, and Aleksander Kotcz. Editorial. *ACM SIGKDD Explorations Newsletter*, 6:1–6, 6 2004. ISSN: 1931-0145. doi:10.1145/1007730.1007733.

[9] Andrew A Cook, Göksel Mısırlı, and Zhong Fan. Anomaly detection for iot time-series data: A survey. *IEEE Internet of Things Journal*, 7(7):6481–6494, 2019.

[10] Mohamed Medhat Gaber. Advances in data stream mining, 1 2012. ISSN: 19424795. doi:10.1002/widm.52.

[11] Mohamed Medhat Gaber, Arkady Zaslavsky, and Shonali Krishnaswamy. Mining data streams: A review, 2005.

[12] Joao Gama. A survey on learning from data streams: current and future trends. *Progress in Artificial Intelligence*, 1:45–55, 2012.

[13] João Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. Learning with drift detection. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 3171:286–295, 2004. ISSN: 16113349. doi:10.1007/978-3-540-28645-5$_2$9.

[14] Margherita Grandini, Enrico Bagli, and Giorgio Visani. Metrics for multi-class classification: an overview. 8 2020. doi:10.48550/arxiv.2008.05756.

[15] Meng Han, Xilong Zhang, Zhiqiang Chen, Hongxin Wu, and Muhang Li. Dynamic ensemble selection classification algorithm based on window over imbalanced drift data stream. *Knowledge and Information Systems*, 65:1105–1128, 3 2023. ISSN: 02193116. doi:10.1007/s10115-022-01791-5.

[16] David J Hand. A simple generalisation of the area under the roc curve for multiple class classification problems, 2001.

[17] Haibo He, Yang Bai, Edwardo A. Garcia, and Shutao Li. Adasyn: Adaptive synthetic sampling approach for imbalanced learning. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pages 1322–1328, 2008. doi:10.1109/IJCNN.2008.4633969.

[18] Steven CH Hoi, Doyen Sahoo, Jing Lu, and Peilin Zhao. Online learning: A comprehensive survey. *Neurocomputing*, 459:249–289, 2021.

[19] Adriana Sayuri Iwashita and João Paulo Papa. An overview on concept drift learning. *IEEE Access*, 7:1532–1547, 2019. ISSN: 21693536. doi:10.1109/ACCESS.2018.2886026.

[20] Syed Muslim Jameel, Manzoor Ahmed Hashmani, Hitham Alhussain, Mobashar Rehman, Universiti Tunku Abdul Rahman Perak, and Malaysia Arif Budiman. A critical review on adverse effects of concept drift over machine learning classification models. *IJACSA) International Journal of Advanced Computer Science and Applications*, 11, 2020.

[21] Botao Jiao, Yinan Guo, Dunwei Gong, and Qiuju Chen. Dynamic ensemble selection for imbalanced data streams with concept drift. *IEEE Transactions on Neural Networks and Learning Systems*, 2022. ISSN: 21622388. doi:10.1109/TNNLS.2022.3183120.

[22] Harsurinder Kaur, Husanbir Singh Pannu, and Avleen Kaur Malhi. A systematic review on imbalanced data challenges in machine learning: Applications and solutions. *ACM Comput. Surv.*, 52(4), aug 2019. ISSN: 0360-0300. doi:10.1145/3343440.

[23] Taiwo Kolajo, Olawande Daramola, and Ayodele Adebiyi. Big data stream analysis: a systematic literature review. *Journal of Big Data*, 6:1–30, 12 2019. ISSN: 21961115. doi:10.1186/S40537-019-0210-7/FIGURES/4.

[24] Lukasz Korycki and Bartosz Krawczyk. Concept drift detection from multi-class imbalanced data streams. *Proceedings - International Conference on Data Engineering*, 2021-April:1068–1079, 4 2021. ISSN: 10844627. doi:10.1109/ICDE51399.2021.00097.

[25] Vincent Lemaire, Christophe Salperwyck, and Alexis Bondu. A survey on supervised classification on data streams. volume 205, pages 88–125. Springer Verlag, 2015. ISBN: 9783319175508. doi:10.1007/978-3-319-17551-5$_4$.

[26] Dan Li, Dacheng Chen, Baihong Jin, Lei Shi, Jonathan Goh, and See-Kiong Ng. Mad-gan: Multivariate anomaly detection for time series data with generative adversarial networks. In *International conference on artificial neural networks*, pages 703–716. Springer, 2019.

[27] Weike Liu, Hang Zhang, Zhaoyun Ding, Qingbao Liu, and Cheng Zhu. A comprehensive active learning method for multiclass imbalanced data streams with concept drift. *Knowledge-Based Systems*, 215:106778, 3 2021. ISSN: 0950-7051. doi:10.1016/J.KNOSYS.2021.106778.

[28] Jesus L. Lobo, Javier Del Ser, Albert Bifet, and Nikola Kasabov. Spiking neural networks and online learning: An overview and perspectives. *Neural Networks*, 121:88–100, 2020. ISSN: 0893-6080. doi:https://doi.org/10.1016/j.neunet.2019.09.004.

[29] Mr Rushi Longadge, Ms Snehlata, S Dongre, and Dr Latesh Malik. Class imbalance problem in data mining: Review, 2013.

[30] Jie Lu, Anjin Liu, Fan Dong, Feng Gu, Joao Gama, and Guangquan Zhang. Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering*, 31:2346–2363, 12 2019. ISSN: 15582191. doi:10.1109/TKDE.2018.2876857.

[31] Osama A Mahdi, Eric Pardede, and Jinli Cao. Combination of information entropy and ensemble classification for detecting concept drift in data stream. 2018. doi:10.1145/3167918.3167946.

[32] Dianne SV Medeiros, Helio N Cunha Neto, Martin Andreoni Lopez, Luiz Claudio S. Magalhães, Natalia C Fernandes, Alex B Vieira, Edelberto F Silva, and Diogo M F. Mattos. A survey on data analysis on large-scale wireless networks: online stream processing, trends, and challenges. *Journal of Internet Services and Applications*, 11:1–48, 2020.

[33] Dinithi Nallaperuma, Rashmika Nawaratne, Tharindu Bandaragoda, Achini Adikari, Su Nguyen, Thimal Kempitiya, Daswin De Silva, Damminda Alahakoon, and Dakshan Pothuhera. Online incremental machine learning platform for big data-driven smart traffic management. *IEEE Transactions on Intelligent Transportation Systems*, 20(12):4679–4690, 2019. doi:10.1109/TITS.2019.2924883.

[34] Ronaldo C. Prati, Gustavo E.A.P.A. Batista, and Diego F. Silva. Class imbalance revisited: a new experimental setup to assess the performance of treatment methods. *Knowledge and Information Systems*, 45:247–270, 10 2015. ISSN: 02193116. doi:10.1007/s10115-014-0794-3.

[35] S. Priya and R. Annie Uthra. Deep learning framework for handling concept drift and class imbalanced complex decision-making on streaming data. *Complex and Intelligent Systems*, 1: 1–17, 7 2021. ISSN: 21986053. doi:10.1007/S40747-021-00456-0/FIGURES/15.

[36] Foster Provost. Well-trained pets: Improving probability estimation trees. 11 2000.

[37] Foster Provost, Provost@acm Org, and Pedro Domingos. Tree induction for probability-based ranking, 2003.

[38] Raihan Seraj and Mohiuddin Ahmed. Concept drift for big data. *Advanced Sciences and Technologies for Security Applications*, pages 29–43, 2020. ISSN: 23639466. doi:10.1007/978-3-030-35642-2$_2$/$FIGURES$/1.

[39] Bingqing Shen, Jingzhi Guo, and Yilong Yang. Medchain: Efficient healthcare data sharing via blockchain. *Applied Sciences*, 9(6), 2019. ISSN: 2076-3417. doi:10.3390/app9061207.

[40] Jafar Tanha, Yousef Abdi, Negin Samadi, Nazila Razzaghi, and Mohammad Asadpour. Boosting methods for multi-class imbalanced data classification: an experimental review. *Journal of Big Data*, 7:1–47, 2020.

[41] Amin Ullah, Khan Muhammad, Ijaz Ul Haq, and Sung Wook Baik. Action recognition using optimized deep autoencoder and cnn for surveillance data streams of non-stationary environments. *Future Generation Computer Systems*, 96:386–397, 2019. ISSN: 0167-739X. doi:https://doi.org/10.1016/j.future.2019.01.029.

[42] Jin Wang, Yaqiong Yang, Tian Wang, R Simon Sherratt, and Jingyu Zhang. Big data service architecture: a survey. *Journal of Internet Technology*, 21(2):393–405, 2020.

[43] Shuo Wang and Leandro L. Minku. Auc estimation and concept drift detection for imbalanced data streams with multiple classes. *Proceedings of the International Joint Conference on Neural Networks*, 7 2020. doi:10.1109/IJCNN48605.2020.9207377.

[44] Shuo Wang, Leandro L Minku, Davide Ghezzi, Daniele Caltabiano, Peter Tino, and Xin Yao. Concept drift detection for online class imbalance learning, .

[45] Shuo Wang, Leandro L Minku, and Xin Yao. Dealing with multiple classes in online class imbalance learning (MOOB/MUOB), .

[46] Shuo Wang, Leandro L Minku, and Xin Yao. Dealing with multiple classes in online class imbalance learning (moob/muob), .

[47] Shuo Wang, Leandro L. Minku, and Xin Yao. Resampling-based ensemble methods for online class imbalance learning. *IEEE Transactions on Knowledge and Data Engineering*, 27:1356–1368, 5 2015. ISSN: 10414347. doi:10.1109/TKDE.2014.2345380.

[48] Ashkan Zakaryazad and Ekrem Duman. A profit-driven artificial neural network (ann) with applications to fraud detection and direct marketing. *Neurocomputing*, 175, 10 2015. doi:10.1016/j.neucom.2015.10.042.

[49] Paweł Zyblewski, Paweł Ksieniewicz, and Michał Woźniak. Classifier selection for highly imbalanced data streams with minority driven ensemble. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11508 LNAI:626–635, 2019. ISSN: 16113349. doi:10.1007/978-3-030-20912-4$_5$7/$TABLES$/1.

[50] Paweł Zyblewski, Robert Sabourin, and Michał Woźniak. Data preprocessing and dynamic ensemble selection for imbalanced data stream classification. *Communications in Computer and Information Science*, 1168 CCIS:367–379, 2020. ISSN: 18650937. doi:10.1007/978-3-030-43887-6$_3$0/$FIGURES$/3.

[51] Ömer Gözüaçık, Alican Büyükçakır, Hamed Bonab, and Fazli Can. Unsupervised concept drift detection with a discriminative classifier. 2019. doi:10.1145/3357384.3358144.

[52] Indrė Žliobaitė. Learning under concept drift: an overview. 10 2010.