Pontifícia Universidade Católica
do Rio de Janeiro

João Guilherme Mattos de Oliveira Santos

A method for interpreting concept drifts in a streaming environment

Dissertação de Mestrado

Dissertation presented to the Programa de Pós-graduação em Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Informática.

Advisor : Prof. Hélio Côrtes Vieira Lopes
Co-advisor: Dr. Thuener Armando da Silva

Rio de Janeiro
April 2021

Pontifícia Universidade Católica
do Rio de Janeiro

João Guilherme Mattos de Oliveira Santos

A method for interpreting concept drifts in a streaming environment

Dissertation presented to the Programa de Pós-graduação em Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Informática. Approved by the Examination Committee.

Prof. Hélio Côrtes Vieira Lopes
Advisor
Departamento de Informática – PUC-Rio

Dr. Thuener Armando da Silva
Co-advisor
Senior Data Scientist – Greensill

Prof. Marcus Vinicius Soledade Poggi de Aragão
Departamento de Informática – PUC-Rio

Dr. Leonardo Dorigo Ribeiro
Otimização – Petróleo Brasileiro S.A.

Prof. Alex Laier Bordignon
Departamento de Geometria – UFF

Rio de Janeiro, April 23rd, 2021

João Guilherme Mattos de Oliveira Santos

Graduated in Electronics and Computer Engineering by UFRJ

Bibliographic data

Mattos de Oliveira Santos, João Guilherme

A method for interpreting concept drifts in a streaming environment / João Guilherme Mattos de Oliveira Santos; advisor: Hélio Côrtes Vieira Lopes; co-advisor: Thuener Armando da Silva. – Rio de janeiro: PUC-Rio , Departamento de Informática, 2021.

v., 73 f: il. color. ; 30 cm

Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática.

Inclui bibliografia

1. Informática – Teses. 2. Mineração de dados;. 3. Detecção de drift;. 4. Entendimento de drift;. 5. Interpretação de drift;. 6. Árvore de decisão. I. Lopes, Hélio. II. Armando da Silva, Thuener. III. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. IV. Título.

CDD: 004

To my family, whose unconditional support made this accomplishment possible.

Acknowledgments

To my adviser Professor Hélio Côrtes for the stimulus and partnership to carry out this work. To my coadvisor Thuener Silva for also helping me with the guidance needed to conduct this research.

To CAPES and PUC-Rio, for the aids granted, without which this work could not have been accomplished.

To Petrobras, whose data and professionals made it possible to develop a research that could support their daily operations and also bring innovation to the field encompassed by this dissertation.

To my family, whose unconditional support made this accomplishment possible.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

Abstract

Mattos de Oliveira Santos, João Guilherme; Lopes, Hélio (Advisor); Armando da Silva, Thuener (Co-Advisor). **A method for interpreting concept drifts in a streaming environment**. Rio de Janeiro, 2021. 73p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

In a dynamic environment, models tend to perform poorly once the underlying distribution shifts. This phenomenon is known as Concept Drift. In the last decade, considerable research effort has been directed towards developing methods capable of detecting such phenomena early enough so that models can adapt. However, not so much consideration is given to explain the drift, and such information can completely change the handling and understanding of the underlying cause. This dissertation presents a novel approach, called *Interpretable Drift Detector*, that goes beyond identifying drifts in data. It harnesses decision trees' structure to provide a thorough understanding of a drift, i.e., its principal causes, the affected regions of a tree model, and its severity. Moreover, besides all information it provides, our method also outperforms benchmark drift detection methods in terms of false-positive rates and true-positive rates across several different datasets available in the literature.

Keywords

Data Mining; Drift Detection; Drift Understanding; Drift Interpretation; Decision Trees

Resumo

Mattos de Oliveira Santos, João Guilherme; Lopes, Hélio; Armando da Silva, Thuener. **Um método para interpretação de mudanças de regime em um ambiente de streaming**. Rio de Janeiro, 2021. 73p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Em ambientes dinâmicos, os modelos de dados tendem a ter desempenho insatisfatório uma vez que a distribuição subjacente dos dados muda. Este fenômeno é conhecido como Concept Drift. Em relação a este tema, muito esforço tem sido direcionado ao desenvolvimento de métodos capazes de detectar tais fenômenos com antecedência suficiente para que os modelos possam se adaptar. No entanto, explicar o que levou ao drift e entender suas consequências ao modelo têm sido pouco explorado pela academia. Tais informações podem mudar completamente a forma como adaptamos os modelos. Esta dissertação apresenta uma nova abordagem, chamada Detector de Drift Interpretável, que vai além da identificação de desvios nos dados. Ele aproveita a estrutura das árvores de decisão para prover um entendimento completo de um drift, ou seja, suas principais causas, as regiões afetadas do modelo e sua severidade.

Palavras-chave

Mineração de dados; Detecção de drift; Entendimento de drift; Interpretação de drift; Árvore de decisão

Table of contents

List of figures

List of tables

List of algorithms

Wassily Kandinsky, *Regards sur le passé.*

1
Introduction

Along the years, the constant evolution of the CPUs, led to an extreme reduction of its physical dimension and its market cost. Nowadays, it is possible to embed processing power to basically any product one may wish. Simultaneously, the transformation of the internet, which is day-by-day becoming faster and ubiquitous, made it possible for machines to communicate not only with humans but also with other machines, a phenomenon known as "Internet of Things". Furthermore, the development of Cloud services, where CPUs and other IT services are highly available on demand, enabled many industries to expand the digital transformation of their businesses. Since developing and maintaining these infrastructures is too expensive, consuming it as a service became the appropriate option for many manufacturers. All these factors combined provided the perfect scenario for the development of a world driven by data. The Global Datasphere, which corresponds to the summation of data generated, captured or replicated by the core (traditional and cloud datacenters), the edge (enterprise-hardened infrastructure, like cell towers) and the endpoints (PCs, smartphones and IoT devices), was in 2018 over 33 ZettaBytes (ZB) of data what is equivalent to 33 billions of Terabytes. It is expected to reach the amount of 175 ZB by 2025. Moreover, 49% of the world data is also expected to reside in public cloud services by 2025 as well (29).

Given the portrayed scenario, being able to learn from previous data and apply the acquired knowledge to unseen data became a crucial activity for industries to improve productivity. Algorithms engineered to perform these tasks are called Machine Learning algorithms. They work by identifying relevant patterns in data and generalizing these patterns to unseen data in order to infer a specific behaviour for them. There are three learning scenarios:

– Supervised Learning

  The target variable, which the algorithm aims to learn, is available for every observation. So patterns that correlate the target and feature variables are learnt by the algorithm and expressed as a function that is later used to predict the target variable for new observation.

– Semi-Supervised Learning

  The target variable is not available for every observation. There are normally much more unlabeled data than labeled data in this scenario, what generally makes learning patterns that correlate target and feature

variables much harder than in a supervised context. Algorithms normally combine supervised and unsupervised techniques to express a function that predicts the target for every observation.

– Unsupervised Learning

No labeled data is provided. Patterns are extracted from the relation among the observations of the data. Normally, clustering techniques and dimensionality reduction algorithms are used to infer groups of observations of similar characteristics.

Nowadays, there are numerous real world applications where such algorithms are applied, e.g., fraud detection in credit cards, email filtering, user preference prediction, etc. Environments where data is generated by a multitude of devices in a sequential manner and near real-time are denominated Data Streams. Developing learning algorithms for streaming data is challenging due to the unknown characteristics of the arriving instances in the stream. Many assumptions that are made for static data are not valid for streaming data.

Traditional Machine Learning algorithms assume static data, i.e., their characteristics are previously known and the algorithms are designed to process the whole dataset at once. The dataset is first loaded into memory and then used to feed an algorithm whose job is to fit a model according to the data at hand and the set of hyperparameters chosen. Thus, the observations are assumed to be i.i.d.(independent and identically distributed), the distribution of each feature is known beforehand, there is no time constraint for fitting a model - what favors the usage of complex techniques - and the whole dataset can easily be retrieved by the system, enabling the model to process the same instance several times to result in better performance. On the other hand, Stream Learning, which is commonly referred to as "Data Stream Mining", has a set particularities that make developing a model a more complex task. As anticipated, data streams generate data in a sequential and continuous fashion (37). As new data is always arriving, the models have to process them faster than the arrival rate in order to remain relevant for the stream. Therefore, less time-consuming - i.e., low computational complex techniques - are required. Another fundamental aspect of data streams is data unpredictability. Unlike static data, where we know the behaviour of the entire dataset at the beginning, dealing with data streams involves understanding that a given optimal model developed at a given timestamp, may not be adequate for the current data, so training a new model or adapting the current one may be required in order to reestablish the proper quality for the models. In other words, if we consider two data windows from the same Data Stream set some time apart from each

other, we may observe distributions that are statistically significantly different. Thus, we usually say that this Data Stream is non-stationary. We name this significant variation in the distribution of the data: *Concept Drift* and it is the fundamental topic of this thesis.

1.1 Concept Drift

Concept Drift is the phenomenon associated with changes on the statistical properties of a target domain in a streaming environment (23). Considering a data stream that provides a sequence of tuples (x_t, y_t) sampled from an unknown joint probability distribution $P_t(X, y)$, where x_t is a R^D vector in timestamp t that represents the D feature variables from the process simulated by our data and y_t is the discrete target variable at instant t. Concept Drift is defined as the change suffered over time by the joint probability distribution (6). Formally, given two timestamps t_0 and t_1, Concept Drift between them is represented by the following inequality:

$$P_{t_0}(X, y) \neq P_{t_1}(X, y)$$

According to the Bayesian Decision Theory (10), one can state that a classification problem can be described by the posterior probability of a target variable $p(y|X)$, its prior probability $p(y)$, the conditional probability density function $p(X|y)$ and the evidence factor $p(X)$ in the following mathematical expression:

$$p(y|X) = \frac{p(y)p(X|y)}{p(X)},$$

where the target variable, y, assumes different discrete values, and the evidence factor is a scale factor that constrains the sum of the posterior probabilities of all the values y can assume to be equal to one: $p(X) = \sum_{i=1}^{c} p(y_i)p(X|y_i)$, where y_i denotes each of the c distinct values of y. To make appropriate decisions, a learner considers the posterior probability of each class y_i given the feature variables X and chooses the one whose posterior probability is the highest.

The joint probability distribution $P(X, y_i)$ is equivalent to the product of the prior probabilities $p(y_i)$ and the conditional probability density function $p(X|y_i)$, so it is possible to reformulate the Bayesian Decision Theory for the joint probability distribution in the following two ways:

$$P(X, y_i) = p(y_i)p(X|y_i)$$

$$P(X, y_i) = p(y_i|X)p(X)$$

By looking at the problem through this perspective, since the evidence factor, $p(X)$, is just a scale factor in our formulation, it is evident that the joint probability distribution is affected by three independent components: the posterior probability, $p(y|X)$, the prior probability, $p(y)$, and the conditional probability density function, $p(X|y)$. As stated by Khamassi et al. (20), each of these components is responsible for a specific type of Concept Drift:

1. Virtual Concept Drift

   The variation of the joint probability distribution over time, $P_{t_0}(X, y) \neq P_{t_1}(X, y)$, is associated specifically with changes in the conditional probability, $p_t(X|y)$. Normally, this kind of drift is not prejudicial to the learner, since only the distribution of the feature variables are changing, not affecting the decision boundaries for the target. This situation is illustrated in Fig. 1.1b.

2. Real Concept Drift

   In this case, $P_{t_0}(X, y) \neq P_{t_1}(X, y)$ is due to changes in the posterior probability distribution, $p_t(y|X)$. This kind of drift poses huge risk to the learner's performance, since class distributions are changing even though the distribution of the feature variables remain the same. It should be detected as soon as possible and an action should be taken in order to reestablish appropriate decision boundaries for the classes. Normally, adaptive techniques are used to accommodate the new concept to the current model or a completely new model is trained in order to replace the old one. It is illustrated in Fig. 1.1a.

3. Class Prior Concept Drift

   Drifts related to prior probability, $p(y)$, is also associated with other common problems in the literature, i.e., class imbalance, novel class emergence or existing class fusion. Many authors don't differentiate these drifts since they can be considered as Virtual Drifts - class imbalance - or Real Drifts - novel class emergence and existing class fusion (20). These kind of drifts are shown in Fig 1.1c.

Concept Drifts can also be classified in terms of how they appear in a data stream. Regarding this matter, drifts can be divided into four categories: Abrupt, Gradual, Incremental, and Recurrent and are presented in fig. 1.2.

Abrupt Drifts are associated with a sudden shift in the data, e.g., in an Oil & Gas refinery, it could be associated with malfunctioning sensors,

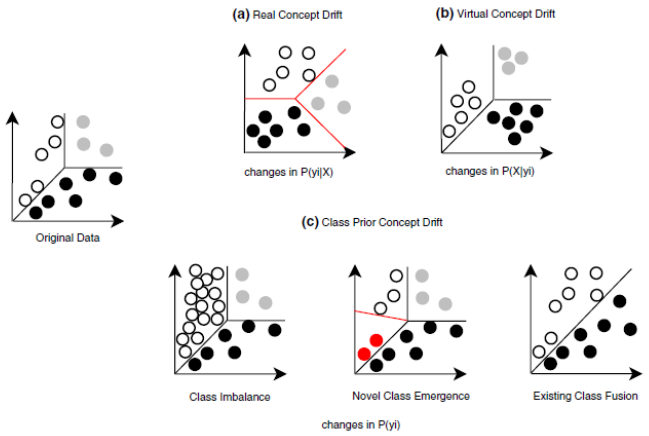

Figure 1.1: Illustration of the different types of Concept Drift: Virtual Drift, Real Drift and Class Prior Drift. This figure was extracted from (20)

sudden change in the input composites or other factors in the process capable of promoting a sudden shift in the joint probability distribution. These drifts pose a huge burden on the learner, since their performance rapidly deteriorates. As they occur in a specific timestamp, detecting these drifts are usually easier than detecting Gradual drifts.

Gradual and Incremental Drifts represent a change from one concept to another over some period of time. As the change happens in a slower pace, determining exactly when it started is quite difficult. There is a transient state where both concepts are active, therefore, the algorithms experience difficulties distinguishing noise in the data from the gradual changes in the concept and it slows the detection process. Many strategies have been developed to deal with these gradual or incremental changes, we investigate them on the next chapter.

Finally, Recurrent Drifts are normally Abrupt Drifts which appear seasonally in the data stream. Dealing with these drifts, over a concept drift detection perspective, is equivalent to dealing with Abrupt Drifts.

The latest surveys in Concept Drift pose new challenges to the area. Instead of only focusing on identifying the specific timestamp where a drift occurred, which is the central point of most algorithms developed for detecting drift so far, it also emphasizes that new researches should not only aim at Concept Drift Detection but also at Concept Drift Understanding, which is a broader perspective. Gama et al. (24) state that Concept Drift Understanding focus on answering three specific questions about Drifts: When, How and Where. “When” is extensively explored in the academia and refers to answering when the drift actually occurred. “How”, on the other hand, is appropriate for defining the adaptation strategy that should be adopted for the learner and

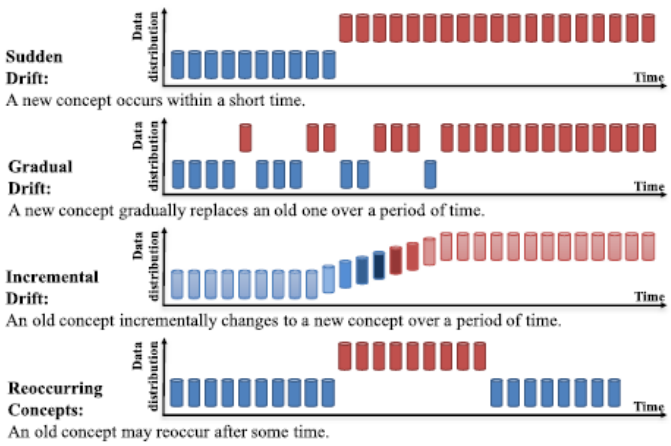

Figure 1.2: Illustration of the different types of Concept Drift: Abrupt, Gradual, Incremental and Recurrent. This figure was extracted from (24).

stands for how severe the drift is. Normally, severity is measured by comparing joint probability distributions and calculating the discrepancy between them. Being able to quantify drift severity is very important for disqualifying a previous model or just adapting it according to recent data. At last, “Where” stands for the regions in the conflicting zone between the old concept and the new concept. Identifying these regions is important in order to understand where most error-prone observations lie in the hyperspace.

There is still a fundamental topic when dealing with Concept Drift in real world scenarios that remains quite unexplored in the literature: “Concept Drift Interpretability”. To the knowledge of the author, only the work of Zheng et al. (41) explores the subject for the specific scenario of a dutch bank. Concept Drift Interpretability relates to identifying the root causes of a drift. This sort of information brings insights to the businesses that cannot be overlooked. Specifically, in the context of any manufacturer, they might help elucidate malfunctioning equipment, changes in the whole process, new characteristics of the feedstocks and many other aspects of these dynamic environments. Once a drift is identified, not only the models should be replaced or adapted, what is the main concern of Concept Drift Understanding, but also their root causes should be known, so the operators can take the proper actions in order to avoid it from happening again or mitigating it’s consequences. Adding interpretability to Gama et al. (24) definition of Concept Drift Understanding could be simply summarized as posing a new question for drift detectors: "Why did the drift happen?". Although this question has been briefly explored in the literature, it is of great importance for the industry in general.

1.2

Concept Drift Handling Strategies

Many methods are engineered to detect Concept Drifts in data streams and then provide appropriate adaptation mechanisms for the models: an approach known as *Active strategy*, which we detail thoroughly in the next chapter. However, Active strategies are not the only way to design learning algorithms in non-stationary environments. If we read the literature on Data Stream Learning, we find many authors that are more focused on developing drift resilient algorithms. Hao et al. (17), e.g., combine different Online Learning methods in an ensemble and weigh their predictions according to their assertiveness to form a final prediction. Furthermore, they still bind these techniques with Active Learning, in order to reduce the cost of querying labels. Note that this approach by no means tries to identify where a drift actually occurred. It only forces their algorithms to keep learning so that new concepts are gradually incorporated to their knowledge. There are plenty of strategies that use this methodology and they are denominated *Passive strategies*. This strategy is not concerned with identifying drifting zones but maintaining an up-to-date model at all times. It can be divided into *single* and *ensemble classifiers*.

Regarding *single classifiers*, an acknowledged approach for Data Stream Mining is Very Fast Decision Trees (VFDT). This algorithm was proposed by Hulten et al. (8) and it constructs an online tree according to the Hoeffding bounds, which determines based on an user defined threshold the number of examples needed to agree on a split in order for it to be consolidated in the tree. Later on, a new algorithm called Concept-adapting Very Fast Decision Trees (CVFDT) improved drift handling by maintaining a sliding window in memory and using it to train an alternative sub-tree. Whenever this sub-tree performs better than its original counterpart in the stable tree, the original sub-tree is replaced by the new one (19).

Ensemble classifiers cope with non-stationary environments by maintaining a pool of classifiers trained at different timestamps and combining their predictive power into a single prediction. Ensembles have been widely adopted because of their generalization ability. They can be classified by different aspects (20):

1. *block-based* or *incremental-based* according to how they process the data, either in "blocks", i.e., small chunks of data, or incrementally, processing observations sequentially one-by-one. Normally, for block-based strategies defining the size of the block is a bit tricky, since the classifiers should be able to generalize seasonal characteristics of the data

but not drifting trends, so outdated classifiers can be replaced by new trained ones and improve the overall performance of the ensemble.

2. *fixed-size* or *variable-size* according to the number of classifiers an ensemble accepts. Some strategies control the overall complexity of the model by fixing the number of classifiers allowed. Others aim at being resilient to recurrent drifts, so they allow an infinite number of classifiers and formulate appropriate techniques to adjust their weights, e.g. evolutionary-based optimization algorithms, in order to minimize misclassification rates. These techniques allow an old classifier to become relevant again once the appropriate concept emerges back into the stream.

Kotler and Maloof (21) proposed the Dynamic Weighted Majority algorithm, which is still a benchmark for recent studies in ensemble classifiers for streaming data. In their work, they reduce the weight of every base classifier that make a mistake, and add a new classifier every time the ensemble fails to detect the true label. The classifiers whose weights are smaller than an user defined threshold are then removed from the ensemble. Other relevant techniques in this field are Learn++.NSE, presented by (11), which are resilient to recurrent concepts as well. These two approaches are block-based passive strategies. Regarding incremental-based strategies, some of the most recognized works are Streaming Ensemble Algorithm (SEA) (32), Accuracy Weighted Ensemble (AWE) (34) and Accuracy Updated Ensemble (AUE) (4).

In general, Data Stream Learning can be segmented into the former two main strategies: *Active* and *Passive*. The *Active Strategy* focus on determining drifting zones through statistical analysis of data windows or online error rate of current learners. Some of the techniques widely used in these methods are: statistical hypothesis tests, dissimilarity measures between distributions (Hellinger distance, Kullback-Leibler Divergence, etc), Bootstrapping, Permutation tests, among many other techniques that are deeply discussed in the next chapter, since it is the strategy adopted for this work.

Testing these algorithms designed to work with streaming data requires appropriate techniques. The two most common are *Holdout* and *Interleaved Test-Then-Train* or *Prequential* Evaluation. Holdout Evaluation corresponds to withholding a subset of data for training and then another one for testing and iterate over this process while the data stream is available. There are a few issues with this approach. First, selecting samples for training in a streaming context. This is normally solved by selecting samples at varying time intervals, reducing the temporal dependence of the subset. Second, determining

the appropriate size of the train/test subsets is also a challenge. And third, the concept during training should be the same as the one during testing, in order to provide a fair evaluation of the models. How to guarantee that concepts are not changing during evaluation at real-time (13). All these problems should be considered when opting by a Holdout Evaluation process for the models.

Interleaved Test-Then-Train or Prequential Evaluation corresponds to the process of using each instance first for testing the algorithms and then for training them. This approach is highly used with sliding windows and decaying factors in passive approaches for improving classification models in streaming environments (37).

Having introduced some of the most fundamental aspects of Concept Drift Detection methods, we are now able to dive deeper into the research presented by this dissertation. Our work aims at identifying, understanding and interpreting concept drifts for the available datasets in the literature, joining together points raised in "Concept Drift Understanding" and "Concept Drift Interpretability". Altogether, we aim to respond the final set of questions regarding a drift: "When", "How", "Where" and "Why". "When", "How" and "Where" relate to understanding concept drifts and allow us to train relevant algorithms according to the different scenarios presented at any complex and dynamic environment, e.g., an Oil & Gas refinery. "Why" relates to interpreting drifts, i.e., finding it's root causes, and is of great importance in order to develop strategies that mitigate their risks or even prevent them from happening again. As interpretability is highly required in our research, we opt to use decision trees as base models. Besides that, we deeply explore the literature of Active Strategies to comprehend the dynamics of data streams, understanding and interpreting the drifts that present themselves along the data.

We organize this dissertation in the following way: Chapter 2 reviews the literature on Concept Drift Detection methods (Active Strategy). Chapter 3 presents the research conducted. Chapter 4 brings the results achieved by applying our method to synthetic and real-world datasets available in the literature and Chapter 5 reports our findings and presents future steps for the area.

2
Previous Work

There are many ways for machine learning practitioners to deal with evolving data. Nonetheless, if they wish to understand how data is evolving and acquire knowledge from the data generating process, change detection methods suit the purpose very well. The first detectors were based in sequential analysis, more specifically, in Sequential Probability Ratio Test (SPRT) (26). Given two distribution, P_0 and P_1, and a timestamp w, which represents the exact moment the distribution of the sequence $X_n = \{x_1, x_2, ..., x_w, x_{w+1}, ..., x_n\}$ switches from P_0 to P_1, the probability of observing subsequencies after w under distribution P_1 is significantly higher than observing subsequencies under P_0. This concept can be mathematically expressed by the log-likelihood ratio between the two pdfs (13):

$$T_w^n = log\frac{P(x_w...x_n|P_1)}{P(x_w...x_n|P_0)} = \sum_{i=w}^{n} log\frac{P_1(x_i)}{P_0(x_i)} = T_w^{n-1} + log\frac{P_1(x_n)}{P_0(x_n)}$$

A change is detected when T_w^n reaches a user-defined threshold. This idea is adopted by two very famous change detectors proposed by Page in 1954 (27):

– CUSUM (Cumulative Sum)

  CUSUM test is represented by $g_t = max(0, g_{t-1} + (x_t - \delta))$, where $g_0 = 0$. Here, x_t is the current observation, δ is the allowed magnitude of change and g_t is the cumulative test statistic. When it is higher than an user-defined threshold, an alarm is signaled. Note that only positive changes are reflected by the aforementioned formula. If negative changes should be detected instead of using max in the formula, we use min.

– Page-Hinkley

  PH test is represented by $m_T = \sum_{t=1}^{T} x_t - \bar{x_T} + \delta$, where $\bar{x_T} = \frac{1}{T}\sum_{t=1}^{T} x_t$. Again, δ and x_t are the allowed magnitude of change and current observation. This test compares the cumulative magnitude of change, m_T with the minimum observed value for $M_T = min(m_t)$, where t=$\{1, 2, 3, ...T\}$. When $m_T - M_T$ is greater than a user-defined threshold, the test signals an alarm.

Although these algorithms are not used alone to identify drift anymore, there are still very recent researches that combine them with other strategies to compose relevant drift detectors.

2.1 Concept Drift Detection

Most of the algorithms recently developed for actively detecting drifts in data streams can be explained by the framework proposed by Gama et al. (24), as fig. 2.1 illustrates. Their work decomposed concept drift detection algorithms into four main stages:

- *Stage I (Data Retrieval)* defines how data is retrieved by the stream. Since a single instance is not enough to infer properties from the current pdf, data is normally retrieved in chunks. How these chunks are built is defined by this step.
- *Stage II (Data Modeling)* pre-process the data in order to identify key features that better represent the current chunk. Normally, this step consists of applying dimensionality reduction or sample size reduction algorithms to the data. That way, a smaller and more meaningful representation can be abstracted from the original data. This step is not implemented by many concept drift algorithms.
- *Stage III (Test Statistic Calculation)* defines a dissimilarity measurement for comparing the historical data and the current data. In other words, quantifies the severity of the drift between the two windows. Defining a robust and accurate measurement has been extensively studied in the literature, but remains an open question.
- *Stage IV (Hypothesis Testing)* formulates hypothesis tests for evaluating statistical significance of the test statistic derived from the previous stage. Some of the most used hypothesis tests are: estimating the distribution of the test statistic, bootstrapping, permutation test and Hoeffding's inequality based bound identification.

According to Gama et al. (24), the different strategies for developing active concept drift detectors can be categorized into: *error rate-based*, *data distribution-based* and *multiple hypothesis test detectors*. We further analyze each of these methodologies, point out their most acknowledged methods and evaluate how they relate to drift understanding concepts, i.e., detecting when drift occurs ("When"), their severity ("How") and the regions affected by them ("Where").

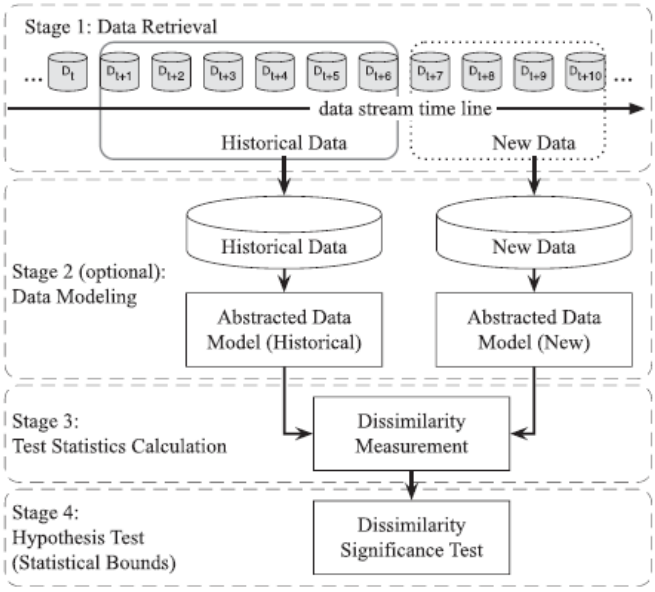

Figure 2.1: Illustration of the four stages that compose a concept drift detection algorithm. This figure was extracted from (24).

2.1.1 Error Rate-Based Detectors

Methods classified as error rate-based are normally designed for supervised classification problems. They develop their test statistic based on metrics from their corresponding classifier's confusion matrix, so they need a target in order to assess how their predictions are drifting from the real target values. As these methods track online error rates, most of them process data incrementally, i.e., they normally use landmark windows for identifying drifts.

One of the most acknowledged concept drift detectors is the DDM (Drift Detection Method) (14). This method works as follows: first, a binary classifier is trained on a reference window. After that, the instances of the data stream - represented by (X_t, y_t) at instant t, where $y_t \in \{0, 1\}$ - are processed sequentially in a landmark window (Stage I). The feature variables, X_t, are then abstracted by the learner's prediction, $\hat{y}_t$, composing the pair $(\hat{y}_t, y_t)$ that represents the data (Stage II). Once we have these pairs, a dissimilarity measure based on the online error rate $(y_t \neq \hat{y}_t)$ is created for evaluating possible drifts in the data through a hypothesis test (Stage III). In order to perform the hypothesis test, it needs to estimate a Normal distribution of the error rate. So, the algorithm simulates Bernoulli processes in order to generate a sequence of binary random variables for representing the errors and successes of the classifier. These Bernoulli processes form a Binomial distribution whose mean, p_t, is the error rate and the standard deviation is given by $s_t = \sqrt{\frac{p_t(1-p_t)}{t}}$. For a sufficient large number of examples, the Binomial distribution can be

approximated by a Normal distribution with the same mean and variance. That way, every time a new instance i is processed, the algorithm updates the Normal distribution and verify if $p_i + s_i < p_{min} + s_{min}$. If it is true, the current values for p_{min} and s_{min} are updated. Otherwise, it checks if $p_i + s_i \geq p_{min} + \alpha_{war} * s_{min}$, where α_{war} is the boundary for the warning level. From that point, the algorithm starts to build a new window for retraining the model, in case a future observation j presents $p_j + s_j > p_{min} + \alpha_{det} * s_{min}$, where $\alpha_{det} > \alpha_{war}$ and indicates the confirmation zone for a drift (Stage IV).

Under the assumption that DDM did not work very well for slowly gradual drifts, Beana et al. (1) developed the Early Drift Detection Method (EDDM). Instead of considering the online error rate, EDDM tracks the average distance between two sequential classification errors. While the model is learning the behaviour of the data, the distance between errors tend to increase. At each error, the algorithm calculates the mean, p_i, and standard deviation, s_i, of the distances and stores the highest value for the sum, $p_i + 2s_i$, as p_{max} and s_{max}. As this sum $(p_i + 2s_i)$ gets smaller, it means that errors are occurring at a higher frequency, therefore a new concept might be emerging. For the hypothesis test, EDDM uses the ratio: $\frac{p_i+2s_i}{p_{max}+2s_{max}}$. Warning and drifting zones are stipulated based on this ratio.

Numerous other works were developed based on the DDM algorithm in order to improve performance under particular conditions. For instance, the RDDM (Reactive Drift Detection Method) (2) brings a new approach for dealing with gradual drifts in DDM. It discards older instances during a gradual drift to overcome memory overflows in a warning zone. The DDM-OCI (36) seeks to improve the performance of the EDDM algorithm in imbalanced datasets. It assumes that drift in imbalanced data occurs when the minority class recall changes, what might not be true to all cases. There are changes to the distribution of the data that don't affect the minority class recall (37). Other approaches based on the DDM algorithm are the HDDM (Hoeffding's Drift Detection Method) (12), which uses Hoeffding's inequality for a two-sample statistical test in order to identify drift and the FW-DDM (Fuzzy-Windowing Drift Detection Method) (22) that implements a fuzzy window instead of using a landmark one for processing the data.

Another very successful approach for error rate-based detectors is the Adaptive Window (ADWIN) algorithm (3). This algorithm is normally combined with a learner, whose predictions are used in order to measure drift by the ADWIN approach. ADWIN considers sliding windows of size W and evaluate possible splits in W that could result in two subwindows W_0 and W_1 with significantly different distribution means $\mu_{\hat{W}_0}$ and $\mu_{\hat{W}_1}$. The distance be-

tween means is compared with a threshold, ϵ_{cut}, by the following inequality: $|\hat{\mu_{W_0}} - \hat{\mu_{W_1}}| \geq \epsilon_{cut}$, where ϵ_{cut} is calculated using the theory of Hoeffding's bounds. Whenever the inequality is satisfied, the oldest window is discarded and only the new one is kept in memory for retraining or adapting the current learner. That way, windows do not have fixed size, hence the name Adaptive Windows. ADWIN evaluates W splits and discards instances one by one according to the inequality formerly presented. Bifet et al. (3) also proposes ADWIN2, where instead of evaluating every instance individually, it compresses the instances into bigger sequences and evaluates these sequences according to their means in order to check for drifts in the data. This approach presents a lower computational complexity, since $O(logW)$ splits are considered for a binary classifier.

At last, EWMA Charts Concept Drift Detection (ECDD) (31) also proposes a unique way of detecting drift in non-stationary data streams. Exponentially Weighted Moving Average (EWMA) was first proposed by Roberts et al.(30) for detecting a significant increase in the mean of a sequence of random variables: $X_1, ..., X_t$. Suppose this sequence has mean μ_0 up to a change point and μ_1 after that point. It downweights the older data, using a parameter λ, in order to come up with an estimate of the mean more biased towards the recent values of the sequence. It is described by the following formulation:

$$Z_0 = \mu_0,$$

$$Z_t = (1 - \lambda)Z_{t-1} + \lambda X_t$$

The sequence of random variables, $X_1, ..., X_t$, represents the misclassification rate of the learner and can be viewed as a sequence of Bernoulli random variables, where the mean represents the probability of incorrectly classifying an example in the stream. Besides that estimation, ECDD develops another estimate for the same mean that does not downweight older values, given by:

$$\hat{p_{0,t}} = \frac{1}{t}\sum_{i=1}^{t} X_i = \frac{t-1}{t}\hat{p_{0,t-1}} + \frac{1}{t}X_t$$

That way, Z_t is more sensitive to changes in the mean and hence represents an accurate measure of it's current value, while $\hat{p_{0,t}}$ is less sensitive to recent changes, therefore representing an estimate of the pre-change value. These two estimate are then combined in order to raise drifts in the distribution of the misclassification rate, when the following inequality is satisfied: $Z_t > \hat{p_{0,t}} + L\sigma_{Z_t}$, where L is a time-varying threshold and σ_{Z_t} is the standard deviation of the EWMA estimation. ECDD has a very competitive performance

when compared to other error-rate based detectors. Besides that, this approach allows the rate of false positive detection to be controlled.

Although these algorithms have proven to be effective in identifying the moment a drift occurs, their ability to quantify drift and to detect drift regions are still very limited. Drift quantification could be measured by comparing overall accuracy in the historical window and the recent window, but the fact is that none of these approaches have successfully come up with an adequate technique for assessing drift severity or the appropriate region a drift occurs. Therefore, it is harder to gain deeper insights on the drift by only applying error rate-based techniques. Other approaches have proven to be more successful in analyzing drifts, nevertheless they might not be as accurate on detecting the time instant a drift occurs. As we investigate in the next section, data distribution-based detectors can be very useful for understanding drifts.

2.1.2 Data Distribution-Based Detectors

A concept drift can also be evaluated by the distributional characteristics of a non-stationary data stream. Data distribution-based detectors propose to measure dissimilarity between historical and current data distributions by using distance metrics and comparing them through distribution estimation or other techniques, as we further investigate on this section. Drifts detected by these methods are also referred to as *Distribution Drifts* and are known to represent their root cause. That way, besides detecting the time it occurs, most of them also provide knowledge regarding their severity and a few identify the main regions of the models that are affected by the drifts. Nevertheless, all this information also incurs in higher computational cost.

Under the perspective of the framework, illustrated by Fig 2.1, we can summarize most of the works developed in Data Distribution-Based methods by their main characteristics for each stage.

Normally, these methods adopt sliding windows techniques in Stage I, since they need to infer current characteristics of the data stream. They keep a fixed window - used to train models running in production - and a sliding window for capturing current concepts of the stream. These windows are compared and once a drift is detected, the fixed window is replaced and the models are retrained or adapted according to the new fixed window.

Although Stage II is not implemented by every method, the ones designed to work with high-dimensional datasets, normally, tend to abstract the real data by adopting multidimensionality reduction algorithms. This approach enables these methods to lower their computational burden to the system.

Nevertheless, abstracting real data may compromise drift interpretability, which is a disadvantage for these methods.

The most suitable dissimilarity measuresments for these detectors, defined in Stage III, are based on distance metrics between distributions. As Goldenberg et al. (15) reinforces in their work, a distance metric should satisfy the following rules:

- $D(x, y) \geq 0$ (Non-Negativity)
- $D(x, y) = 0 \iff x \equiv y$ (Identity of indiscernibles)
- $D(x, y) = D(y, x)$ (Symmetry)
- $D(x, y) + D(y, z) \geq D(x, z)$ (Triangle inequality)

Based on these rules, the different techniques used for quantifying dissimilarity between distributions are also investigated in (15). The most common ones exploited by data distribution-based methods are:

- *Kullback-Leibler divergence*

  KL divergence measures disparity between distribution - P and Q - through an information-based criteria:

  $$D_{KL}(P, Q) = E(-\log_2(\frac{P}{Q}))$$

  It can be interpreted as the amount of information lost for approximating P to Q. This measurement does not satisfy symmetry nor triangle inequality rules. Hence, it is not a distance metric. Authors normally adapt KL divergence in order to use it as a distance metric in their researches.

- *Hellinger distance*

  Given two probability measures, P and Q, which are continuous with respect to λ in a measurement space Ω, the Hellinger integral (also known as Bhattacharya coefficient) is defined as:

  $$H(P, Q) = \int_{\Omega} \sqrt{\frac{dP}{d\lambda}} \sqrt{\frac{dQ}{d\lambda}} d\lambda$$

  The Hellinger distance is defined based on this integral as:

  $$D_H(P, Q) = \sqrt{1 - H(P, Q)}$$

  The Hellinger distance has a closed-form calculation for univariate Normal and Poisson distributions. Furthermore, it can be approximated

for multivariate Normal distribution and unknown distribution of low-dimensional data. Thus, it is a very appropriate distance metric for estimating concept drift between two time windows.

– *Kolmogorov-Smirnov statistic*

The Kolmogorov-Smirnov statistic can be used as a distance, a goodness of fit measure or a hypothesis test. The KS distance is defined as the difference between two univariate distributions. Suppose we have a distribution X of n i.i.d. observations. We denote the cumulative density function (CDF) from this distribution by $F(x)$. Furthermore, we also calculate an empirical cumulative distribution function (ECDF) based on the following indicator function: $I_{[-\infty,x]}(X_i) = \begin{Bmatrix} 1, if X_i > x \\ 0, if X_i \leq x \end{Bmatrix}$. The ECDF can then be represented by the following formula:

$$F_n(x) = \frac{1}{n} \sum_{i=1}^{n} I_{[-\infty,x]}(X_i)$$

The KS distance between the CDF and ECDF is given by the norm L-infinity of their difference:

$$D_n = \sup_x |F(x) - F_n(x)|$$

– *Hotelling's T^2 distance*

Hotelling's T^2 distance has a few similarities with the Mahalanobis distance. Suppose we want to test if a p-dimensional sample X of size n comes from a distribution with mean μ. Given the sample mean $\bar{X}$ and variance-covariance matrix S - assumed to be singular -, we calculate the T^2 distribution with dimension p and n degrees of freedom through the following formula:

$$T^2(p, n) = n(\bar{X} - \mu)' S^{-1} (\bar{X} - \mu)$$

This distribution is proportional to the following F-distribution:

$$F_{p,n-p} = \frac{n-p}{p(n-1)} T^2$$

This distribution can then be used for a hypothesis test in order to assess if the sample mean $\bar{X}$ is statistically significantly different than distribution mean μ. Furthermore, when n is sufficiently large, the T^2

distribution can be approximated to a Chi-Squared (χ^2) distribution with p degrees of freedom and hence be used as a distance metric between the distributions. This distance is unitless and is measured according the standard deviation of the Chi-Squared distribution.

Having presented all these metrics, finally, the last stage, Stage VI, is responsible for developing the hypothesis test. At this stage, different strategies are considered, e.g., bootstrapping, permutation tests, distribution estimation and even algorithms based on sequential probability ratio test as the Page-Hinkley algorithm. Next, we bring the core ideas behind a few acknowledged data distribution-based methods and analyze how they relate to drift understanding concepts.

Dasu et al. (5) propose an information theoretic approach for detecting drifts in multidimensional data. Their research works with two sliding window models: *adjacent windows* and *fix-slide windows*. The first better captures rate-of-change, while the second is suitable for cumulative changes over time. We focus our explanation in the second model. Their approach uses a space partitioning scheme, called *kdq-tree*, that subdivides the hypercube composed by the features into cells. The *kdq-tree* algorithm is applied to the fixed window and, according to the resulting rules of this tree, the instances of the sliding window are placed into the appropriate leaves. For each leaf, the Kullback-Leibler divergence, $\hat{d}$, between samples of the fixed and the sliding windows is calculated. After that, Bootstrapping techniques are used to recalculate the Kullback-Leibler divergence several times. Based on the distribution of these KL-divergence estimates, which is a Normal distribution, a critical interval, (d_{hi}, ∞), is inferred according to a statistical parameter α for which the null hypothesis that both samples come from the same distribution is rejected. So, if $\hat{d}$ falls in the interval (d_{hi}, ∞), k consecutive times, where k is defined according to the number of instances in the sample and a threshold parameter, the algorithm assumes that a drift has occurred. Once a drift has occurred, by applying *Kulldorf spatial scan statistic*, it is possible to identify the regions that differ the most between the fixed window and the sliding window. All in all, this method is appropriate for detecting when and where a drift occurrs. Nevertheless, since Kullback-Leibler is not a distance metric, it does not provide any information regarding the severity of the drift.

Hellinger Distance Drift Detection Method (HDDDM) is another approach for identifying drift by analyzing data distribution (7). Ditzler et al. propose to approximate the baseline window and the sliding window distribution by a histogram with $\sqrt{N}$ bins, where N is the cardinality of the current data batch. The Hellinger distance is computed for each feature of the dataset

and then the average of these distances: $\delta_H(t)$ is stored. Next, the algorithm calculates the difference between the current average distance and previous average distance: $\epsilon(t) = \delta_H(t) - \delta_H(t-1)$. The difference between the distances, $\epsilon(t)$, is compared to an adaptive threshold in order to claim whether the change is really a drift or not.

Dos Reis et al. (9) propose a drift detection method through an incremental Kolmogorov-Smirnov Test. They adopt the same strategy of fixed-slide windows. Considering the reference set, A, and the current set, B, with n and m observations respectively, the KS-test is defined as: $D > c(\alpha)\sqrt{\frac{n+m}{nm}}$, where $c(\alpha)$ comes from a known table for a given significance level α and D is the KS-statistic ($D = \sup_x |F_A(x) - F_B(x)|$) as previously introduced. Once the inequality is satisfied, the algorithm rejects the null hypothesis the two samples A and B come from the same distribution. The proposed incremental KS-test is applied to each feature of the dataset individually. If the test fails for a given attribute, a drift is signaled. One downside of this approach is that drifts may happen without failing any of the tests. A multivariate test would result in more accurate results in drift detection. Nevertheless, it does not provide any information regarding drift regions or drift severity.

A PCA-Based change detection framework is proposed by Qahtan et al. (28). Their work uses the reference and sliding window paradigm for comparing data distribution. They suggest the utilization of PCA for extracting the k most important principal components of the reference window, i.e., the k components accounting for 99,9% of data variance. After that, the data of the reference and current windows are projected into these k principal components and their PDFs are approximated by a density estimator - histograms and KDE-Track are the ones used by their work. Furthermore, the method still proposes 3 different divergence metrics for comparing the current and reference sets: a modified symmetric KL-divergence; the intersection area under the curve of the two density functions; and a modified version of the LLH metric, which measures the likelihood of data samples from current window belonging to the density function estimated for the reference window. A change detector is created by selecting one of these metrics and testing it to the univariate estimated density functions of the projected reference and current sets into each principal component. A change-score, based on the selected metric, is then computed for each component and the highest one is kept in memory and represents the level of disagreement between the windows at that time. The Page-Hinkley algorithm is then applied for the series of highest change-scores in order to detect drift. This approach also relies on univariate distribution comparison for detecting drifts. That way, many gradual drifts and smaller

drifts might go undetected. The metrics, considered by the authors, can be used as a measure of drift severity in order to appropriately handle model adaptation.

2.1.3 Multiple Hypothesis Test Detectors

Multiple Hypothesis Test Detectors are also either online error-rate based or data-distribution based detectors. Their main difference to these approaches are the multiple hierarchical or parallel layers used for detecting drift.

Weng et al. (35) propose the Linear Four Rates (LFR) drift detector. This method relies on a binary classifier for identifying drifts, that way it is also an online error-rate based detector. It uses a landmark window that grows incrementally as new observations arrive. The LFR algorithm monitors 4 rates of a classifier's confusion matrix: *True Positive Rate*, *True Negative Rate*, *Positive Predicted Value* and *Negative Predicted Value*. For each of these rates, it keeps in memory a test statistic which numerically translates information from the sequence of prediction errors and successes. At each iteration, the algorithm performs a parallel verification on each monitored rate. The ones that have changed from one iteration to the next are subjected to a hypothesis test where Bernoulli random variables are used to simulate a sequence of predictions according to the current rate. The test statistic is then compared with the numerically approximated Normal distribution translated from the simulated sequences in order to test the hypothesis that the it actually comes from this distribution (Null Hypothesis) or not (Alternative Hypothesis). The null hypothesis is rejected if the test statistic lies beyond a user-defined threshold for the distribution, and hence a new drift is acknowledged. A less restrictive user-defined threshold is the drift warning zone, i.e., where the algorithm starts to build a new window with the arriving data. If the null hypothesis is rejected later, the model is retrained with the new window.

In order to reduce the rate of false positive detection raised by the LFR, Yu et al. (38) developed the Hierarchical Linear Four Rates (HLFR). Besides the parallel tests implemented by the LFR, HLFR performs hierarchical tests. The first layer applies the LFR algorithm to the data in order to detect drift. Once a drift is detected, it follows to a second layer, where the drift is validated. The second layer performs a permutation test with the window built by the first layer. It first divides the window into training and testing sets, then it trains a benchmark classifier in the training set and tests it in the testing set, and computes the *zero-one loss* for this classifier, $\hat{E_{ord}}$. After that, the same process is repeated P times for other new classifiers with the observations of

the window shuffled. Ultimately, the user defines a threshold η to validate the drift by verifying the following inequality: $\eta \geq \frac{1+\sum_{i=1}^{P} 1[\hat{E_{ord}} \leq \hat{E}_i]}{1+P}$, where $\hat{E}_i$ is the ith classifier *zero-one loss*. This approach significantly reduces the number of false positives from the LFR algorithm.

The work proposed by Yu et al. (39) adopts a similar Hierarchical Hypothesis Testing (HHT) framework. In their research, two different methods are designed: HHT with Classification Uncertainty (HHT-CU) and HHT with Attribute-wise "Goodness-of-fit" (HHT-AG). These methods avoid requiring true labels at every iteration, since they might not be immediately available depending on the application. Furthermore, although they mostly work in an unsupervised manner, they were developed specifically for classification problems.

The first method, HHT-CU, uses the Hoeffding's inequality in its Layer-I to monitor the moving average of the classification uncertainty measurement u_t, which is calculated by the l2-norm of the difference between the model's prediction, $\hat{y}_t$, and the posterior probability estimated by the classifier, $\hat{P}(y_t|X_t)$: $u_t = ||\hat{y}_t - \hat{P}(y_t|X_t)||_2$. Hoeffding's inequality does not require any assumption regarding the probabilistic distribution and is therefore appropriate in data stream learning scenarios. Once a drift is suspected, Layer-II runs a permutation test based on the *zero-one loss* the same way as formerly explained to the HLFR algorithm.

HHT-AG compares a baseline window, W_1, and a sliding window, W_2, with the same size N. For each feature of the data stream, $x^k|_{k=1}^{d}$, a "Goodness-of-fit" test, based on Kolmogorov-Smirnov distance measure, is conducted in order to determine whether the features of sets W_1 and W_2 come from the same distribution. If the test concludes that they don't come from the same distribution, Layer-I signals a drift to Layer-II. The second layer requires the true label of each instance from both windows, W_1 and W_2, and based on the features and true labels a 2D Kolmogorov-Smirnov test is now conducted to validate the drift.

2.2 Summary

Although error rate-based methods are the best option for precisely determining a concept drift, their accuracy is tightly coupled to the corresponding learner's performance. Therefore, they are very limited in terms of applicability, being suited for fully supervised classification problems where true labels are immediately available after a new instance is processed, which is not common in real world scenarios. Despite its limited applicability in the real world,

it lacks in understanding properties of the detected drift which may also be a problem since it becomes difficult to come up with an adequate strategy to restore a model's performance, rather than retraining the whole model. Data distribution-based methods, on the other hand, could perfectly be applied to non-supervised or semi-supervised scenarios as well as fully supervised scenarios, too. They rely on larger windows for capturing properties in the data and use these properties to infer drifts, not needing a particular model to assist them. Nevertheless, they are more exposed to false alarms, since changes in the data distribution may not necessarily affect the overall performance of these models. They could simply represent a purely virtual drift. These methods provide a lot more information regarding a drift than the ones based on learners.

The algorithm we propose in this work, as we detail in the next chapter, brings aspects from both sort of drift detectors: error rate-based and data distribution-based. Error rate-based, since our algorithm analyzes how the accuracy of distinct nodes in a decision tree varies along time, resembling what is defined as "real drifts", and data distribution-based, since it also takes into account how the behaviour of the variables associated with these nodes' rules is changing along time, what resembles the definition of "virtual drifts".

3
Proposal

In complex and dynamic environments, understanding the behavior of the data brings a lot of advantages to the related businesses. Apart from being able to detect a drift rapidly - which is crucial to keep models updated with the underlying distribution of the data - measuring drift severity and the main regions affected by it are also very important to provide mechanisms for adapting models more accurately than retraining them entirely. Furthermore, if we can relate a drift to a specific variable or to a set of variables, we are actually providing insights for businesses to correlate the real-world with the data that translates it. That way, drifts could be handled accordingly, not needing necessarily an algorithmic intervention to correct them, e.g., the drift could be associated with an uncalibrated sensor or a specific change in the sensor's position. In both cases, an operational approach could be adopted to replace the bad sensor with a calibrated one or moving the sensor back to its original position. So, understanding what led to the drift is crucial in order to adopt the right strategy for dealing with it. Our method aims at providing all this information by leveraging the structure of tree models.

Our method looks for drifts inside each node of the trained decision tree. That way, our knowledge regarding the whole scope of variables provided by the dataset is restricted to the ones selected by the algorithm to be a part of the tree. Consider the scenario of an Oil & Gas refinery. These environments are highly sensorized. Hence, analyzing their data brings a lot of information on the diverse sensitive operating conditions of a refinery. Imagine for instance that in a given moment the crude oil feedstock changed and their new characteristics imposed different temperatures and pressures to the distillation towers and other processes of the plant. If our model selected the features responsible for measuring the characteristics of the feedstock, it is probable that our method would be able to detect the root causes of this drift, which is indeed the changed oil feedstock. Nevertheless, if these features are not selected, but affected pressures and temperatures are, we would still be able to infer the root causes with the help of expert operators, since the model would accuse drifts on variables highly correlated with the type of feedstock used, i.e., the pressures and temperatures. On the other hand, if none of these variables were actually selected by the model, our method would then accuse drifts in nodes that have little or almost no relation to the true root causes themselves providing no real knowledge for the operators of a refinery. In other words,

by abstracting the feature data by the tree model, we are actually amplifying the universe of non-observable variables, even though some of them might be inferred indirectly by variables of the model. These non-observable variables, which could be inferred indirectly by observable variables, are called latent or hidden variables. Nonetheless, there are still other variables that will remain unknown to the digital modelling of the refinery and drifts occurring to them might go undetected or poorly detected by our method. It is a clear limitation of the strategy adopted by our research.

Despite the discussion on non-observable variables, capturing drifts in the data is crucial for developing models that remain as accurate as possible over extensive periods of time without any human intervention. By capturing the drifts, we aim at not only determining the moment it occurred, but also understanding its severity and assessing the most affected regions of the current learning model - an area presented by Gama et al. (24) as *Drift Understanding*. By processing this information, it is possible to develop accurate strategies for adapting the current model or retraining a new one. Furthermore, this research also addresses another relevant question to this matter: "why has the drift actually occurred?". To this specific question, we suggest a new term: *Drift Interpretation*. We observe that most research in the field of Concept Drift Detection has no intent in informing the real reasons behind a specific drift. However, comprehending these reasons makes a huge difference for businesses, since it translates to more information for the operation teams in the most diverse industries. We believe that concepts like *Drift Understanding* and *Drift Interpretation* are still not fully explored and can be used by businesses to have a massive impact on the way they operate on a daily basis.

3.1 Assessing Drift Criticality

In order to present the ideas behind our algorithm, we use a known stream generator, called Random Radial Basis Function. We define 10 features and 50 different centroids with random central positions, a standard deviation and a class label - we use a binary target variable for our analysis. The samples are then generated by randomly picking one centroid and offsetting the attributes in some direction farther from the center of this centroid. This generator creates a hypersphere of data around each centroid. Fig. 3.1 illustrates the generated dataset with 20000 samples. The number of instance were chosen in order to produce 4 distinct periods of significant length to train a decision tree or to analyze drifts along them. Indexes 0 to 9 correspond to a single feature in the dataset, and the last row (index 10) represents the target variable. In order

to evaluate the behaviour of each variable along the stream, we normalize the features to a $[0, 1]$ interval by applying a min-max scaler to each of them. That way, we are able to analyze the behaviour of each feature in relation to the others as well.

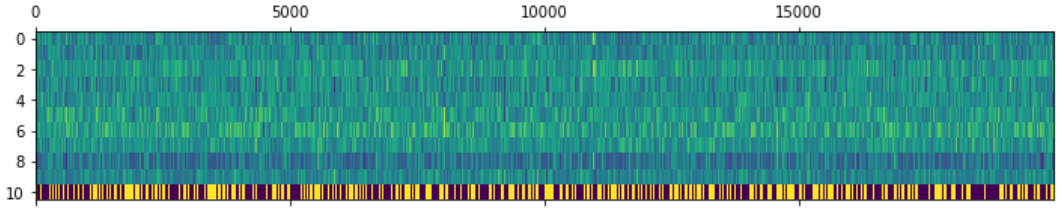

Figure 3.1: The heat map illustrates the features (indexed 0 to 9) and target (index 10) variables along all the 20000 observations of the dataset. The variables were scaled to the $[0, 1]$ interval by a min-max scaler. The target only assumes values 0 or 1 and so does not need to be scaled.

In order to analyze drifts appropriately, we perturb variables indexed by 0, 3, 5 and 8. Each of these variables were perturbed by adding a specific Gaussian noise. Variables 0 and 5 received a signal with mean, μ, equal to -1 and standard deviation, σ, equal to 0.2, while variables 3 and 8 received noises with $\mu = 1$ and $\sigma = 0.2$. Furthermore, the target variable (index 10) is also perturbed. In order to perturb this discrete variable, we use Bernoulli trials with success probability of 0.9. Fig. 3.2 illustrates the drifts added to the data. Note that each variable is perturbed during a specific interval: variable 0, during the interval $[6000, 9000]$; variable 3, for the interval $[12000, 15000]$; 5, from 10000 to 13000; 8, from 7000 to 9000; and the target variable is perturbed for the interval $[16000, 19000]$.

The dataset is divided into 4 distinct windows with 5000 instances each. The first window is used to train our base model, which is a decision tree. The training set is separated from the test set in fig. 3.2 by the red vertical line. The following three windows encompass the test set: second window, interval $[5000, 10000]$; third window, $[10000, 15000]$, and fourth window, $[15000, 20000]$. We aim to analyze how the trained decision tree performs on the test data on two specific aspects: accuracy and frequency. Note that for the last window, we only leave the perturbation on the target variable. That way, we are able to measure how the change in the labels influenced the different nodes in the tree and assess how the tree behaves when such a drift presents itself. The noises added to the feature variables should impact only the sub-trees rooted by the nodes that use these variables for their hyperspace segmentation. A node's rule may become inadequate, with frequencies for its child nodes completely different from the trained scenario, i.e., the first window. Thus, their accuracies may also deviate from previous standards.

The decision trees used in this exploratory analysis were trained with maximum depth of 4 and minimum number of samples per leaf of 5%. The maximum depth is chosen in order to enable a visual comprehension of the model, while the samples per leaf is a pruning parameter that avoids the generation of unsubstantial leaves in the tree. That way, we limit the complexity of our model and are able to interpret what is happening to it. Decision trees are actually constructed by using an heuristic for splitting the feature space that maximizes node purity. Every time a split is made, it is guaranteed that the variable and threshold chosen are the ones that yield the maximum combined node purity of the resulting child nodes. Therefore, the addition of a split to a node not necessarily means that the resulting nodes have different predictions. It only means that the overall node purity increased by the segmentation. The node purity is calculated either by the Gini index criterion or by the Cross-Entropy criterion. Our decision tree uses the Gini index, which is a measure of the total variance of a node's data across the different labels of the underlying target variable.

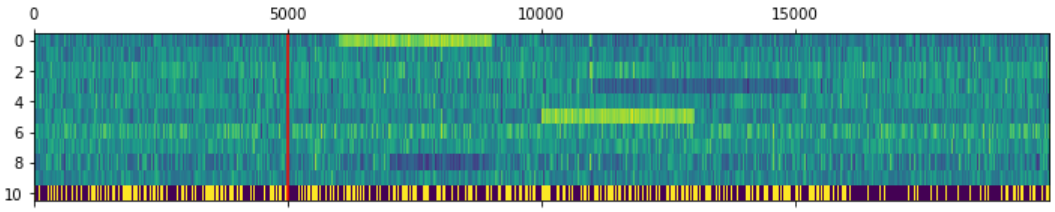

Figure 3.2: The heat map of the dataset with the Gaussian noises added to the feature variables: 0, 3, 5 and 8, and the noise added to the discrete target variable through Bernoulli trials.

3.1.1 Node Frequency Analysis

The Node Frequency Analysis aims at monitoring how frequent each node is accessed across the different periods in the dataset. By identifying periods where a node is much less accessed than it was during the training set, we suspect that the given rule attributed to that node no longer is a valid rule for appropriately segmenting the hyperspace in the stream. That way, we state that high variations to the frequencies of a node are correlated to the occurrences of drifts in the data. Fig. 3.3 shows how the node frequencies of a tree model behaves for the training set. The blue color becomes lighter the less frequent a node is. As expected, the tree is very well balanced since it was trained on this subset of the data.

Fig. 3.4 indicates the first period of the test set. During this period two distinct features (index 0 and 8) were perturbed. These features are used by

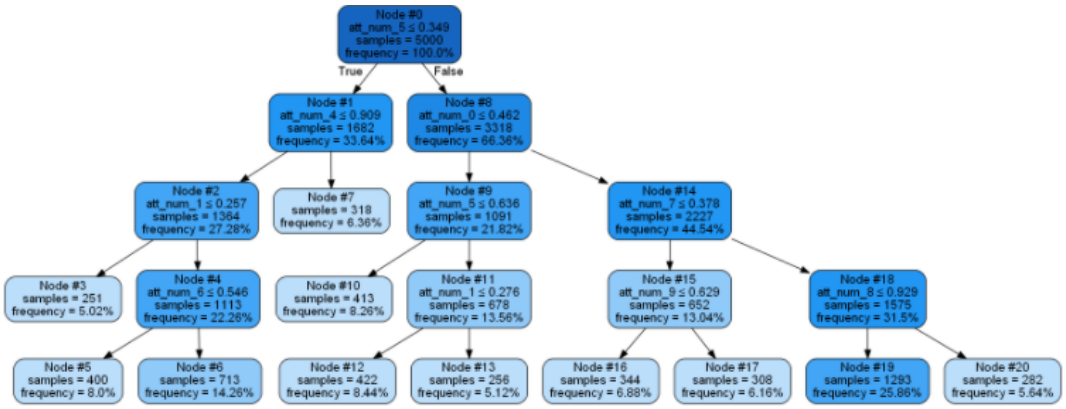

Figure 3.3: Node frequency visualization of the trained decision tree. A palette that goes from blue to white is used to represent the frequency a node is accessed. The less frequent a node is the lighter it's internal color becomes.

nodes #8 and #18. If we compare the data segmentation of these nodes for the first test set and the training set, we observe that it changed drastically for node #8, but not the same happened for node #18. Nodes #9 and #14, which are child nodes from #8, have respectively 50.04% and 16.82% of the data in the current period, while 21.82% and 44.54% during the training set. Although we already know that a drift happened for "att_num_0" - the variable used by #8 -, just observing these drastic variations in the frequencies of it's child nodes is sufficient to suspect the occurrence of a drift in this period. The same logic is applicable for node #18 that uses variable 8 ("att_num_8"). Nevertheless, it is harder to visualize the data segmentation change for this node, since the drift in "att_num_8" is shorter than the one in "att_num_0" and the node is deeper in the tree, so less observations pass through it. Furthermore, the effect of the first analyzed drift is also reflected into this second drift, since node #18 resides in the lighter sub-tree of node #8. If we compare the frequencies of nodes #19 and #20, which are child nodes from #18, for this period and for the training set, we observe that their ratio changed slightly from $25,86\%$ (#19) and $5,64\%$ (#20) in the training set to $9,4\%$ (#19) and $2,44\%$ (#20) in the current period.

During the second period of the test set, drifts were added to variables 3 ("att_num_3") and 5 ("att_num_5"). The variable "att_num_5" is used twice in the tree for nodes #0 and #9, while "att_num_3" is not used by any node at all. That way, since this last variable is not reflected in the model, perturbations happening to it are not acknowledged by our node frequency analysis, i.e., it is a non-observable variable for us. On the other hand, the drift added to "att_num_5" drastically impacts the tree model changing the whole configuration of it's node frequencies. By observing fig. 3.5, we are able to identify severe changes in the frequencies of the child nodes from #0 (nodes #1

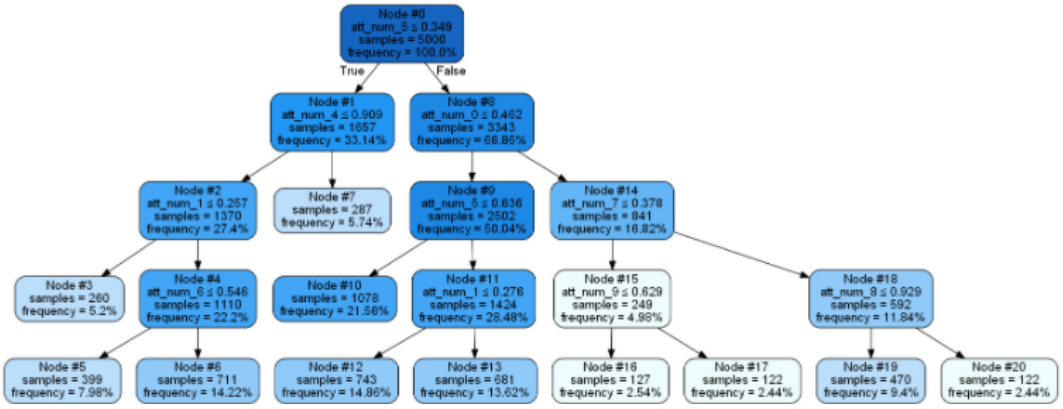

Figure 3.4: Node frequency visualization of the trained decision tree for the first period of the test set.

and #8). During the training set, illustrated by fig. 3.3, nodes #1 and #8 had respectively 33, 64% and 66, 36%, while in this period, these values changed to 74% and 26%. In the lighter sub-tree of node #0, variable "att_num_5" is also used in #9. Nevertheless, the majority of instances that reside in this lighter sub-tree are posterior to the drift in "att_num_5", so no drastic variations in the frequencies of nodes #10 and #11(child nodes from #9) are noticed.

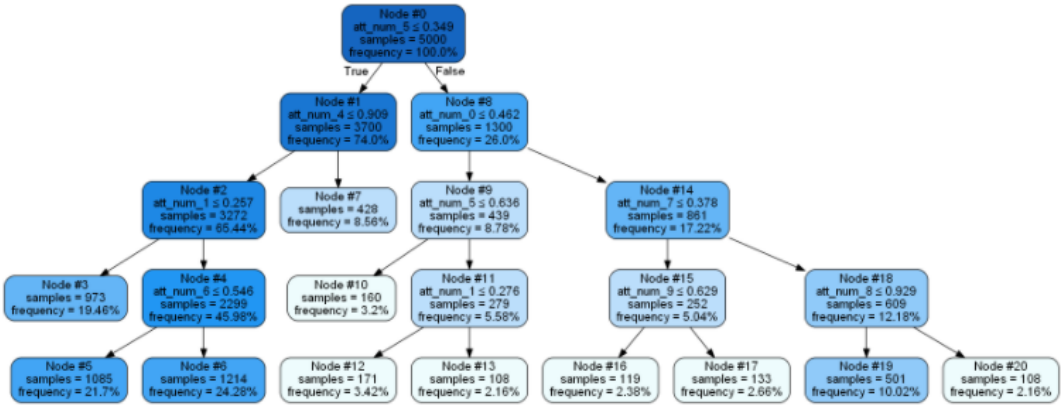

Figure 3.5: Node frequency visualization of the trained decision tree for the second period of the test set.

Fig. 3.6 shows the behaviour of the tree model for the last period of the test set. No drifts were added to the feature variables during this period, so it is expected that no drastic variations in the nodes' frequencies are noticed in this tree. Although there were perturbations to the target variable, they did not impact the node frequency analysis, and are therefore not identified in fig. 3.6. The tree reestablished the frequency behaviour observed for the training set.

Visually inspecting the trees can help understand where drifts are happening in the data. Nevertheless it can be a very overwhelming task, since in a real scenario, we would not know which variables are drifting and when

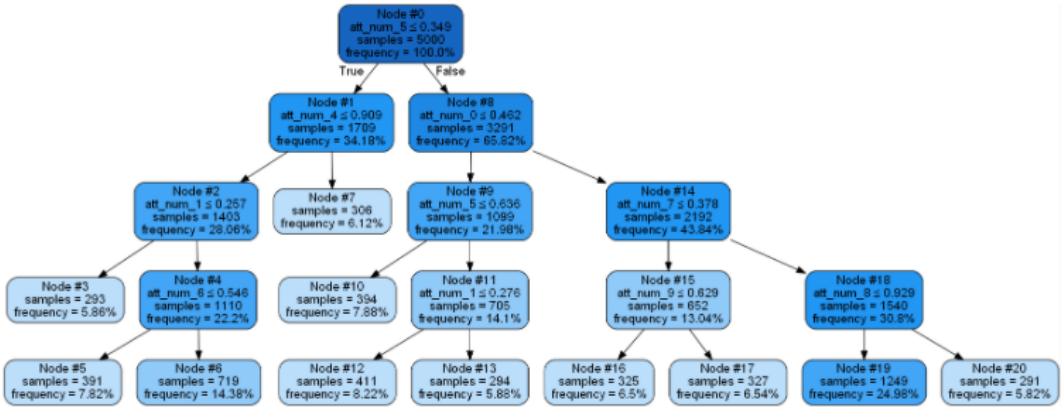

Figure 3.6: Node frequency visualization of the trained decision tree for the third period of the test set.

they are drifting. That way, in order to develop a thorougher analysis on drifts that does not rely on a visual inspection of the trees, we use bootstrapping techniques to estimate confidence intervals for a given test statistic during the training set, and then determine whether the same statistic computed for the test set lies inside the stipulated interval. We start our approach by collecting the instances from the training data passing through the nodes that contain a feature chosen for drift evaluation. We then extract 500 samples - a parameter that was chosen empirically - by block-bootstrapping each node's data subset and calculate for each sample it's child nodes' frequencies based on the feature rule of the parent node. The sampling distribution of these frequencies forms a Normal distribution, based on which we estimate the appropriate confidence intervals. For this analysis, 95% confidence intervals are stipulated ($2-\sigma$ rule).

This statistical analysis is conducted on one child node from each of the following nodes: #0, #8, #9 and #18. As visually inspected, the rules of these nodes are based on variables that have drifted along the stream: "att_num_0" is used by #8; "att_num_5" is used by #0 and #9; and "att_num_8" is used by node #18. That way, selecting one of their child nodes and conducting the formerly explained statistical analysis on them is sufficient to assert that a drift happened. We just need one child since the frequencies are complementary, i.e., the instances passing through the parent node can either go to the left child or to the right child, so analyzing any of them is the same. We selected the right child from each node: #8 (right child of #0), #14 (right child of #8), #11 (right child of #9) and #20 (right child of #18). Fig. 3.7 shows the sampling distribution for the frequency of each child node. These sampling distributions were generated by calculating the test statistic - i.e. the frequency of a node in the tree - in each block-bootstrapping sample of the training set. These samples are of size equal to $\frac{2}{3}$ the size of the training subset passing through

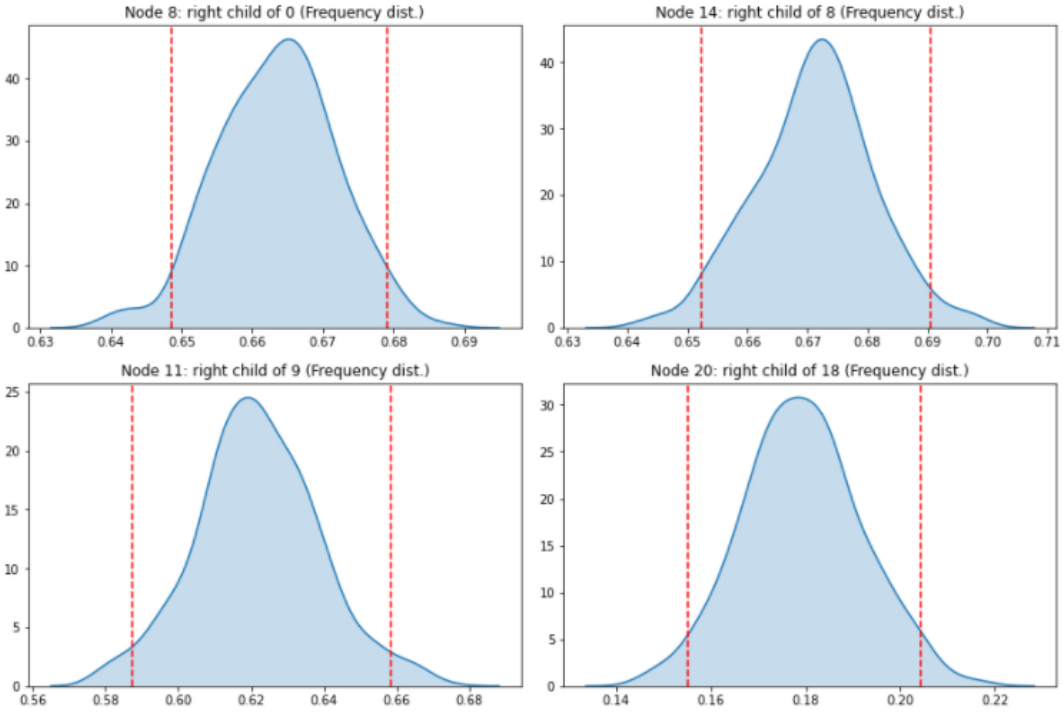

Figure 3.7: Sampling distribution of the frequency of each node selected for drift evaluation based on the training data. The red vertical lines denote the 2.5 and 97.5 percentiles of the distributions. The area between these lines correspond to accepting values for the true frequencies on the test set in order to reject the hypothesis that a drift happened.

the corresponding node. If during the test set we observe frequencies that lie outside the 95% confidence intervals, we have a more reliable indication that a drift happened.

	Training Data	Testing Data		
Nodes	95% CI	[5000, 10000]	[10000, 15000]	[15000, 20000]
Node #8	[0.648, 0.679]	0.668	**0.260**	0.658
Node #14	[0.652, 0.690]	**0.251**	0.662	0.666
Node #11	[0.587, 0.658]	0.569	0.635	0.641
Node #20	[0.155, 0.204]	**0.206**	0.177	0.189

Table 3.1: The 95% confidence intervals for the training data and the true frequencies for the different segments of the test data are shown by the distinct nodes analyzed. The frequencies of the test set periods that lie outside the intervals are marked in bold and they mean that a drift might have happened for the given node at the given period.

By analyzing Table 3.1, we observe that the frequencies that crossed the boundaries of the two-sigma interval are marked in bold. It happened for three nodes in two distinct periods. Note that the frequencies for Node #8 in the second period and for node #14 in the first period deviate extremely from their expected intervals. These are clear indications that drifts occurred

to the features attributed to their parent nodes. Furthermore, there is still a slight deviation for node #20, which one could argue that a three-sigma interval would not have considered it as a drift. So it could definitely be due to normal variations in the data and not actually a tendency itself, as a drift suggests. Nevertheless, if we refer back to fig. 3.2, we observe that features "att_num_0" and "att_num_8" drifted for the first period of the test set, while "att_num_3" and "att_num_5" drifted for the second period. The variables "att_num_0", "att_num_5" and "att_num_8" are used respectively by nodes #0 (parent of node #8), #8 (parent of node #14) and #18 (parent of node #20). That way, we can definitely state that node #20 crossed the boundary due to a drift in the variable "att_num_8". Since this node is very deep in the tree less data pass through it and hence analyzing drifts the same way as we do for shallower nodes is a bit unfair. We discuss a better approach for this issue further on this chapter.

3.1.2 Node Accuracy Analysis

The Node Accuracy Analysis is highly correlated to real drifts in the data. A significant drop in the accuracy of a specific node means that the set of rules that compose the path to this node is not reliable anymore to approximate the target set passing through the node. That way, if a node's frequency has not changed for the period but the accuracy has, it basically means that a real drift happened to the data, which is not associated with a change in a node's frequency.

Figure 3.8: Node accuracy visualization of the trained Decision Tree. A palette that goes from green to red is used for representing the accuracy of a node. If the accuracy is lower than 60%, we use a red palette, otherwise a green palette is used.

Fig. 3.8 illustrates the decision tree for the training set with respect to their nodes' accuracies. The accuracy of the decision tree is $80,24\%$ and is represented by the accuracy of the root node (node #0). The colors used to represent these accuracies are green and red. Once they drop below 60% a red palette is used, otherwise the nodes are painted in a green palette. Note that some leaves - specifically nodes #7, #10 and #20 - are painted in red, indicating that not every prediction is highly reliable even for the training set. Although they are less trustworthy, their accuracies are obviously above the 50% threshold, otherwise the algorithm would choose the complementary value of the binary target to classify them.

Figure 3.9: Node accuracy visualization of the trained decision tree for the first period of the test set.

The nodes' accuracies of the trained decision tree on the first test set are illustrated by fig. 3.9. Note that the overall accuracy of the tree dropped to 72%. It is a clear evidence that the drifts attributed to variables "att_num_0" and "att_num_8" affected the overall performance of the tree. If we look deeper into the tree, we observe that the highest drop in accuracy is associated with the sub-tree rooted by node #8 that uses the drifting variable "att_num_0". Their child nodes' frequencies are significantly impacted by the drift, what makes them more error-prone to the new observations they classify. It happened for node #9, a child node from #8, which received around 22% of the data for the training set and started receiving around 50% of the data for the first period of the test set. It affected it's accuracy which dropped from $77,09\%$ to $62,55\%$. On the contrary, node #18, which uses the drifting variable "att_num_8", did not suffer any loss in accuracy. It is due to a lower frequency for this node during the period and to it's already poor performing child node #20.

For the second period of the test set, illustrated by fig. 3.10, we observe

Figure 3.10: Node accuracy visualization of the trained decision tree for the second period of the test set.

that the other sub-tree from the root node #0, the sub-tree rooted by #1, is deeply affected by the drifts added during this period. Node #1, which used to receive 33% of the data - as figs. 3.8 and 3.9 illustrate - received 74% of the data during this period. It is a clear consequence of the drift added to variable "att_num_5", used by the root node #0. As previously explained, this sudden change to a node's frequency can also jeopardize it's accuracy, affecting the whole tree. Specially when it happens to the shallower nodes of the model, since they commonly receive a higher percentage of the data.

Although we separate the accuracy analysis from the frequency analysis, it is very common to see them both happening at the same time. Normally, the changing behaviour of a variable can jeopardize the accuracy of the whole sub-tree rooted by the node that makes use of that variable, and so both frequency and accuracy are indeed impacted. Further, we analyze the effects of another sort of drift: the drifts added to the target variable. Our synthetic dataset contains this sort of drift for the third period of the test set, which can be visualized by analyzing the inteval $[16000, 19000]$ in fig. 3.2. Note that the target variable for this interval is mostly painted in violet with thin yellow lines, what denotes a region of imbalanced target. Our last tree focus on identifying the effects of this sort of drift to the accuracies of the nodes.

Fig. 3.11 shows the tree for the last period of the test set. At first glance, we notice that the nodes' accuracies dropped drastically in comparison to the training set tree. A more careful analysis would also imply that the frequencies from the training set tree have been preserved, evidencing that only the target variable was perturbed for the period. If we isolate the sub-tree rooted by node #8, we observe that every red painted node is comprised by this sub-tree, making them the main responsible for the drastic drop in accuracy. At

Figure 3.11: Node accuracy visualization of the trained decision tree for the third period of the test set.

this tree, it is also possible to observe nodes where accuracy is below 50%, as it is the case for nodes #10, #17, #18, and #19. Although they seem a bit more problematic, if we compare the overall accuracies of the resulting trees for all the drifting periods, we observe that they actually did not vary that much: $72,56\%$, $67,36\%$ and $67,06\%$, respectively. That way, whether the drift manifest itself as a change to the target variable or to the feature variables, the tree accuracy can indeed be drastically impacted, so none of these drifts should be disregarded by any drift detection algorithm.

Analogously to the Node Frequency Analysis, a statistical evaluation of the drift is also performed for the Node Accuracy Analysis. Although the principles are the same, this one is a bit simpler, since we are actually using the own node's accuracy as the test statistic for the estimation of the sampling distribution. We again block-bootstrap 500 samples of a predefined size based on the subset from the training set passing through the corresponding node, and calculate that node's accuracy for each sample, forming a sampling distribution. Fig. 3.12 illustrates the sampling distribution for a few selected nodes from the tree: #0, #1, #2, #8, #14, and #18. The red vertical lines delimit the 95% confidence intervals. Based on them, we assess whether the data drifted by analyzing corresponding nodes' accuracies for the different test set periods, as table 3.2 elucidates. The accuracies that fall outside the stipulated confidence intervals are marked in bold, denoting that the underlying variables should be analyzed since they are suspected of having drifted for the periods. Moreover, once the drift is confirmed, the model could be retrained in order to suit the new characteristics of the data and restore the adequate performance.

The next subsection presents an algorithm capable of identifying regions

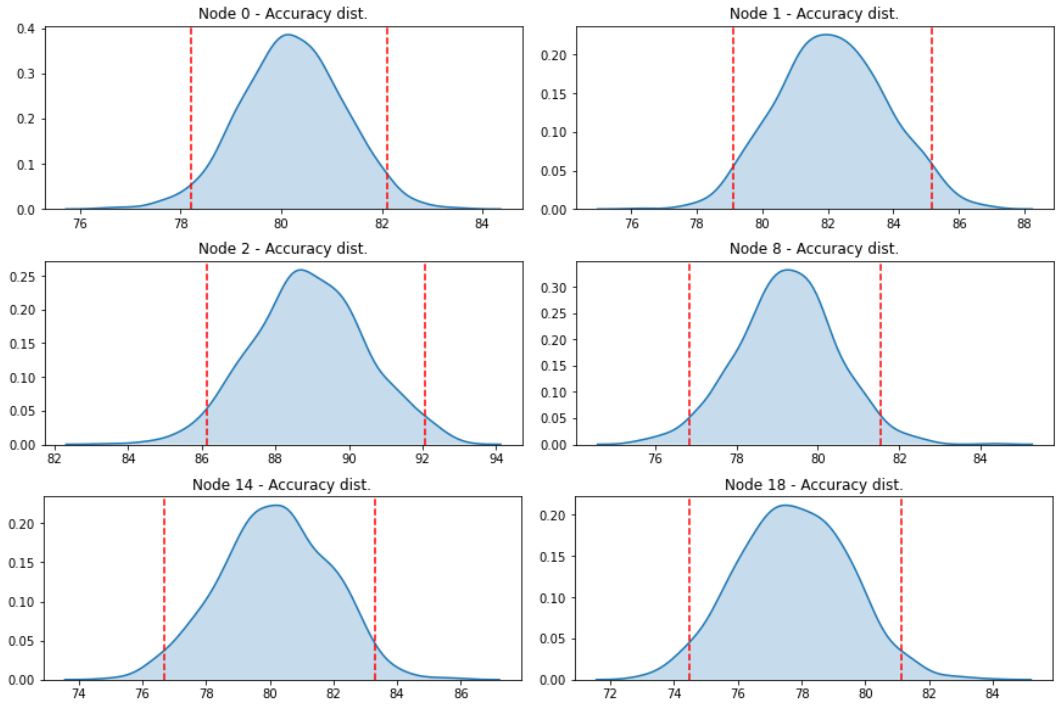

Figure 3.12: Sampling distribution of the accuracy of each node selected for drift evaluation based on the training data. The red vertical lines denote the 2.5 and 97.5 percentiles of the distributions. The area between these lines correspond to the accepting values for the accuracy on the test set in order to reject the hypothesis that a drift happened.

of a tree model affected by the drift, assessing the drift's severity, estimating the period the drift last, and foremost, diagnosing why the drift happened. This algorithm is called *Interpretable Drift Detector* and is entirely based on the analysis introduced in these subsections (3.1.1 and 3.1.2).

	Training Data	Testing Data		
Nodes	95% CI	[5000, 10000]	[10000, 15000]	[15000, 20000]
Node #0	[78.21, 82.11]	**72.56**	**67.36**	**67.06**
Node #1	[79.11, 85.18]	83.16	**63.00**	87.47
Node #2	[86.12, 92.07]	89.19	**64.15**	90.59
Node #8	[76.85, 81.56]	**67.30**	79.76	**56.45**
Node #14	[76.68, 83.29]	81.45	79.90	**51.45**
Node #18	[74.48, 81.14]	77.53	77.83	**45.84**

Table 3.2: 95% confidence intervals for the accuracy of each node selected for drift evaluation in the training data and their accuracies for each period of the test data. The test accuracies that lie outside the confidence intervals are marked in bold.

3.2

Interpretable Drift Detector

As previously stressed, the adoption of decision trees as base learners brings a lot of interpretability to algorithms suited for drift detection. We aim at deeply exploiting all these intrinsic characteristics of a tree model in order to provide the most information possible regarding any drift in the data.

Although previously we were able to detect drifts by just visually inspecting the trees (see section 3.1), it is not easy to come up with the right intervals to plot the perfect trees that would clarify drifts along the stream. If the drifts are not known beforehand, it would be an overwhelming task to find these intervals, and very possibly we would never find them. Moreover, the purpose of developing a drift detector is to be able to infer drifts in an unknown dataset without any human intervention, so that the model's performance is restored to optimal standards, if it is indeed affected by a drift. The advantage of our method is that we diagnose the drift thoroughly, instead of just detecting it, so users are able to trace the right strategy to correct the drift, whether by retraining the model or acting on other factors of their digital environment.

Our methodology is called *Interpretable Drift Detector* and is mostly based on the concepts introduced in the previous section (see section 3.1). The *Interpretable Drift Detector* defines a training set, for which the base model - either a regression or a decision tree - is trained. We explored the case where the base models are decision trees, since the dataset presented earlier, which is used as an example for our explanation, brings a classification problem. Once the model has been trained, the algorithm goes through every node of the tree, subsetting the training set with the observations that pass through the given node. That way, for each node, there is a subset of the training set, which is used to block-bootstrap a fixed number of samples, n_{samp}, with a predefined size chosen based on the node's subset cardinality, $|W|$. For our algorithm, we define the size of the samples to be $\frac{2}{3}$ of $|W|$. Once all n_{samp} samples have been generated, we calculate the appropriate test statistic, which is either one of the child nodes' frequencies (see subsection 3.1.1) or the node's accuracy (see subsection 3.1.2). After calculating both metrics for each sample of a given node, we generate the node's sampling distributions for both analysis. Based on them, we calculate the means, μ_{freq} and μ_{acc}, and the standard deviations, σ_{freq} and σ_{acc}. Algorithm 1 elucidates the process of computing the appropriate test statistics for each node of the trained decision tree.

Algorithm 1: Computing node sampling distribution parameters

Input: Training Data: (X_{train}, y_{train}); Maximum tree depth: max_depth; Number of samples: n_{samp}; Size of a sample: $size$

Output: Node array for frequency std: std_{freq}; Node array for accuracy std: std_{acc}; Node array for mean frequency: $mean_{freq}$; Node array for mean accuracy: $mean_{acc}$; Trained decision tree: dtc;

```
samplingDistributionTreeStats(X_train, y_train, n_samp, size):
    dtc ← DecisionTreeClassifier(max_depth).fit(X_train, y_train)
    for i ← 1 to n_samp do
        X^i_samp, y^i_samp ← sampleWithReplacement(X_train, y_train, size)
    end
    samples_freq, samples_acc
     ← newFloat[length(dtc.nodes)][n_samp]
    for i ← 1 to n_samp do
        W ← length(y^i_samp)
        for node ∈ dtc.nodes do
            subset ← setPassingThrough(X^i_samp, y^i_samp, node)
            freq ← size(subset)/W
            acc ← sum(dtc.predict(subset.X) == subset.y)/W
            samples_freq[node][i] ← freq
            samples_acc[node][i] ← acc
        end
    end
    std_freq, std_acc, mean_freq, mean_acc ← newFloat[length(dtc.nodes)]
    for node ∈ dtc.nodes do
        std, mean ← calcStdAndMean(samples_freq[node])
        std_freq[node], mean_freq[node] ← std, mean
        std, mean ← calcStdAndMean(samples_acc[node])
        std_acc[node], mean_acc[node] ← std, mean
    end
    return std_freq, std_acc, mean_freq, mean_acc, dtc
end
```

The next step is a bit different than we previously did in subsections 3.1.1 and 3.1.2, when we estimated confidence intervals for the sampling distributions. For our algorithm, we calculate the z-scores according to a sliding window that moves through the data. That way, instead of having a fixed threshold for determining drifts based on a node's previous behaviour, we can

define more flexible thresholds according to different aspects of a drift, as we detail further on this section. After calculating the z-scores, we grade every node for drift evaluation. We disconsider the leaves of the tree, since they do not contain a rule for segmenting the data, and hence, are not directly associated with any variable from the dataset.

The process of grading nodes is composed of the z-scores, a drift threshold and a sigmoid function which normalizes the results to the $[0, 1]$ interval. The dataset is processed by a sliding window of a user-defined size, k. In every slide from the window, the algorithm recalculates the frequency and accuracy for each node, $\mu_{freq_{test}}$ and $\mu_{acc_{test}}$. With that information, we are able to obtain the z-scores for the Node Frequency and the Node Accuracy analysis. The z-scores are calculated in the following way: $z_{score_{acc}} = \frac{\mu_{acc_{test}} - \mu_{acc}}{\sigma_{acc}}$ and $z_{score_{freq}} = \frac{\mu_{freq_{test}} - \mu_{freq}}{\sigma_{freq}}$. After the z-score calculation, we define the drift threshold for the nodes, i.e., the maximum value for the z-score for which we ignore the variation in the frequency or in the accuracy. The drift threshold is calculated based on the node's weight, w, which is the ratio between the number of observations passed through the node and the total number of observations from the window. The drift threshold is a linear function of the node's weight: $d_{max} = \alpha * w + \beta$, where α and β are constants in our analysis. After empirically testing different set of values for α and β, we opted by setting $\alpha = 4$ and $\beta = 2$. That way, the $d_{max}(w) = 4 * w + 2$. The grade is then initially defined by the equation: $g(w, z_{score}) = w * 2^{z_{score} - d_{max}(w)}$. At last, we apply a sigmoid function to $g(w, z_{score})$, in order to normalize the results. So the final grade is calculated by the following equation: $grade(w, z_{score}) = \frac{1}{1+e^{-g(w, z_{score})}}$. Note that if $g(w, z_{score}) = 0.0$, then the grade attributed to the node is 0.5. Therefore, grades below or equal to 0.5 should not be taken into account for drift evaluation. Algorithm 2 details the mechanics of the method.

Algorithm 2: Interpretable Drift Detector

Input: Data: $(X_t, y_t)|_{t=1}^{\infty}$; trained decision tree: *dtc*; sliding window size: W;

Output: Frequency grades matrix (nodes x instances): m_f; Accuracy grades matrix (nodes x instances): m_a;

```
interpretableDriftDetector(X, y, dtc, W, mean_freq, std_freq, mean_acc, std_acc):
    for t ← 1 to ∞ do
        if t ≤ W then
            m_f[dtc.nodes, t] ← 0.0
            m_a[dtc.nodes, t] ← 0.0
            sw[t] ← (X[t], y[t])
        else
            sw.remove(1)
            sw[W] ← (X[t], y[t])
            for node ∈ dtc.nodes do
                subset ← setPassingThrough(sw, node)
                freq ← size(subset)/W
                acc ← sum(dtc.predict(subset.X) == subset.y)/W
                z_score_freq ← (freq − mean_freq) / std_freq
                z_score_acc ← (acc − mean_acc) / std_acc
                m_f[node, t] ← calcGrade(z_score_freq, freq/W)
                m_a[node, t] ← calcGrade(z_score_acc, freq/W)
            end
        end
    end
    return m_f, m_a
end
calcGrade(z_score, w):
    if w ≤ 0.05 then
        return 0.0
    else
        d_max ← 4w + 2
        g ← 2^(z_score − d_max)
        return 1/(1 + e^(−g))
    end
end
```

In order to exemplify a use case of the *Interpretable Drift Detector*, we

use the synthetic dataset introduced earlier in this chapter. The dataset is particularly interesting since it presents several drifts to different variables and a final one to the target variable (see fig. 3.2). Furthermore, recall that this dataset contains 20000 instances, from which the first 5000 were used for training the decision tree. The method uses a sliding window of size 2500 for processing the data. Figs. 4.1 and 4.2 bring the drifting grades from the Node Frequency and the Node Accuracy Analysis, respectively. Note that the nodes are shown in the y-axis, while the instances, in the x-axis. The palette denotes grades in the interval $[0.5, 1.0]$. The purple color indicates the lowest grade possible, 0.5, and represents a non drifting zone. The yellow color, on the other hand, indicates high drifting zones and is associated with the highest grade, 1.0.

By analyzing fig. 4.1, it is possible to observe extremely high drifting grades for nodes #0, #1, #2, #4 (all in the second test interval: $[10000, 15000]$) and #8 (in the first test interval: $[5000, 10000]$). There are other relevant drifting grades for nodes #9, #11(at a similar period to node #8) and #18(in the end of the first period and beginning of the second). The last 5000 instances do not raise any suspicion of drifts by this analysis. Note that the grades reflect a very similar scenario to the one observed by analyzing the trees (see subsection 3.1.1). In the first period, variables "att_num_0" (used by node #8) and "att_num_8" (used by #18) drifted for the intervals $[6000, 9000]$ and $[7000, 9000]$, respectively. In the second period, variables "att_num_5" (used by #0 and #9) and "att_num_3" (not used by any node) drifted for intervals $[10000, 13000]$ and $[11000, 15000]$, respectively as well (see fig. 3.2).

Assessing the nodes' behaviour, we notice that node #8 presented the highest drifting grades (≈ 1.0) during the approximate interval $[6150, 11800]$. So it clearly detected the drift pretty early but did take a while to acknowledge the end of it. The high grades seen for nodes #9 and #11 are just a reflection from the drift observed in #8. These nodes started receiving much more data than expected, what led to changes on their child nodes' frequencies. On the other hand, the right sub-tree of #8 which contains node #18 received almost no data for the same period, making the drift to variable "att_num_8" undetectable by our algorithm. A similar analysis can be made for the second period as well. Node #0, which makes use of the drifting variable "att_num_5", presented maximum drifting grades approximately for the period $[10050, 15400]$. Again, the start of the drift is very precise, while the end of it is a bit inaccurate. Nodes #1, #2 and #4 are also reflections from this drift, since node #0 forwarded almost all the data to it's left sub-tree. Analogously to nodes #9 and #11 in the previous analysis, these nodes

also started receiving unexpected data, leading to heavily altered frequencies of most nodes encompassed by the right sub-tree of #0. Although node #9, which belongs to the left sub-tree, also uses the drifting variable "att_num_5", no variation in it's drifting grades are observed in the heat map. It is an obvious consequence of the data segmentation in #0. This data segmentation also impacted node #18, leading to a small and short drifting behaviour. All in all, it is possible to have a sense of most drifting behaviours from the variables by analyzing how the frequencies of their associated nodes alter along the stream. Nevertheless overlapping drifts can be a challenge, since one node may influence the behaviour of another node in a tree.

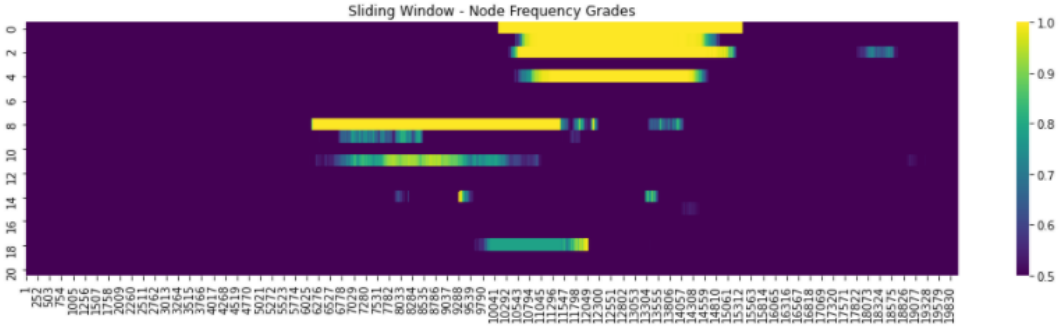

Figure 3.13: Heat map of node frequency grades for the entire tree model along the dataset. The nodes are shown in the y-axis, while the instances processed in the x-axis. The algorithm uses a sliding window of size 2500 for determining the grades.

Fig. 4.2 brings the result of the node accuracy analysis by our algorithm. Note that the drifting variables clearly affect the accuracies of the nodes, as we had also concluded for the plotted trees (see subsection 3.1). In the first test set, nodes #0, #8, #9 and #11 presented extremely high drifting grades for the accuracy. Specifically, for #8, #9 and #11 the drifting period is around the interval $[6100, 11500]$, with nodes #8 and #9 starting slightly later and ending slightly earlier. For node #0 the high grades period maintains, since it's associated variable drifts for the second test set interval. Along with node #0, nodes #1, #2 and #4 also exhibit high drifting grades for their accuracy analysis. The high grades period is very similar for these nodes as well, starting at instance 10050 and ending around 15300. Finally, the last period is the one for which the target variable was perturbed. The perturbation lasts from instance 16000 till 19000. The grades clearly reflect that only the right sub-tree of #0 was affected by the target drift, since nodes #8, #9, #14, #15 and #18 are the ones with high drifting grades. The Node Accuracy Analysis attests the same results (see subsection 3.1.2). Since this drift was performed by substituting the real values from the target with a Bernoulli variable with

"success" ($y = 1$) probability of 0.9, we conclude that the right sub-tree of node #0 was mostly associated with the other value for the target variable (value 0), while the left sub-tree was mainly classified by 1's. Note that again the nodes detect the drift pretty early - around instance 16100 -, but fail to precisely identify the moment the drift ends. They kept the drifting grades till the end of the dataset, or instance 20000.

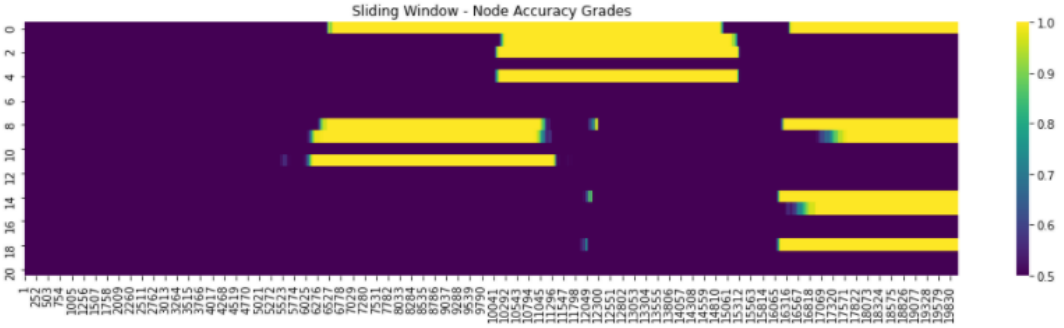

Figure 3.14: Heat map of node accuracy grades for the entire tree model along the dataset. The nodes are shown in the y-axis, while the instances processed in the x-axis. The algorithm uses a sliding window of size 2500 for determining the grades.

Along this section, we presented the *Interpretable Drift Detector*. The algorithm was able to precisely detect the moment the drifts started, assess their severity by the grading mechanism, understand the regions of the tree model most affected by them, and correlate the drifts to their root causes. Although it is capable of providing all these information, it has some limitations, as detailed formerly. First, it presents some difficulty in detecting concurrent drifts to different feature variables. It happens specifically for the case, where one node that makes use of a drifting variable lies in the sub-tree of another node associated with another drifting variable. Secondly, not every feature is represented by the tree model. The model is an abstraction of the feature data, and therefore may be limited to detecting drifts to variables that are part of it or at least correlated to it. Thirdly, it took time for the algorithm to realize that a drift had ended. It happened because the process of reestablishing the standard behaviour of a node is much smoother than the process of adding an abrupt drift to a node. Furthermore, the sliding windows could also be tuned in order to reduce the inaccuracy in detecting the end of a drift as well. Reducing the size of the sliding window would lead to more responsive grades at the cost of adding noisier segments to the grading heat maps. In the next chapter, we investigate this trade-off in an experiment performed in a real-world dataset. Furthermore, we also compare how an adaptation of our algorithm performs in relation to benchmark drift

detectors in terms of drift identification for a set of synthetic datasets vastly used in the literature.

4
Results

As previously presented, our algorithm, *Interpretable Drift Detector*, was designed to correlate drifts happening to different variables along the stream to nodes of a tree model. Thus, once the behaviour of a node starts to deviate from a previous standard behaviour stipulated during the training set, the grades attributed to that node start to rise, elevating the probability of a drift happening to the associated variables. In order to be able to accurately detect most drifts, the algorithm grades a node by it's accuracy and the distribution of it's data subset between it's child nodes, i.e., it's frequency analysis, as we described in chapter 3. That way, in order to compare our algorithm to other benchmark drift detectors, it is essential that we first adapt the *Interpretable Drift Detector* to point drifts at exact instants, as other drift detectors do. The adaptation of our method is done by raising a drift in the first instant the combined mean z-score of all nodes - excluding the leaves, which are not graded - are higher than 3 (3-σ rule) for either one of both analysis: frequency and accuracy. Once the *Interpretable Drift Detector* detects a drift, it enters a drifting zone, which it only exits when the combined mean z-score drops below 2.5, indicating that the drift has indeed ended. Note that, in this adaptation, only the z-scores are used and all nodes are equally weighted, differently then the previous approach where a grading mechanism based on their weight and test statistic was applied (see section 3.2). Nevertheless, these simplifications do not compromise the reliability of our method in terms of drift detection, what is actually measured in this experiment.

In order to assess how our algorithm performs along different datasets in relation to the variety of benchmark methods, we compare the *Interpretable Drift Detector* to the implementations available on *Scikit-Multiflow*: $ADWIN$, DDM, $EDDM$, $HDDM_{A_{test}}$, $HDDM_{W_{test}}$, $KSWIN$ and *Page-Hinkley* (25). These methods are used with the default parameters established by the *Scikit-Multiflow* library. Furthermore, we still add the Linear Four Rates (LFR) algorithm to the list of benchmark methods with the following parametrization: $\sigma_* = 1/100$, $\epsilon_* = 1/100K$ and $\eta_* = 0.99$ (35). The $HLFR$, which is an improvement of the LFR algorithm, is left out of our experiment, since it mostly represents the addition of a second layer permutation test to the LFR. A similar approach could be easily adapted to most algorithms analyzed in this experiment, therefore we opt to only compare single-layer drift detectors. For our method, we define a sliding window with 1000 observations

and train the decision tree with the first 10000 observations of each dataset. The dataset generators are also available on *Scikit-Multiflow*. The datasets generated in our experiment are: SEA, $Sine$, $AGRAWAL$, $RandomRBF$, $Hyperplane$, $Mixed$, $RandomTree$ and $STAGGER$ (25). Each of them is generated with 40000 instances and 5 drifts, which are added to specific intervals: 4 of them are added to the target variable (intervals $[10500, 11500]$, $[14500, 15500]$, $[19500, 20500]$ and $[29500, 30500]$), while 1 of them is added to a randomly selected feature variable (interval $[25000, 26000]$).

Figs. 4.1, 4.2, 4.3 and 4.4 illustrate the performance of the drift detectors along the synthetic data streams generated. Clearly, the default parametrization for DDM and $EDDM$ did not manage to detect the drifts added to the streams. DDM failed in most datasets, not raising any relevant drifts for the first 5 ones (SEA, $Hyperplane$, $RandomRBF$, $AGRAWAL$ and $RandomTree$). It improved a little bit for the last 3 datasets ($Mixed$, $Sine$ and $STAGGER$), but still very far from an acceptable performance. $EDDM$, on the other hand, failed miserably in every dataset. Besides these two algorithms, $Page\text{-}Hinkley$, which also performed badly in most cases, at least identified a few drifts late, what leads us to the opinion that a better parametrization would possibly enable it to detect these late detections correctly, i.e., in the stipulated intervals. $ADWIN$, which is a very well-known algorithm for drift detection, was able to detect some drifts correctly and some others lately. It presented a very poor performance for the $RandomTree$ dataset, and raised a lot of false positives for the last 3 ($Mixed$, $Sine$ and $STAGGER$), although it was able to correctly detect all 4 target drifts in these cases. In the upper half performing algorithms, LFR was very assertive in the first 4 datasets, failing only to detect the feature variable drifts. On the other hand, in the last 4, it performed poorly, detecting only 2 drifts for the $RandomTree$, and raising an enormous amount of false positives for the last 3 ($Mixed$, $Sine$ and $STAGGER$). $KSWIN$ and $HDDM_W$ had very similar performances, except for the first 2 datasets ($Hyperplane$ and SEA), where $HDDM_W$ performed significantly better than $KSWIN$. On $RandomRBF$, $AGRAWAL$ and $RandomTree$, they both managed to detect almost every drift, including the feature variable ones. Nevertheless, they also presented a reasonable number of false positives. Curiously, their performance was stabler in the last 3 ($Mixed$, $Sine$ and $STAGGER$), raising very few false positives then. It is the opposite from what we observed for other benchmark methods, as LFR and $ADWIN$. $HDDM_A$ had a reasonable and stable performance in most cases. It normally detected drifts in the end of the stipulated interval with a few late detections and some miss detections. Furthermore, it managed to keep a very

low amount of false positives. $HDDM_A$ failed to detect feature variable drifts. At last, our methodology performed well in almost every case. It raised almost none false positives, and missed 1 drift for the *RandomTree* and 3 drifts for the $STAGGER$ dataset. Apart from $STAGGER$ where our methodology was overcome by a few other benchmark methods, it is unquestionable that our algorithm performed best for every other dataset analyzed.

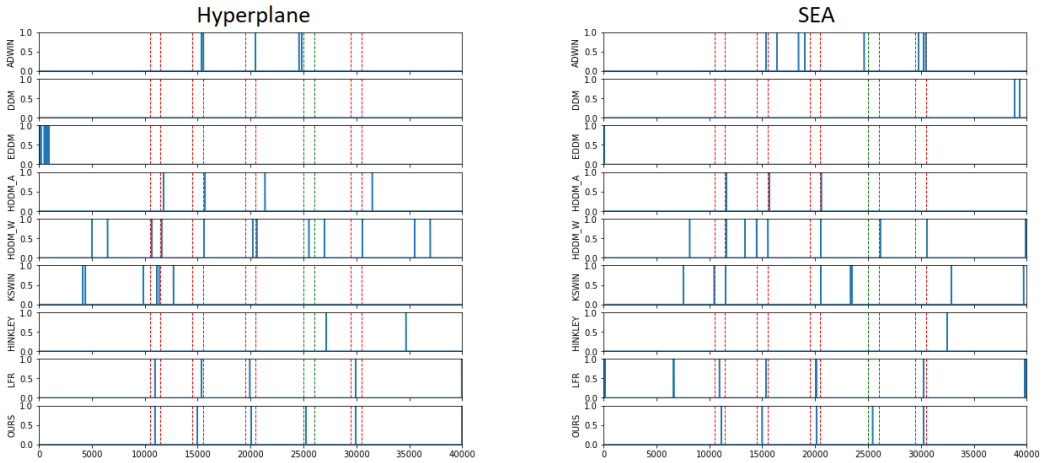

Figure 4.1: Drift identification comparison among drift detection methods along *Hyperplane* and SEA data streams. The red dashed lines indicate the beginning and the end of each target variable drift, and the green dashed lines indicate the feature variable drift. The blue vertical lines correspond to drifts raised by each algorithm.

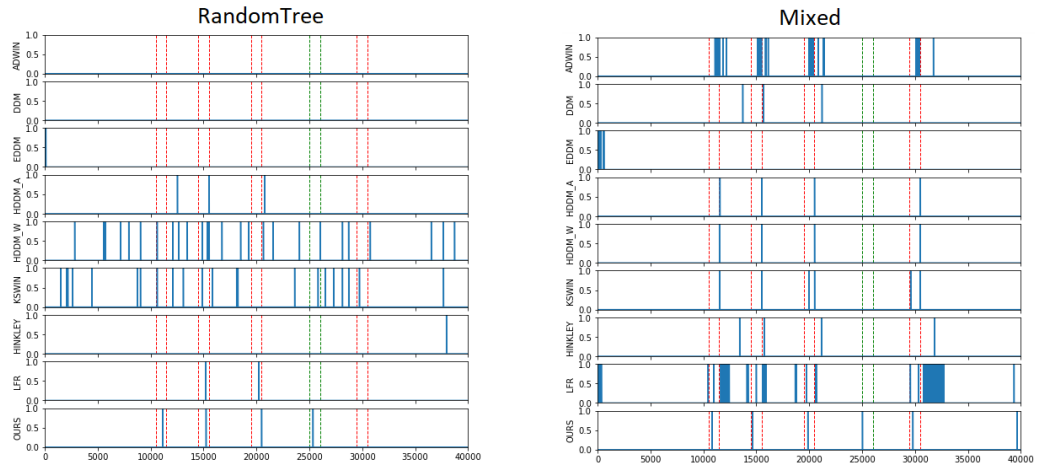

Figure 4.2: Drift identification comparison among drift detection methods along *RandomTree* and *Mixed* data streams. The red dashed lines indicate the beginning and the end of each target variable drift, and the green dashed lines indicate the feature variable drift. The blue vertical lines correspond to drifts raised by each algorithm.

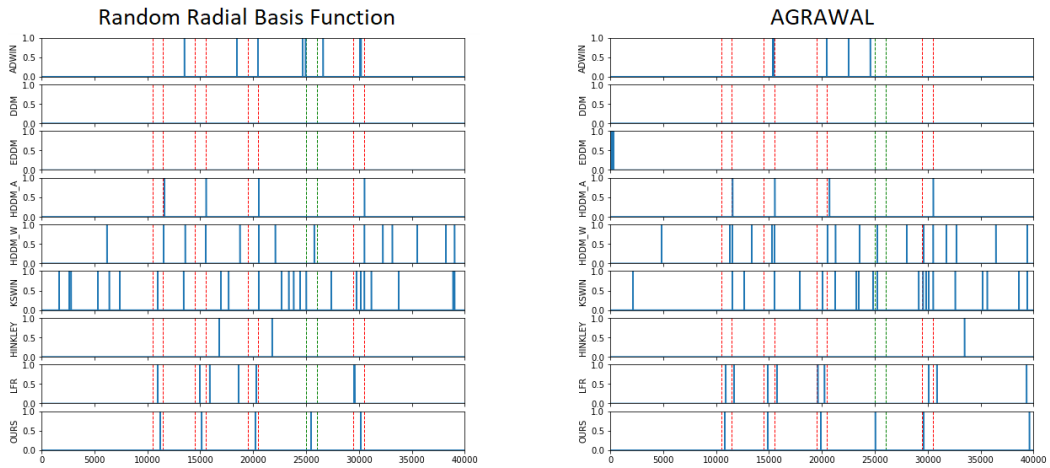

Figure 4.3: Drift identification comparison among drift detection methods along *RandomRBF* and *AGRAWAL* data streams. The red dashed lines indicate the beginning and the end of each target variable drift, and the green dashed lines indicate the feature variable drift. The blue vertical lines correspond to drifts raised by each algorithm.

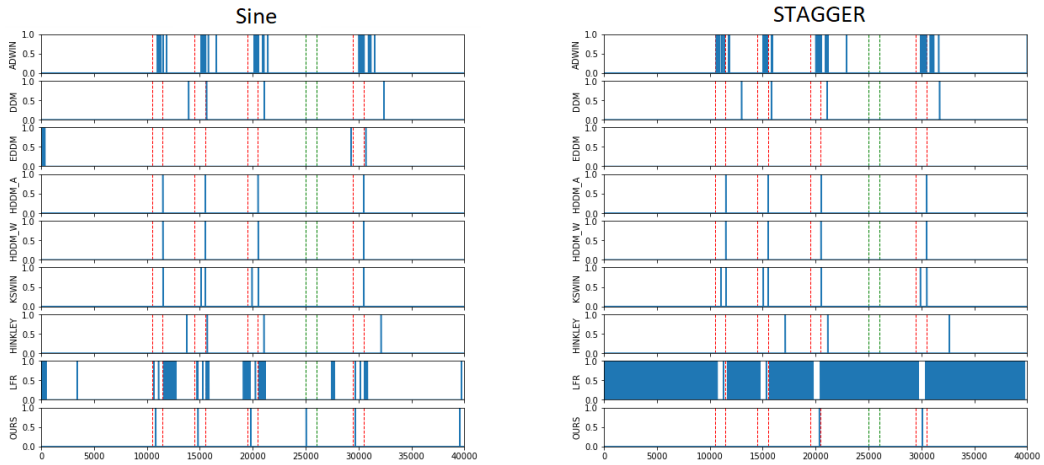

Figure 4.4: Drift identification comparison among drift detection methods along *Sine* and *STAGGER* data streams. The red dashed lines indicate the beginning and the end of each target variable drift, and the green dashed lines indicate the feature variable drift. The blue vertical lines correspond to drifts raised by each algorithm.

Although the results shown in the previous figures (figs. 4.1, 4.2, 4.3 and 4.4) highlight the assertiveness of our algorithm for the designed data streams, it only corresponds to a single execution of the experiment. In order to provide a less biased comparison of the algorithms, we execute the experiment a hundred times, each time changing the seed for generating the data streams for all the different sorts of synthetic data (*SEA*, *Sine*, *AGRAWAL*, *RandomRBF*, *Hyperplane*, *Mixed*, *RandomTree* and *STAGGER*). At each

	AGRAWAL			HYPERPLANE			MIXED			RANDOM_RBF		
	TP	FP	FN	TP	FP	FN	TP	FP	FN	TP	FP	FN
ADWIN	2.46 ± 0.96	3.92 ± 1.82	2.54 ± 0.96	1.74 ± 1.26	2.73 ± 2.14	3.26 ± 1.26	4 ± 0	32.03 ± 3.32	1 ± 0	2.98 ± 0.96	7.35 ± 3.25	2.02 ± 0.96
DDM	0.05 ± 0.22	$\mathbf{0.35 \pm 0.72}$	4.95 ± 0.22	0.01 ± 0.1	$\mathbf{0.14 \pm 0.51}$	4.99 ± 0.1	0 ± 0	3.3 ± 0.94	5 ± 0	0.07 ± 0.26	$\mathbf{0.8 \pm 1.06}$	4.93 ± 0.26
EDDM	0 ± 0	5.72 ± 5.44	5 ± 0	0.01 ± 0.1	4.99 ± 5.73	4.99 ± 0.1	0 ± 0	3.65 ± 4.85	5 ± 0	0 ± 0	4.67 ± 4.75	5 ± 0
HDDM_A	0.03 ± 0.17	3.84 ± 0.61	4.97 ± 0.17	0.01 ± 0.1	2.92 ± 1.37	4.99 ± 0.1	0.02 ± 0.14	3.96 ± 0.2	4.98 ± 0.14	0.01 ± 0.1	3.74 ± 0.54	4.99 ± 0.1
HDDM_W	1.4 ± 0.9	15.57 ± 2.67	3.6 ± 0.9	1.01 ± 1.13	9.34 ± 4.12	3.99 ± 1.13	0.11 ± 0.31	4 ± 0	4.89 ± 0.31	0.62 ± 0.78	6.73 ± 2.49	4.38 ± 0.78
PAGE-HINKLEY	0.12 ± 0.33	2.37 ± 1.14	4.88 ± 0.33	0.2 ± 0.4	2.02 ± 1.07	4.8 ± 0.4	0 ± 0	3.84 ± 0.37	5 ± 0	0.11 ± 0.31	2.26 ± 0.97	4.89 ± 0.31
KSWIN	2.59 ± 1.26	15.13 ± 3.69	2.41 ± 1.26	1.91 ± 1.24	8.49 ± 4.32	3.09 ± 1.24	1.88 ± 1.09	4.92 ± 1.08	3.12 ± 1.09	1.58 ± 1.16	6.22 ± 2.87	3.42 ± 1.16
LFR	4.01 ± 0.27	5.7 ± 1.4	0.99 ± 0.27	3.82 ± 0.52	0.92 ± 1.35	1.18 ± 0.52	4.17 ± 0.38	147.85 ± 107.09	0.83 ± 0.38	3.97 ± 0.22	7.13 ± 9.63	1.03 ± 0.22
OURS	$\mathbf{4.39 \pm 0.83}$	2.24 ± 3.05	$\mathbf{0.61 \pm 0.83}$	$\mathbf{4.69 \pm 0.61}$	0.37 ± 0.93	$\mathbf{0.31 \pm 0.61}$	$\mathbf{4.4 \pm 0.91}$	$\mathbf{3.16 \pm 4.27}$	$\mathbf{0.6 \pm 0.91}$	$\mathbf{4.54 \pm 0.56}$	1.05 ± 1.6	$\mathbf{0.46 \pm 0.56}$
	STAGGER			SINE			SEA			RANDOMTREE		
	TP	FP	FN	TP	FP	FN	TP	FP	FN	TP	FP	FN
ADWIN	4.01 ± 0.1	41.62 ± 3.57	0.99 ± 0.1	4.03 ± 0.17	36.31 ± 2.36	0.97 ± 0.17	2.22 ± 0.93	3.2 ± 1.68	2.78 ± 0.93	1.62 ± 1.44	2.91 ± 3.37	3.38 ± 1.44
DDM	0 ± 0	4 ± 0	5 ± 0	0.04 ± 0.2	3.69 ± 0.68	4.96 ± 0.2	0.02 ± 0.14	$\mathbf{0.36 \pm 0.7}$	4.98 ± 0.14	0.03 ± 0.17	$\mathbf{0.31 \pm 0.71}$	4.97 ± 0.17
EDDM	0 ± 0	$\mathbf{0 \pm 0}$	5 ± 0	0.22 ± 0.42	9.4 ± 6.01	4.78 ± 0.42	0 ± 0	4.31 ± 5.58	5 ± 0	0 ± 0	7.15 ± 7.54	5 ± 0
HDDM_A	0.02 ± 0.14	3.93 ± 0.26	4.98 ± 0.14	0.01 ± 0.1	3.98 ± 0.14	4.99 ± 0.1	0.01 ± 0.1	3.48 ± 0.67	4.99 ± 0.1	0.03 ± 0.17	2.92 ± 1.39	4.97 ± 0.17
HDDM_W	0.16 ± 0.44	3.99 ± 0.1	4.84 ± 0.44	0.03 ± 0.17	4.01 ± 0.1	4.97 ± 0.17	0.77 ± 0.79	7.95 ± 2.11	4.23 ± 0.79	1.5 ± 1.14	16.43 ± 6.35	3.5 ± 1.14
PAGE-HINKLEY	0 ± 0	3.01 ± 0.1	5 ± 0	0 ± 0	3.91 ± 0.29	5 ± 0	0.2 ± 0.4	1.8 ± 1.01	4.8 ± 0.4	0.12 ± 0.33	2.18 ± 1.21	4.88 ± 0.33
KSWIN	1.3 ± 1.1	4.2 ± 0.49	3.7 ± 1.1	1.83 ± 1	4.48 ± 0.72	3.17 ± 1	1.75 ± 1.04	6.37 ± 2.68	3.25 ± 1.04	2.35 ± 1.13	15.68 ± 6.58	2.65 ± 1.13
LFR	$\mathbf{5 \pm 0}$	985.6 ± 15.34	$\mathbf{0 \pm 0}$	4.15 ± 0.36	187.91 ± 85.91	0.85 ± 0.36	3.91 ± 0.29	6.65 ± 5.71	1.09 ± 0.29	3.74 ± 0.69	4.44 ± 15.07	1.26 ± 0.69
OURS	2.51 ± 1.94	1.36 ± 1.93	2.49 ± 1.94	$\mathbf{4.95 \pm 0.22}$	$\mathbf{3.16 \pm 4.96}$	$\mathbf{0.05 \pm 0.22}$	$\mathbf{4.74 \pm 0.46}$	0.61 ± 1.48	$\mathbf{0.26 \pm 0.46}$	$\mathbf{4.02 \pm 1.23}$	1.2 ± 3.1	$\mathbf{0.98 \pm 1.23}$

Table 4.1: Results of a hundred executions of the experiment. All data sets have 40000 instances and the drifts are set always on the same intervals. For each method and data set, we monitor three distinct metrics: True Positive (TP), False Positive (FP) and False Negative (FN). The best results in each one are marked in bold.

round, we calculate three distinct parameters: the true positive rates (TP), which indicates the drifts an algorithm correctly identifies; the false positive rates (FP), which indicates the number of times an algorithm raises a false drift detection, and the false negative rate (FN), which represents the number of drifts an algorithm missed. After the completion of all the rounds, we estimate the mean and standard deviation for the three metrics. Table 4.1 shows the results from all the hundred executions of the experiment. For each metric in each dataset, the best results are marked in bold. Nevertheless, the best algorithm for each dataset is defined by combining both metrics: TP and FP, since one wants it's detector to identify the most number of drifts with the least number of false positives. Based on this trade-off, we can state that the *Interpretable Drift Detector* performs best for all datasets in the experiment, except for $STAGGER$. For this one, the answer is a bit more subjective, since there are algorithms capable of identifying more drifts at the cost of a lot more false positives. However, if we consider the ratio TP/FP to base the decision, it would still indicate our method as the best one for this dataset as well.

In order to evaluate how our algorithm performs on real-world datasets, we conduct another experiment using the Electricity dataset (18). This dataset is a time-series, collected from the Australian New South Wales Electricity Market, which contains variables related to demand, transfer and price for electricity from two neighboring states: New South Wales and Victoria. The class label measures whether the price for electricity in New South Wales increased (UP) or decreased (DOWN) in relation to a moving average of the last 24 hours. The whole dataset contains 45312 and each observation corresponds to a period of 30 minutes. The time related variables are left out of the model, since they are not related to drifts and would only add noise to

our analysis.

This experiment starts by training a simple decision tree with maximum depth of 2 for the first 10000 observations. The resulting decision tree can be visualized in fig. 4.5. Note that it only uses two features from the dataset: "nswdemand" (node #0) and "nswprice" (nodes #1 and #4). Based on the tree model, we run the *Interpretable Drift Detector* three times, varying the size of the sliding window parameter (5000, 2500 and 1000). We then compare the results and choose the most appropriate size for the sliding window. At last, we verify if the results could be explained by inspecting the time-series data of the feature variables.

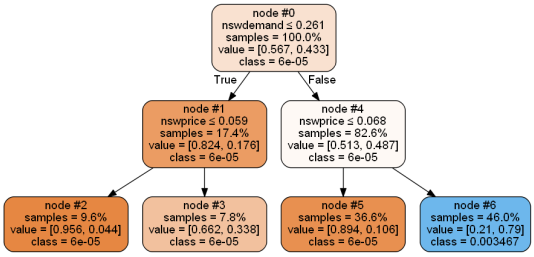

Figure 4.5: Decision tree trained with the first 10000 observations of the Electricity dataset. Note that only the variables "nswdemand" and "nswprice" are used.

The results from the execution of our algorithm to the Electricity dataset are illustrated on figs. 4.6 and 4.7. Fig. 4.6 shows how the tree nodes' frequencies behave along the stream (Node Frequency Analysis) when we run the algorithm with three specific values for the sliding window W (5000, 2500 and 1000). It becomes evident that there are two major drifts on the data: one on node #1 starting around the interval $[14000, 18000]$ and lasting till the end of the stream, and another one, on node #4 starting around $[18000, 23000]$ till $[36000, 38000]$. There are other minor drifts, specially for the middle and bottom charts from fig. 4.6 ($W = 2500$ and $W = 1000$ respectively), but they seem to be a bit irrelevant and could actually be attributed to noisier segments of the data. Fig. 4.7 brings the results of the Node Accuracy analysis, also performed by our algorithm. By comparing these heat maps, it becomes clear that the grades in node #0 are actually a reflection from a drift in #4, which lasts from $[19000, 23000]$ till $[35000, 38000]$. From observing both analysis, we can state that the noisiness reflected in the grading heat maps are tightly coupled with the size of the sliding window. At first sight, it seems for us that the execution of our algorithm with $W = 5000$ is the one that suits the data best. It brings a cleaner visualization of the grading heat map with more homogeneous drifting segments. On the other hand, larger sliding windows tend to be less responsive to changes in the data, and hence their drifting

periods tend to be larger, becoming less accurate than smaller choices for the sliding window. Tuning the sliding parameter is a crucial activity for the best performance of our algorithm, since it controls the trade-off between accuracy and noisiness in the resulting heat maps.

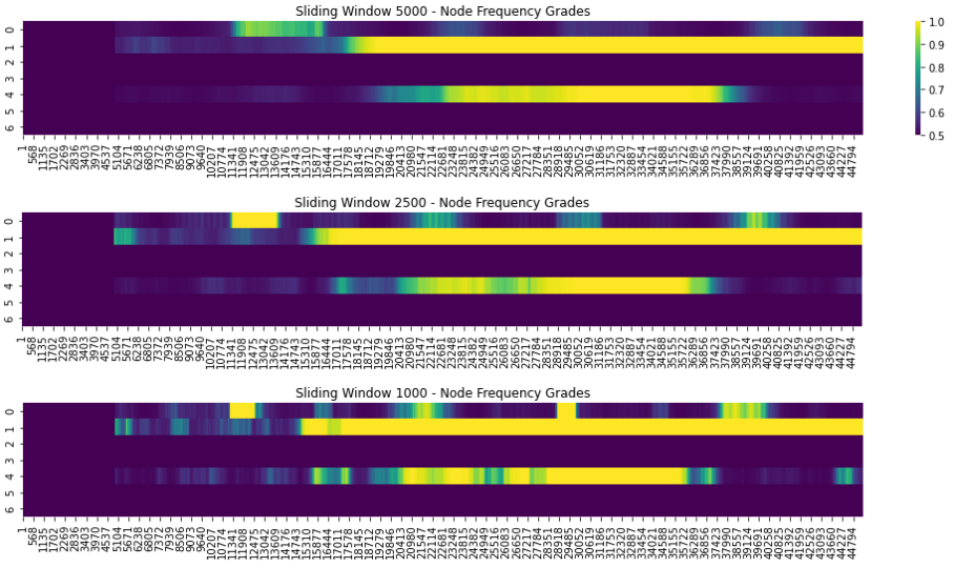

Figure 4.6: The grading heat maps from the node frequency analysis of the *Interpretable Drift Detector* with 3 specific values for the sliding window size, W. The top chart represents the execution of the algorithm with $W = 5000$; the middle one the execution where $W = 2500$, and for the bottom chart, the algorithm is run with $W = 1000$.

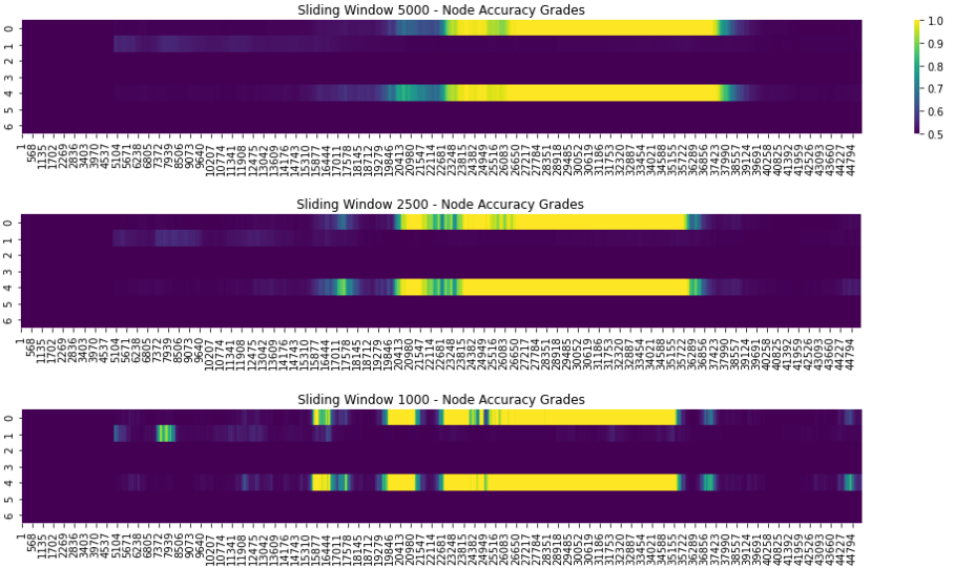

Figure 4.7: The grading heat maps from the node accuracy analysis of the *Interpretable Drift Detector* with 3 specific values for the sliding window size, W. The top chart represents the execution of the algorithm with $W = 5000$; the middle one the execution where $W = 2500$, and for the bottom chart, the algorithm is run with $W = 1000$.

Fig. 4.9 illustrates the 48 observations moving averages for the feature data of every node that segments the dataset in the tree model (#0, #1 and #4). The horizontal line represents the rule set for the node (magenta), while the vertical lines represent the separation of the training set from the test set (black), and the start (green) and end of a drift (red). We consider that a drift starts for a node once its grades are higher than 0.95 and it ends when it gets smaller than 0.70. We only evaluate the execution of the *Interpretable Drift Detector* with $W = 5000$. It is possible to correlate the drifts detected for nodes #1 and #4 (middle and bottom charts respectively) to changes in the behaviour of the feature data. Specifically for node #1, it is clear that the rule learned during the training set is not appropriate for the test set, since the moving average of the feature data lies almost entirely under the horizontal line after the black vertical line. The algorithm detected this drift on instance 18357 and it last till the end of the stream. For node #4, the drift is less obvious, nevertheless it is possible to observe a change in the pattern of the feature data for the interval around $[20000, 36000]$. The algorithm detects the drift on instance 23647 and reports it's end on instance 38453. Although the detections are a bit late, the drifts are very stable and no false positive detections are found. That way, running the algorithm with a sliding window of size equal to 5000 is actually a good choice.

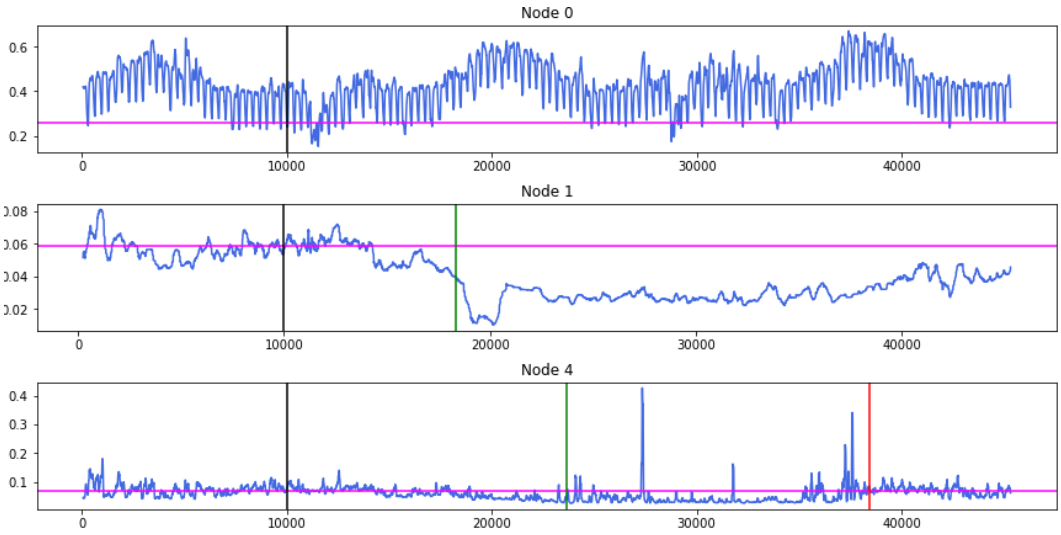

Figure 4.8: Each chart represents a moving average with 48 observations of the feature data entering the corresponding node along the stream. The vertical lines denote the end of the training set (black), the start of a drift (green) and the end of a drift (red), which are estimated based on the grades resulting from the node frequency analysis of the *Interpretable Drift Detector*.

Fig. 4.9 illustrates correct predictions (attributed the value 0) and the wrong predictions (attributed the value 1) through a heat map. The yellow

color represents the correct predictions, while the purple represents the wrong ones. There are also three distinct vertical lines: the black line, which denotes the separation between training and test set; the green line, which shows the start of a drift, and the red line, which represents it's end. By observing the changes between the density of both colors (yellow and purple) along the stream, it is possible to distinguish drifting zones from non drifting zones. We only need to project the training set pattern onto the test set and observe if they actually match. It is however very difficult to precisely determine the starting point of a drift by that analogy. The *Interpretable Drift Detector* identifies a drift starting at instance 23207 and ending at 38141, which are very similar to the period determined by the node frequency analysis for node #4, what leads us to the conclusion that this behavioral change in the feature variable "nswprice" deeply affected the accuracy of the tree as well.

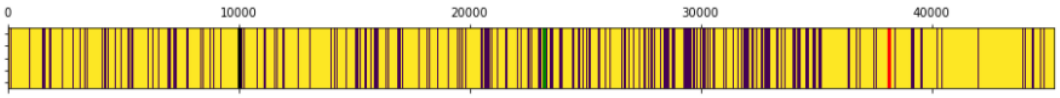

Figure 4.9: Heat map shows the correct (yellow) and wrong (purple) predictions of the tree model for the Electricity dataset. There are other 3 vertical lines that denote the separation of the training and test set (in black), the start of a drift (in green) and the end of a drift (in red).

This experiment brings a real-world scenario for which the application of the *Interpretable Drift Detector* reveals interesting facts regarding the target variables and how they affected the predictions of the tree model. By inspecting figs. 4.5 and 4.6, we observed that nodes #1 and #4 suffered severe frequency drifts, but specially the one suffered by node #4 impacted the accuracy of the entire tree model. Then, we were able to confirm these points by observing the moving average of the feature variable "nswprice", segmented by the nodes for which it is used (middle and bottom charts of fig. 4.7). By comparing the moving average before and after the black vertical line, it becomes evident that the distribution of the variable "nswprice" suffered changes along the stream. So, the algorithm was able to return the instant the drift started, the affected regions of the model, the severity of the drift and after all the root causes of the drift as well. It proves the effectiveness of adopting the *Interpretable Drift Detector* for data stream mining.

5
Conclusion

Tree-based models are the utmost technique for acquiring interpretability on how feature and target variables are related. Ultimately, these models have been largely used in the academia with the focus of enhancing interpretability of other complex models. Thibaut et al. (33), for instance, developed an algorithm capable of approximating tree ensembles by a single tree, Zhang et al. (40) proposed to interpret the predictions of a CNN model through a decision tree, and many other researchers are developing great works centered on interpretability and tree-based models. In other words, Interpretable Machine Learning has been gaining a lot of attention from the academia, and our research relates to that topic in the sense that it leverages tree models in order to enhance interpretability of a fundamental topic in data stream mining: concept drifts.

Concept Drift Interpretability is still an unexplored topic. Although there are methods capable of providing a lot of information regarding a drift - as it is the case for most "Data Distribution-based Drift Detectors" -, most methods are actually only concerned about identifying the moment the drifts occur, taking for granted all the extra information relevant to properly deal with them. Our research introduces a new method, the *Interpretable Drift Detector*, which aims at providing answers to a set of questions regarding a drift: "When" (the moment the drift happened); "How" (the severity of the drift); "Where" (the parts of the model affected by the drift), and "Why" (the root causes of the drift). The first three questions are first introduced by the work of Gama et al. (24), while the last one is a step forward into diagnosing the drift, i.e., identifying the main variables responsible for it.

We compare our method to benchmark drift detectors, available on Scikit-Multiflow library with their default parametrization (25). The methods are compared on eight synthetic datasets generated with five distinct drifts: four, added to the target variable, and one to a randomly selected feature variable (see chapter 4). The results of this experiment clearly shows that our method outperforms benchmark detectors in terms of false-positive and true-positive rates for the majority of the datasets analyzed. Moreover, another experiment conducted on a real-world dataset presents how the different sorts of drifts detected by our method correlates to a variable of the dataset, therefore providing the proper understanding of what is actually happening in the data that led to the drift. Besides all these advantages, our method also

has its drawbacks. First, it is computationally more expensive than the other methods, since it navigates through each node of the tree in order to compare the frequency and accuracy metrics of the current sliding window to these same metrics calculated during the training set. Another drawback is that it is only capable of detecting drifts to variables that are selected by the tree model, or at least correlated to the selected ones.

Identifying drifts is a valuable information, however deciding to retrain a model based on that information alone has many shortcomings. Since the root causes are not known, a retraining scheme may actually lead to a worse model after a short period of time. Moreover, knowing a model's most affected regions and the intensity of the drift are also relevant information in order to provide an appropriate retraining scheme for the tree-based model. Besides developing better models, we strongly believe that a thorougher analysis on drifts can considerably benefit a lot of manufacturers. It exposes possible problems in their data, new behaviours for the feature and target variables, and enables the correlation between the real-world scenario and the one translated by the data. A drift may actually raise attention to problems occurring in the manufacturer's processes and therefore require a better investigation on their real-world scenario. The *Interpretable Drift Detector* is a step forward into conducting a deeper analysis on concept drifts. We aggregate distinct concepts inside the algorithm, i.e., *Concept Drift Identification*, *Concept Drift Understanding* and our main goal, *Concept Drift Interpretability*.

As future works, we intend to further exploit the visual representation of concepts drifts and develop effective mechanisms of retraining a tree-based model, leveraging all the extra information provided by the algorithm presented by this dissertation, the *Interpretable Drift Detector*.

6
References

[1] BAENA-GARCIA, M.; DEL CAMPO-ÁVILA, J.; FIDALGO, R.; BIFET, A.; GAVALDA, R. ; MORALES-BUENO, R.. **Early drift detection method**. In: FOURTH INTERNATIONAL WORKSHOP ON KNOWLEDGE DISCOVERY FROM DATA STREAMS, volumen 6, p. 77–86, 2006. 2.1.1

[2] BARROS, R. S.; CABRAL, D. R.; GONÇALVES JR, P. M. ; SANTOS, S. G.. **Rddm: Reactive drift detection method**. Expert Systems with Applications, 90:344–355, 2017. 2.1.1

[3] BIFET, A.; GAVALDÀ, R.. **Learning from time-changing data with adaptive windowing**. volumen 7, 04 2007. 2.1.1

[4] BRZEZIŃSKI, D.; STEFANOWSKI, J.. **Accuracy updated ensemble for data streams with concept drift**. In: HYBRID ARTIFICIAL INTELLIGENT SYSTEMS, p. 155–163. Springer Berlin Heidelberg, 2011. 1.2

[5] DASU, T.; KRISHNAN, S.; VENKATASUBRAMANIAN, S. ; YI, K.. **An information-theoretic approach to detecting changes in multi-dimensional data streams**. In: IN PROC. SYMP. ON THE INTERFACE OF STATISTICS, COMPUTING SCIENCE, AND APPLICATIONS. Citeseer, 2006. 2.1.2

[6] DITZLER, G.; ROVERI, M.; ALIPPI, C. ; POLIKAR, R.. **Learning in non-stationary environments: A survey**. IEEE Computational Intelligence Magazine, 10(4):12–25, 2015. 1.1

[7] DITZLER, G.; POLIKAR, R.. **Hellinger distance based drift detection for nonstationary environments**. In: 2011 IEEE SYMPOSIUM ON COMPUTATIONAL INTELLIGENCE IN DYNAMIC AND UNCERTAIN ENVIRONMENTS (CIDUE), p. 41–48. IEEE, 2011. 2.1.2

[8] DOMINGOS, P.; HULTEN, G.. **Mining high-speed data streams**. In: PROCEEDINGS OF THE SIXTH ACM SIGKDD INTERNATIONAL CONFERENCE ON KNOWLEDGE DISCOVERY AND DATA MINING, p. 71–80, 2000. 1.2

[9] DOS REIS, D. M.; FLACH, P.; MATWIN, S. ; BATISTA, G.. **Fast unsupervised online drift detection using incremental kolmogorov-**

smirnov test. In: PROCEEDINGS OF THE 22ND ACM SIGKDD INTERNATIONAL CONFERENCE ON KNOWLEDGE DISCOVERY AND DATA MINING, p. 1545–1554, 2016. 2.1.2

[10] DUDA, R. O.; HART, P. E. ; STORK, D. G.. **Pattern classification. john wiley & sons**. Inc., New York, 2, 2001. 1.1

[11] ELWELL, R.; POLIKAR, R.. **Incremental learning of concept drift in nonstationary environments**. IEEE Transactions on Neural Networks, 22(10):1517–1531, 2011. 1.2

[12] FRÍAS-BLANCO, I.; D. CAMPO-ÁVILA, J.; RAMOS-JIMÉNEZ, G.; MORALES-BUENO, R.; ORTIZ-DÍAZ, A. ; CABALLERO-MOTA, Y.. **Online and non-parametric drift detection methods based on hoeffding's bounds**. IEEE Transactions on Knowledge and Data Engineering, 27(3):810–823, 2015. 2.1.1

[13] GAMA, J.; ŽLIOBAITĖ, I.; BIFET, A.; PECHENIZKIY, M. ; BOUCHACHIA, A.. **A survey on concept drift adaptation**. ACM computing surveys (CSUR), 46(4):1–37, 2014. 1.2, 2

[14] GAMA, J.; MEDAS, P.; CASTILLO, G. ; RODRIGUES, P.. **Learning with drift detection**. In: BRAZILIAN SYMPOSIUM ON ARTIFICIAL INTELLIGENCE, p. 286–295. Springer, 2004. 2.1.1

[15] GOLDENBERG, I.; WEBB, G. I.. **Survey of distance measures for quantifying concept drift and shift in numeric data**. Knowledge and Information Systems, p. 1–25. 2.1.2

[17] HAO, S.; HU, P.; ZHAO, P.; HOI, S. C. ; MIAO, C.. **Online active learning with expert advice**. ACM Transactions on Knowledge Discovery from Data (TKDD), 12(5):1–22, 2018. 1.2

[18] HARRIES, M.; WALES, N. S.. **Splice-2 comparative evaluation: Electricity pricing**. 1999. 4

[19] HULTEN, G.; SPENCER, L. ; DOMINGOS, P.. **Mining time-changing data streams**. In: PROCEEDINGS OF THE SEVENTH ACM SIGKDD INTERNATIONAL CONFERENCE ON KNOWLEDGE DISCOVERY AND DATA MINING, p. 97–106, 2001. 1.2

[20] KHAMASSI, I.; SAYED-MOUCHAWEH, M.; HAMMAMI, M. ; GHÉDIRA, K.. **Discussion and review on evolving data streams and concept drift adapting**. Evolving Systems, 2018. (document), 1.1, 3, 1.1, 1.2

[21] KOTLER, J.; MALOOF, M.. **Dynamic weighted majority: A new ensemble method for tracking concept drift**. In: IEEE INTERNATIONAL CONFERENCE ON DATA MINING, p. 123–130, 2003. 1.2

[22] LIU, A.; ZHANG, G. ; LU, J.. **Fuzzy time windowing for gradual concept drift adaptation**. In: 2017 IEEE INTERNATIONAL CONFERENCE ON FUZZY SYSTEMS (FUZZ-IEEE), p. 1–6. IEEE, 2017. 2.1.1

[23] LU, N.; ZHANG, G. ; LU, J.. **Concept drift detection via competence models**. Artificial Intelligence, 209:11–28, 2014. 1.1

[24] LU, J.; LIU, A.; DONG, F.; GU, F.; GAMA, J. ; ZHANG, G.. **Learning under concept drift: A review**. IEEE Transactions on Knowledge and Data Engineering, 31(12):2346–2363, 2018. (document), 1.1, 1.2, 2.1, 2.1, 3, 5

[25] MONTIEL, J.; READ, J.; BIFET, A. ; ABDESSALEM, T.. **Scikit-multiflow: A multi-output streaming framework**. Journal of Machine Learning Research, 19(72):1–5, 2018. 4, 5

[26] PRICE, D. O.. **Sequential Analysis. By Abraham Wald. New York: John Wiley and Sons, Inc., 1947.** Social Forces, 27(2):170–171, 12 1948. 2

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

[28] QAHTAN, A. A.; ALHARBI, B.; WANG, S. ; ZHANG, X.. **A pca-based change detection framework for multidimensional data streams: Change detection in multidimensional data streams**. In: PROCEEDINGS OF THE 21TH ACM SIGKDD INTERNATIONAL CONFERENCE ON KNOWLEDGE DISCOVERY AND DATA MINING, p. 935–944, 2015. 2.1.2

[29] REINSEL, D.; GANTZ, J. ; RYDNING, J.. **Aim@shape shape repository**. `https://www.seagate.com/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf`, 2018. Accessed: 2020-10-08. 1

[30] ROBERTS, S.. **Control chart tests based on geometric moving averages**. Technometrics, 42(1):97–101, 2000. 2.1.1

[31] ROSS, G. J.; ADAMS, N. M.; TASOULIS, D. K. ; HAND, D. J.. **Exponentially weighted moving average charts for detecting concept drift**. Pattern Recogn. Lett., 33(2):191–198, Jan. 2012. 2.1.1

[32] STREET, W. N.; KIM, Y.. **A streaming ensemble algorithm (sea) for large-scale classification**. In: PROCEEDINGS OF THE SEVENTH ACM SIGKDD INTERNATIONAL CONFERENCE ON KNOWLEDGE DISCOVERY AND DATA MINING, p. 377–382. Association for Computing Machinery, 2001. 1.2

[33] VIDAL, T.; SCHIFFER, M.. **Born-again tree ensembles**. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING, p. 9743–9753. PMLR, 2020. 5

[34] WANG, H.; FAN, W.; YU, P. S. ; HAN, J.. **Mining concept-drifting data streams using ensemble classifiers**. In: PROCEEDINGS OF THE NINTH ACM SIGKDD INTERNATIONAL CONFERENCE ON KNOWLEDGE DISCOVERY AND DATA MINING, p. 226–235. Association for Computing Machinery, 2003. 1.2

[35] WANG, H.; ABRAHAM, Z.. **Concept drift detection for streaming data**. In: 2015 INTERNATIONAL JOINT CONFERENCE ON NEURAL NETWORKS (IJCNN), p. 1–9. IEEE, 2015. 2.1.3, 4

[36] WANG, S.; MINKU, L. L.; GHEZZI, D.; CALTABIANO, D.; TINO, P. ; YAO, X.. **Concept drift detection for online class imbalance learning**. In: THE 2013 INTERNATIONAL JOINT CONFERENCE ON NEURAL NETWORKS (IJCNN), p. 1–10. IEEE, 2013. 2.1.1

[37] WARES, S.; ISAACS, J. ; ELYAN, E.. **Data stream mining: methods and challenges for handling concept drift**. SN Applied Sciences, 2019. 1, 1.2, 2.1.1

[38] YU, S.; ABRAHAM, Z.. **Concept Drift Detection with Hierarchical Hypothesis Testing**, p. 768–776. 06 2017. 2.1.3

[39] YU, S.; WANG, X. ; PRINCIPE, J. C.. **Request-and-reverify: Hierarchical hypothesis testing for concept drift detection with expensive labels**. arXiv preprint arXiv:1806.10131, 2018. 2.1.3

[40] ZHANG, Q.; YANG, Y.; MA, H. ; WU, Y. N.. **Interpreting cnns via decision trees**. In: PROCEEDINGS OF THE IEEE/CVF CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, p. 6261–6270, 2019. 5

[41] ZHENG, S.; VAN DER ZON, S. B.; PECHENIZKIY, M.; DE CAMPOS, C. P.; VAN IPENBURG, W.; DE HARDER, H. ; NEDERLAND, R.. **Labelless concept drift detection and explanation**. In: NEURIPS 2019 WORKSHOP

ON ROBUST AI IN FINANCIAL SERVICES: DATA, FAIRNESS, EXPLAINABILITY, TRUSTWORTHINESS, AND PRIVACY, 2019. 1.1