POLITECNICO DI MILANO
Master Degree in Computer Science Engineering
School of Industrial and information engineering

INCREMENTAL QUANTTREE

Relatore: Prof. Giacomo Boracchi

Correlatore: Luca Frittoli

Candidato:
Daniele Lunghi

Anno Accademico 2019-2020

Ai miei genitori

Ringraziamenti

Non capita molte volte di laurearsi, e vorrei approfittare dell'occasione per ringraziare una serie di persone, che più di tutte mi hanno aiutato durante questo mio percorso.

Innanzi tutto vorrei ringraziare Giacomo per avermi dato l'opportunità di lavorare a un progetto che mi ha dato molte soddisfazioni, per avermi incoraggiato a continurare con la ricerca e per i mille consigli dati durante la stesura della tesi. Ringrazio Luca per i suggerimenti e l'aiuto datomi, senza di lui questa tesi sarebbe decisamente peggiore.

Vorrei ringraziare Erika, Andrea, Seba e Ilan per essere stati miei amici dall'inizio alla fine, ciò che è importante dura a lungo, Un grazie a Giulia, come amica e per avermi dato suggermimeti e aiuto per la tesi. Vale doppio. Grazie a Emanuele, per anni di consigli e piani sul percorso universitario, pare abbiano funzionato. E soprattutto grazie a Sara. Sono una persona migliore da che ti conosco, e gran parte dei traguardi che ho raggiunto negli ultimi anni portano anche tua sua firma. Grazie di essermi vicina. Sono felice di averti incontrata.

Grazie ai miei nonni, tutti e quattro. Ringrazio nonna Egle, che penso più di tutti avrebbe voluto vedere questo momento, nonno Roberto, che è un po' l'ingegnere che vorrei diventare, nonna Vanna, che è capace di rendermi sempre di buon umore e nonno Mario, che avrei voluto conoscere meglio.

Grazie mamma, per essere un esempio di vita e la persona che più ammiro.
Infine più di tutti vorrei ringraziare Anna. Le parole non rendono sempre l'idea, ma sono davvero molto fortunato ad essere tuo fratello. Ti voglio bene.

Abstract

Change Detection on data streams, defined as the problem of monitoring a stream of data to detect whether a distribution change has occurred, is a relevant topic in machine learning, with applications ranging from wildfires monitoring to quality control. In this thesis we address the problem by proposing a novel non parametric algorithm for change detection on data streams.

We begin by analyzing literature on this problem, discussing the change detection algorithms proposed over the years, with a focus on those capable of monitoring multivariate data streams. We then start from QuantTree, a non parametric, histograms-based algorithm for multivariate data streams to build our contributions to the problem, consisting of a novel change detection algorithm, Incremental QuantTree (IQT). IQT requires a very small training set to begin monitoring a data stream, continuously increasing its detection accuracy over time. Furthermore it can be combined with of an Exponentially Weighted Moving Average chart, which controls the false alarms of the algorithm over time, detecting if a change occurred.

Our experiments show that Incremental QuantTree is able to obtain a performance comparable to that of QuantTree when having access to a much smaller training set, and is also in line with Hotelling - Change Point Methods.

Abstract

La rilevazione dei cambiamenti (Change detection) su flussi di dati, fefinito come il problema di monitorare un flusso di dati per stabilire se è avenuto un cambiamento nella distribuzione che genera quei dati, è un problema rilevante di Machine Learning, con applicazioni in campi come il monitoraggio degli incendi, il controllo del traffico e il controllo di qualità. In questa tesi proponiamo un nuovo algoritmo non parametrico per Change Detection su stream di dati.

Prima di tutto analizziamo la letteratura sul tema, discutendo le tecniche principali proposte nel corso degli anni. Partendo poi da QuantTree, un algoritmo non parametrico per change detection basato su istogrammi, abbiamo creato un nuovo algoritmo per fare change detection, Incremental QuantTree (IQT). IQT è un algoritmo in grado di iniziare a monitorare un flusso di dati partendo da un training set di dimensioni estremamente ridotte, migliorando progressivamente la propria capacità di individuare cambiamenti. Inoltre può essere esteso con un controllo con un diagramma di Exponentially Weighted Moving Average, che controlla i falsi allarmi dell'algoritmo nel tempo, individuando in questo modo dei cambiamenti.

I nostri esperimenti mostrano che Incremental QuantTree è in grado di ottenere una performance comparabile a QuantTree, avendo accesso a un training set molto più ridotto. Inoltre ha una performance in linea con il Change Point Method che fa uso della statistica Hotelling.

List of Figures

Contents

Chapter 1

Introduction

Change Detection is a relevant topic in machine learning, and it can be defined as the problem of identifying differences in the state of an object or phenomenon by observing it at different times [1]. We focus on change detection of data stream, which can be formulated as determining as quickly as possible if a monitored data stream encounters a distribution change [2]. The underlying assumption is that data can be modeled as generated by a stationary distribution ϕ_0 until a certain time τ, typically referred to as the change point, and as generate by a different distribution ϕ_1 afterwards.

In supervised learning change detection is closely related to the problem of concept drift detection, i.e. determining whether the statistical properties of an observed process vary over time, possibly reducing the performance of learners trained on past data [3]. Concept drift is a common problem in online machine learning, concerning applications such as fraud detection [4] [5], financial time series analysis [6] and predictive maintenance [7].

In recent years an increasing number of applications rely on data streams monitoring, some examples being wildfires monitoring [8], security [9] and quality control [10]. In doing so it is important to properly control the False Alarms Rate, because any detection might trigger possibly time-consuming and costly interventions, including the re-training of the change-detection algorithm.

Usually, change detection must be done in a semi-supervised manner. In fact it is common to have access to a relatively small training set TR_{small} of points from ϕ_0, while having access to data drawn from the post change distribution is often impossible [11]. Another challenge is that the potentially infinite lenght of the data stream does not allow to store the observations and process them all together. Moreover, changes in data streams must be found at run time, which is particularly important for high frequency streams.

A large number of change detection techniques in the literature are parametric and require the distribution ϕ_0 to belong to a known family, which is not always the case. Non parametric techniques based on ranks [12] or on the cumulative function [13] are not applicable to multivariate data. Kernel methods [14] can process multivariate data and control the False Alarms , but require a large training set. This, depending on the situation, can become a severe limitation when the size of the available training set is not large enough.

Histograms are non parametric models, often used to describe and compare multivariate probability distributions. However, they are often implemented over regular grids and require a number of bins that grows exponentially with the data dimension. A histogram-based methods that allows to control the number of bins for highly multi-dimensional data is QuantTree, which builds its histogram via binary splits on a single covariate, where the cutting points are defined by the quantiles of the marginal distributions [11]. However, a large training set is required to build the histogram, leading to the same drawbacks discussed for [14].

Our contribution is Incremental QuantTree (IQT), a non parametric change detection algorithm for multivariate data streams, which requires solely a very small training set. IQT is able to learn from the data stream, increasing the detection power over time, controlling meanwhile the False Positives. The components of IQT are:

- A technique to continuously update a QuantTree (QT) histogram whenever new data drawn from the stationary distribution ϕ_0 are received, maintaining the control over the False Alarms, proper of QuantTree [11].
- A statistical test that is performed iteratively on the incoming data to assess the goodness-of-fit with respect to the model. The monitored data stream is divided into batches of fixed size ν, and analyzed batch-wise.
- A procedure to compute the threshold for the described statistic in $O(1)$, to allow a proper functioning of the algorithm for high frequency data streams. To this end, a Neural Network is trained using Monte Carlo Simulations, leveraging the fact that the distribution of any statistic defined over a QT histogram does not depend on ϕ_0 [11].

Furthermore, we discuss the possibility of using an Exponentially Weighted Moving Average (EWMA) chart to monitor the Alarms raised by an Incremental QuantTree. Here change detection boils to the monitoring of a binary classifier whose behaviour in stationary condition is known, an Incremental QuantTree, and this task can be effectively performed by an EWMA [15] [12].

Our experiments (Chapter 5) show that Incremental QuantTree achieves detection performance comparable with QuantTree starting with a significantly smaller training set, while maintaining the control over the False Alarms. Furthermore, we show that IQT has good detection performance for high dimensional data streams.

1.1 Problem formulation

We consider a possibly infinitely long data stream $x_1, x_2, ... \in \mathbb{R}^d$, where data become available one at time. We assume data in stationary conditions to follow a distribution ϕ_0, and we define the change point τ as the unknown time when the distribution changes, i.e.:

$$x_t \sim \begin{cases} \phi_0 & t < \tau \\ \phi_1 & t \geq \tau \end{cases} \tag{1.1}$$

We consider a batch-wise analysis, where the data stream is divided into batches of fixed size. Each batch is analysed by means of statistical tests as soon as it is available, to tell whether its observations are generated from the same distribution ϕ_0 as the training set.

We assume a small training set $TR_{small} = \{w_1, w_2, .., w_M\}$ to be provided at time $t = 0$, and to receive successive data as a continuous stream $x_1, x_2....$ When all the observations in the training set follow ϕ_0, i.e. if $w1, w2, .., w_M \sim \phi_0$, the problem is semi-supervised, while when there is no guarantee that all the points have been generated by ϕ_0, the problem is unsupervised. This work focuses on the semi-supervised scenario, where the algorithm can model the distribution ϕ_0 using the training set.

For change detection on data streams, the performance of our algorithm is measured in terms of the Average Run Lenght before a False Alarm (ARL_0) and the Average Run Lenght before detecting a change, after it has happened (ARL_1). The goal of change detection algorithms on data streams is to minimize the ARL_1, while controlling the ARL_0.

Our intuition is that, when having initially access to a small training set TR_{small}, the model of ϕ_0 fitted on TR_{small} is imprecise, thus the detection power is low. Therefore, it is important to improve the quality of the model upon the arrival of new observations, providing a better model of ϕ_0 and thus improving the ability of the algorithm to detect changes. As we are considering change detection by means of one-shot statistics, this translates into increasing the detection power, expressed as the probability of detecting as different from the stationary data a batch of points

drawn from a different distribution ϕ_1, controlling meanwhile the False Positive Rate (FPR). It should be noted that a change could potentially happen at any time, hence the algorithm used must be able to analyse data before using them to update the model.

1.2 Structure of the Thesis

This thesis is structured as follows:

- In Chapter 2 we review the literature on change detection and we discuss the challenges faced when performing it on data streams
- In Chapter 3 we give a detailed explanation of QuantTree, the building block of our research work.
- In Chapter 4 we present Incremental QuantTree algorithm, its properties and the implementation of a minor threshold computation algorithm to ease the training of its neural network.
- In Chapter 5 we test the performance of the algorithm, introducing the datasets used and the methodology; we also discuss the obtained results.
- In Chapter 6, we summarize the contributions of our work, pointing out possible improvements and presenting our conclusions.

Chapter 2

State of the art

During time, multiple approaches for change detection have been developed. This is caused both by the large amount of research on the topic and the variety of forms in which data are available, which influences the monitoring approaches. Typically most techniques are characterized by the following structure[11]:

- A model for the observed process, typically extracted from the training set
- A test statistic T used to assess the difference between the monitored data and the training set.
- A decision rule $T \rightarrow Decision$, which monitors the test statistic in order to detect a change.

In particular, [16] points two main approaches: batch-wise and continuous monitoring. The first group consists of those algorithms which group incoming data in fixed lenght intervals called batches,which are analysed independently by means of a one-shot statistical test. Continuous algorithms instead incrementally build the value of the statistic, updating it whenever new observations are available. The value of the statistic used at time t depends therefore on its value at time $t-1$, and the detection is based on all data received up to the current moment.

Given the vast amount of researches published and solutions proposed, we give an overview of the main families change detection algorithms can be split into, and explain where our work fits in the current state of the art. It must be noted that they are all built on statistical tests, as they are built on the assumption of monitoring the realization of a random variable, possibly unknown. The first major split can be made between supervised settings, in which the algorithm receives labelled samples from both stationary and non stationary data, and semi-supervised ones, where only stationary data are used. Most techniques operate in a semi-supervised manner

[16], as non stationary samples are often not available during the training phase [11], because they often represent an anomalous and out of control situation In fact it is not known which change might happen, and the change detection algorithm should in principle be able to detect any possible change, even those that have never occurred in the past.

A second notable division splits the techniques between those that require the data to be univariate and those which don't need this assumption. We discuss the most important families of algorithms used, starting from the division between those that can only operate on univariate data streams and those that don't require this situation.

2.1 Parametric methods

Most change detection methods in history are parametric, and use a set of parameters to model the observed distribution ϕ_0. They are built on the assumption that ϕ_0 belongs to a known family of distributions, and they use the training phase to set the parameters. Parametric change detection methods often use graphs called control charts to study how a process changes over time. Control charts were first introduced by Shewart (1926-1927) and were used to monitor the fraction of non conformities. [17]. Shewart charts monitor that all single observations remain inside a given interval. It is a model based on a zero order polynomial, which is constructed on the assumption that the variations lying inside the control limits are the results of random causes and the variations lying outside the control limits are the results of assignable causes[18]. The main limitation lies in the fact that each observation is evaluated independently from the others, which makes Shewart charts not very effective on small sustained changes [19].

To address this problem, concept drift methods based on the contemporary evaluation of multiple observations have been created. The CUSUM chart was first introduced by Page [20]. It assumes that the change in the distribution can be modeled as an instantaneous change of a single parameter θ [21], whose values before and after the change are known. We call θ_0 the value of the parameter prior to the change, and θ_1 the value after. CUSUM chart sequentially monitors the log-likehood ratio s_t, expressed as:

$$s_t = \ln \Lambda(x) = \ln \frac{p(x_t, \theta_1)}{p(x_t, \theta_0)}. \tag{2.1}$$

The CUSUM statistic $S(t)$ consists of the cumulative sum of the log likehood ratio s_t, and is characterized by the formula:

$$S(t) = \max(0, S(t-1) + s_t) \tag{2.2}$$

The decision rule is given by confronting the statistic S with a threshold γ, according to the rule $S(t) > \gamma$. In order to guarantee a desired ARL_0, the threshold γ can be computed analytically [20], by means of an approximation or using the Markov Chain Approach [21]. The CUSUM methods are proved to have lower detection times than the corresponding Shewhart-type charts in presence of small changes [22].

A third method is the Exponentially Weighted Moving Average (EWMA) chart, which computes a weighted average of the observed data, where the weights decrease geometrically with time [23]. The EWMA chart was introduced in 1959 as a memory-based tool to detect small changes [24]. The formula for the EWMA is the following,

$$Z_t = (1 - \lambda) \cdot Z_{t-1} + \lambda \cdot X_t \tag{2.3}$$

where the parameter λ is called the smoothing factor, and determines the influence over the model of old observations.

Multivariate data Not all methods that work well on univariate problems can be transposed to the multivariate situation, as some of them explicitly require the observations to be scalar. The first multivariate control chart was introduced by Hotelling in 1947 as the direct multivariate equivalent of a Shewart chart. It assumes that sample data follow a p-variate normal distribution with known mean μ and known covariance matrix $\mathbf{\Sigma}$. We discuss an algorithm based on the Hotelling statistic in 5.1.

An other common parametric change detection method consists of iteratively monitoring the log likehood of the observed data with respect to the distribution ϕ_0. They are based on the assumption that the family to which the distribution ϕ_o belongs is known a priori, and that it can be modeled by means of tuning the value of one or more parameters $\theta_1, \theta_2.. = \Theta$. A training phase is used to estimate the true value of the parameters and, consequently, the distribution ϕ_0. We call $\hat{\phi_0}$ the estimation built on Θ.

The log likehood of the observed data $x(t)$ is given by [25]

$$L(x(t)) = \log(\phi_0(\hat{x}(t)) \tag{2.4}$$

Denoting with L the sequence of log likehood observations, we observe that in stationary conditions L contains i.i.d data drawn from a random scalar variable. When X is subject to a change, we expect L to change itself, and a change is detected by continuously monitoring L. However, the detectability of a change worsens significatively when the dimension increases, because of the linear relationship between the variance of the log likehood and the dimension of the data space [26].

2.2 Non parametric methods

It is not always possible to make assumptions about the data distribution, or the assumptions made might be incorrect, therefore in some situations a model which does not require them is necessary. As changes can happen both in the location and the scale of the observed variable, different techniques have been developed during the years to identify one or both.

Pettitt [27] implements a statistic based on the maximization over multiple Mann-Witney statistics, which allows to detect changes in the location of the observed parameter for fixed sized batches. It is important to notice that the choice of the thresholds is built on top of an asymptotic reasoning, which limits the effectiveness of the algorithm when multiple changes happen.

An extension of the same method has been proposed by [19], based on the Change Point Formulation. It recursively applies the T-test between left and right sections of the sequence, maximized across all possible change points. Common choices for the threshold in this case comes from the Bonferroni inequality and the Monte Carlo sampling. In [28] the authors propose a Ranks-based method for finding changes in both location and scale for univariate data streams, without assuming any prior knowledge about the distribution observe.

Multivariate data The non parametric algorithms described so far are designed for change detection on univariate data. We present here the main techniques for multivariate data. Principal Component Analyisis(PCA) [29] projects the data on a smaller set of dimensions (like the ones associated to the maximum variance), and features like the proximity of the points in the subspace or the reconstruction error when bringing data back to the original features are monitored. A major issue is that the results are often hard to interpret and much depends on the analyst's skills to identify the key components to represent changes and select the thresholds.

Kernel methods are a strong candidate to detect in highly multidimensional data. Naming $x_1, x_2.., x_\tau$ the set of observations of the data stream prior of the change point τ, and $y_1, y_2...$ the observations $x_{\tau+1}, x_{\tau+2}$ after the change, such that $y_i = x_{\tau+i}$, we define the Maximum Mean Discrepancy as $MMD = sup_{finF} E_x[f(x)] - E_y[f(y)]$, which represents the squared distance between the embeddings of the two distributions ϕ_0 and ϕ_1 on a reproducing kernel Hilert Space (RKHS). A common approach is to estimate the Maximum Mean Discrepancy by means of the U-statistic, an unbiased estimator built using kernels [30]. However, the computational cost of computing the U-statistic is $O(T^2)$, which for data streams can become a severe limitation. To address the performance problems given by this statistics a test with

complexity $O(N)$, called B-test, was proposed in [14]. However, kernel methods require a large training set to obtain a good performance, which, depending on the situation, can become a severe limitation to the applicability of these methods.

Histograms represent a common and strong method to find changes in multivariate data streams. However their common implementation over regular grids scales badly when dimensionality grows, as the number of bins grows exponentially with the dimension, which makes them a problematic choices when dealing with highly multivariate data [11]. Among the histogram-based algorithms, QuantTree [11] allows to control the number of bins regardless of the dimensionality of the data and provides a False Positive Rate which is independent from the underlying distribution. The next chapter of our work is dedicated to the description and analysis of QuantTree.

Chapter 3

Background

We present here the main instruments that have been used for our research. The first algorithm discussed is QuantTree. This is particularly important, as Incremental QuantTree is built on top of QuantTree [11], therefore a detailed explanation of its structure and properties is provided to illustrate our contributions.

The Exponentially Weighted Moving Average (EWMA) [12], is the main component of our second contribution, IQT-EWMA. We discuss the EWMA charts as a general change detection tool. We then proceed to go into the details of the settings that are useful for our work, in particular the case where the monitored random variable follows the Bernoulli distribution.

3.1 QuantTree

3.1.1 Algorithm overview

QuantTree is a non parametric change detection algorithm for multivariate data, which relies on a histogram h, called QuantTree histogram, to model the initial distribution ϕ_0 from a training set TR. It detects changes by means of a test statistic T, applied on batches $W_1, W_2...$ of data to tell whether they follow the distribution ϕ_0. In particular, the batch W_i is considered as following a different distribution when

$$T(W_i) > Thr \quad i \in 1, 2, ... \tag{3.1}$$

where the threshold Thr for the statistic T is computed via Monte Carlo Simulations, thanks to the fact that the distribution of T is independent from ϕ_0, as shown in [11].

As a consequence, QuantTree can control the False Positive rate, which is achieved regardless of the family to which ϕ_0 belongs. Furthermore, the procedure adopted

to create h allows to select the number of bins for the histogram a priori, which is important as it avoid the exponential growth of the number of bins with the dimensions of the data.

3.1.2 Implementation details

Histogram Construction

$$\hat{\pi}_k = \frac{n_k}{N} \tag{3.2}$$

where n_k are the points of the training set TR falling inside S_k.

Algorithm 1 Histogram Construction

Require: TR, $\{\pi_k\}_k$
  return $\{S_k, \hat{\pi}_k\}_k$
  $t \leftarrow 0$
  $L_t \leftarrow TR$
  $\bar{L}_t \leftarrow size(L_t)$
  for $k \in 1, 2..K$ **do**
    Choose a random component $j \in \{1, 2, ...d\}$
    $n_k \leftarrow round(\pi_k \cdot N)$
    Draw $\gamma \in \{0, 1\}$ from a Bernoulli(0.5)
    Define $z_j = [x_i]_j \forall x_i \in TR$
    Sort $\{z_j\} : z_{(1)} \leq z_{(2)} ... \leq z_{(\bar{L}_t)}$
    if $\gamma = 0$ **then**
      Define $S_k = \{x \in L_t : [x]_j < z_(n_k)\}$
    else
      Define $S_k = \{x \in L_t : [x]_j > z_(L_k - n_k + 1)\}$
    end if
    Remove the points assigned to S_K from L_t: $L_t \leftarrow L_t - S_k$
    set $\hat{\pi}_k = n_k/N$
  end for

The procedure to compute the histogram is the following. First of all a new subset of the training set, called L is created, containing all the points not yet assigned to any bin S_k for $k \in K$. At the beginning it coincides with the full training set, i.e. $L = TR$ and we call the size of this set at time t as $\bar{L}_t$. Iteratively, a random dimension $j \in d$ is chosen with uniform probability. All data $y_1, t_2.. \in L$ are then sorted over their j_{th} dimension. We denote as $z_j = [x_i]_j$ the value of the point $x_i \in TR$ on its d-th dimension. A value y is drawn from a Bernoulli distribution.

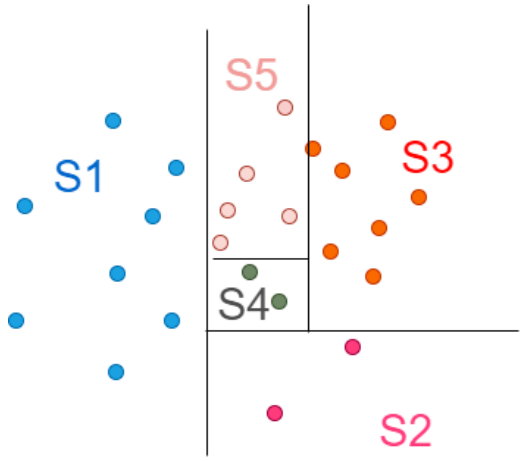

Figure 3.1: QuantTree cuts. $K = 5$. The target probabilities $\{\pi_k\}_k$ are $\{\frac{1}{3}, \frac{1}{12}, \frac{7}{24}, \frac{1}{12}, \frac{5}{24}\}$. The first bin computed is S_1, where the cut is made along the x-axis. The other bins are then iteratively created by partitioning the space over a random axis and assigning n_k points to each bin S_k, according to algorithm 1

Depending on the value y the first or last n_k points are selected to belong to the bin S_k, where $n_k = round(\pi_k * N)$, and a cut $x_j = s[n]$ or $x_j = s[\bar{L}_t - n]$ is selected as the limit for the bin. The assigned points are then removed from L. The procedure is repeated for each bin of the histogram.

An example of the results is shown in figure 3.1 where $d = 2$ and the number of bins is 5. The number of the training set is $N = 24$, and the probability π_k assigned to each bin can be computed from the image as n_k/N. In this case the first bin to be assigned is $S1$, where the dimension used to cut is the horizontal one, and the first n_k points have been selected. The procedure has then iteratively assigned all the other points, reaching the final configuration in figure.

Threshold computation

QuantTree has a fundamental property: "the distribution of any statistic defined over the resulting histograms does not depend on the data generating distribution" [11]. This means that we can numerically compute the thresholds Th for any statistic T defined on the histogram h, provided the size N on the training set used to create it, the size ν of each batch, and the target probabilities $\{\pi_1, \pi_2..\pi_K\}$ associated to each bin. Remarkably, knowing the distribution ϕ_0 and the number of dimensions d is not required to set the threshold.

The thresholds are computed using Monte Carlo simulations on a conveniently chosen 1-dimensional distribution ψ_0, exploiting the fact that a threshold valid for ψ_0 is valid for any distribution, including ϕ_0. The procedure, implemented in algorithm

Algorithm 2 Computation of thresholds

Require: Test statistic T, arbitrarily chosen ψ_0, number B of datasets and batches to compute the thresholds, number of points ν for every batch, N, K, $\hat{\pi}_k$, the desired α

return The value Th of the threshold

for $b = 1, 2, ..B$ **do**

Draw from ψ_0 a training set TR_b of N samples

Use QuantTree to compute the histogram h_b with K bins and target probabilities $\{\pi_k\}_k$ over TR

Draw a batch W_b containing ν points from ψ_0

Compute the value $t_b = T(W)$

end for

Compute the threshold Thr as in (3.3)

2, is the following. At first B training sets $\{TR_b\}_{b=1,..B}$ are generating by drawing N points from ψ_0. For each training set, a histogram h_b is built using algorithm 1. For each h_b a batch W_b of ν points is drawn from ψ_0, and the result of the statistic $T_b = T(h_b, W_b)$ is stored. Finally, the threshold Thr is estimated from the results of the statistics, i.e from the set $G = \{T_1, T_2...T_B\}$, as the $1 - \alpha$ quantile of the empirical distribution of Thr over the generated batches, i.e:

$$Thr = \min \{T_i \in G : \#\{T_j \in G : T_j > T_i\} \leq \alpha B\} \tag{3.3}$$

To take full advantage of the distribution-free nature of the procedure, we use an univariate uniform distribution as ψ_0. This allows to obtain high accuracy on the estimation of the thresholds, since the low computational cost allows to use a very large number B of batches for the simulation.

3.2 EWMA and Bernoulli monitoring

3.2.1 EWMA charts

We provide here a more detailed analysis on Exponentially Weighted Moving Average charts, with a particular their use to monitor Bernoulli variables. As stated in Chapter 2, Exponentially Weighted Moving Average is one of the oldest methods in the Change detection literature, having been proposed in 1959.

It is based on an estimator Z, called EWMA estimator, built to monitor a sequence of independent random variables $X_1, X_2, ...$ Assuming the mean μ_0 and the variance σ_0 an to be known, Z is defined as:

$$Z_0 = \mu_0 \tag{3.4}$$

$$Z_t = (1 - \lambda) \cdot Z_{t-1} + \lambda \cdot X_t, \quad \forall t > 0 \tag{3.5}$$

where the parameter λ control the weight assigned to new observations compared to the previous ones. This EWMA estimator allows to form a "recent" estimate for the mean μ_t of the observed distribution [12], aiming at detect when this diverges from the initial mean μ_0. It can be proved [24] that, independently from the distribution ϕ_0, the mean and standard deviation of the estimator Z_t are give by:

$$\mu_z = \mu_t \tag{3.6}$$

$$\sigma_z = \sqrt{\frac{\lambda}{2 - \lambda} \cdot (1 - (1 - \lambda)^{2 \cdot t})} \cdot \sigma_0 \tag{3.7}$$

The value of Z_t starts from μ_0 and goes towards $\mu_1 \neq \mu_0$, when a change occurs. This can be used to set a control limit L to flag a change, which is detected when:

$$Z_t > \mu_0 + L \cdot \sigma_{Z_t} \tag{3.8}$$

3.2.2 Bernoulli monitoring

A common problem in online classification happens when a change in the data happens, and the error rate $X = \{X_1, X_2...\}$ of the classifier increases accordingly. In this case change detection boils down to the monitoring of a Bernoulli, and in particular of the parameter p regulating the behaviour of the distribution. In this case, expressing the initial value of the parameter p as p_0 ,the variance of the estimator Z_t previously defined can be rewritten as:

$$\sigma_{z_t} = p_0 \cdot (1 - p_0) \sqrt{\frac{\lambda}{2 - \lambda} \cdot (1 - (1 - \lambda)^{2 \cdot t})} \tag{3.9}$$

ARL_0	Regression Estimate of L
100	$2.76 - 6.23\hat{p}_0 + 18.12\hat{p}_0^3 - 312.45\hat{p}_0^5 + 1002.18\hat{p}_0^7$
400	$3.97 - 6.56\hat{p}_0 + 48.73\hat{p}_0^3 - 330.13\hat{p}_0^5 + 848.18\hat{p}_0^7$
1000	$1.17 + 7.56\hat{p}_0 \mid - 21.24\hat{p}_0^3 + 112.12\hat{p}_0^5 - 987.23\hat{p}_0^7$

Figure 3.2: EWMA table: $\lambda = 0.2$. The values of L for different values of the desired ARL_0 are expressed as a polynomial function of $\hat{p}_0$, i.e. of the estimated value of the parameter p of the monitored Bernoulli distribution..

3.2.3 Unknown parameter case

When the value of the parameter p is not known, it must be estimated from the stream along with σ_0. To this end [12] proposes the use of a second estimator of p_0, denoted as $\hat{p}_{0,t}$, defined as:

$$\hat{p}_{0,t} = \frac{1}{t}\sum_{i=1}^{t} X_i = \frac{t-1}{t} \cdot \hat{p}_{0,t-1} + \frac{1}{t} X_t \tag{3.10}$$

Unlike Z_t, the $\hat{p}_{0,t}$ estimator gives the same weight to all the observations, and is therefore less reactive to the new ones. This means that, when a change occurs, Z_t tends towards the new value of the parameter p faster than the less reactive $\hat{p}_{0,t}$, and a change can be detected by measuring the difference between the two statistics. A change is therefore detected when the difference between the estimators exceeds a certain threshold, i.e. when

$$Z_t > p_{0,t} + L \cdot \sigma_z \tag{3.11}$$

The pre-change standard deviation can be estimated using:

$$\hat{\sigma}_{0,t} \cdot (1 - \hat{p}_{0,t}) \tag{3.12}$$

The new formula for the EWMA estimator's standard deviation becomes therefore:

$$\sigma^2_{X_t} = \hat{p}_{0,t} \cdot (1 - \hat{p}_{0,t}) \cdot \sqrt{\frac{\lambda}{2-\lambda} \cdot (1 - (1-\lambda)^{2 \cdot t})} \tag{3.13}$$

The choice of the threshold is important and they must be carefully set to guarantee the desired ARL_0. The determination of the value L depends on the standard deviation σ_x, which must be estimated, when the probability p for the Bernoulli is not known. In this case it is necessary to rely on the estimate $\hat{p}_{0,t}$, which can however vary in time. Therefore, in order to keep the expected rate of false positives constant, the control limit must be recomputed every time $\hat{p}_{0,t}$.

A possible solution for this problem is to use Monte Carlo simulations, as follows. Naming $f(p_0, ARL_0)$ the function that returns the value of L corresponding to a

desired ARL_0 for some value p_0, it is possible to approximate this function by a polinomial, using regression techniques to approximate the polynomial. This process generates a look-up table, containing the values of L_t for the current estimate $p_{0,t}$. The polynomial for this procedure can be generated as follows. For a given value of the ARL_0, the values of L corresponding to various values of p_0 can be estimated by fitting a degree m polinomial, of the form: $L = c_0 + x_1 \cdot p_0 + ... + c_m \cdot p_p^m$.

The results obtained fitting a degree 7 polynomial, with $\lambda = 0.2$ are available on the original paper [15],. As they are needed to compare the performance of Incremental QuantTree during the experiments,they are reported in table 3.2.

Chapter 4

Incremental QuantTree

In this chapter we present Incremental QuantTree (IQT), a novel online change detection algorithm for multivariate data streams. Section 4.1 is dedicated to an introduction of the algorithm. We explain the goal of the algorithm and we provide an overview of its structure. Sections 4.2 and 4.3 are dedicated to the analysis of the two main components of IQT: the histogram update procedure and the threshold estimation technique. Each section provides the analysis of the problem addressed and of the implementation, and a discussion over Section 4.4 analyses the application to online streaming scenario and discusses the possibility of controlling the ARL_0 of the algorithm In section 4.5 we study the properties of the threshold estimation procedure discussed in 4.3 when the number of observations N grows and we propose an algorithm to make it more efficient. We than discuss the choice of the statistic to use in section 4.6, analysing the pro and cons of the main possible choices. Finally, we discuss in 4.7 a possible combination of Incremental QuantTree with EWMA charts for change detection on data streams.

4.1 Introduction

As stated in the introduction, we address here the problem of change detection on data streams. We do so by extending an existing change detection algorithm (QuantTree), to work in a streaming environment. In particular, we aim at maintaining the FPR control proper of QuantTree while providing it with a much smaller training set, while improving the capability of the QT histogram to describe ϕ_0, hence the detection power.

In streaming settings it might not be possible to divide the training phase from the monitoring one. Most change detection algorithms become more powerful when they have access to a larger amount of data, and the same holds for Incremen-

tal QuantTree. Fitting a good model for a highly multivariate distribution ϕ_0 can require a huge amount of data. This may cause problems due to to memory constraints. In this case, being able to build a model on a small training set and adapt it progressively can become a key feature, as it allows to have in memory at any time only a part of the training set used, thus never exceeding the memory constraints.

Furthermore it is not always possible to predict the size of the available data for training. In data streams, data arrive continuously, and the size of the stream is unbounded. In this case, to learn from these data, it is required to continuously process data, as storing all the observations is infeasible due to memory constraints.

To initialize the Incremental QuantTree algorithm, first a QuantTree histogram h is built from a small training set, using the procedure described in chapter 3. The additions brought by Incremental QuantTree can be split in two major parts. First, a procedure to update the histogram whenever new training data are available is provided. This allows IQT to begin with an extremely small training set and continuously learn from incoming data, continuously boosting the detection power of the algorithm. Meanwhile, a novel threshold computation procedure allows IQT to have a proper FPR control, regardless of the data generating distribution ϕ_0, thanks to the QT properties.

4.2 Histograms update

QuantTree histogram consists of a tuple $h = \{(S_k, \hat{\pi}_k)\}_{k=1,..,K}$, associating a probability $\hat{\pi}_k$ to each bin $S_k \in \mathbb{R}^d$. This is the component of QuantTree that is built during the training phase, to model the distribution ϕ_0. The aim is to improve the histogram h whenever new data become available, to better model ϕ_0. There are two natural approaches to modify the histogram upon the arrival of one or multiple data are the following:

- Keep the bins $\{S_k\}_k$ fixed and modify the probabilities $\{\hat{\pi}_k\}_k$
- Modify the bins $\{S_k\}_k$ while keeping the probabilities$\{\hat{\pi}_k\}_k$ constant

Incremental QuantTree uses the former, the rest of the section is dedicated to the analysis of both solution and the explanation for the choice made.

4.2.1 Update procedure chosen: modify the probabilities

The goal is to associate to each bin K a probability π_k such that the true probability $\hat{\pi_k}$ for an observation x to fall inside S_k is π_k, i.e $\hat{\pi_k} = \pi_k$. This would be trivial if the old points were always available, as it would be enough to recompute the value associated to each bin as the number of points fallen inside it divided by the total number of observations. However, due to memory constraints, this is not possible, and old data must be discarded.

We need therefore to resort to a different method. By construction, when the first histogram is generated, π is the estimator of the true value of $\hat{\pi}$. After that, whenever a new datum is coming, we sum 1 to the number of points falling into that bin and re-normalize by dividing each value by $N + 1$, which is the new value of N. It is easy to see that also the new histogram keeps the same property, which makes the algorithm reiterable multiple times. Algorithm 3 illustrates the process in detail.

Algorithm 3 modify Probabilities

Require: data_number, $\{S_k, \pi_k\}_k$, dataSet

return $\{S_k, pi_k\}_k$

for point in dataSet **do**

$index \leftarrow findBin(point)$

$\pi_{index} \leftarrow \frac{\pi_{index} \cdot dataNumber+1}{dataNumber+1}$

$data\_number \leftarrow data\_number + 1$

end for

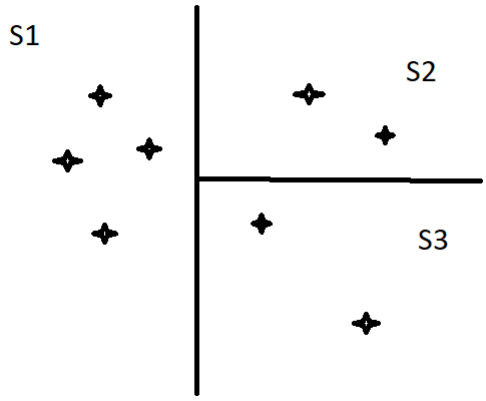

Figure 4.1: Initial situation, a QuantTree histogram of four bins has been created. $\hat{\pi}_1 = \frac{1}{2}$

4.2.2 Comments on the alternative

There is a second way to update a QuantTree histogram, which is maintaining the probabilities fixed and modify the partition. This has a series of advantages. First of all, the uniform-probabilities tree is the most tested. QuantTree does not require a precise combination $\pi_1, \pi_2, ..\pi_k$ to be implemented, but in practice it has been always presented paired with a uniform histogram. This implies that it is by far the most tested combination so far. Finally, not being obliged to recompute the threshold at each round means that once the grid is recomputed, the algorithm is immediately able to start monitoring.

However,it must be noted that modifying the partition of the input space is by no means trivial, and its complexity grows linearly with the number of bins. This is properly shown by the following example. Let us imagine that we have three bins (Figure 4.1) such that the values of the probabilities are: $[1/2, 1/4, 1/4]$. S2 shows what happens if two new points arrive (Figure 4.2): the cut between S1 and the other two bins must be moved to keep π_1 constant. This brings to a situation in which S2 has much more points that S3, which triggers the movement of the line between S2 and S3, as shown in Figure(4.3). Correcting the boundaries of those will trigger the same problem for all the adjacent bins. The task must in principle be repeated for each bin S_k, making it a potentially very expensive solution for large histograms with many bins. This shows that the reduction of enlargement of a given bin will automatically modify the boundaries of the bins next to it, thus their estimated probabilities.

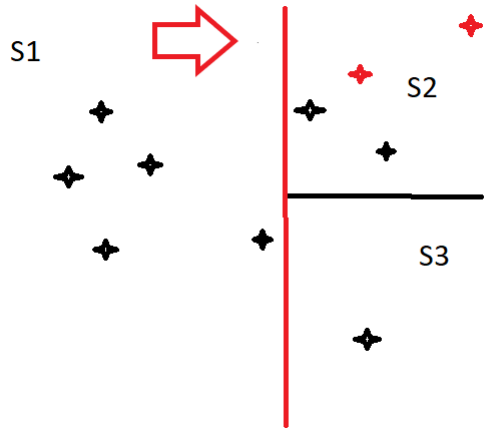

Figure 4.2: As two new points arrive, it is necessary to move the boundary between S_1 and the adjacent bins to maintain $\hat{\pi}_1$ constant.

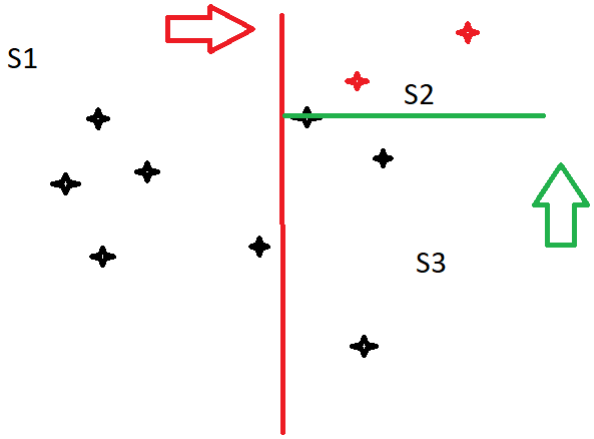

Figure 4.3: The value of $\hat{\pi}_3$ has changed, therefore the boundary between S_2 and S_3 must be moved to maintain the empirical probabilities constant.

4.3 Thresholds computation

QuantTree uses Monte Carlo (MC) simulations to compute the threshold for a given histogram. This procedure does not work well when the probabilities of the histogram must be continuously updated, as Monte Carlo simulations require a too long time. A novel threshold procedure is therefore required, and the speed is an important property to evaluate it.

To this end we propose the use of Neural Networks to estimate the simulations performed by QuantTree. The reason is that, once trained, a neural network is much faster than a simulation, and it is therefore compatible with a high frequency data streaming scenario.

4.3.1 Implementation details

The threshold procedure of QuantTree shows that, in order to compute the threshold for the monitoring phase, only the following values are required: the vector of histogram's probabilities $\pi = \{\pi_1, \pi_2, .., \pi_k\}_k$, the number of points observed so far N, the size of the batches ν, the number of simulations to perform B and the statistic used. Most of them are parameters that can be fixed in advance, such as the number of the simulations, the statistic used and the size of the batches. Therefore we can model the outcome of each simulation i as a function of the only two remaining variables, i.e $Thr_i = f(\pi, N)$, where f is the simulation process and π_i the target probabilities $\pi = \{\pi_1, \pi_2, ..\pi_k\}$ used for simulation i. The process of approximating the function f by means of a faster procedure can be modeled as a Supervised Learning Problem, which can be solved with standard Machine Learning Techniques. In particular, we propose to use a regression Neural Network.

We start by generating combinations of $Thr = (\pi, N)$, we can treat the process of approximating the simulation with a parametric function as a classic supervised learning problem of machine learning. The combinations for the training set π can be generated randomly. It should be noted that any estimate about the value of the threshold associated to a histogram should not depend on the order of its bins. In fact each bin indicates the probability of a point drawn from the training set distribution to fall inside one of the bins generated by the cuts performed by QuantTree. Given the partition and the probabilities associated to each bin, there is no privileged order to write them in the histogram, as long as we keep the same order when evaluating new data. We generate multiple random probabilities combinations, associate to each combination a random integer N representing the number of the observations, and compute the associated threshold by means of the threshold

computation algorithm of QuantTree. We than use the data collected this way to train a neural network to predict the outcome of the simulations given the input, and used the predictions in place of the simulations outcomes during the life cycle of the algorithm. Given that the input space coincides with the one given to the thresholds computation algorithm, and that there is no feature influencing the threshold that is not used to train the Neural Network, the capacity of the network to properly emulate the algorithm depends only on the size of the training set and the complexity of the model used.

A second problem is the choice of the features to feed into the Neural Network. In fact an obvious solution is to use the probabilities π (independently from the order) and the number of points N. Multiple order-independent statistics might also be used, such as the variance of π. This has the advantage of potentially reducing the number of features used to train the model, and therefore the training time and the quantity of data required. However, the problem lies in the limitations that it imposes on the creation of the histograms. In fact, when the number of bins grows, the empirical variance of the distribution converges to the variance of the distribution from which we created the histogram. Therefore, the values of the features of different histograms tend to be very similar to each other, making it impossible to perform an analysis based on them. We opted to a different approach: sorting the histograms. We sort them in decreasing order of probabilities, making the analysis order-independent.

To learn the best approximation for the Monte Carlo Simulations we used two Multi-layer Perceptron regressors. Recalling that N is the number of observations used to build and update the histogram, we train one regressor with the simulations computed according to the QuantTree algorithm for different values of N such that $N < N_{max}$, and the other one trained with the simulations performed with cut_o_space. The former is used to estimate the probabilities when the total number of observation N is smaller than N_{max}, the latter for larger N.

4.4 Online monitoring

Incremental QuantTree is an algorithm designed to monitor data streams. As such, it must work in a streaming environment, and its performance is measured in terms of the ARL_0 and the ARL_1, i.e the average run lenght before a false alarm and between the occurrence of a change and its detection. However the technique presented so far relies on statistical tests, and is in fact measured in terms of the FPR and the detection power.

In this section we discuss how the threshold Thr can be selected to ensure a

controlled ARL_0, starting from the results obtained in the previous sections. We denote as

$$p(T(W_t) > Thr_i | T(W_i) \leq Thr \ \ \forall i < t) \tag{4.1}$$

the probability that a process stops at time t provided that it has not detected it yet at time $t-1$, and we recall that if

$$p(T(W_t) > Thr_i | T(W_i) \leq Thr \ \ \forall i < t) = \bar{p} \quad \forall t > 0 \tag{4.2}$$

the average time T before the process stops can be computed as

$$E[T] = \frac{1}{\bar{p}} \tag{4.3}$$

Measuring the time in batches, i.e using a time t' such that $t' = \nu \cdot t$, the probability of Incremental QuantTree to stop at time t', assuming that the batch analysed at time t' is composed of points drawn from ϕ_0, is equal to α. Recalling that the average run lenght of a change detection algorithm under no change coincides with the ARL_0, and that a batch is analysed every ν observations, we have that:

$$ARL_0 = \frac{1}{\alpha} \cdot \nu \tag{4.4}$$

Therefore, controlling the FPR for Incremental QuantTree directly translates into controlling the ARL_0, according to (4.4).

4.5 Asymptotic analysis

Monte Carlo simulations are costly and time-consuming operations. Even though Incremental QuantTree performs them before deployment, and it is not limited by their execution time in the analysis of high frequency data streams, it can still benefit from a faster simulation procedure, to facilitate the training phase and the making of the experiments. This is particularly true when working with large data sets, as the threshold computation procedure implemented by the threshold computation algorithm of QuantTree becomes progressively slower as the training set size N increases.

To address this problem we introduce cut_on_space, an algorithm able to compute the thresholds with a precision comparable to the threshold computation algorithm of QuantTree and in a notably shorter time, provided that the number of observations N is above a certain threshold. By proving that the threshold obtained using cut_on_space is aymptotically equal to the one obtained with the original algorithm, we show that it can be used to efficiently train the neural network of Incremental QuantTree.

4.5.1 cut_on_space

Theorem 1 of QuantTree states that the value of any statistic computed on a QuantTree histogram depends only on the probabilities π and on the number of observations N_t. The dependency on N_t does however depend on time. In this case, since we learn the bin probabilities online, the number of observations grows overtime, Figure 4.4 plots the average over different combinations π of the thresholds computed with the the threshold algorithm of QuantTree with respect to the size N of the training set. Larger N bring to higher thresholds, but it is possible to see how this saturates over a certain N. Furthermore, it shows that for $\lim N \to \infty$ it tends to a constant value.

There are multiple ways to exploit this property. The first and simplest one is to set a large value of N_{max} and to train the neural network only to learn the dependency on N up to that points, assuming that $Thr(\pi, N) = Thr(\pi, N_{max}) if N > N_{max}$. We empirically verify that this assumption is true, and that this procedure is feasible.

There is however a better way, which takes advantages of the fact that the simulations are performed using a uniform distribution $U[0, 1]$. This results cut_on_space is described in algorithm 4.

First a variable L, representing the portion of the interval $[0, 1]$ yet to be assigned

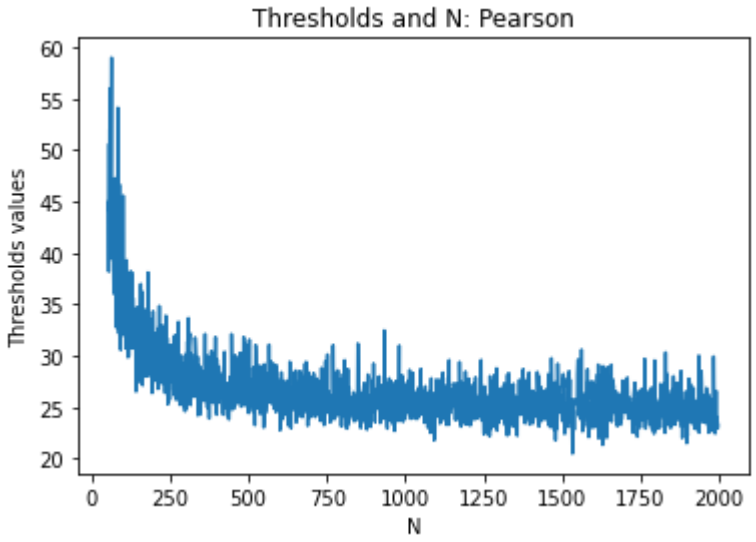

Figure 4.4: The plot represents the dependency of the thresholds computed with QuantTree threshold procedure with respect to the size N of the training set. For each value N, multiple random combinations of target probabilities $\{\pi_k\}_k$ are drawn, and for each of them the threshold is computed. The value plotted is the average of the thresholds obtained this way.

Algorithm 4 Cut_on_space

Require: $\pi_k > 0 \forall\ k$
  return $\{S_k, \hat{\pi}_k\}_k$
  $bern \leftarrow Bernoulli(0.5)$
  $bins \leftarrow \{[None, None]\}_k$
  $L \leftarrow [0, 1]$
  for $\pi_k\ in \{\pi_k\}_k$ **do**
    if $bern == 0$ **then**
      $bins_k \leftarrow [L[0], L[0] + \pi_k]$
      $L \leftarrow [L[0] + \pi_k, L[1]]$
    else
      $bins_k \leftarrow [L[1] - \pi_k, L[1]]$
      $L \leftarrow [L[0], L[1] - \pi_k]$
    end if
  end for

to any bin is set to comprehend the whole interval $[0, 1]$. For each target probability π_k, a value g is drawn from a Bernoulli of parameter $p = 0.5$. Depending on the value of g, the first or last portion of L is assigned to the bin S_k, and the size of the assigned portion is set to be equal to π_k.

The cut scheme proposed follows the same pattern of the one used for threshold computation in QuantTree. QuantTree compute the the cut points by drawing N points and computing the empirical quantiles of the marginal distributions. Cut_on_space is instead based on the fact that the expected value for the quantiles of the univariate uniform distribution ψ used to compute the histograms has a lenght of exactly π, where π is the probability associated to the bin.

Proposition The threshold computed by the standard QuantTree algorithm tends to value obtained using by the cut_on_space when N tends to ∞.

Proof The cut_on_space and the algorithm used in QuantTree for threshold computations are used on a uniform distribution on the interval [0, 1] for the Monte Carlo simulations. We recall the main properties of this distribution.

1. U has a cumulative function G^{-1} given by

$$G^{-1}(p) = p \; for \; p \subset [0, 1) \tag{4.5}$$

2. The probability that a uniformly distributed random variable falls within any interval of fixed length is independent of the location of the interval itself (but it is dependent on the interval size), so long as the interval is contained in the distribution's support.

We first prove that the expected size of the bins computed using the QuantTree algorithm tends to the value computed using cut_on_space, when $N \to \infty$. We then prove that this is a sufficient condition for the asymptotical equality of the two threshsolds.

We know that the first cut points using quantTree corresponds to the value of the point $\bar{x}$ with the $\pi \cdot N$th largest (or smallest) value. For the law of the Large Numbers, as $N \to \infty$, we know that that the value of $\bar{x}$ is exactly the π (or $1 - \pi$) quantile of the uniform distribution. From property 2 we know that the size of the π and the $1 - \pi$ quantiles are the same, and from property 1 we can tell that for both it is exactly equal to π (which is the same value that we obtain when performing cut_on_space). The same reasoning holds for any other cut point, therefore the size of each bin S_k, characterized by a target probability π_k, computed with cut_on_space is

asymptotically equal to the size of a a bin with equal target probability π_k computed using the QuantTree histogram creation algorithm on a uniform [0,1] distribution. We can therefore can tell that asymptotically the two algorithms perform exactly the same cuts. We know that the value of the statistic whose quantile is the threshold is given by the variation between a batch of points drawn from the uniform [0,1] and the tree performed using the algorithm, which in turn is made by the couple $S_k, \hat{\pi}_k$. We proved that the tree is asymptotically the same for the two algorithm, and we know that the batch does not depend on the algorithm we used to compute the tree, therefore we can tell that the expected value of the threshold is the same for the two algorithms, when N tends to infinite.

4.5.2 Comment on the results

There is a two-fold advantage in the implementation of cut_on_space. The first result is that the simulation of a threshold for large training set can be done efficiently by means of cut_on_space, thus reducing the computational cost of the simulations.

More importantly, the fact that the threshold computed with this technique is asymptotically equal to that obtained using the threshold computation algorithm of QuantTree, means that the threshold for any set of probabilities π converges to a constant value as $N \to \infty$, and that there exists a number N_{max} such that, provided enough simulations to estimate the threshold for any $N < N_{max}$, there is no need to make any simulation for any $N \geq N_{max}$.

4.6 Statistics choice

Incremental QuantTree uses the output of a Neural Network to compute the threshold for the statistic. The training set for the network is generated by computing for multiple different histograms and number of observations the thresholds according to the procedure implemented in QuantTree. It is therefore important that the Monte Carlo simulations of QuantTree algorithm return the correct value for all the probabilities π used to train the Neural Network. In this section we study the performance of the algorithm for different probabilities π and for the two statistics used in [11], i.e the Pearson statistic and the Total Variation.

4.6.1 Pearson

The Pearson statistic for a histogram h computed on a distribution ϕ_0 is defined by:

$$T(W) = \sum_{k=1}^{K} \frac{(y_k - \nu \cdot \hat{\pi_k})^2}{\nu \cdot \hat{\pi_k}} \tag{4.6}$$

where y_k is the number of points of the analysed batch that fall into the bin S_k, $\hat{\pi_k}$ the probability for a point drawn from ϕ_0 and ν the number of points of the batch. It represents the covariance of two variables, divided by the product of their standard deviations, and it is a measure of the linear correlation between two sets of data. The Pearson statistic is the standard choice adopted in [11], and in most situations it guarantees a proper FPR control.

However, the threshold procedure implemented in QuantTree, when using Pearson, returns in some situations values on a different scale from the others. We call this values outliers. The following properties hold for these outliers:

1. Outliers are systematic: The outliers are associated to some histograms in particular and do not depend on the precision of the simulations.

2. Outliers are errors: These outliers can be considered errors and they should not be used.

3. Outliers are caused by the Pearson's formula. The reason the simulations using the Pearson statistic can bring to outliers lies in the form of the histograms used and the Pearson's formula

Outliers are systematic The outliers are not randomly caused by variance in the process involving Monte Carlo Simulations that is used to compute the thresholds, but are highly correlated to certain combinations of probabilities $\{\pi_k\}_k$. To prove

it, we generated multiple random combinations π, and computed the threshold associated to each of them using the standard QuantTree. Repeating the operation multiple times with the same π, the outliers are always associated to the same combinations of probabilities. Therefore, the outliers are caused by some properties of the target probabilities used for the histogram.

Outliers are errors We analyse the performance of a QuantTree, whose histogram has been built on some target probabilities $\{\pi_k\}_k$, which in our experiments cause the insurgence of an outlier. If the thresholds were correctly computed, the behaviour of the algorithms should be the same as the one of a tree built on a different combination of probabilities. We noticed instead that in those situations our algorithm is much less powerful than usual.

The problem is the following: choosing such an high threshold, we will never be able to notice any change which does not cause a point to fall in the low probability batch, as the rise in the Pearson's value won't probably be enough to reach the threshold, which could be orders of magnitude higher than the average Pearson's value. This leads to a severe loss in the algorithm's power.

We show an example of the performance of a QuantTree initialized with a set of probabilities that lead to an outlier, compared to a QuantTree initialized with random probabilities. It is possible to see that the ability to detect a change of the former is dramatically lower. We can therefore state that the presence of outliers is not a correct behaviour of the Monte Carlo Simulations. This is important, as it means that the neural network if Incremental QuantTree should not learn this outliers.

Outliers are caused by the Pearson's formula We want to know if it possible to predict which π cause the insurgence of outliers. We found out that there is an inverse correlation between the size of the smallest probability of a histogram and its threshold. We proved that explanation lies in the Pearson formula, which with alpha small enough bring to extremely high values. We explain the reason by means of the following example.

We suppose to be given a set of probabilities $\pi_1, \pi_2.. = \pi$, such that there is a very small probability $\pi_m in$ for a given point to fall in a certain bin. We use a uniform distribution ϕ to run the simulations, as done in QuantTree 2. In order to compute the threshold by means of the Monte Carlo Simulation, we we first generate a large set of data N and we create a histogram on top of them. For the sake of semplicity, let us assume that we had enough computational power to use a number of data N large enough to have that the true probability for any point to fall inside

S_k is exactly equal to π_k. We now generate multiple batches and compute the value of the statistic for each of them, then we sort the values obtained and we select the $1 - \alpha$ empirical quantile.

For every batch, the probability that at least a point of the batch falls inside the bin with the smallest probability is given by the product of the probability for a single point to fall inside S_{min}, that is π_{min} and the number of points in the batch ν. If the number of batches observed B is large enough, at least one batch will have this characteristic The Pearson statistic computed on a batch where at least a single point falls in that batch has a very high level, as we have a denominator close to zero in the formula. This means that, when sorting the values before choosing the threshold, there will be a discontinuity between the first $B - x$ values and the last x, where x is the number of batches which have at least a point falling in S_{min}. Therefore, if α is small enough that the α empirical quantile falls inside this region, a threshold with a potentially extremely high value is returned.

4.6.2 Total Variation

The other analysed statistic is Total Variation, characterized by the formula:

$$T_h(W) = \frac{1}{2} \cdot \sum_{k=1}^{K} |y_k - \nu \cdot \pi_k|$$

Total variation comes with its own problems. Being a discrete statistics, it can assume a finite set of possible values. This leads to low precision, because the thresholds associated to different α may be the same value, reducing therefore the possibility of precisely controlling the FPR.

It has to be noticed that Total Variation has a better performance when working with large batches of data, as it can assume more values and provides a more precise estimation for the correct value of the threshold to associate to alpha. However relying on batches of many points reduces the precision with which the change is located and increases the expected detection delay even for huge changes, as they might occur at the beginning of the batch and we would be waiting ν points before making a decision.

A much less costly solution is the following: if we use non symmetric probabilities we have much more values to choose among and we obtain a threshold able to determine precisely the FPR. The reason is that, when bins are symmetric, there are multiple changes that can bring to the same final value, according to Total Variation. In fact if two bins S_1 and S_2 are associated to the same probabilities, any combination in which S_1 has X points and S_2 has Y leads to the same value. This does not happen when π_1 and π_2 are different.

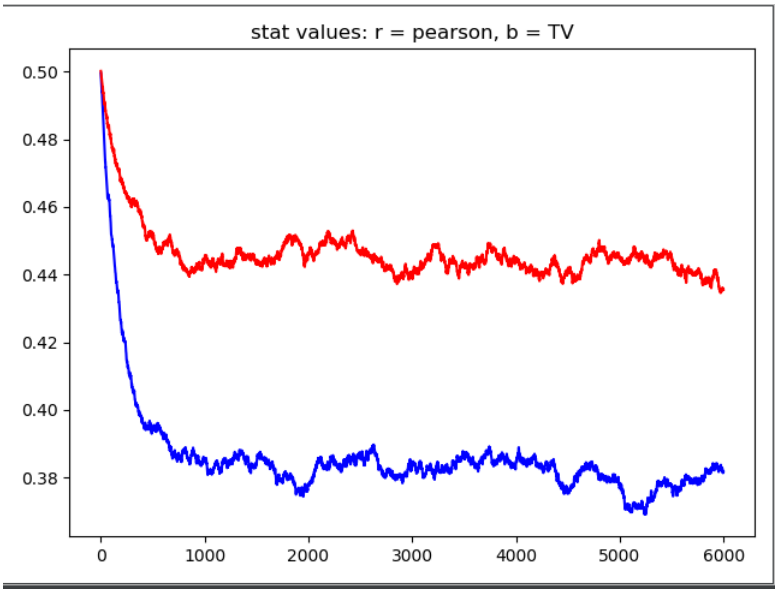

Figure 4.5: Monitorig a QuantTree with an EWMA chart, it is possible to see that, if the QuantTree has been built with uniform target probabilities $\{\pi_k\}_k$, the effective FPR when using the Total Variation statistic is smaller than the one obtained using Pearson, although they have been set to yield to the same desired FPR $\alpha = 0.5$

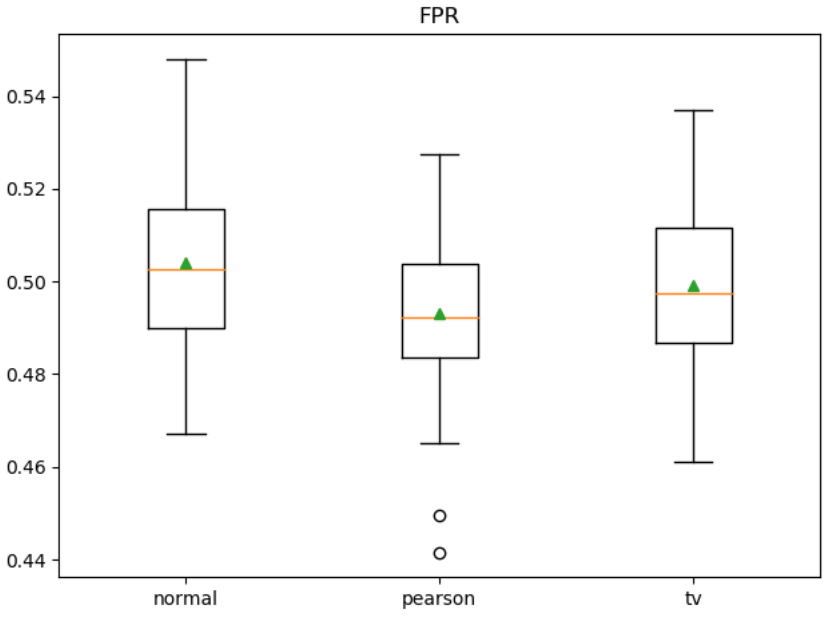

Figure 4.6: When the probabilities $\{\hat{\pi}\}_k$ are iteratively built by modifying an Incremental QuantTree built on a sufficiently large training set, the probabilities $\{\pi_k\}_k$ are generally not symmetric, ant the performance of the Total Variation improves as a consequence.

In practice this reduces a lot the combination of possible values the statistic can assume when dealing with constant cuts, strongly reducing the capability of the learner to effectively control the False Positive Rate.

Choices adopted for the algorithm

We analysed the behaviour of the statistics used in different situations, proposed a viable way to use both and proved experimentally that Total Variation, when properly tuned, can be a properly used to solved the tasks required for our work. In the end we opted to use Total Variation for our experiments, as it requires less assumptions about the possible situations to perform properly. In fact it is not always possible to predict in advance whether the final histogram will be unbalanced, while it is safe to assume that it won't be symmetric, as the probability for a random set of probabilities $\{\pi_k\}_k$ to be symmetric is extremely low.

4.7 Incremental EWMA-QuantTree

We introduce here Incremental EWMA-QuantTree, a change detection algorithm based on the combination of an Incremental QuantTree and an EWMA chart. Our goal is to adapt the QuantTree algorithm to base the decision on all the data observed so far, while maintaining the batch wise statistic. This is obtained by applying an EWMA chart to the decisions made by an Incremental QuantTree on a sequence of batches. Thanks to the properties of QuantTree discussed in chapter 3, the probability to detect a change on a batch following ϕ_0 can be set a priori to be equal to α

4.7.1 Algorithm overview

As discussed in chapter 3, a common use of the EWMA charts is to continuously monitor the classification error of a binary classifier [31] [23] [32]. Assuming the probability error to be a parameter α, the problem boils to the continuous monitoring of a Bernoulli variable of parameter α. Naming y_t the variable stating whether the prediction made by the classifier at time t is correct, we say that $y_t = 0$ if the prediction was correct and $y_t = 1$ otherwise.

We consider QuantTree and Incremental QuantTree as classifiers, which classify a batch as anomalous or normal. Under stationary conditions, we know that the correct assignation to each batch is to classify it as normal: we say then that $y_t = 1$ if the batch is labeled as anomalous, and $y_t = 0$ otherwise. Recalling that for a batch-wise monitoring change detection algorithm the FPR is the probability for a batch to be labeled as anomalous when it was drawn from ϕ_0, we can state that,

$$FPR = E[T_t] \tag{4.7}$$

For Incremental QuantTree, the False Positive Rate is controlled and set to be equal to a previously chosen parameter α, therefore we can expect that:

$$E[T_t] = \alpha \tag{4.8}$$

4.7.2 Implementation

The construction of the algorithm is composed of the following steps:

1. The choice of the monitored value, that is the proportion of positives returned by QuantTree

2. The creation the monitored statistics and of the final statistic used to detect a change.

3. The computation of a threshold for the previously created statistic.

We discuss in details the choices made for all of the three points highlighted.

Choice of the values to monitor

The values monitored according to the described scheme are the detections made by a QuantTree or Incremental QuantTree on batches of data. This consists of a Bernoulli (whose expected value for the parameter p is α) under ϕ_0, and represents the proportion of wrong choices made by the tree. This is coherent to the setting for which the monitoring scheme in [32] is proposed.

Statistic computation

Algorithm 5 ALG1: compute EWMA

Require: QuantTree QT, memory factor λ, False Positive Rate α for QT

$t \leftarrow 1$
$Z_0 \leftarrow \alpha$
for $t = 1, 2...$ **do**
  $x_t \leftarrow QT.acceptbatch$
  $Z_t \leftarrow (1 - \lambda) \cdot Z_{t-1} + \lambda \cdot x_t$
end for
return $Z - t$

The structure of the algorithm is the same used in [32], with the main difference coming from the observed process. We compute two statistics, the EWMA and the Y-statistic. The formula to compute the EWMA statistic Z is the following

$$Z_t = (1 - \lambda) \cdot Z_{t-1} + \lambda \cdot x_t \tag{4.9}$$

The value of the statistic Z(t) is give y the exponentially weighted sum of its previous value ad the new observation, where λ is the forget factor. The second statistic computed is the Y-statistic, which formula is:

$$Y(t) = \frac{(t-1) \cdot Y(t-1)}{t} + \frac{x_t}{t} \tag{4.10}$$

Both statistics asymptotically tend, under ϕ_0 to the FPR α of the monitored tree. Unlike EWMA, the Y(t) estimator does not give more weight to recent observations. Therefore EWMA is more sensitive to changes in p0, and it gives a value closer to current value of α.

Algorithm 6 ALG2: compute Y-statistic

Require: QuantTree QT, memory factor λ

$t \leftarrow 1$, $Ewma[0] \leftarrow QT.alpha$

$y[t] \leftarrow QT.acceptbatch$

$Z_t \leftarrow \frac{(t-1)\cdot Y(t-1)}{t} + \frac{x_t}{t} * \lambda$

$t \leftarrow t + 1$

return Y

Whenever a change happens the FPR of QuantTree-based algorithms increase, and the monitoring statistics tend towards the new value α_1. EWMA is faster to converge to the new value, therefore we can interpret a significant difference between the two statistics as a prove that a change has happened. We detect a change whenever:

$$Z_t > Y_t + L \cdot \sigma_{Z_t} \tag{4.11}$$

σ_{Z_t} is the variance of the Z estimator, and it is given by The choice of the limits for the statistics boils down to the choice of L.

Control Limits

We reproduce the control limits choice applied in [32]. First of all, it should be noted that we are assuming the FPR to be unknown. The monitored statistics are both time-dependent, as at time $t = 0$ they both begin from the value of 0, while tending to a different value α. This means that obtaining a constant rate of false positive (and therefore controlling the ARL_0) is possible only if we allow the control limit to be time varying. From here on we hence name the control limit L_t to highlight its dependence from time. The following procedure is used: Suppose $f(p_0, ARL_0)$ the function that returns the value of L corresponding to the desired ARL_0 for some desired P. We can approximate this function using a polynomial by means of standard regression techniques, that can be used to estimate the thresholds. This is a computationally costly procedure, but it can be computed once, and then used for any run of the algorithm.

This means that a look up table can be created, and finding the threshold on it requires a constant time. The table is generated by means of a Monte Carlo simulation. It should be noticed that the table of the original work from [32] is also available, which cuts even more the cost of the operation.

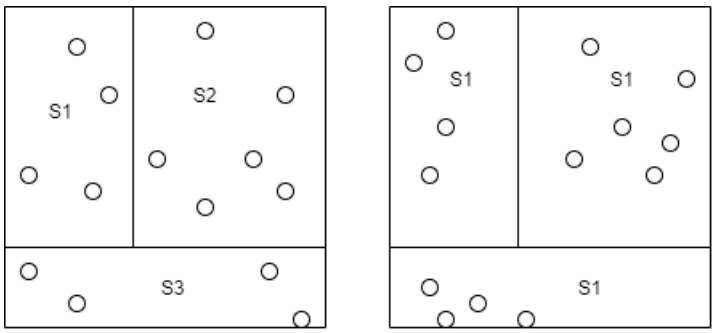

Figure 4.7: The points drawn from ϕ_0 (figure on the left) are used to build a QuantTree histogram $h = \{S_k, \hat{\pi_k}\}_k$. The points drawn from ϕ_1 (figure on the right) have a different distribution in the space of the data, but tend towards the center of the bins $\{S_k\}_k$

4.7.3 Comments on the results

The key innovation brought by the coupling of Incremental QuantTree and an EWMA monitoring scheme comes from the fact that the whole procedure happens in an unsupervised manner. All existing schemes based on the monitoring of a classifier error rate require the truth about the target values associated to at least some observations, which are necessary to assess the error rate of the monitored classifier and, in doing so, detect a change. In practice this is not always feasible. QuantTree instead is fully unsupervised, which eliminates the need for labels during the monitoring. This is a factor of key importance, because it allows to extend the family of situations where it is feasible to apply this family of solutions.

Degenerate cases analysis In addition to what stated so far, EWMA QuantTree is improves dramatically the performance of QuantTree when dealing to a degenerate type of changes, which can never be noticed by classic QuantTree. In particular, it is in principle possible that data distribution changes without moving from a bin to an other, for example concentrating inside the bins (figure 4.7). In this case the lower sensibility to the noise will bring to lower values of the statistic, which can never be noticed by QuantTree or Incremental QuantTree. It is a very peculiar and probably unlikely situation, but if we suspect that it may happen, we can build a two-tailed test on the EWMA chart to detect any anomalous reduction of the positive rate compared to the expected one.

4.7.4 Paired learners

QuantTree paired with EWMA can be used as a stand alone change detection algorithm. It is able to guarantee a prefixed ARL_0, and is able to detect changes thanks to the threshold computation mechanism. However we think that, at a practical level, the most interesting application can be as a stable learner paired to a more reactive one, such as an Incremental QuantTree. This would both solve the problem of the slow detection of big changes from the EWMA, which is caused by its continuous nature, and the risk that an Incremental QuantTree learns from data coming from ϕ_0 (which might in principle happen if the shift between phi_0 and ϕ_1 is small, although it has not happened yet during our experiments). Some studies in this direction have been made, but we consider them to be outside of the scope of this thesis.

Chapter 5

Experiments

The goal of this chapter is to compare the performance of Incremental QuantTree with existing methods in the literature, in particular with QuantTree and the Hotelling statistic for change detection. Furthermore, we show that IQT is always able to control the False Positives.

The rest of the chapter is organized as follows:

- The first section is dedicated to the introduction and discussion of the principal instruments used to design the tests. These are the algorithm we compare with IQT to compare them and assess the performance, that is H-CPM, and the procedure used to generate the data for the analysis.
- The second section is dedicated to the analysis of the figures of merits used.
- In the third section, we introduce the setting used for our experiments
- In the last section the results are displayed and commented.

5.1 Instruments used

5.1.1 The Hotelling test statistic

Hotelling's T^2 statistic is a widely used statistic in statistical process control and change detection. T^2 is used in change detection as a tool for parametric statistical tests. Assuming that data come from a multivariate normal distribution, i.e. $\phi_0 \sim \mathcal{N}(\mu, \Sigma)$, the T^2 statistic can be expressed as

$$T^2 = (X_t - \mu)' \cdot \Sigma^{-1} \cdot (X_t - \mu). \tag{5.1}$$

The values of the parameters μ and Σ can be assumed to be known a priori [33], or more realistically must be estimated during the training phase [34]. However, estimating the parameters during the training phase requires a large training dataset, which is not always available. An online change detection algorithm leveraging the Hotelling statistic has been proposed in [35].

Change Point Model

The change point model defines the change in case of a d-variate normal case, where $\phi_0 \sim \mathcal{N}_p(\mu, \Sigma)$ and $\phi_1 \sim \mathcal{N}_p(\mu_1, \Sigma_1)$.

The change point model divides the task of detecting the change point τ in two statistical tasks: a testing task aiming at deciding whether there has been a change in the analysed data, and an estimation task, with the goal of detecting the exact location of the change point τ [19]. We call Phase I the task of detecting whether a change has occurred in the analysed data, and Phase II the detection of the most likely location of the change τ.

We focus here on the situation when the assumption $\Sigma = \Sigma_1$ holds, so that $\phi_0 \sim \mathcal{N}_p(\mu, \Sigma)$ and $\phi_1 \sim \mathcal{N}_p(\mu_1, \Sigma)$, and define as:

$$\bar{X}_{j,m} = \frac{1}{m-j} \cdot \sum_{t=j+1}^{m} X_t \tag{5.2}$$

the mean of the data observed between the time $t = j + 1$ and $t = m$. Calling n the number of data observed so far, and W_k the pooled variance structure for the considered time $\bar{t}$, we have that:

$$W_{\bar{t}} = \{\sum_{i=1}^{k}(X_t - \bar{X}_{0,\bar{t}})(X_t - \bar{X}_{0,\bar{t}})' + \sum_{i=\bar{t}+1}^{n}(X_t - \bar{X}_{0,\bar{t}})(X_t - \bar{X}_{\bar{t},n})'\}/(n-2) \tag{5.3}$$

It is important to assume $n > d + 1$ to ensure full rank. [35].

Defining the standardized difference between the pre-shift and post-shift observations as:

$$Y_{\bar{t}} = [\bar{t} \cdot (n - \bar{t})/n]^{1/2} \cdot (\bar{X}_{0,\bar{t}} - \bar{X}_{\bar{t},n}) \tag{5.4}$$

and assuming a possible change point $\bar{t}$ the Hotelling T^2 statistic for testing the difference between pre-shift and post-shift data for that point is given by:

$$T_k^2 = Y_{\bar{t}}' W_{\bar{t}}^{-1} Y_{\bar{t}}, \quad \bar{t} = 1, ...n - 1 \tag{5.5}$$

The most suitable candidate for a change is the maximum likehood estimator (MLE) for the change, named $\hat{\tau}$, which is defined as the time $\bar{t}$ maximizing:

$$\hat{\tau} = \arg\max_{\bar{t}} T_{\bar{t}}^2 \tag{5.6}$$

Algorithm description

The authors of [35] proposed an algorithm for change detection, relying on the Change Point Formulation, and able to work in a framework where no training set is provided.. In the experiments we will refer to it as H-CPM, where H stays for Hotelling. A common application of the Change Point Formulation is when the analysed data have a fixed size, especially for Phase I. However, as we are discussing the applications of the T^2 statistic for change detection on data streams, the size of the data to analyse is unknown. To this end the authors of [35] proposed an adaptation of the described model to to deal with this task, removing the boundaries between Phase I and Phase II. Assuming the parameters μ and Σ not to be known a priori, the process whenever a new observation X_n is available is the following:

- The T^2 for each possible split, and the $T^2_{max,n}$ is computed
- If $T^2_{max,n}$ exceeds some threshold $h_{n,p,\alpha}$, a change is detected. The time of the change is identified as the k values leading to maximum T^2.
- Otherwise there is not enough evidence to detect a change, and the process continues

In order to guarantee a fixed ARL_0, it is enough to guarantee a constant false positive probability α, and the sequence must satisfy [35]:

$$P[T^2_{max,n} > g_{n,p,\alpha} | T^2_{max,j} \leq h_{j,p,\alpha}; j < n] = \alpha \tag{5.7}$$

These values can be computed via simulations. It must be noted that the lookup tables with the result of the simulations is available at [**?**].

5.1.2 Controlling the Change Magnitude

Our experiments are performed using multi-variate gaussian distributions. In particular, we generate the post-change distribution ϕ_1 as a roto-translation of ϕ_0, according to the "Controlling the Change Magnitude" (CCM) framework [36]. Notably, a change performed via a roto-translation encompasses both changes affecting the expected value μ of the distribution and the correlation among the data components. CCM allows to generate a new distribution ϕ_1 from ϕ_0, with a fixed symmetric Kullback-Leibler (SKL) divergence between the two distributions, by means of iterative algorithms, which are guaranteed to converge [36].

5.2 Figures of merit

False Positive Rate Given batch b drawn from the stationary distribution ϕ_0 used to train a model h, we define the False Positive Rate FPR as the probability for the batch to be detected as generated from a different distribution ϕ_1, i.e.

$$FPR = p(T(h, b) > Thr | b \sim \phi_0) \tag{5.8}$$

where T is the statistic used and Thr the threshold for the statistic T. We evaluate the False Positive Rate by applying a statistical test to multiple batches drawn from ϕ_0 and computing the False Positive Rate as the empirical mean of the detections as the number of alarms divided by the number of batches drawn from ϕ_0 analysed.

Detection Power We compute the detection power as the counterpart of the False Positive Rate when the points composing the batch b come from a distribution $\phi_1 \neq \phi_0$, i.e.

$$Detection\ Power = p(T(h, b) > Thr | b \sim \phi_1 \neq \phi_0) \tag{5.9}$$

where T is the statistic used and Thr the threshold used for the statistic T. As for the False Positive Rate, we evaluate the Detection Power by applying a statistical test to multiple batches drawn from ϕ_1 and computing the False Positive Rate as the empirical mean of the detections, i.e. as the number of alarms divided by the number of batches drawn from ϕ_1 analysed.

Average Run Lenght We define the Average Run Lenght (ARL_0) as the expected run lenght of the algorithm, when analysing data from the stationary distribution ϕ_0 used to train the model. More formally defining t' as the random variable representing the stop time

$$t' = \min t : T_t > Thr \tag{5.10}$$

where T is the statistic used, T_t the valued assumed by T at time t, and Thr the threshold for the statistic T, we define the ARL_0 as:

$$ARL_0 = E[t'] \tag{5.11}$$

To compute the ARL_0 we build multiple algorithms on different training sets of a certain lenght N, and we monitor a data stream x_1, x_2.... For each tree we compute the stop time and we define the ARL_0 as the empirical mean of the stop time of the algorithms.

Detection Delay We define the Detection Delay (ARL_1) for an algorithm as the the expected value of the interval between the change point τ in a data stream and the stop time of the algorithm analysing it. More formally:

$$ARL_1 = E[t' - \tau] \tag{5.12}$$

where τ is the change time and t' is the random variable representing the stop time, defined as:

$$t' = \min t : T_t > Thr \tag{5.13}$$

To compute the ARL_1 we build multiple algorithms on different training sets of a certain lenght N, and we monitor a data stream x_1, x_2.... We eliminate from the pool all the algorithm stopping before the change point τ, and compute the ARL1 as the average of the time intercurred between the stop time and the change point τ for the algorithm which had not stopped at τ.

5.3 Design of the experiments

Our experiments are conducted using synthetically generated data, drawn from multivariate gaussian distributions. In particular we use a d-dimensional gaussian distribution as the pre-change distribution ϕ_0, and a roto-translation of ϕ_0 as the post-change distribution ϕ_1. We recall that the magnitude of the change can be measured by means of the SKL value [36].

The first experiment we run has the goal to prove that Incremental QuantTree is able to control the FPR regardless of the size of the initial training set and of the distribution ϕ_0. To this end we test IQT with different combinations of data dimensions d, change magnitude SK, size of the initial training set N and desired FPR α. We then proceed to prove that the control over the FPR effectively translates into the control over the ARL_0. To prove it, we use the same setting and measure the ARL_0 of an Incremental QuantTree, with the goal of having $ARL_0 = \nu \cdot \alpha$.

The third group of experiments concerns the comparison of the detection power of QuantTree and Incremental QuantTree, measured as the percentage of batches effectively recognized as post change when drawn from the post-change distribution ϕ_1. We show that an Incremental QuantTree starting with a small training set TR_{small} has a performance comparable with that of a QuantTree trained on a significantly larger training set.

Finally we compare the performance of Incremental QuantTree and H-CPM. As H-CPM does not require a training set, we confront it with an Incremental QuantTree having access to an extremely reduced training set, composed of only 4 batches.

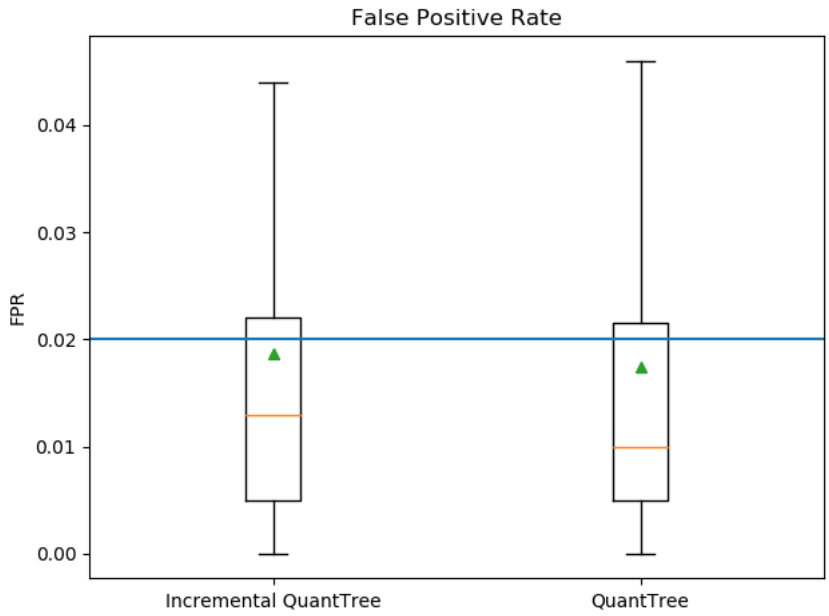

Figure 5.1: False Positive Rate

5.4 Comments on the results

We report here the results of our experiments. We dedicate a paragraph to the analysis of the performance of Incremental QuantTree for all the figures of merit discussed before. All our experimets have bee performed using $d = 8$ dimensions, $\nu = 32$ bins per batch $M = 4 \cdot \nu$ as the size for small training sets and $N = 100 \cdot \nu$ as the size of the large ones, $SKL = 2$ as the change of the magnitude and $\alpha = 0.02$ as the desired value for the False Positive Rate, in order to guarantee a desired $ARL_0 = \frac{\nu}{\alpha} = 1600$.

False Positive Rate We show here the comparison between an Incremental QuantTree trained with a small training set $TR_{small} = \{w_1, w_2, ..w_M\}$ and a QuantTree trained with a larger training set $TR = \{w_1, w_2, .., w_N\}$. We then use both algorithms to analyze the False Positive Rate, according to the described procedure. Figure 5.1 shows that Incremental QuantTree is able to control the FPR as QuantTree does.

Detection Power We compare the detection power of QuantTree and Incremental QuantTree. To this end we train a QuantTree with a training set $TR = \{w_1, w_2, .., w_N\}$, and an Incremental QuantTree with a subset a training set $TR_{small} \subset TR$. We used the $N - M$ observations belonging to TR and not to TR_{small} to iteratively update the histogram h of Incremental QuantTree. We then compared the

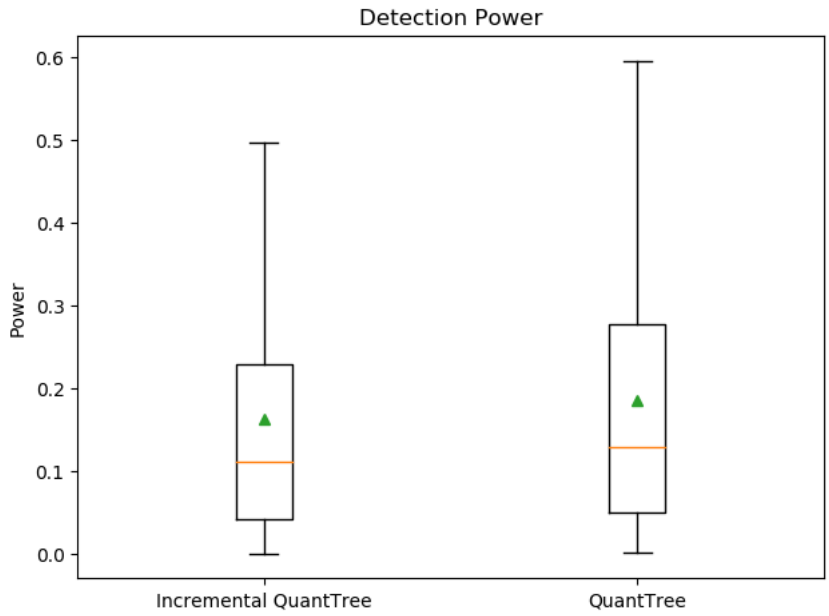

Figure 5.2: Detection Power

detection power on a series of batches drawn from a different distribution ϕ_1, according to the formula described in Section 5.2 Figure 5.2 shows that the Detection Power of Incremental QuantTree is in line with that of QuantTree, even when the size of the training set initially provided to IQT is very small.

Average Run Lenght We compare here the Average Run Lenght (ARL_0) of QuantTree and Incremental QuantTree. To this end we provided QuantTree with a large training set $TR = \{w_1, w_2, ..w_N\}$, and Incremental QuantTree with a subset $TR_{small} = \{w_1, w_2, .., w_M\}$, where $N < M$. We then tested both algorithms on a data stream $x_1, x_2... \sim \phi_0$, comparing the average stop time of the two algorithms. Figure 5.3 shows that both algorithms can be used to control the ARL_0. In particular, Incremental QuantTree is able to control the ARL_0, even when trained with very small Training set.

We then compare the ARL_0 of an Incremental QuantTree trained with the same small training set $TR_{small} = \{w_1, w_2, ..w_M\}$ and of an $H - CPM$, which does not require a training set. We show in Figure 5.4 that the Incremental QuantTree has a better control over the ARL_0.

Detection Delay We compare here the detection delay of QuantTree and Incremental QuantTree, according to the framework descried in Section 5.2. We use the same initial training set $TR_{small} = \{w_1, w_2, ..w_M\}$ and TR used for the experiment

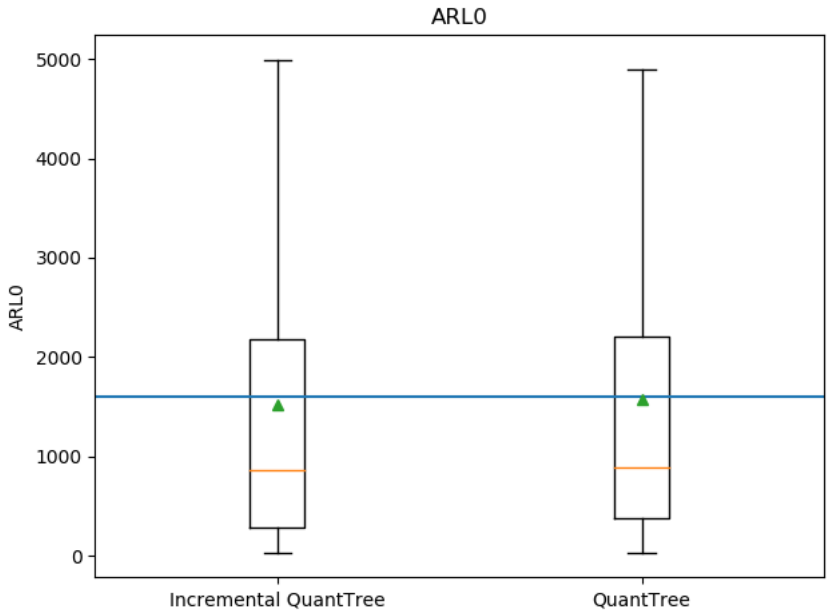

Figure 5.3: Average Run Lenght before a false alarm, QuantTree and Incremental QuantTree. $N = 3200$, $D = 8$, $M = 128$, the desired value for the ARL_0 is 1800.. The green triangle represents the empirical mean of the two distributions, while the yellow line represents the median.

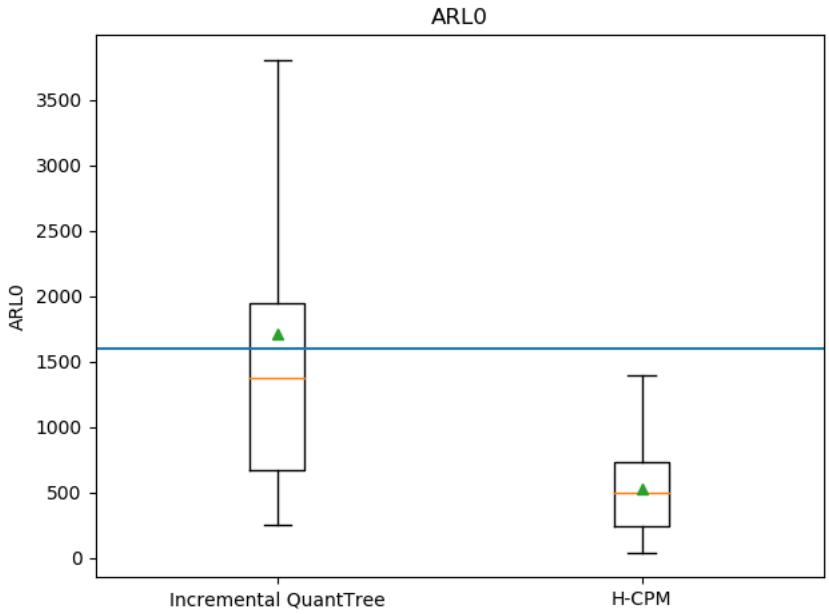

Figure 5.4: Average Run Lenght: Incremental QuantTree and H-CPM. Incremental QuantTree controls better the Average Run lenght.

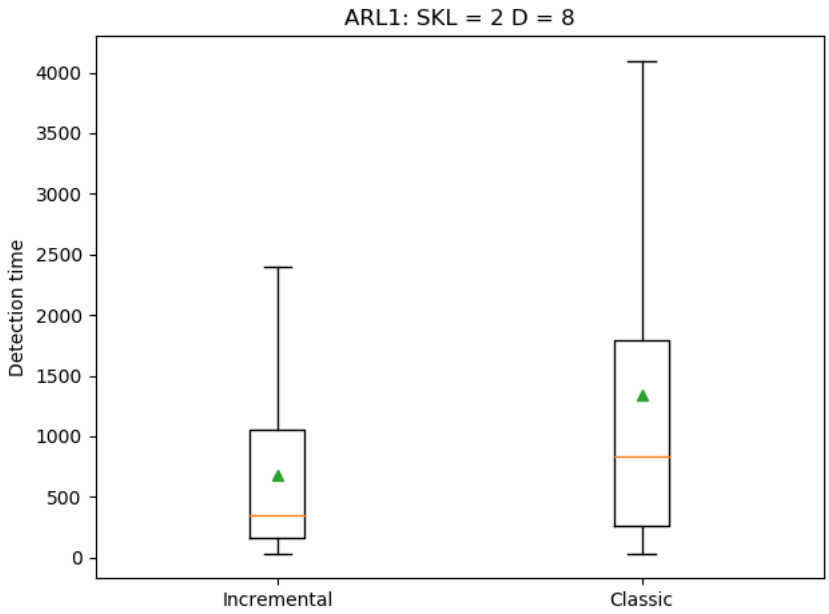

Figure 5.5: Detection Delay: Incremental QuantTree and QuantTree. The green triangle represents the empirical mean of the two distributions, while the yellow line represents the median.

for the ARL_0 and we continuously update the Icremental QuantTree histogram with the observations $\{w_{M+1}, w_{m+2}...w_N\}$ of TR. We then compare the Run Lenght of the two algorithms on a data stream $x_1, x_2..$, where each observation $x_i \sim \phi_1$, and compute the Detection Delay as the average run lenght of the two algorithms before a detection. We show in Figure 5.5 that the performance of an Incremental QuantTree initially trained with very few data, is comparable to that of a QuantTree trained with a considerable training set

Chapter 6

Conclusions

This thesis addressed the problem of change detection on data stream. Starting from an analysis of the literature, we showed that there are only few algorithms that can monitor multivariate data streams, whose distribution ϕ_0 is unknown.

We propose a novel change detection algorithm, Incremental QuantTree, able to start monitoring a data stream without having access to a large training set. We designed the algorithm as an extension of QuantTree, a histogram-based algorithm for batch-wise change detection on multivariate data.

Our contributes can be divided into three main components: a series of techniques to enable an algorithm based on a QuantTree histogram to analyse online data streams, a theoretical analysis of the algorithm and of its properties, and a series of experiments assessing the performance of the algorithm. In particular, we first introduced an algorithm to update a QuantTree histogram h upon the arrival of new data (Section 4.2) and one (Section 4.3) to recompute the thresholds whenever the histogram h is modified.

We then exploited a property of the thresholds computation algorithm of QuantTree to propose a novel and faster threshold computation algorithm for large N, and proved that the results of the two algorithms are asymptotically equal (Section 4.5). Furthermore, we studied the performance of Incremental QuantTree Neural Network based on the statistic used, and we suggested to use the Total Variation (Section 4.6). Then we discussed a possible combination of an EWMA chart with an Incremental QuantTree as an alternative change detection algorithm for data streams (Section 4.7).

Finally we assessed the performance of Incremental QuantTree, proving that it is possible to extend QuantTree to work with extremely small training sets, controlling the False Alarm Rate and reaching a detection power in line with that of a QuantTree trained on a large training set.

Future works involve a further investigation in the use of an EWMA chart to monitor an Incremental QuantTree histogram, and the use of ensemble learners using QuantTree and EWMA charts. Finally, we believe that there is the possibility of further analysing the use of Incremental QuantTree as an online learner, possibly making it able to detect multiple changes in a fully online manner, re-initializing upon the detection of a change.

Bibliography

[1] Ashbindu Singh. Review article digital change detection techniques using remotely-sensed data. *International journal of remote sensing*, 10(6):989–1003, 1989.

[2] Daniel Kifer. *Change Detection on Streams*, pages 317–321. Springer US, Boston, MA, 2009.

[3] Indrė Žliobaitė, Mykola Pechenizkiy, and João Gama. *An Overview of Concept Drift Applications*, volume 16, pages 91–114. 01 2016.

[4] Andrea Dal Pozzolo, Giacomo Boracchi, Olivier Caelen, Cesare Alippi, and Gianluca Bontempi. Credit card fraud detection: a realistic modeling and a novel learning strategy. *IEEE transactions on neural networks and learning systems*, 29(8):3784–3797, 2017.

[5] Fabrizio Carcillo, Andrea Dal Pozzolo, Yann-Aël Le Borgne, Olivier Caelen, Yannis Mazzer, and Gianluca Bontempi. Scarff: A scalable framework for streaming credit card fraud detection with spark. *Information Fusion*, 41:182–194, 2018.

[6] Rodolfo C. Cavalcante and Adriano L. I. Oliveira. An approach to handle concept drift in financial time series based on extreme learning machines and explicit drift detection. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2015.

[7] Jan Zenisek, Florian Holzinger, and Michael Affenzeller. Machine learning based concept drift detection for predictive maintenance. *Computers Industrial Engineering*, 137:106031, 2019.

[8] Yanjun Li, Zhi Wang, and Yeqiong Song. Wireless sensor network design for wildfire monitoring. In *2006 6th World Congress on Intelligent Control and Automation*, volume 1, pages 109–113, 2006.

[9] A. G. Tartakovsky, B. L. Rozovskii, R. B. Blazek, and Hongjoong Kim. A novel approach to detection of intrusions in computer networks via adaptive sequential and batch-sequential change-point detection methods. *IEEE Transactions on Signal Processing*, 54(9):3372–3382, 2006.

[10] Daniel Kifer, Shai Ben-David, and Johannes Gehrke. Detecting change in data streams. pages 180–191, 04 2004.

[11] Giacomo Boracchi, Diego Carrera, Cristiano Cervellera, and Danilo Maccio. Quanttree: histograms for change detection in multivariate data streams. In *International Conference on Machine Learning*, pages 639–648. PMLR, 2018.

[12] Gordon J. Ross, Dimitris K. Tasoulis, and Niall M. Adams. Nonparametric monitoring of data streams for changes in location and scale. *Technometrics*, 53(4):379–389, 2011.

[13] Gordon J. Ross and Niall M. Adams. Two nonparametric control charts for detecting arbitrary distribution changes. *Journal of Quality Technology*, 44(2):102–116, 2012.

[14] Shuang Li, Yao Xie, Hanjun Dai, and Le Song. M-statistic for kernel change-point detection. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28, pages 3366–3374. Curran Associates, Inc., 2015.

[15] Gordon J. Ross, Dimitris K. Tasoulis, and Niall M. Adams. Online annotation and prediction for regime switching data streams. In *Proceedings of the 2009 ACM Symposium on Applied Computing*, SAC '09, page 1501–1505, New York, NY, USA, 2009. Association for Computing Machinery.

[16] Rosana Noronha Gemaque, Albert França Josuá Costa, Rafael Giusti, and Eulanda Miranda Dos Santos. An overview of unsupervised drift detection methods. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 10(6):e1381, 2020.

[17] William H. Woodall. Control charts based on attribute data: Bibliography and review. *Journal of Quality Technology*, 29(2):172–183, 1997.

[18] Nicholas P. Cheremisinoff. S. In Nicholas P. Cheremisinoff, editor, *Condensed Encyclopedia of Polymer Engineering Terms*, pages 268–300. Butterworth-Heinemann, Boston, 2001.

[19] Douglas M Hawkins, Peihua Qiu, and Chang Wook Kang. The changepoint model for statistical process control. *Journal of quality technology*, 35(4):355–366, 2003.

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

[21] Pierre Granjon. The cusum algorithm-a small review. 2013.

[22] William H. Woodall. Control charts based on attribute data: Bibliography and review. *Journal of Quality Technology*, 29(2):172–183, 1997.

[23] Marcus B Perry. The exponentially weighted moving average. *Wiley Encyclopedia of Operations Research and Management Science*, 2010.

[24] S. W. Roberts. Control chart tests based on geometric moving averages. *Technometrics*, 1(3):239–250, 1959.

[25] Ludmila I Kuncheva. Change detection in streaming multivariate data using likelihood detectors. *IEEE transactions on knowledge and data engineering*, 25(5):1175–1180, 2011.

[26] Cesare Alippi, Giacomo Boracchi, Diego Carrera, and Manuel Roveri. Change detection in multivariate datastreams: Likelihood and detectability loss, 2016.

[27] A. N. Pettitt. A non-parametric approach to the change-point problem. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(2):126–135, 1979.

[28] Gordon J Ross, Dimitris K Tasoulis, and Niall M Adams. Nonparametric monitoring of data streams for changes in location and scale. *Technometrics*, 53(4):379–389, 2011.

[29] Dengsheng Lu, Paul Mausel, Eduardo Brondizio, and Emilio Moran. Change detection techniques. *International journal of remote sensing*, 25(12):2365–2401, 2004.

[30] Arthur Gretton, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 13(25):723–773, 2012.

[31] Gordon J Ross, Niall M Adams, Dimitris K Tasoulis, and David J Hand. Exponentially weighted moving average charts for detecting concept drift. *Pattern recognition letters*, 33(2):191–198, 2012.

[32] Muhammad Aslam Masood Amjad Khan Hina Khan, Saleh Farooq. Exponentially weighted moving average control charts for the process mean using exponential ratio type estimator. *ournal of Probability and Statistics*, 2018.

[33] H. Hotelling. Multivariate quality control-illustrated by the air testing of sample bombsights. 1947.

[34] Nola D. Tracy, John C. Young, and Robert L. Mason. Multivariate control charts for individual observations. *Journal of Quality Technology*, 24(2):88–95, 1992.

[35] K. D Zamba and Douglas M Hawkins. A multivariate change-point model for statistical process control. *Technometrics*, 48(4):539–549, 2006.

[36] Diego Carrera and Giacomo Boracchi. Generating high-dimensional datastreams for change detection. *Big data research*, 11:11–21, 2018.