Politecnico di Torino

Corso di Laurea Magistrale
in Ingegneria Informatica (Computer Engineering)

Master thesis

Anomaly detection through stream processing

Supervisor:
prof. Paolo Garza

Candidate:
Paolo Cristofanelli

December 2017

Abstract

Anomaly detection is a crucial problem in many contexts. This project is about detecting abnormal events, which patterns are unknown, in the context of streaming data. For this purpose, we selected two data-driven methods (*Adwin* and *Page-Hinkley*) able to monitor for changes in a stream. With the help of several data-sets, we tested the performances and the main features of these algorithms. Evaluating and comparing the performances when there is not a well-defined ground truth available is complex. Thus, we have introduced some modification on the data-sets that allowed a more rigorous evaluation of the algorithm's performances while enabling comparisons. The experiments demonstrate that the Adwin is more flexible than the Page-Hinkley. Overall its performances are better that the Page-Hinkley ones. However, the Page-Hinkley is less expensive in terms of computational cost and memory. Moreover, it is able to obtain better performances in cases where it is possible to clearly separate the normal behavior of the stream from the abnormal one. Finally, the Adwin is not able to handle single outliers while the Page-Hinkley does.

Keywords: streaming data, anomaly detection, outlier detection

Acknowledgements

I would like to thank my thesis supervisor at Chalmers, Marina Papatriantafilou and the assistant supervisor, Ioannis Nikolakopoulos for the help provided during this work. Additionally, I would like to thank the supervisor at Polytechnic of Turin, Paolo Garza, for having followed me through the work.
A special thank goes to my family that supported me by every possible means through all my university studies.

Paolo Cristofanelli, Gothenburg, October 2017

Contents

List of Figures

List of Tables

1

Introduction

We are living in a world where the advance of technologies such as smartphones, computers, sensors, and Internet of things seems unstoppable. These technologies are with us almost every moment of our lives and the interaction between us (or the environment) and them creates continuous streams of data. Just think about the financial market, the water management, interconnected cities and so on. This data often contains valuable information that we must be able to process in order to extract meaningful knowledge that may help us make decisions. Consider the example of a sensor network monitoring the temperature of different phases of a chemical process. Each sensor will produce a stream of data where each record contains the temperature recorded by the sensor and the time of the measurement. In this case, we want to be able to analyze these readings in a continuous fashion in order to get information that we may use to improve the process and/or to recognize some abnormal situation that can cause some sort of damage.

1.1 Purpose

The aim of this thesis is to study, implement and evaluate streaming anomaly and outliers detection methods. These methods have been studied from the literature. The purpose is to understand the properties of these algorithms and their suitability for detecting different types of anomalies. Moreover, this study focuses on data-driven methods for anomaly detection and not on approaches where the anomalies that we are searching have known signature.

After that, we are interested in comparing the results and evaluate the methods with different metrics, such as delay in detection, throughput, latency, false positive and true positive rate.

The final goal is to have a detailed view of the algorithms' behavior and performances and, in some cases, suggest some modification. We will use Apache Flink [1] as streaming processing engine and, through its Datastream API, we will define the user-defined transformations implementing our methods.

1.2 Context

Handling streams of data raises a series of challenges. Since a stream is unbounded by definition it is not possible to store all the related information. So, there is a need for methods able to operate in an online fashion, with stringent latency and bandwidth constraints.

It may happen that, during a stream, the incoming data can invalidate existing patterns or/and introduce new ones. Also, it is possible that the data is being collected from different sensors in a network. For these reasons, streaming methods should be both adaptive (able to learn the new patterns) and distributed (able to improve the scalability of the problem by moving the computation to the nodes).

Consider for example the case of the sensor network monitoring a chemical process. Each sensor can perform some basic analysis over the collected data before sending it to a central elaboration unit. In this way we are reducing the amount of data sent over the network while moving some of the computation to the nodes (i.e. sensors).

More specifically, this thesis analyzes algorithms that address the problem of detecting data changes, outliers and anomalies in a data stream. It is important to define and clarify the differences between them: an outlier is one or few values that are considerably different from the others. On the other hand, an anomaly is a longer sequence of values, generated by an underlying distribution, that deviates from the normal one. After the end of the anomaly, our stream will return to its standard behavior. Finally, a data change is defined as a *semi-permanent* variation of the stream´s behavior, i.e. the standard behavior of the stream changes. Figure 1.1 shows the difference between an anomaly and a change in the case of mono-dimensional data.

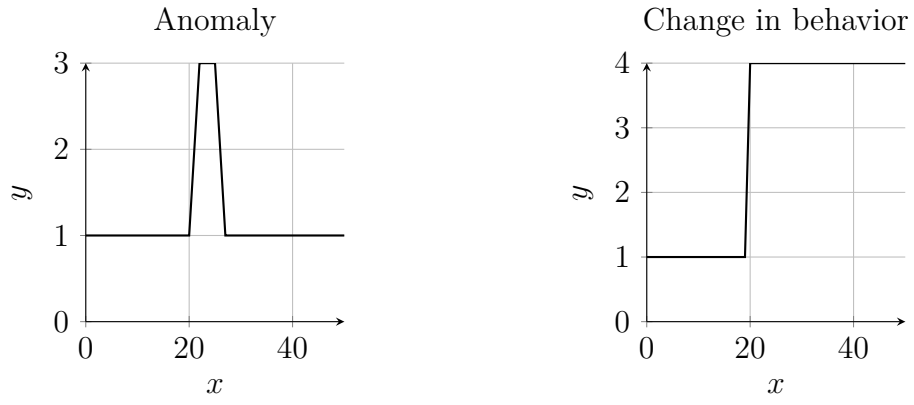


Figure 1.1: Difference between an anomaly and a change in behavior

Detecting outliers, anomalies and changes is a critical task since they are often correlated with the interesting events of the system. For example, in case of abnormality or malfunction, a sensor will give readings that deviate significantly from the aver-

age. If our methods are able to detect this anomaly it will be possible to identify and replace the faulty sensor.

Another interesting application is represented by the Intrusion Detection Systems. In this case, the purpose is to distinguish between normal network traffic and the suspicious one. For example, one of the most common way to perform a denial of service attack is the ping flooding. It consists in sending a huge amount of ping packets to a target server. Therefore, the server consumes all its resources to process the ping packets and it is not able to provide other services. By using an algorithm that can identify when the number of ping packets is abnormal, it is possible to identify and react to the attack.

Outliers and anomalies should not be confused with noise. The firsts two represent events that are useful to the study of the data. On the other hand, noise is unwanted data and does not have any utility to the analysis [7]. If the data presents some noise then, before applying an anomaly detection method, the use of some noise removal technique is required.

The thesis works towards understanding properties of the methods. For this purpose, different data sets are used, with different properties, that are supposed to shed light to the properties and complexity of the methods with regards to detecting different types of anomalies.

As previously mentioned, these methods will be implemented inside Apache Flink using its DataStream APIs. Flink is able to process streaming of data while providing interesting properties such as scalability, fault tolerance and parallelism management.

1.3 Overview of the methods

This section provides a short description of the methods that are going to be analyzed in this work. The first analyzed method is the Adwin algorithm [6]. It can be used for three main purposes: to detect changes, to store updated statistics of the stream and to keep some data that allows rebuilding the model after a change. The Adwin algorithm keeps a data structure in the memory storing the most recent values of the stream.

After that, the Page-Hinkley approach will be considered [Page,1954; Hinkley, 1969, 1970, 1971; Basseville, 1988]. This technique considers a variable that stores the cumulative difference between the observed values and their mean till the current moment. It is possible to use this variable in order to detect changes. This test does not require to store any data-structure in memory and it is very fast. On the other hand, it requires defining a threshold and the magnitude of accepted change (i.e. if the difference between observed value and the mean is lower than the accepted

change we do not increase the variable storing the cumulative difference).

1.4 Thesis Outline

The thesis is composed of seven main chapters. In the **Problem Description** the theory´s fundamentals that are needed to understand the next chapters are given. The **Theory**´s chapter provides a more accurate explanation of the methods, underlining the idea and the theory behind them. After that, the **Implementation**´s chapter illustrates how the methods have been implemented in Java and inside the streaming processing engine. This chapter also defines how the data-sets have been modified in order to evaluate the performances of the methods. The following chapter, **Results**, has three purposes: the first one is to clarify how the algorithms have been evaluated. The second goal is to illustrate and explain their behavior. For this reason, it will analyze in detail the results obtained by the algorithms on the available data-sets. The third purpose is to collect measurements about latency, throughput, false positives and false negatives. Then the **Discussion** chapter sums up and examines the results. Finally, the **Conclusion** chapter presents the major findings of the thesis.

2

Problem Description

2.1 Stream Processing

The following section contains a brief description of the stream processing paradigm.

The starting point is a *data source* that produces streaming data, usually seen as a sequence of *tuples*. On the opposite a *data sink* is a consumer of results. A tuple is something similar to a database row since it has a set of named and typed attributes. These attributes are defined by the *data schema*. Formally a stream is an unbounded sequence of tuples t_0, $t_1 \ldots t_n$sharing the same schema.

Due to the unbounded nature of the problem, it is not possible to store all the data. There are two main alternatives: the first one consists in creating summaries and synopsis that are used to perform queries. The second one is to use a window-based stream processing. Only by storing all the data it would be possible to have exact results. Since this is not possible the results will be approximate. There is a trade-off between the size of the stored synopsis and the level of approximation of the query. There are two types of approximation:

- ϵ Approximation: the answer is correct with some small fraction ϵ of error
- (ϵ, δ) Approximation; the answer is within $1 \pm \epsilon$ of the correct result, with probability $1 - \delta$

As previously said, the level of approximation influences the size of the synopsis So a small ϵ or/and δ would cause a greater memory usage.

The other common approach used in order to handle data streams consists in keeping a *sliding window*, i.e. a structure that stores just the recently seen data and discards the old ones. This approach needs to define the size i and the advance j of the window. The window will keep the most recent i values of the stream and it will be updated every j values.

There are two ways for defining the size: the first one sets the number of stored elements. The second one is based upon the concept of *timestamp*, the window will store all the tuples that are not older than a certain value (a parameter, equivalent to the size).

2.2 Apache Flink - Stream processing engine

As previously mentioned the SPE (Stream Processing Engine) that we are going to use is Apache Flink [1]. Flink has been specifically designed for streaming application. It relies on a *streaming execution model*. Clearly, an alignment between the execution model and the type of processed data creates many advantages. The purpose of Flink is to handle distributed stream environment while providing accurate results with high throughput and low latency. Moreover, Flink handles flexible windowing (based on timestamp, count...) and event time semantics. It supports three different notions of time:

1. Processing time, i.e. the system time of the machine that is executing the operations.

2. Event time, i.e. the time that each individual event occurred on its producing device. This time is usually embedded within the record.

3. Ingestion time, i.e. the time that events enter in Flink.

In this thesis, the *event time* notion has been considered. It allows extracting a timestamp from each tuple and assigning it to the record.

Consider an example where the tuples refer to some kind of measurements. Each tuple has the timestamp of the measurement and its value. The window should contain the most recent measurement even if the data is out of order (i.e. a tuple with an older timestamp is processed after a tuple with a more recent timestamp). With this notion, it is possible to assign to each input tuple their embedded timestamp. So Flink can use it to define a temporal order of the data.

If the notion of ingestion time is used and the tuples arrive out of order (with respect to their embedded timestamp) Flink will not be able to recognize the correct order. On the other hand, when using the event time notion Flink uses a *watermark* mechanism that is able to deal with out of order data. Basically, a watermark guarantees that all the events up to a certain timestamp have arrived.

2.3 Outlier/Anomalies

Consider a stream S as a list of sequences $<S_1, S_2... S_n>$ where each sequence S_i is generated by a distribution D_i ([6, p.66]). The purpose of the change detector algorithms is to recognize when the distribution D_{i+1} is different from D_i. Whenever this happens the information extracted from the distribution D_i does not reflect anymore the state of the stream. So the algorithm needs to discard D_i.

One issue faced by these algorithms is to distinguish whether the new data is different because of some noise or because the stream is changing its underlying distribution. Usually, the best approach is to wait since the algorithms have processed enough data and they are able to decide. This means that we have to take into account a delay of detection.

Outliers/Anomaly detection in a distributed streaming environment has many applications. Possible examples are the aforementioned detection of faulty sensors and intrusion detection systems.

There are several definitions of outliers, one of the most known is the *Distance-based outliers* introduced by Knorr [12]: given parameters k and R, an object is a distance-based outlier if less than k objects of the input data set lie within distance R from it. This definition is using the intuitive idea that an outlier is just an observation sufficiently far from the others. One of the main advantages of this interpretation is that does not need any prior knowledge of our data.

Distance-based outliers are very useful when the user is able to specify the parameter k accurately. Unfortunately, this is not possible in all the cases where the data space presents regions with different densities.

For this reason, different definitions of outliers have been introduced. For example, the *local metrics-based outliers'* definition [16] enables the usage of automatic cut-offs for the outliers, avoiding the necessity of choosing a threshold.

2.4 Limitations

This thesis will consider only single-dimensional data. More specifically, in our data-sets, the values considered refer to electricity consumption/demand. Another important aspect to underline is that anomaly detection approaches vary significantly from case to case. Also, there are differences between the definition of an anomaly when considering different data-sets. For these reasons, it is usually hard to compare methods when tested on different data-sets [14]. This is to say that it is not possible to generalize all the obtained results. However, some characteristics

of the analyzed algorithms may be retrieved in all the situations.

2.5 Evaluation Criteria

We do not have any rigorous information about the anomalies/outliers present in the data-sets. For this reason, it has been decided to proceed with two approaches. Firstly, each data-set has been analyzed by means of the selected algorithms. By studying case per case, we will gain an in-depth knowledge of the data-sets and of the methods' behaviors. After that, the data-sets have been manipulated by introducing several types of anomalies. Having accurate information about the inserted anomalies allows running the algorithms several times (while varying their parameters) and measuring metrics such as: False Positives, True Positives, False Negatives, Latency, Throughput and Delay in detection. Since these metrics are related to how the anomalies have been inserted they will be explained in detail in the first section of the Results chapter. These metrics have been defined in a way such that it is possible to compare the results.

2.6 Data-sets

These data-sets have been collected during the Finesce project [3]. For the goals of this thesis each record is treated as streaming tuple. The data-sets have been collected in two trial-sites: Aachen and Terni.

2.6.1 Electricity consumption in a manufacturing environment

The data-sets collected from the first trial site contain the power consumption of several machines of a factory located in Aachen, Germany. These data-sets represents a more general case of electricity consumption in a manufacturing environment. In this thesis, we will refer to each data-set with the names of the machines that are: "Robot Welter", "Bending", "Laser Shaping", "Laser Cutting" and "Smart Meter".

The behavior and the values of these files are quite similar. Nevertheless, the "Smart Meter" data-set is an exception. It is not related to the energy consumption of a machine but keeps track of the overall power consumption of the factory (a cumulative sum).

Each record in the data-set consists of a measurement of the machine´s state and has two information: the timestamp and the related power consumption, expressed

in Megawatt hour (MWh).

Usually, there is a reading for each minute. However, there may be intervals without any measurement. For this reason, whenever a temporal window over the stream is defined the average of the values inside it should be considered (and not their sum).

The first four data-sets show higher power consumption during the daytime while not consuming any energy overnight. For this reason, they show a high correlation. We have considered two data-sets, Laser Shaping and Robot Welter. Then we aggregated them for days and computed the *Pearson correlation coefficient* over each day using the formula 2.1 [17]. After that, we have taken the average of the correlation between the days. The final result is 0.67 and it is very close to 0.7, that is the required value for a *strong* correlation.

$$\rho(X,Y) = \frac{nS_{\mathrm{xy}} - S_{\mathrm{x}}S_{\mathrm{y}}}{\sqrt{nS_{\mathrm{xx}} - {S_{\mathrm{x}}}^2}\sqrt{nS_{\mathrm{yy}} - {S_{\mathrm{y}}}^2}} \tag{2.1}$$

where

$$S_{\mathrm{xy}} = \sum_{t=1}^{n} x_{\mathrm{t}} y_{\mathrm{t}}, \qquad S_{\mathrm{xx}} = \sum_{t=1}^{n} {x_{\mathrm{t}}}^2, \qquad S_{\mathrm{yy}} = \sum_{t=1}^{n} {x_{\mathrm{y}}}^2, \qquad S_{\mathrm{yy}} = \sum_{t=1}^{n} {x_{\mathrm{y}}}^2, \tag{2.2}$$

$$S_{\mathrm{y}} = \sum_{t=1}^{n} x_{\mathrm{y}}, \qquad S_{\mathrm{x}} = \sum_{t=1}^{n} x_{\mathrm{x}} \tag{2.3}$$

The formula 2.1 has been derived from the standard one in order to handle stream series. It allows to compute the ρ incrementally by maintaining the sum, square sum and inner product of each series.

2.6.2 Power demand/supply profiles of customers in different energy sectors

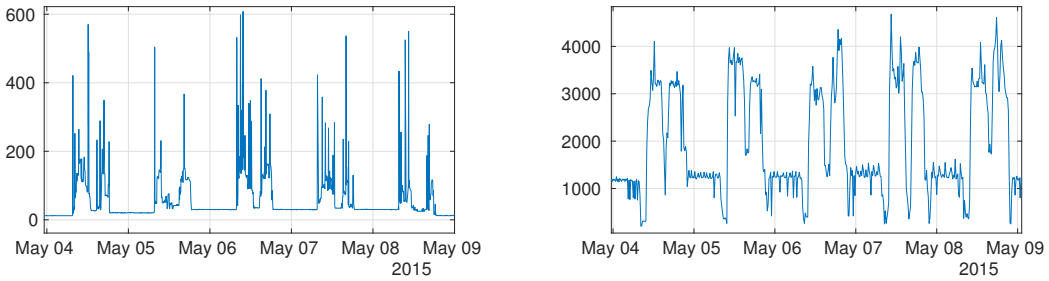
The other group of data-set collects information about the power demand of different offices and industries situated in Terni, Italy. These data-sets represent a more general case of power demand/supply profiles of customers in different energy sectors. In this case the name of the data-set is taken from the type of the customer. They are: "Commercial 1" and "Office 1".

Similarly to the Aachen´s data-sets also these files consist of measurements with a timestamp and a value. However, in this case, the measurements have not been taken every fixed interval of time. In order to understand what it is happening we need to perform some kind of temporal aggregation over the stream generated by these data-sets.

For example, consider what would happen in a situation where all the readings have the same power demand value but the first one hundred have been taken in 10 minutes while the next fifty readings are condensed in the next 20 seconds. We are clearly witnessing an increment of the power demand. However, without a temporal aggregation, our algorithms will just see equal values and would not detect any change.

In these data-sets, it is possible to observe a periodic behavior. Nevertheless, as shown in figure 2.1, the periodic behavior between the files is not so similar. In both the graphs of the figure we aggregated our values using a window of one hour (so each point represents the sum of all the reading inside that hour).

A few of these data-sets present some interesting situations, like outliers and anomalies, that we will analyze in the results section.



(a) Customer Industrial 1

(b) Customer Commercial 1

Figure 2.1: Section of the power consumption of two data-sets

3

Theory

This chapter gives the reader a theoretical background about the streaming methods that have been considered in this thesis. These methods have been selected since they are able to monitor for changes in a data stream. Moreover, these approaches are adaptive, i.e. they are able to adjust when the behavior of the stream change. The problem of detecting anomalies and changes in a stream is addressed also by other methods such as the *OnePassSample* [15]. Its authors claim that this algorithm is oblivios of the underlying data distribution and inexpensive in terms of computational cost and memory. Moreover, it has performances comparable to the Adwin´s ones. Another interesting method has been introduced by Shai Ben-David *et Al.* [5]. This method aims to provide a reliable detection of change as well as a meaningful description of them. In addition this technique does not need any parameter.

3.1 ADWIN

The Adwin algorithm has been introduced to handle three main tasks: keep an updated average of the stream, detect changes in the nature of the data and store data that allows to revise the stream's model after a change.

The Adwin uses a window of variable size. This window adjusts its length whenever a change in the average of values inside the windows is detected. The average inside the window can be considered as a reliable measure of the current average of the stream.

Using a window of fixed length would be the simplest approach. However, in this case, the size of the window has to be specified as a parameter. In order to make an optimal choice, the rate of change should be known. A too large size would lead to a window that does not reflect the current state of the stream. On the other hand, a small size means less example to work on and therefore, less stability. Some methods use a *decay function* (e.g. [8]) that helps them reduce the importance of the past values. Nevertheless, the problem of knowing the rate of change still exists

when choosing the strength of the decay function.

Approaches based on windows of variable size are rare and most of them are based on an heuristic approach, therefore they are not able to guarantee rigorous performance bounds. On the contrary, despite using a window of variable size the Adwin algorithm is able to provide theoretical guarantees.

The main idea behind the algorithm is to keep a sliding window W over the stream. Then it checks once in a while if inside W there are two "large-enough" sub-windows (called W_0 and W_1) that exhibit "different enough" statistics. If this happens the algorithm shrinks the window by dropping the oldest values. In this way it is possible to maintain statistics over the window that are a reliable approximation of the stream's current behavior.

The formula used to decide if the sub-windows are "different enough" enables the definition of rigorous performance bounds.

The only parameter of the algorithm is a confidence value $\delta \in (0, 1)$. This parameter plays an important role in the definition of the "different enough" and in the theoretical guarantees of the the Adwin.

For each possible split of the window in two sub-window such that $W = W_0 \cdot W_1$ the algorithm compare the sub-windows' statistics. If the comparison says that the statistics are different enough then the last values (i.e. the oldest ones) of the window will be dropped.

The Adwin defines n as the size of the window, n, n_0 and n_1 as the sizes of the window and the two sub-windows, σ^2 as the overall variance of the values stored in the window, μ_{W_0} and μ_{W_1} as the averages of the two sub-windows. Then it is possible to define the following equations:

$$m = \frac{1}{\frac{1}{n_0} + \frac{1}{n_1}} \qquad and \qquad \delta' = \frac{\delta}{\ln n} \tag{3.1}$$

$$\epsilon_{\text{cut}} = \sqrt{\frac{2}{m} \cdot \sigma^2{}_{\text{W}} \cdot \log \frac{2}{\delta'}} + \frac{2}{3 \cdot m} \log \frac{2}{\delta'} \tag{3.2}$$

$$|\hat{\mu}_{W_1} - \hat{\mu}_{W_0}| \geq \epsilon_{\text{cut}} \tag{3.3}$$

If the inequality 3.3 holds it is possible to say that the two sub-windows are different enough. This means that the distribution that generates the data is changing. So

the algorithm drops the older input tuples that do not reflect anymore the current stream values. Depending on our goals, we need to alert an anomaly or train a new decision model with just the data referring to the new distribution. Formula 3.2 is composed by two additive terms: the first one exploits the fact that $|\mu_{W_0} - \mu_{W_1}|$ tends to a normal distribution for large windows sizes. The second one is useful when the window size is small and it does not follow the normal distribution.

The first version of the Adwin (ADWIN0) stores all the values that are inside the window. For this tasks it needs $O(n)$ memory. Additionally, it checks for every possible split such that $W = W_0 \cdot W_1$. This is computationally expensive (since there are $O(n)$ possible splits).

The author provides a better implementation of the Adwin. This new version does not keep the window explicitly. The window is compressed by using a variant of the exponential histogram technique [9] [10, p.53-54].

The idea of this technique is to consider a simplified data stream model where each element comes from a data source and it is either 0 or 1. The goal is to count the number of 1's over a window of size N using just $O(\log N)$ space. The solution consists in defining some buckets to store the data. Each bucket is associated with a timestamp so that it is possible to decide when it falls outside the window. Every time a new input tuple arrives a new bucket of size one is created. If there are more than a certain number of buckets of the same size the algorithm merges the two oldest ones (parents). The merged bucket takes the oldest timestamp of its parents and combines their statistics about the number of 1's.

This approach defines two variables: TOTAL (that stores the total size of the buckets) and LAST (that stores the size of the oldest bucket). The estimation of the number of 1's in the sliding window is : TOTAL - LAST/2. Before computing the estimation the algorithm checks for buckets which timestamp falls outside the sliding window. Then it drops these buckets.

The number of the buckets of the same size is a parameter of the method and in the case of the Adwin it is called *M*. By setting this parameter we also decide the amount of used memory and the number of checks. However, unlike the choice of the of the window's size in the approaches with a window of fixed length, the parameter M does not reflect any assumption about the rate of change.

By using a variant of this technique the Adwin is able to keep statistics about a window of length W using only $O(logW)$ memory and $O(logW)$ processing time per item, rather than the $O(W)$ one expects from a naïve implementation. The data-structure that the Adwin uses has the following properties :

- *It uses* $O(M \cdot \log \frac{W}{M})$ *memory word*
- *It can process the arrival of a new element in O(1) amortized time and* $O(\log W)$

worst-case time

- *It can provide the exact counts of 1's for all the sub-windows whose lengths are of the form* $(1 + \frac{1}{M})^i$*, in O(1) time per query.*

As already mentioned, one of the main feature of the Adwin algorithm is that it is able to ensure formal theoretical guarantees:

1. (False positive rate bound) *If* μ_t *remains constant within W, the probability that the Adwin shrinks the window at this step is at most* $\frac{M}{n} \cdot \log \frac{n}{M} \cdot \delta$

2. (False negative rate bound) *Suppose that for some partition of W in two sub-windows (*W_0 *and* W_1*) we have* $|\mu W_0 - \mu W_1 > 2\epsilon_{cut}|$*. Then with probability* $1 - \delta$ *the algorithm shrinks W to* W_1 *or shorter*

The proof can be found at [6, p.68-69].

3.2 Page-Hinkley

The Page-Hinkley test is a sequential analysis technique used for monitoring change detection. It is a variant of the *cumulative sum algorithm* (CUSUM) introduced by Page(1954).

The goal of the CUSUM algorithm is to give an alarm whenever the mean of the input data is significantly different then zero. Consider a stream S defined as $<x_0, x_1 ..., x_t>$ where x_t represents an input tuple with real value x at time t. Let v be a parameter of the algorithm. It is possible to define the CUSUM test as follows :

$$g_0 = 0 \tag{3.4}$$

$$g_t = max(0, g_{t-1} + (x_t - v) \tag{3.5}$$

For each new input tuple x_t the algorithm increments g_t, also defined as cumulative sum. Then it checks if g_t is greater than a user specified threshold λ. If it is the algorithm will alert a change and sets g_t=0.

The PH test is a sequential adaption of the detection of an abrupt change of a Gaussian signal [13]. Consider the stream S as defined for the CUSUM and let $\bar{x}_T$ be the average of the values observed till the current moment.

The basic idea of the PH is to compute the cumulative sum of differences (m_T) and alerting for a change whenever $\mathrm{m_T}$ goes above a user defined threshold λ (a parameter). Along with λ, the approach needs also another parameter: the magnitude of accepted change (δ). For each value $\mathrm{x_t}$ of the stream, the test adds to $\mathrm{m_T}$ the difference between $\mathrm{x_T}$ and $\bar{x}_\mathrm{T}$ as shown in formula 3.6.

$$m_\mathrm{T} = \sum_{t=1}^{T}(x_\mathrm{t} - \bar{x}_\mathrm{T} - \delta) \; where \; \bar{x} = \frac{1}{T}\sum_{t=1}^{T} x_\mathrm{t} \tag{3.6}$$

Then the PH defines $M_\mathrm{T} = min(m_\mathrm{t}, t = 1...T)$. As a final test it monitors the difference between $\mathrm{m_T}$ and $\mathrm{M_T}$. If it is above the threshold λ the algorithm alerts a change.

The parameters used by the PH are two : (δ and λ). The δ defines the magnitude of accepted change, i.e. how much two consecutive values can be different without increasing the cumulative sum. Instead, the threshold λ is defined by considering the admissible false alarm rate. With a low threshold, the algorithm would detect almost all the changes but it may incur in false alarms. On the other hand, a high threshold may lead the PH method to miss some changes.

When dealing with streams the old observations are less relevant. So it is worth considering the use of the fading factors as a forgetting mechanism. The integration of the PH with the fading factor has already been studied [10, p.75-77] [11]. A fading factor is a parameter and assumes values between [0,1]. In order to use it inside the algorithm, $\mathrm{m_T}$ is calculated as follows (for each input tuple):

$$m_\mathrm{T} = fadingFactor * m_\mathrm{T} + (x_\mathrm{t} - \bar{x}_\mathrm{T} - \delta) \tag{3.7}$$

With these formulas the test is able to detect only positive changes (i.e. the average of the stream is increasing). In the implementation chapter this approach has been complemented in order to detect also negative changes.

4

Implementation

The purpose of this chapter is to explain how the methods have been implemented and integrated in Apache Flink by means of its DataStream APIs. These APIs offer the common building blocks for data processing, like various forms of user-specified transformations (also called operators).

In this work we used version 1.3.1 of Apache Flink and 1.8 of Java. As IDE Eclipse has been selected.

Since the early operators of the program are equals for both the algorithms they will be explained first. The first operator of the program is in charge of reading the data-set and producing a stream of strings. Then these strings are parsed by means of a FlatMap operator. This transformation allows to convert the meaningful strings (the power consumption/demand measurements) of the stream into Tuple objects and discard the others (usually the ones at the beginning of the data-sets that specify the meaning of the different fields).

After that, before applying any window over the stream, it is necessary to specify the time associated with every tuple. As already mentioned in the problem description, each tuple has an embedded timestamp that refers to the time the measurement has been collected. With a specific operator Flink allows extracting this timestamp and relating it to the tuple. In this way, every time the program defines a temporal window over the stream, the tuples are ordered basing on their embedded timestamp. The windows defined by Flink are not to be confused with the Adwin one. We will see that both the Adwin and the Page-Hinkley are implemented within a map operator. So, they receive and analyze one value at a time. This is possible thanks to the streaming nature of the analyzed algorithms.

4.1 ADWIN

This section illustrates the behavior and the Java implementation of the second, improved implementation of the Adwin algorithm [2] [6, p.62-66].

4.1.1 Data structures

In the theory chapter a general explanation of the window composition was provided. This section explains how the window has been implemented in Java code.

The Adwin's window is represented by a list where each item i contains a vector. Each cell of the vector contains two doubles: the first one keeps the variance of the input tuples referring to that cell. The second one stores the sum. The cells are also called *buckets*.

Each vector is statically declared with fixed size of *maxbucket+1*, where maxbucket is the M mentioned in the 'Methods' chapter. Each bucket contains the grouped statistics of 2^i input tuples, where i is the position of the item in the list. Consider for example a bucket where the sum's statistics is 128 and it is inside the third item of the list. So the sum will refer to 2^3 input tuples and their average will be $128/8 = 16$. If the bucket had been in item 0 the average would have been 128.

In order to check for changes in the stream, the algorithm needs to divide our window/list into two contiguous sub-windows called left and right sub-windows. The left one keeps the more recent input tuples.

One of the main problems of the first version of the Adwin algorithm was that it checked all the possible divisions of the window. On the contrary, this implementation consider as cut-points only the border of the buckets, i.e. the two sub-windows' valid lengths are of the form $M \cdot (1 + 2 + ... + 2^{i-1})$.

For example, a possible composition of the two sub-windows would be: the left sub-window contains all the buckets of item 0 and one buckets of item 1. The right sub-window includes the remaining buckets of item 1 and all the buckets of the other items. The composition of the window is shown in Figure 5.1.

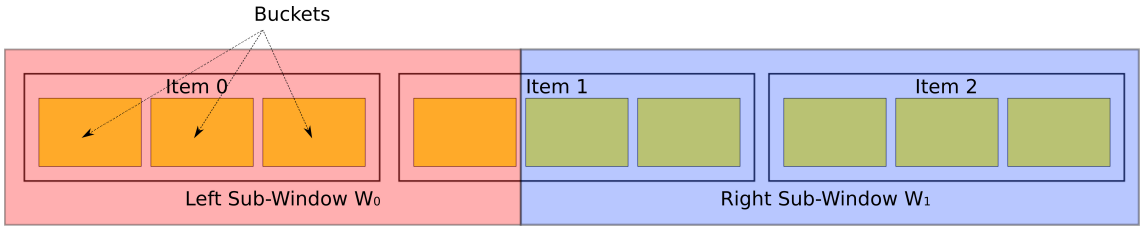


Figure 4.1: Example of window composition

Moreover, there is also another limitation to the possible splits: each one of the two sub-window must contain the statistics of a minimum number of input values. This value is another parameter used in this implementation and it is called 'minimum length of the sub-window'.

4.1.2 Algorithm

Algorithm 1 [6, p.63] shows the pseudo-code of the Adwin. It starts by initializing the window and three variables that refer to: the length of the window(WIDTH), the overall variance of all the elements inside the window (VARIANCE) and their overall sum (TOTAL).

For each input tuple it calls the SETINPUT method that inserts the new tuple inside the Adwin window, then it checks for all the possible splits (that satisfy the aforementioned constraints).

For each new input tuple the algorithm creates a bucket. Then, it updates the new bucket by setting the value of the tuple as sum and zero as variance. After that, the bucket is inserted in the first item of the list and WIDTH, VARIANCE and TOTAL are updated.

At this point the algorithm checks if all the items of the list have not more than *maxbucket* buckets. This is done by the COMPRESSBUCKETS function. The function starts from item zero and whenever finds an item with *maxbucket+1* buckets merges the two oldest ones. The merged bucket is then inserted in the next item of the list. If we have already reached the end of the list a new item is created.

Consider for example the case where the *maxbucket* parameter is equal to 3 and three values have been processed. So the window is composed by three buckets belonging to item number zero. When a new input value arrives the Adwin creates a new bucket and adds it to the item zero. However, now item zero has more than *maxbucket* buckets. So the compress function merges the oldest two, creates item number 1 of the list and inserts the merged bucket inside it. As a result the window is composed by two item: the first has two bucket and stores statistics for two inputs. The second has one bucket and stores statistics for two inputs.

One implication of this approach is that the older data's statistic is compressed in bigger buckets.

After having inserted the input tuple in the list the algorithm searches for splits of the window such that the two sub-windows are different enough (basing on formulas 3.2 and 3.3). Moreover, the sub-windows should store statistics for more than "minimum sub-window length". If the algorithm finds a split that satisfy these conditions it drops the last buckets then it starts over searching for a new split. Otherwise it is ready to process a new tuple. Checking multiple times for all the possible splits on a window may be computationally expensive. Thus, the Adwin uses a build-in parameter (CLOCK) that defines how often the check is performed. For example, if the CLOCK is set to 128 the Adwin will check the window's splits every 128 input tuples. This value is the default one that has been used for the experiments (if not specified otherwise).

Algorithm 1 ADWIN pseudo-code from [6]

```
1: function mainFunction
2:     Initialize W as an empty list of buckets
3:     Initialize WIDTH,VARIANCE and TOTAL
4:     for each t>0 do
5:         SETINPUT (x_t,W)
6:         return output μ̂_W as TOTAL/WIDTH and ChangeAlarm
7: function SETINPUT
8:     INSERTELEMENT(e,W)
9:     if CLOCK then
10:        repeatDELETEELEMENT(W)
11:        until |μ̂_{W_0} − μ̂_{W_0} < ε_cut| holds
12:        for every split of W into W = W_0 · W_1
13:
14: function INSERTELEMENT
15:    create a new bucket b with content e
16:    W ← W∪ b (i.e. add e to the head of W)
17:    update WIDTH, VARIANCE and TOTAL
18:    COMPRESSBUCKETS(W)
19: function DELETEELEMENT
20:    remove a bucket from tail of list W
21:    update WIDTH, VARIANCE and TOTAL
22:    ChangeAlarm ← true
23: function COMPRESSBUCKETS(List W)
24:    Trasverse the list of buckets in increasing order
25:    while there are more than M buckets of the same capacity do
26:        merge buckets
27:        COMPRESSBUCKETS (sublist of W not traversed)
```

4.2 Page-Hinkley

The Page-Hinkley method has been developed to monitor for changes in the average of Gaussian signals. However, each of the analyzed data-sets presents specific behavior that may not be related to a Gaussian signal.

For this reason, the thesis defines a main procedure, but results and consideration, obtained with variants of this approach, are considered.

We will see that one of the most important aspects to consider is the granularity of the data that the algorithm is processing. As for the Adwin, the PH is implemented by means of a map operator that processes individually each tuple of the stream. Algorithm 2 shows the pseudo-code of the main implementation. This approach uses two distinct cumulative sum (*mtH* and *mtL*), one for each kind of change. Each of

them is associated with a threshold (λ_{H} and λ_{L}), a fading factor (*fadingFactorH* and *fadingFactorL*) and an allowed change (δ_{H} and δ_{L}). The thresholds, fading factors, and allowed changes are parameters of the algorithm. The need for separate parameters for the two cumulative sum has been motivated in the results chapter.

Algorithm 2 Page-Hinkley

```
1:  function mainFunction
2:      Initialize average, mtH, mtL, mtAbove, mtBelow
3:      for each t>0 do
4:          processElement (x_t)
5:  function processElement
6:      Set aboveAverage and underAverage to false
7:      if (x_T − average > δ) then
8:          mtH+ = x_t − average − δ
9:          set aboveAverage true
10:     else if (average − x_t > δ) then
11:         mtL+ = x_t − average + δ
12:         set underAverage true
13:     if  not aboveAverage then
14:         mtH∗ = fadingFactorH
15:     if not underAverage then
16:         mtL∗ = fadingFactorL
17:     mtAbove = max(MtAbove, mtL)
18:     mtBelow = min(Mtbelow, mtH)
19:     checkChange(W)
20: function checkChange
21:     if (mtH − mtBelow > λ_H) then
22:         resets mtH, mtL, mtAbove, mtBelow
23:         set average equal to the input value x_t
24:         alert positive change
25:     else if (mtAbove − mtL > λ_L then
26:         resets average, mtH, mtL, mtAbove, mtBelow
27:         alert negative change
```

For each input value, the algorithm calls the processElement function that is in charge of updating the two cumulative sums. More specifically *mtH* and *mtL* are updated only when the input value is different of at least δ from the average. Without this condition, the algorithm would increase *mtH* and *mtL* also when the value is equal to the average. Then the two cumulative sum are multiplied for the respective fading factor. After that, the algorithm recomputes *mtAbove* and *mtBelow*. These variables are used by the checkChange function in order to detect a change. If a change is detected all the variables of the PH are set to zero with the exception of the average that takes the value of the input value.

4.2.1 Page-Hinkley with time-slots

This implementation of the PH refers specifically to the case of electricity consumption in a manufacturing process.

Usually, during a day we have time-slots when the machine is running at its full potential, with a very high power consumption.There are also time-slots where the machine is partially working or not working at all.

It may be of interest to see if the power consumption inside these time-slots remains constant during the time. For example, by monitoring this consumption it is possible to identify situations where the machine worked overnight.

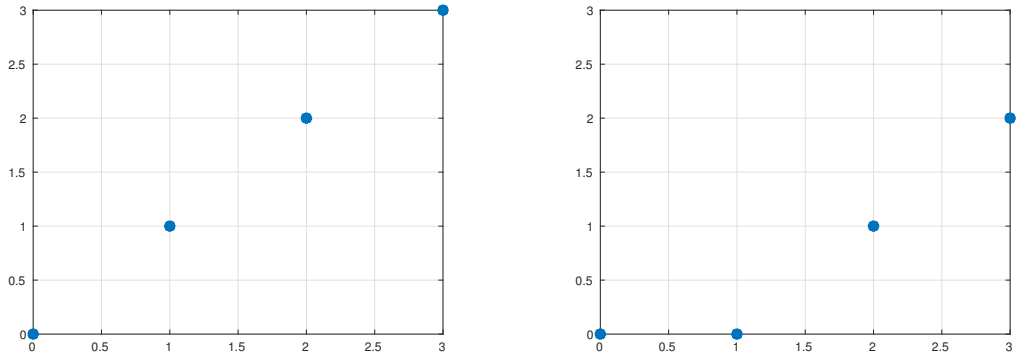
For this reason, a new version of the PH (*PH with time-slots*) has been provided. This approach keeps statistics about each time-slot and checks each new value against the time-slot's average. More specifically, after the aforementioned initial steps (parse data, assign timestamp), the algorithm aggregates the stream by computing the sum of the power consumption for each hour. The time-slot average is the average between the hours belonging to it.

For each input value, the method updates the statistics of the related time-slot. Then it performs the Page-Hinkley test between the input (hour's consumption) and the average of the time-slot. At this point, the behavior of the method is the same to the one illustrated before. Also, when a change is detected the average is reset.

It is possible to decide whether to keep a cumulative sum for each time-slot or one in common for all time-slots. The result obtained would be quite different. In the first case, the algorithm would detect a change only when the behavior has changed inside the specific time-slot. In the second case, the algorithm is still able to detect changes inside a time-slot. However, it can also identify the days where the change is distributed over the time-slots. In order to underline the differences an example has been provided (figure 4.2).

The two graphs refer to a case with four time-slots. The graph on the left shows the average of each time-slot (the x-axis plots the time-slot number).

The graph on the right plots the value of four (aggregated) tuples of the stream. The x-axis represents the time-slot they belong. The y-axis represents their value (energy consumption). The time-slots 1,2 and 3 shows a difference of 1 between the average of the time-slots (graph a) and the current analyzed values (graph b). If we just keep a cumulative sum for each time-slot with threshold greater than one we will not alert any changes. On the other hand, a common cumulative sum will consider all the variations. If these variations are considered alone they are not enough to declare a change. On the contrary, if the algorithm considers the sum of these variations it will detect a change.



(a) TimeSlotAverage

(b) Values of the stream

Figure 4.2: Values of power consumption inside the time-slot

Due to this reason, our implementation uses a general cumulative sum. The number of time-slots has been set to 6 time-slots. They divide equally the days. A more accurate time-slot's division would have been possible if we had had some rules about the power consumption/demand (e.g. usually the factory open at five o'clock).

4.3 Anomaly insertion

The purpose of inserting anomalies in the data-set is to simulate a ground truth that can be used to evaluate our algorithms. This section illustrates how the anomalies have been inserted in the data-sets. It has been decided to injected anomalies on the data-sets about power consumption in a manufacturing environment. There are two main reasons for selecting these data-sets: the first one is that they are longer than the data-sets about the power demand profiles. So more anomalies can be inserted. The second one is that they show similar and comparable behavior, thus making possible to define anomalies that are valid for multiple data-sets.

In order to test our algorithms in different situations, this section defines several kinds of anomalies. More specifically, six types of anomalies have been defined. They can be coupled in two groups. The first group of anomalies is about small and realistic deviations from the standard behavior, e.g. a machine of the factory consumes power when it is usually supposed to be shut down.

On the other hand, the second group is about anomalies which values and behavior are very distant from the normal ones. These kinds of anomalies may be related to faulty sensors, power leakage, broken machines and so on.

Although some of these anomalies may be not realistic for the studied case, they are useful to check the algorithm's behavior (e.g. in case of negative power consump-

tion).

4.3.1 First group of anomalies

Since these anomalies are based on a small deviation from the standard behavior, there is the need to have a quick look at it. The graph on figure 4.3 shows the values of the machine's power consumption for the laser shaping data-set. Usually, the power consumption is different from zero from the 6:00 and the 15:00 of each day. We will refer to this time interval as working hours/time of the machine.

It is possible to identify three different ranges of values that the stream has: the first one is between 0.05 and 0,08 and it is assumed by the stream several times during the working time. The second behavior is overlapping with the first one and consists of intervals with a power consumption ranging between 0.003 and 0.005. Finally, the last interval range is composed of just one value, zero. It is the standard power consumption values outside the working hours.

Keeping this in mind three kinds of anomalies have been defined:

1. Type A : it sets the tuples belonging to the first interval (0.05 to 0.08) to a smaller value (range [0.02-0.03]). However, the new values are still above the overall power consumption average (around 0.02). This situation emulates a reduced power consumption during the working hours.

2. Type B : as the type A also type B modifies the tuples that belongs to the first interval. In this case the new power consumption values belong to the interval [0.005, 0.01]. The main difference with type A is that the modified values are below the average.

3. Type C : it sets the power consumption belonging to the last interval range (the one with just zeros) to values belonging to the interval [0.05, 0.08]. This is an example of the machine consuming power outside the working hours.

In order to insert these anomalies, the first operation is to define a window of two days over the stream. Then we associated each anomaly to a number within 1 and 20. Inside each window, the program extracts a random number in the same range. When the extracted number matches one of the anomaly's the window's values are modified according to the type of the anomaly selected.

In order to have more control over the anomalies, a *target*, has been set. The machine should consume exactly this *target* through the two days affected by the anomaly.

More specifically at the beginning of the program, two targets have been defined: the first one is in common for the anomalies of type A and B. The other target is

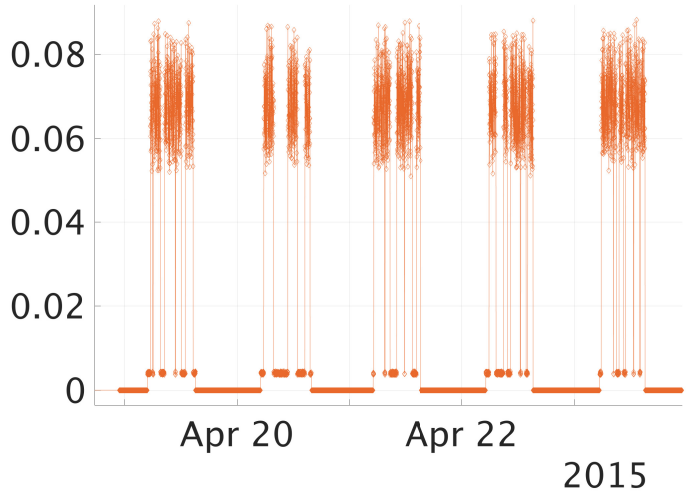


Figure 4.3: Section of the power consumption of the laser shaping data-set

defined for the anomalies of type C.

The reason for which the first two types share the target is that we want to compare the effects of their different distribution, without taking into account diverse overall power consumption.

The values of these two targets are built-in parameters of the anomaly's insertion module. In this case, we defined them by studying the values of the existing anomalies on the Aachen's data-sets. The first target is 20, the second one is 85.

4.3.2 Second group of anomalies

The anomalies of the second group have been inserted by means of a map operator that modifies a certain number of consecutive values. This number is the length of the anomaly. At the start of the program and after each inserted anomaly the map operator waits for a certain time before inserting a new abnormal situation.

Both this time and the anomaly length are expressed in number of processed tuples and they are random numbers within a range.

How the values are modified depends on the *type* of the anomaly. All our anomalies are characterized by two parameter: the *base* and the *multiplicative*. The base is calculated once for each anomaly while the multiplicative is calculated for each input values that belongs to the anomaly. The results of the multiplication between base and multiplicative is the new power consumption value of the tuple.

In this way it is possible to obtain non-constant anomalies with a different overall power consumption.

1. Type D (Big Positive)
    - Anomaly length, range between 800 and 1500
    - Interval length, range between 10000 and 30000
    - Modified values, *base* between 0.05 and 0.4, *multiplicative* between 1 and 8
2. Type E (Big Negative)
    - Anomaly length, range between 300 and 3000
    - Interval length, range between 8000 and 20000
    - Modified values, *base* between 0.05 and 0.4, *multiplicative* between 1 and 8
3. Type F (Oscillating)
    - Anomaly length, range between 500 and 1500
    - Interval length, range between 6000 and 20000
    - Modified values, , *base* between 0.05 and 0.4, *multiplicative* between 1 and 8, *change sign probability* 1%

Type F has a probability of changing the sign of the base of 1% for each input tuple. As it is possible to see the interval is always greater than the anomaly length. In this way it is still possible to recognize a normal, standard behavior.

5

Results

This chapter starts by illustrating how the methods have been evaluated in terms of throughput, latency, true positives, true negatives and false positives. The goal is to give a definition of the used metrics and explain how they have been calculated. These definitions are given in the *evaluation methodology* section.

After that, the ADWIN and the Page-Hinkley algorithms will be tested against several data-sets. More specifically we are interested in:

- monitoring their performances
- checking how they handle different types of anomalies and outliers
- analyzing the effect that their parameters have on the performances

The Adwin algorithm´s implementation has three main parameter: *delta*(δ), *min. length of the sub-window* and *adwinClock* (how often we check our window for shrinks). Also the Page-Hinkley uses three main parameters: *allowed change*(δ), *threshold*(λ) and *fading factor*.

We are also interested in considering different granularity of the data by means of an aggregation operator over a window. Aggregating the data changes the values of the stream, its periodicity and the total number of tuples. For example, when aggregating per minutes the alternation between day and night is to be considered. This behavior would not be visible when aggregating per days.

It is possible to divide the tests into three main parts: the first one focuses on the data-sets about electricity consumption in a manufacturing environment. After having analyzed the results produced by the algorithms the first and second group of anomalies have been inserted. The second part focuses on the data-sets about power demand profiles of customers in different energy sectors. Finally, the last part is about the time performances of the algorithms.

5.1 Evaluation methodology

The abnormalities present in the data-sets do not follow a defined pattern. So instead of defining what it is and is not an anomaly, a different approach has been used. The data-sets have been analyzed by means of aggregation operators so that it was possible to understand the reasons that lead the algorithms to detect (or not) a change. After that, it is possible to test some of these deductions during the evaluation of the results obtained with the injected anomalies.

The computer we used for the tests is running Red Hat 6.9 (Santiago), has four Intel core i5-3470s (2.90 GHz) and 16 GB of RAM.

As a starting point, the Apache Flink's metric system has been considered. Flink exposes some metrics through its web interface. It is possible to create and access new metrics in any user function that extends RichFunction [4]. A number of default metrics are already implemented and their evolution is shown by means of graphs.

These graphs are displayed in the Flink's web-interface. They are a good way to get an insight of how the system is currently working. However, these metrics do not cover all the interesting measurements.

In order to correctly evaluate our algorithms, we would like to measure:

1. The job execution time, i.e. the time that Flink needs to apply the algorithms over all the data-set

2. The latency, defined as time that the map function implementing the Adwin or the Page-Hinkley algorithm needs to process each input tuple

3. The true positives, i.e. the anomalies detected

4. The true negatives, i.e. the anomalies not detected

5. The false positives, i.e. the detections not linked to an anomaly

6. The delay in detection, i.e. the number of input tuples between the start of the anomaly and its detection

5.1.1 Time and Latency

Retrieving the job execution time with Apache Flink is rather trivial. Indeed, when the Flink's job ends it returns a *JobExecutionResults* object that contains several statistics about the job. The execution time is among them. With the help of an

accumulator that counts the number of lines, it is possible to compute the Job's throughput.

Apache Flink exposes also a metric for measuring the latency. Nevertheless, this metric has been implemented quite recently and shows some misbehavior [4]. So, in order to calculate the latency, a different approach has been used.

As previously explained, the algorithms have been implemented inside a map operator. The first thing the map function does is taking the *start time*. However, the operation of returning a tuple may be expensive and it is worth considering it. For this reason is not possible to compute the elapsed time inside the same function. So a new map operator has been introduced right after. This new operator takes the end time for each input tuple. Since the previous method used one of the tuple's field to store the relative start time it is possible to compute the latency of the tuple.

The correctness of this measurement is ensured by the streaming nature of Flink, i.e. it does not create batches or mini-batches but as soon as a tuple is processed it is forwarded to the next operator. Since the process of taking the time is costly (taking twice the system time for each input tuple and the new map operator) we removed these measurements when taking the job execution time. To ensure the validity of the measurements, we have run our algorithms one hundred times (and taken the averages).

The Adwin´s time performances are deeply influenced by its internal clock. The results illustrated in the next chapter have been obtained with an Adwin clock of 128.

5.1.2 TP, FP and FN

When dealing with an anomaly it is possible that our algorithms detect several changes. There is the need to define which of these changes are true or false positives.

The ADWIN adapts its window to the new behavior dropping the oldest bucket. However, it usually does not drop all the tuples linked with the old behavior at once but performs multiple shrinks. Instead, the Page-Hinkley detects a change every time the cumulative sum goes above or below a threshold. Thus if the abnormality is long enough and with values far from the average, the algorithm will detect more changes during the same anomaly.

For the Adwin it has been decided to label as 'good detections' all the shrinks that happen during the anomaly. On the contrary, changes detected when there are not ongoing anomalies are labeled as false positives.

Nevertheless, if there is at least a small amount (ω) of values belonging to the

anomaly inside the Adwin window, the detection will not be labeled as false positive. Indeed, these change alerts are still caused by the anomaly. They represent the algorithm signaling the return to the normal behavior. Considering the length of the anomalies ω has been set to 25.

However, with this definition, the algorithm does not have any control over the time interval where the detected changes are not labeled as false positives. For this reason, a maximum interval after the end of the anomaly was introduced. After the end of this interval the changes are labeled as false positives regardless of the window composition.

A similar concept has been defined also for the Page-Hinkley: all the changes that occur during the anomaly are labeled as 'good detections'. As for the Adwin, the detections related to the return to the normal behavior should not be labeled as false positives. However, it is not possible to use the definition provided for the Adwin since the PH does not define a window. So, in order to define this interval (*ReturnToNormalInterval*) the *delay in detection* has been used. More specifically: $ReturnToNormalInterval = 2 \cdot DetectionDelay + K$ where K is a constant that handles cases where the detection delay is very small.

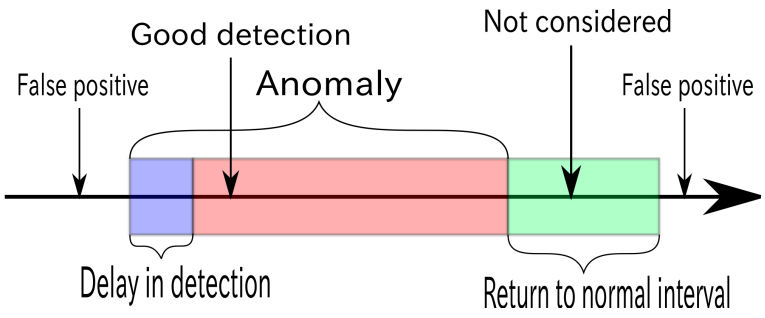


Figure 5.1: Example of the detections' meaning basing on their position in the stream

K and the maximum interval of the Adwin have been set respectively to 50 and 1000 (we will see later on that the Adwin adapts slower than the PH). The detections in this interval are not considered for the evaluation (i.e. they are not considered as good detections).

If for an anomaly there is at least one 'good detection' (i.e. a detection during that anomaly) then it is considered successfully detected (true positive), otherwise it is a false negative. For one anomaly there may be a lot of 'good detections' but the number of true positives is incremented just once. If, on the other hand, the algorithm detects a change outside the anomaly and there are not at least 25 anomaly's values inside the window (if we are using the Adwin) or we are past the *ReturnToNormalInterval* (for the Page-Hinkley) the detection is labeled as false positive.

5.2 Electricity consumption in a manufacturing environment

In this section, we will analyze the most interesting results obtained with the two algorithms on the data-sets about the power consumption in a manufacturing environment.

These data-sets show a high correlation each other and similar values. Each of them is related to the power consumption of a machine in the factory. However, they show several interesting situation, that we are going to analyze.

5.2.1 Robot Welter

Figure 5.2 shows the power consumption for the Robot Welter data-set aggregated per day.

The two tables 5.1 show the comparison between the changes detected by the Page Hinkley (on the right, $\lambda = 25$ and fading factor = 0.998 for the positive cumulative sum and $\lambda = 33$ and fading factor = 0.997 for the negative one) and the Adwin (on the left, with $\delta = 10^{-5}$, minimum length of the sub window = 2000) on this data set. The first column of the two tables contains the timestamp of the detected changes. In the left table the second column represents the change of the average inside the Adwin window. This value has been computed as the mean inside the window before the shrink, minus the one after it. Therefore, a negative sign implies that the average is increasing, a positive one reveals that it is decreasing. Similarly, in the right table the second column consists of a string that identifies whether the change is detected because of an increase or a decrease of the power consumption. As it is possible to see the changes detected by the two algorithms are different.

Table 5.1: Changes detected by Adwin and Page-Hinkley for robot welter

Timestamp	Avg change
2015-04-21 06:00:00	0.00013
2015-04-23 04:56:00	0.00069
2015-04-25 08:08:00	0.00074
2015-06-13 14:14:00	-0.00016
2015-06-14 15:50:00	-0.00041
2015-07-05 11:57:00	-0.00042
2015-07-07 15:09:00	-0.00073
2015-07-09 22:37:00	-0.00022
2015-07-10 15:41:00	-0.00052
2015-07-17 14:44:00	-0.00036

Timestamp	Change
2015-04-20 00:57:00.0	Decreasing
2015-05-11 06:29:00.0	Decreasing
2015-05-12 06:44:00.0	Decreasing
2015-05-22 15:02:00.0	Increasing
2015-06-07 14:25:00.0	Increasing
2015-06-13 15:30:00.0	Increasing
2015-06-16 15:18:00.0	Increasing
2015-07-05 13:52:00.0	Increasing
2015-07-09 21:18:00.0	Increasing
2015-07-16 21:33:00.0	Increasing

The data-set shows several days with an overall power consumption higher than the usual. It is possible to distinguish these days in two groups: the first one contains the days where the machine was used during the normal hours but in a more intensive way (consuming more power). The second group contains the days where the machine worked also during the evening/night. The days belonging to the first group are: June 7th, June 13rd and June 16th. Instead, May 22th, July 7th and July 16th belong to the second group.

From the two tables (5.1) it is possible to see that all the detections linked to an increase of the power consumption (minus sign in the Adwin and 'increasing' for the Page-Hinkley) are related to these days. The Adwin fails to detect the high consumption on May 22nd and on June 6th. This detection is missed because the previous days had a slightly lower power consumption that balanced the next day´s higher consumption.

Both the algorithms detect also a change on July 5th. This change is linked with a singular situation: there are not records with timestamp from 16:21 of July 2^{2nd} until 08:18 of July 5th. This means that it is missing an interval, such as the night, with a low power consumption between two days. So our algorithms will process continuously values related to the higher power consumption of two days.

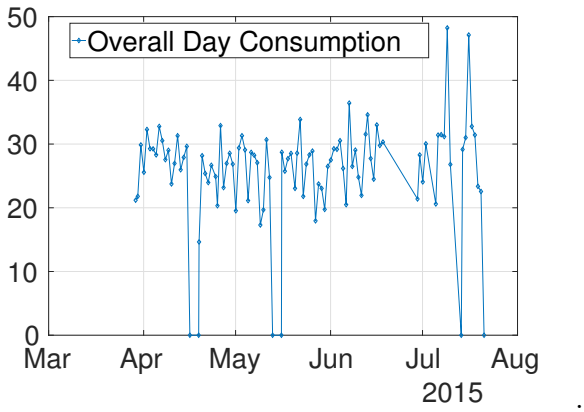


Figure 5.2: Power consumption for robot welter aggregated by days

As it is possible to see from the figure 5.2 the data-set presents also some days with a low power consumption. It is important to notice that the graph shows just the overall consumption. Some of these days have missing tuples (e.g. some of them have just 80 tuples instead of the standard 1440). If a day has few tuples its impact will be smaller. For example, on July 8th the electricity consumption was zero. However, there are only 160 records in this day, so our algorithms do not detect it. The most meaningful reduction of power consumption can be observed from April 20th to April 23th. Both the algorithm detect a change in these days.

5.2.1.1 Page Hinkley Analysis

For this data-set, but also in the others related to the electricity consumption in a manufacturing environment, the change allowed (δ) parameter has been set to zero (if not specified otherwise). The use of this parameter in the current context has proven difficult and did not lead to interesting results. One of the main issues is that it is not possible to find a meaningful value. Indeed, the maximum variation of the stream (from zero to the highest power consumption during the day) is perfectly normal. Nevertheless, this parameter may be useful in specific contexts. For example, if we want to spot a situation where the power consumption reaches higher values than the usual, setting a δ equal to the highest normal value would be an optimal solution. Since in these cases we want to be able to detect all kinds of variations the change allowed has not been used.

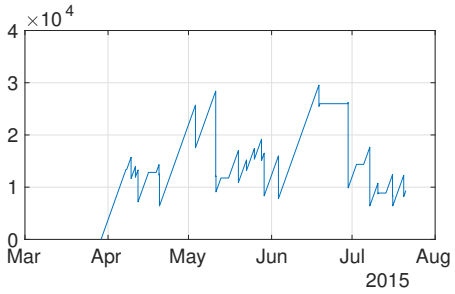
Also, setting the fading factor and the threshold in order to obtain a meaningful result may be not an easy task. For example, with a threshold of 3.25 and a fading factor of 0.985 (for both cumulative sums), the algorithm have a similar number of detections (with respect to the results previously shown). However, they are completely different from the ones obtained before.

With these settings, the PH is more likely to miss the days with a higher power consumption caused by the machine working until late. This is linked with the lower fading factor that makes the algorithm forgets quickly about the old increments (even in the same day). So the cumulative sum will not reach the threshold. On the other hand, since the threshold is lower, the algorithm is still able to detect the days where the machine consumed a lot of power in a small interval of time (i.e. less tuple, fewer times the cumulative sums are multiplied for the fading factor).

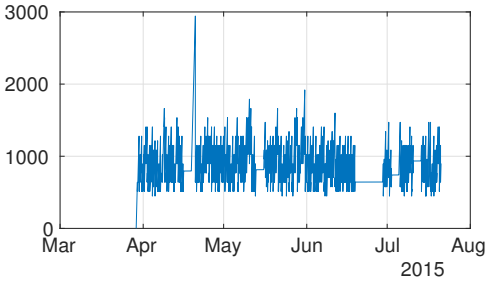
5.2.1.2 Adwin Analysis

The first three detections of the Adwin, shown in the left table of figure 5.1 are related to days with low power consumption. Although May 10^{th} has a similar low consumption it is not detected by the algorithm. In May there are a mix of days with higher and lower power consumption. So there are mixed values inside the Adwin window. For this reason, the algorithm does not detect either the higher consumption on May 22^{nd} and the lower one on May 10^{th} (detected by the PH). On the other hand, with a δ of 10^{-4} the Adwin detects several changes in May (related to both kind of changes, increase and decrease of the power consumption). It has been decided to not show the results with this parameter setting because of the high number of detections.

The two graphs at figure 5.3 show how the behavior of the algorithm change by varying the minimum length of the sub-window. In both the graphs the y-axis plots the size of the ADWIN window. The x-axis is related to the timestamp of the



(a) $\delta = 0.002$, minimum length of the sub window = 2000



(b) $\delta = 0.002$, minimum length of the sub window = 3

Figure 5.3: Evolution of the size of the Adwin window

tuple. In the graph on the right, it is possible to see that the window shrinks very often. It happens because with a *minimum length of the sub-window* equal to 3 the algorithm detects a change every time it compares a sub-window with positive power consumption values with one full of zeroes and vice-versa. For this reason, we expect about two shrinks for day (one at the beginning of the machine´s working time and one at its the end). The analysis of the changes confirmed this hypothesis but shows some delay between the shrink and the start (or the end) of the working time.

A small delay is introduced by the Adwin clock (i.e. we are checking for changes only once every *adwinClock* input tuples) but it is not enough to explain the delay. If we look at the values of ϵ_{cut} we would find that they are very high when one of the two sub-window has size close to the minimum. This happens because ϵ_{cut} is proportional to m. Formula 5.1 shows how m has been defined in the implementation.

$$m = \frac{1}{\frac{1}{n_0 - (minSubWindowLength + 1)} + \frac{1}{n_1 - (minSubWindowLength + 1)}} \tag{5.1}$$

Thus, if one of the sub-windows has size close to the minimum there must be a very high difference between the sub-windows averages to trigger a shrink. This causes the additional delay in detection. However, in the graph on the right, despite the small sub-window length, the maximum value reached by the ADWIN window is 2150. The window grows from the night of April 15$^{\text{th}}$ until the morning of April 20$^{\text{th}}$. Moreover, during this interval the window has a very low average. This happens thanks to a particular situation: in this period the machine shows a very low power consumption or no power consumption at all.

In order to reduce the dependency from the alternation of working time, where the machine is consuming power, and the rest of the day, where it is off, a different

minimum sub-window length has been set. Knowing that the frequency of the readings is of 1 minute we want to have a minimum length of the sub-window grater than $60\,minutes * 24\,hours = 1440$ (plus some margin).

With a min length of the sub-window equal to 2000, the number of shrinks falls from 640 to 21 (with the same δ). Also, the value of the average inside the ADWIN window is way more stable. It is usually between 0.02-0.03 while with the previous settings was ranging from 0 to 0.08.

Using a smaller sub-window (with respect to 1440) and reducing the δ would lead to similar results. With an appropriate parameter setting, the algorithm ignores the change in consumption between day and night even if we are comparing very different sub-window. Moreover, this approach has the advantage of comparing smaller sub-windows and is able to reduce the delay in detection.

5.2.2 Laser Cutting

As already stated, the data-sets regarding the power consumption in a manufacturing environment are quite alike. However, the 'laser cutting' data-set is characterized by a day (May 6^{th}) where the machine did not consume power at all and there are not missing records. Moreover, on July 17^{th} and July 18^{th} the power consumed during the working hours was very low. Also, in this data-set it is possible to observe some days with a higher power consumption: April 10^{th}, May 28^{th}, June 2^{nd} and June 30^{th}.

Table 5.2: Changes detected by Adwin and Page-Hinkley for laser cutting

Timestamp	**Avg Change**
2015-04-11 12:57:00.0	-0.00075
2015-04-11 15:05:00.0	-0.00444
2015-05-06 08:25:00.0	0.00036
2015-05-06 16:57:00.0	0.00064
2015-05-06 19:05:00.0	0.00075
2015-05-06 21:13:00.0	0.00073
2015-05-06 23:21:00.0	0.00175
2015-05-07 03:37:00.0	0.00118
2015-05-07 05:45:00.0	0.00206
2015-05-28 22:31:00.0	-0.00095
2015-06-03 15:03:00.0	-0.00127
2015-07-01 11:46:00.0	-0.00107
2015-07-01 13:54:00.0	-0.00172
2015-07-01 16:02:00.0	-0.00338
2015-07-19 02:45:00.0	0.00170

Timestamp	**Change**
2015-04-10 21:24:00.0	Increasing
2015-05-06 11:58:00.0	Decreasing
2015-05-28 21:27:00.0	Increasing
2015-06-02 22:38:00.0	Increasing
2015-06-13 14:31:00.0	Increasing
2015-06-30 18:13:00.0	Increasing
2015-07-18 04:52:00.0	Decreasing

The table 5.2 shows the detected changes of the two algorithms (left one for the

Adwin, right one for the Page-Hinkley). For the Adwin a $\delta = 10^{-11}$ and a minimum length of the sub- window of 100 have been used. For the PH we used a threshold of 22.5 and a fading factor of 0.999 for the positive changes and a threshold of 36 with a fading factor of 0.997 for the negative ones.

It is possible to see that all the days with a higher power consumption are detected, although the Adwin shows some delay in detection (usually it detects the change in the morning of the day after the anomaly).

The Page-Hinkley alerts also a change on June 13$^{\text{th}}$. This day has a power consumption slightly bigger than the average. It is preceded by two days with a slightly smaller power consumption that lowered the average. For this reason, the impact of June 13$^{\text{th}}$ is bigger than the one it is possible to expect by looking at the values.

As for the changes related to a reduction of the power consumption both the algorithms behaves in a similar way. Indeed, they detect the two most interesting situations (on May 6$^{\text{th}}$ and on July 17-18 $^{\text{th}}$). Yet the Adwin has some delay in detecting the last one.

5.2.3 Laser Shaping

The 'laser shaping' machine presents three main interesting situations: the first one is a higher power consumption on three consecutive days, from May 4$^{\text{th}}$ to May 6$^{\text{th}}$. The second one is about a very low energy consumption on May 22$^{\text{th}}$ and May 24$^{\text{th}}$. The last one happens on June 1$^{\text{st}}$. In this case, the machine consumes power outside its working hour. Moreover, we have a slightly lower power consumption on April 27$^{\text{th}}$ and May 19 $^{\text{th}}$.

We have run the Adwin with a $\delta = 10^{-11}$ and minimum length of the sub-window of 300. Instead, for the Page-Hinkley we used different thresholds and fading factors for the two cumulative sums: a threshold of 27.5 and a fading factor of 0.998 for the positive one. A threshold of 33 and a fading factor of 0.997 for the negative one. The results are shown in the two tables at 5.3 (left one for the Adwin, right one the Page-Hinkley).

Table 5.3: Changes detected by Adwin and Page-Hinkley for laser shaping

Timestamp	**Avg Change**
2015-05-06 15:21:00	-0.00029
2015-05-24 07:28:00	0.00025
2015-05-25 02:40:00	0.00096
2015-06-02 15:28:00	-0.00270

Timestamp	**Change**
2015-04-27 04:55:00	Decreasing
2015-05-06 15:16:00	Increasing
2015-05-19 06:06:00	Decreasing
2015-05-25 06:01:00	Decreasing
2015-06-01 22:07:00	Increasing

Both the approaches successfully detect the increase of consumption on June 1$^{\text{st}}$ and

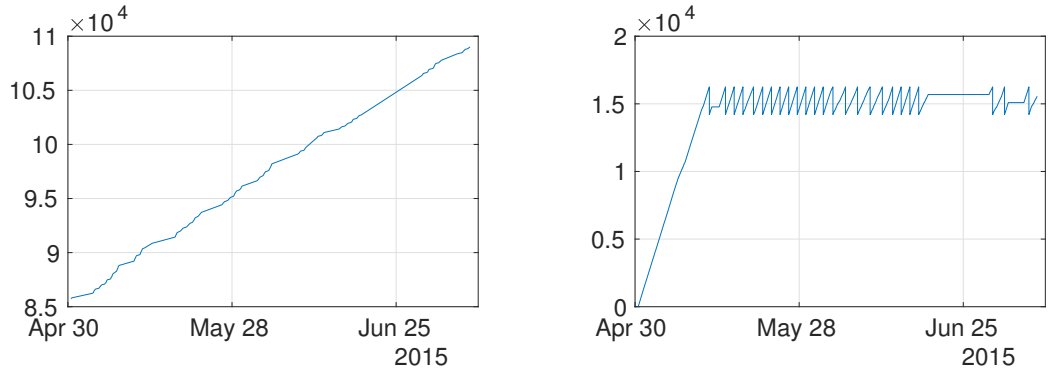
between May 4nd and May 6th. Also, the decrease in power consumption between May 22nd and May 24th is detected by both the algorithms.

The Page-Hinkley detects two more changes related to a lower consumption: on April 27th and on May 19th. Increasing the threshold would lead the algorithm to detect these two changes but to miss the one on May 6 th. Indeed, with this parameter setting, it is harder for the PH to detect gradual but bigger changes (May 4th-6th) than abrupt but smaller ones.

On the other hand, the Adwin detects more easily the latter change. We need to increase the δ to 10^{-9} in order to detect also the changes on April 27th and on May 24th.

5.2.4 Smart-Meter

As already said, the smart meter data-set is quite different from the others related to the power consumption in a manufacturing environment. The left graph of figure 5.4 shows the behavior of the stream generated from the data-set. It is possible to see that this data-set shows an increasing trend.



(a) Power consumption´s values

(b) Evolution of the size of the Adwin window

Figure 5.4: Metrics and results on the smart meter data-set

The right graph in the same figure plots the size of the Adwin window. The graph shows that the ADWIN window is continuously shrinking, trying to adapt to the stream's values. The value of the parameters, in this case, is not relevant. Indeed, if we change our settings the only relevant change we would witness is a variation on the frequency (and number) of changes.

For this example a $\delta = 0.002$ and a minimum Length of the sub-window=8000 have been used.

This data-set has a regular trend, input tuples with close time-stamps have also similar values. Instead, the data-sets regarding the machines consumption showed

a very different behavior. For example, as soon as the machine is turned on the consumption values instantaneously increase. This means that also the variance increases. So, since the variance is proportional to ϵ_{cut} it will more hard to have a shrink. For this reason, the Adwin window shrinks more often in the smart-meter data-set than in the others. Moreover, it may be observed a regular behavior of the window size: the algorithm shrinks the window every time it reaches the value of 16255. This value is not random, but it is the first value for which we have two sub-windows of size greater than 8000, the minimum.

The value is greater than 16000 for two main reasons: first, the algorithm does not check for changes every time it inserts a tuple inside the window. Second, the sub-windows lengths must be of the form $M \cdot (1 + 2 + ...2^i)$.

For example, the check before the one that leads to a shrink is computed when the Adwin window has size of 16128 and there is no split such as both of the sub-windows have size greater than 8000 (the closest split is 8192 for the sub-window with the older input tuples and 7936 for the other one). A detailed description of the Adwin´s window composition may be found in appendix A.1.

In this case, the results obtained with the Page-Hinkley are close to the Adwin´s one. The PH detects several changes in order to adapt to the stream´s trend. The graph in figure 5.5 plots the values of the input tuples of the data-set and the averages computed by the two algorithms. In both cases, the averages follow the trend of the stream. Every time they detect a change their computed average makes a step and became closer to the current stream behavior.

If we increase the delta or decrease the minimum sub-window length the average computed by the Adwin will be more close to the stream´s values. Decreasing the threshold in the PH leads to similar results.

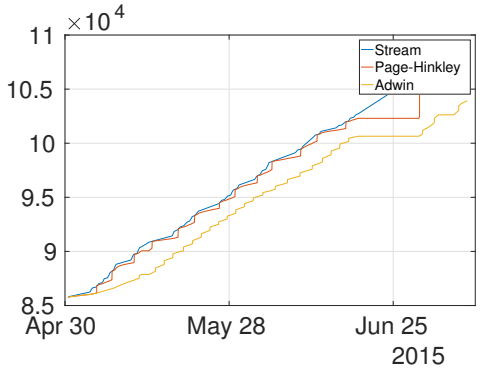


Figure 5.5: Comparison of the averages computed by the two algorithms

5.2.4.1 Smart meter with differences

The nature (i.e. the slow increasing trend) of the smart meter data-set suggests that each value represents the power consumed by the factory until that moment. For this reason it is interesting to elaborate the stream by considering the difference of two consecutive records. As a result of this the stream value´s are between 0.2 and 0.5. However, in few situations there are higher values (up to 3653). Since these higher values are isolated and very different from the rest of the stream, they may be regarded as outliers.

The results obtained by the two algorithms on this modified data-set are very different. The Page-Hinkley manages to detect a change whenever it processes these outliers. On the other hand, the Adwin does not detect any anomaly/outlier and if we change the parameters setting (e.g. by increasing the δ) the detected changes are not related to these values.

5.2.5 Page-Hinkley with time-slots

In this part we will look at the result obtained with our second implementation of the Page-Hinkley method. We remember that this implementation has been designed specifically for analyzing the data-sets about electricity consumption in a manufacturing environment under a different point of view. This method looks at data aggregated per hour and compare these values against the average of a time-slots (of 4 hours).Then it uses the difference to update the cumulative sum in the same way of the standard Page-Hinkley implementation.

The used parameter setting is: $threshold = 5$, $changeAllowed = 1$ and $fadingFactor = 0.95$. The tables 5.4 5.5 5.6 show the changes detected by the algorithm over three data-set (welter, cutting, shaping).

Table 5.4: Changes detected by Page-Hinkley time-slots for robot welter

Hour Consumption	Time-slot avg	Time-slot	Timestamp
0.24	2.98	8-11	2015-04-19 11:00:00.0
0.24	2.25	12-15	2015-05-10 12:00:00.0
0.54	2.54	12-15	2015-05-27 13:00:00.0
3.62	0.03	16-19	2015-07-09 17:00:00.0
3.89	0.02	20-23	2015-07-09 21:00:00.0

The first column shows the consumed power on the analyzed hour. The second one the usual consumed power in that time-slot. The third one the time-slot we are referring to. Finally, the last one contains the timestamp of the input value.

The analysis of the results is straightforward. In table 5.4 the first two rows are

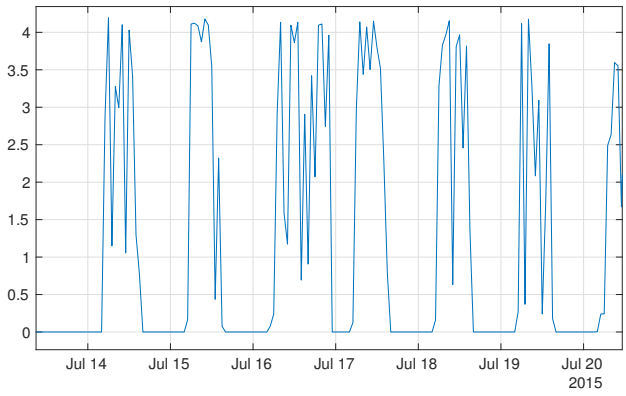


Figure 5.6: Section of power consumption of robot welter data-set

labeled as anomalies because the machine uses a low amount power during a time-slot when it usually consumes a lot more. On the opposite, all the remaining rows refer to situations where the machine is consuming power when it is usually supposed to be shut down. Graph 5.6 plots the power consumption (aggregated per hour) of the last two detected anomalies in the Robot Welter data-set. It is possible to see the odd behavior (with respect to the other days) of July 16th, where the machine continues to consume power until 23 o´clock.

Table 5.5: Changes detected by Page-Hinkley time-slots for laser cutting

Hour Consumption	Time-slot avg	Time-slot	Timestamp
4.17	0.28	16-19	2015-04-10 19:00:00.0
3.89	0.16	20-23	2015-04-10 21:00:00.0
0.24	2.37	12-15	2015-05-01 12:00:00.0
0.00	2.73	8-11	2015-05-06 09:00:00.0
4.17	0.17	16-19	2015-05-28 19:00:00.0
3.89	0.08	20-23	2015-05-28 21:00:00.0
4.11	0.51	20-23	2015-06-02 20:00:00.0
4.10	0.26	16-19	2015-06-30 17:00:00.0

Table 5.6: Changes detected by Page-Hinkley time-slots for laser shaping

Hour Consumption	Time-slot avg	Time-slot	Timestamp
0.24	2.32	12-15	2015-04-26 12:00:00.0
0.24	2.53	12-15	2015-05-18 12:00:00.0
4.17	0.07	16-19	2015-06-01 19:00:00.0
3.89	0.03	20-23	2015-06-01 21:00:00.0

Table 5.6 shows some cases where the machine stopped working during the day (first and second detection). The last two detections of this table refers to June 1st were the machine worked overnight.

All these data-sets refers to machines of the same factory. Moreover, these data-sets shows also a high correlation. For this reason, it is interesting to see if these

anomalies are somehow correlated. For example, a low power consumption during the day may be caused by a malfunction or a power shortage affecting all the factory. Then all the machines should have an anomaly at the same time. However, looking at the tables, it is possible to see how all the detected anomalies for the machines refer to a different day. This suggests that the machines do not show any correlation when it comes to abnormal usage.

For each data-set it can be observed that the changes detected with this approach are quite similar to the ones detected by the Adwin and the previous Page-Hinkley implementation. Nevertheless, some differences can be spotted. For example, on July 18^{18} and 19^{th} we have a gradual power consumption reduction. In this case, the Adwin is able to detect the changes while this (but also the previous) implementation of the Page-Hinkley fails. In order to detect these changes, an idea would be to use a lower fading factor. But the algorithm would end up alerting meaningless changes. This happens because the cumulative sum would not be decremented fast enough and it would increase through time.

5.2.6 Introducing anomalies of the first group

We have analyzed the data-sets, highlighted their anomalies and determined the best parameter setting. This study was preparatory to test our results and considerations about the algorithms in a more extensive way. For this reason all the data-sets (with the exception of the smart-meter) have been manipulated by inserting anomalies of type A, B, and C, as described in the implementation chapter. These anomalies are conceptually similar to the already existing ones. So we expect to detect them with the same parameter configurations.

Tables 5.7 and 5.8 show the results obtained by modifying the data-sets and running the two methods.

Table 5.7: Adwin´s results with anomalies of first group

Interval	**% TP**	**% FN**	**Delay Detection**
14 days	92	8	1492.87
12 days	93	7	1690.53
6 days	79	21	1215.13

Table 5.8: Page-Hinkley´s results with anomalies of first group

Interval	**% TP**	**% FN**	**Delay Detection**
14 days	76.3	23.7	1071.82
12 days	58.6	41.4	810.71
6 days	67.5	32.5	1017.30

As described in the implementation's chapter we want to have a certain number of days between two anomalies. This number has been used as a parameter in

this experiment. It is shown in the first column of the two tables. Moreover, this interval defines the position of the anomalies in the data-set. The second and the third column show respectively the percentage of true positives and false negatives. We remember that whenever the algorithm detects a change during an inserted anomaly we consider the anomaly successfully detected. At the end of the job, the number of detected anomalies expresses the true positives, while the number of not detected ones represents the false negatives.

For each interval value shown in the tables, we both modify the data-set and run our algorithms 100 times. Then the percentages have been computed.

If the interval between two anomalies is too small the results may be affected. This happens because there may be too few normal values between the two anomalies (especially if during the interval some days have missing values). So the algorithms may perceive them as a unique, longer one. For this reason, the performances of the Adwin drops when considering an interval of just 6 days.

On the other hand, the Page-Hinkley does not show this behavior. This discrepancy is correlated with the different delay in detection that the two algorithms show: the Page-Hinkley is faster to adapt to this kind of changes and therefore, handle better close anomalies.

Apart from the delay in detection the tables show that the Adwin algorithm produces far better results. Moreover, if we further analyze the anomalies not detected by the Page-Hinkley (i.e. false negatives) we would see that 85% of them are of Type B.

With the Page-Hinkley is not always possible to understand if the machine consumed less power than the usual during the working hours. Until now, all the reductions of the power consumption were caused by values below the average. For example, recalling figure 4.3, in some days the number of measurements belonging to the interval 0.06-0.08 was very low. At their place there were more measurements belonging to the interval 0.00045-0.0055. These measurements are below the average computed by the PH (usually 0.002-0.003) and then the algorithm modify its negative cumulative sum.

In the anomalies of type B the machine is reducing the consumption by decreasing the power consumed during the working hours. However, the modified value is still greater then the average computed by the Page-Hinkley. Since the algorithm does not modify the negative cumulative sum whenever the input values are greater than the average, it is not able to detect the reduction.

Essentially, it is a limitation of the implementation: if our stream changes behavior from Fig. 5.7 to Fig. 5.8 the algorithm would not detect anything.

Instead, if our stream does the opposite (from Fig. 5.8 to Fig.5.7) we would detect the change (an increasing one). Indeed, the new values would be greater than the

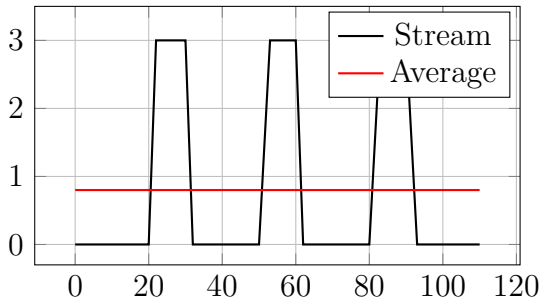


Figure 5.7: Example of consumption

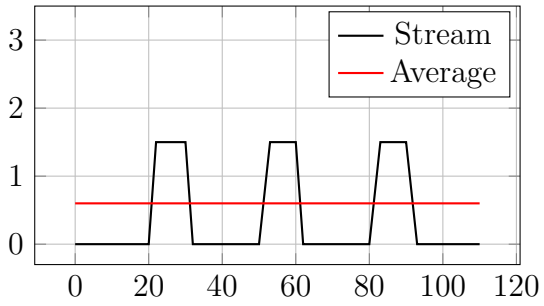


Figure 5.8: Example of consumption

average and the positive sum would be incremented.

In order to address this problem, a different solution has been tested. In this case, the Page-Hinkley uses just one cumulative sum that is updated for every value of the stream. In this solution, the unique cumulative sum m_T follows the stream´s behavior. In figure 5.9 it is possible to see how m_T increases during the working hours and decreases during the night. The increment during the day compensates the decrease during the night. Then whenever the machine consumes less power during a day, m_T should reach a lower value in the night. Figure 5.9 shows that during May 10^{th} and May 11^{th} the machine consumes less power and m_T reaches lower values during the working hours. Nevertheless, during the night the cumulative sum does not reach lower values. Once again, the cause is to be searched in the fading factor.

However, removing the fading factor would lead to meaningless results. Suppose that during a day $\mathrm{m_T}$ was increased to a value very near the threshold. After that, for the next 50 days, the stream observed a normal behavior. Then finally, we had another day with a small increase in power consumption. The algorithm would detect this small increase, even though is not significant (or far less significant than the previous one).

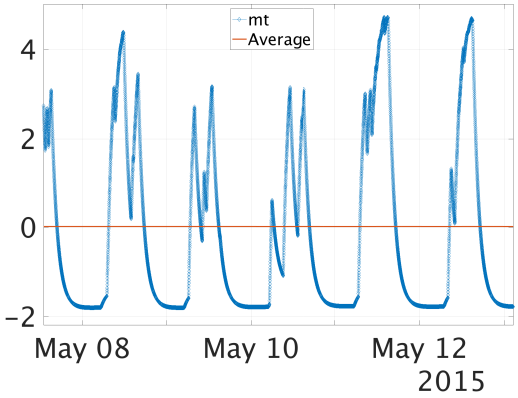


Figure 5.9: Behavior of the Page-Hinkley with a unique cumulative sum on the robot welter data-set

5.2.7 Introducing anomalies of the second group

The anomalies of the second group simulate cases linked with some kind of fault in the system. For this reason, the introduced values are very different from the ones belonging to the normal behavior.

As already said, these anomalies have been inserted on the Aachen´s data-sets since they are longer (i.e. more input tuples) and presents a more stable behavior.

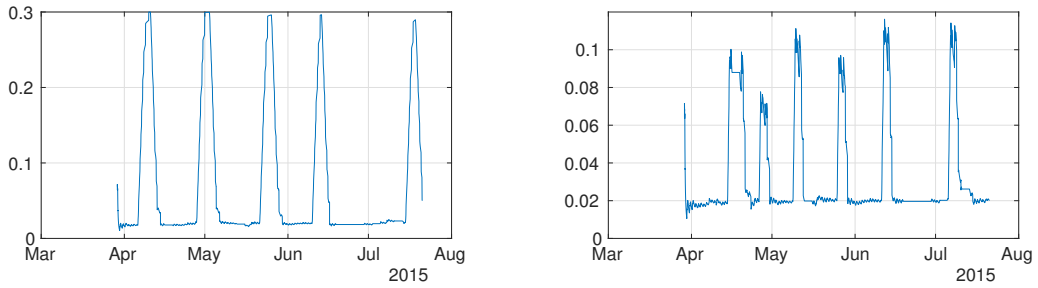
The purpose of these experiments is to analyze the metrics defined in the evaluation criteria. They are: the delay of detection, the number of changes that are correlated to the anomaly (true positives), the ones that are not (false positives) and the false negatives. Additionally, for the Adwin algorithm, we have explored the relationship between the minimum length of the sub-window and the length of the anomaly.

In the Adwin´s handling of the anomaly it is possible to identify three distinct phases. After the beginning of an anomaly the Adwin starts dropping the older buckets containing the normal values. The average of the window adapts to the anomaly´s one. Then there is a second phase where the window is composed only by anomaly values. In this stage the window increases its size until the anomaly ends. Finally, the stream returns to a normal behavior. The Adwin drops the blocks containing abnormal values. Thus, the average of the window returns to pre-anomaly value.

If the anomaly is not longer than two times the minimum length of the sub-window we would not observe the second phase. Figure 5.10 shows two graphs plotting the average inside the Adwin window. These two graphs have been obtained by inserting Type A anomalies. For this case, the values of the anomaly has been set to a constant value of 0.3 (instead of having a variable range). This allows to have a clearer example.

The left graph shows an example where the injected anomalies are longer than two times the minimum sub-window. So for each anomaly, it is possible to distinguish the three phases. On the right graph the length of the anomaly is shorter than the minimum length of the sub window. Then there is not window full of anomaly´s values. Therefore, the maximum value reached by the average of the window is smaller than 0.3.

It is possible to observe that, differently from the other graph, after the increase of the average there is not a stable region, but the average start decreasing right away. Moreover, it is of interest to note, that following a first small decrease the average suddenly increases. After that, there are one or more abrupt change that brings the window average back to the pre-anomaly value.



(a) Anomaly's length between 4000 - 7000

(b) Anomaly's length between 800 - 1500

Figure 5.10: Average of the Adwin window in the laser-bending data-set with $\delta = 0.002$ and minimum length of the sub-window=2000

This behavior can be explained by looking at the Adwin´s window composition after the end of the anomaly, shown in figure 5.11. The figure refers to a case where the length of the anomaly is shorter than two times the minimum length of the sub-window.

As soon as the anomaly ends, the algorithm starts processing normal values. The Adwin compares a left sub-window composed by pre-anomaly and anomaly values with a right sub-window composed by the remaining part. This means that the averages are similar. So no change is detected. Due to this reason, the algorithm detects that the stream has returned to the normal behavior with some delay.

In the first change alert after the end of the anomaly the algorithm detects that the newer values are smaller than the older ones. However, it is interesting to notice that as effect of this shrink the window increases its average. Indeed, the algorithm is dropping the oldest buckets containing the pre-anomalies (and therefore smaller) values. So the new window would have a higher percentage of values related to the anomaly.

Finally, as the algorithm continues to process the after-anomaly values it detects new changes and the average inside the window goes back to the normal value. In

these cases the shrinks drop the anomaly´s values causing a fall in the window´s average.

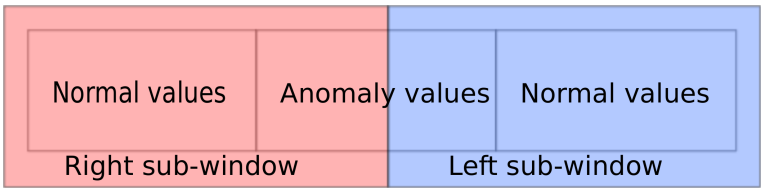


Figure 5.11: Adwin window composition

A small diversity in the streams' behavior does not change the results obtained when dealing with anomalies that greatly differ from the data. For this reason we have chosen to modify just the 'laser-bending' data-set. The results obtained with the others data-sets would be similar.

The anomalies of the first group are linked to possible and somehow predictable situations. So we analyzed their behavior and determined the best parameter setting. In this case, the anomalies are not linked to a predictable behavior of the machines. For example, we have no hints of how a sensor behaves when it is broken. So in the following tables, we provide the results obtained with various parameters settings. Moreover, we used a common threshold and fading factor for the two cumulative sum of the Page-Hinkley. For each anomaly type, we tried several parameters setting. Then for each of them, we run our algorithms 100 times.

In the result tables (5.9, 5.11, 5.13 for the PH and 5.10, 5.12, 5.14 for the Adwin) the first columns refer to the parameters (two for the Adwin, three for the Page-Hinkley).Then we used the measurements about FP, TP and TN to report the recall and precision metrics in the next two columns. Finally, the last column shows the detection delay.

Table 5.9: Changes detected by Page Hinkley with anomalies of type D

Threshold	Fading Factor	Change Allowed	Recall	Precision	Detection Delay
4	0.99	0.00	1.000	0.028	6.11
8	0.99	0.00	1.000	1.000	14.54
8	0.99	0.08	1.000	1.000	24.27
20	0.98	0.00	0.854	1.000	33.67
20	0.99	0.00	0.871	1.000	67.45
50	0.99	0.00	0.717	1.000	53.14

The tables 5.9 and 5.10 shows the results obtained by injecting the anomalies of type A. In this case (but also in the others) the Page-Hinkley is able to recognize the changes faster than the Adwin. On the other hand, it is possible to observe that the latter provides good results even if the chosen parameters are not the best ones. Instead, a non-optimal parameter setting causes the Page-Hinkley to obtain far worse results, i.e. the Adwin ´s behavior is more stable.

Table 5.10: Changes detected by Adwin with anomalies of type D

δ	**Recall**	**Precision**	**Detection Delay**	**Threshold**
10^{-7}	0.998	0.328	95.34	73.80
10^{-9}	1.000	0.662	106.80	70.89
10^{-12}	1.000	0.979	123.81	76.62
10^{-15}	0.997	1.000	141.62	78.52

These trends are confirmed also by the tables 5.11, 5.12, 5.13 and 5.14. On the other hand, the Page-Hinkley always manage to reach a perfect recall and a precision with at least one of the parameter setting. The Adwin does not obtain the same results, although, in the first two tables, with some parameter setting the precision and recall are very close to one.

With the last kind of anomalies, the type F, both the algorithms struggled to obtain the same results. However the Page-Hinkley still reaches the best possible results in the second row. The normal power consumption of the machine is between 0 and 0.09 and its average is around 0.2-0.3. By setting an allowed change of 0.08 we are saying to the algorithm to not consider all the part of the stream that is not an anomaly. So there is no possibility of having false positives. On the other hand with an allowed change of 0.08 we would never be able to detect if the machine completely stops or it starts working all the nights. With the type F anomalies, both the algorithms witness an increase of the detection delay.

From the tables referring to the Adwin algorithm it is possible to see how decreasing the δ improves the precision. However, this also increases the risk of missing some anomalies (i.e. more TN and decrease of the recall). The delta is inversely proportional with ϵ. If we decrease the δ there will be less false positives because we are requiring less accurate detections. The increase of the threshold in the PH have the same effect. It also increments the risk of missing some anomalies.

In these tests (with the anomaly of the second group) the PH detects the anomalies after few tuples its starts. So the fading factor multiply the cumulative sum lesser times (the detection delay reaches a maximum of 84 for the anomalies of type F while for the anomalies of the first group the average delay was of 966). For this reason the effect of the fading factor is not as relevant as it was in the anomalies of the first group.

5.3 Power demand/supply profiles of customers in different energy sectors

As already described in the problem description, in order to extract valuable information from these data-sets we need to perform a temporal aggregation. For these

Table 5.11: Changes detected by Page Hinkley with anomalies of Type E

Threshold	Fading Factor	Change Allowed	Recall	Precision	Detection Delay
4	0.99	0.00	1.000	0.301	7.41
8	0.99	0.00	1.000	1.000	11.07
8	0.99	0.08	1.000	1.000	17.93
20	0.98	0.00	0.881	1.000	29.62
20	0.99	0.00	1.000	0.988	43.81
50	0.99	0.00	0.745	1.000	53.23

Table 5.12: Changes detected by Adwin with anomalies of type E

δ	Recall	Precision	Detection Delay	Threshold
10^{-7}	1.000	0.580	94.94	81.22
10^{-9}	1.000	0.853	106.89	85.62
10^{-12}	0.999	0.996	123.94	86.18
10^{-15}	0.998	1.000	141.62	85.73

Table 5.13: Changes detected by Page Hinkley with anomalies of type F

Threshold	Fading Factor	Change Allowed	Recall	Precision	Detection Delay
4	0.99	0.00	1.000	0.0319	6.82
8	0.99	0.00	0.999	1.000	13.59
8	0.99	0.08	1.000	1.000	24.3
20	0.98	0.00	0.873	1.000	48.24
20	0.99	0.00	0.988	1.000	84.65
50	0.99	0.00	0.767	1.000	70.96

Table 5.14: Changes detected by Adwin with anomalies of type F

δ	Recall	Precision	Detection Delay	Throughput
10^{-7}	0.984	0.449	197.52	81.07
10^{-9}	0.971	0.811	226.42	82.17
10^{-12}	0.940	0.988	264.70	74.55
10^{-15}	0.912	1.000	271.61	76.80

cases, a temporal aggregation of 1 hour have been chosen. Differently, from the Aachen´s data-sets, that showed an explicit periodic behavior (the machines start to consume power at around 15:00 and stop at 15:00-17:00), the Terni´s data-sets does not have such a regular behavior.

5.3.1 Customer-Office 1

In figure 5.12 we can see the temporal evolution of the power demand aggregated by hours of the Customer-Office 1 data-set. During the day the data-set shows a quite constant power demand with two peaks: one at lunchtime and the other one later in the evening. However, these two peaks do not have regular starts and end hours. Also, their values may be quite different from day to day.

The data-set is characterized by a big increment of power demand from 12:00 on June 20$^{\text{th}}$, 2014 until 8:00 on July 2$^{\text{nd}}$, 2014. Furthermore, there are other interesting situations that we would like to detect: from 8 August 2014 to 13 August 2014, the power demand decreases. Then there is an interval, from the second half of December 2014 until the beginning of March 2015, where the power demand is higher than the previous months. The increase in power demand starting on 23 July 2015 represents the last interesting situation.

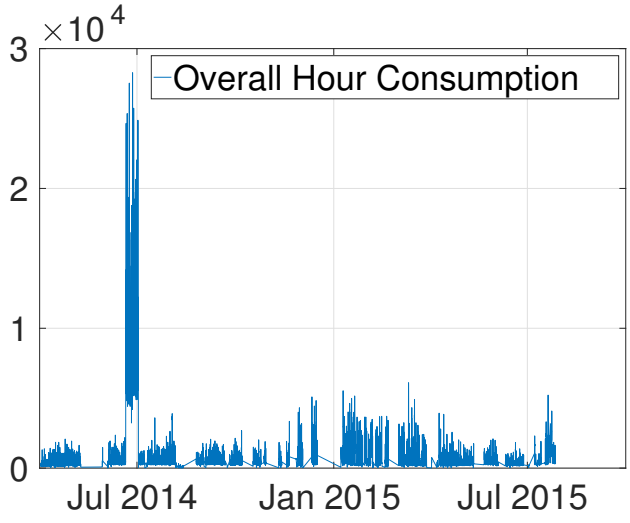


Figure 5.12: Power demand of customer office 1 data-set aggregated by hours

By confronting the graph at figure 5.12 with the right graph of figure 5.13 it is possible to compare the Adwin´s average with the values of the stream. The two graphs show how the Adwin average follows the stream´s behavior. The left graph of figure 5.13 shows the evolution of the window size. Every abrupt change of the average estimated by the Adwin is correlated with a shrink of the Adwin´s window.

When the Adwin starts processing the anomaly starting on June 20$^{\text{th}}$ the average increases slowly. As soon as there are two different enough sub-windows, the Adwin drops the oldest bucket causing a quick change in the average.

The two tables at 5.15 show the anomalies detected by the two algorithms. With a δ of 10^{-9} and a minimum length of the sub-window of 50 the Adwin detects 16 changes. Three of these are related to the aforementioned anomaly (one for alerting the anomaly and two for alerting the return to the normal behavior).

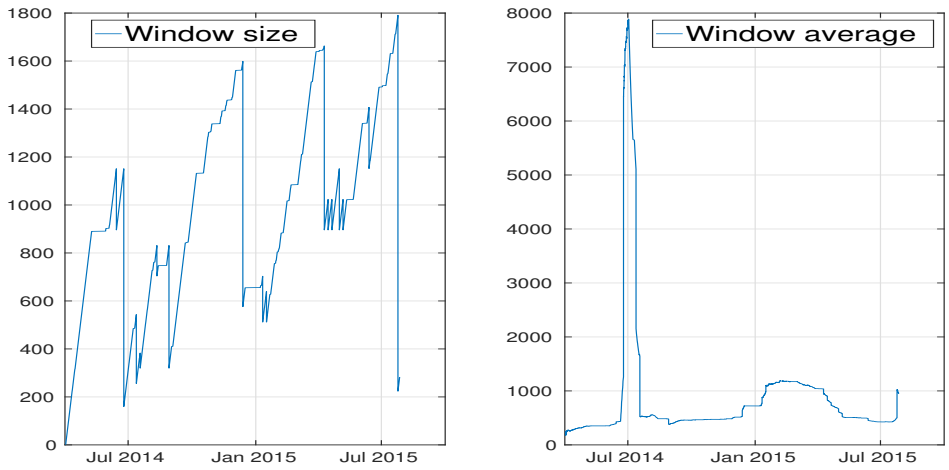


Figure 5.13: Size (left) and average (right) of the Adwin with a $\delta = 10^{-9}$ and minimum length of the sub-window = 50 for the customer office 1 data set

Table 5.15: Changes detected by Adwin and by Page-Hinkley for customer office 1

Timestamp	Avg Change
2014-06-14 03:00:05	-33.412
2014-06-24 19:00:12	-5194.679
2014-07-12 21:04:06	2953.330
2014-07-18 10:04:48	1113.566
2014-08-11 12:07:22	-4.631
2014-08-28 22:04:54	96.939
2014-12-13 09:04:41	-112.071
2015-01-10 23:04:43	-102.203
2015-01-16 11:00:07	-123.231
2015-04-09 19:00:13	97.609
2015-04-15 06:04:39	71.090
2015-04-20 19:00:16	48.859
2015-05-01 11:04:58	58.924
2015-05-06 19:04:56	61.457
2015-06-13 05:04:51	35.746
2015-07-24 22:04:41	-517.566

Timestamp	Change
2014-06-20 14:00:11	Increasing
2014-06-21 09:00:05	Decreasing
2014-06-21 17:04:35	Increasing
2014-06-22 03:01:36	Decreasing
From 06 - 22 to 07-02	Oscillating
2014-07-02 16:00:11	Decreasing
2014-08-10 04:00:25	Decreasing
2015-01-15 23:04:57	Increasing
2015-03-04 07:04:38	Decreasing
2015-03-25 19:05:00	Decreasing
2015-07-23 22:00:00	Increasing

With a threshold of 30000, a fading factor of 0.9 and change allowed set to zero, the Page-Hinkley detected 27 changes. The anomaly is the cause of 22 detected changes (the table does not report all of them).

Figure 5.14 shows the evolution of the negative and positive cumulative sum for the Page-Hinkley. It is possible to see how the anomaly greatly influences the two sums. During the anomaly, they go multiple times above (or below) the threshold. After that, the decreasing of the power demand on August 8, 2014, makes the negative sum go below the threshold. In December the power demand increases and our positive sum does the same, causing another change detection. Then the return to the previous behavior is detected in May as a decrease of the negative sum. Finally, the last detection is related to the increase of power demand on July 23$^{\text{th}}$ 2015.

The Adwin algorithm detects some additional changes on: June 14$^{\text{th}}$, 2014 , May 1$^{\text{st}}$, 2015, May 6$^{\text{th}}$, 2015, and June 13$^{\text{th}}$, 2015. The change on June 14$^{\text{th}}$ makes the window drop 256 values. These values had a slightly lower average than the rest of the window. Depending on what we want to consider as meaningful changes we may need to decrease the δ in order to avoid this detection (with a $\delta = 10^{-12}$ the first detection is related to the anomaly).The other mentioned changes are related to the increment of power consumption from the half of December 2014 to the beginning of March 2015. This means that there are three months between the end of the high power consumption interval and the last detection correlated to it (i.e. the shrinks on June 2015 happens because the Adwin window still contains values from the interval December 2014 - March 2015).

Since the decrease in power consumption is gradual the Adwin window slowly follows the trend with several small shrinks. This behavior of the algorithm when dealing with slow gradual changes has been also analyzed in [10].

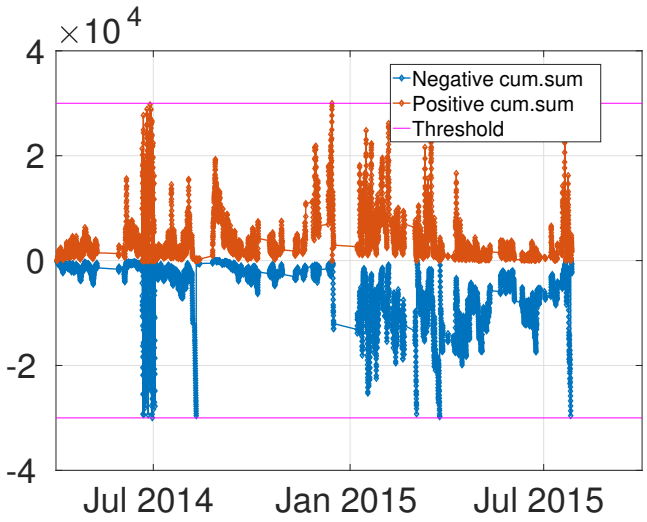


Figure 5.14: Page-Hinkley results on customer office 1

5.3.2 Customer Commercial 1

We have now tested our algorithms against a data-set containing an anomaly that differs greatly from the normal behavior. After that, we would like to see how they perform in presence of outliers (i.e. just one or few abnormal values). To this purpose, we analyzed the "Customer Commercial 1" data-set that is characterized by the presence of an outlier at 19:00 on March 11th. As usual for the data-sets belonging to the Terni group the stream has been pre-aggregated per hour.

Figure 5.15 shows the evolution of the power consumption for this data-set. Since the outlier reaches a value of 699330, a maximum value of the y-axis to 40000 has been set.

The data-set is characterized by a higher power consumption during the weekdays and a lower one during the weekends. Each weekday presents higher power demand from (roughly) 09:00 to 20:00. Moreover, after the end of March, the data-set shows a general decrease of power consumption. According to this behavior, the data-set can be conceptually divided into two parts: the first one starts from the beginning of the stream and ends at the end of March. The second one covers all the rest of the stream.

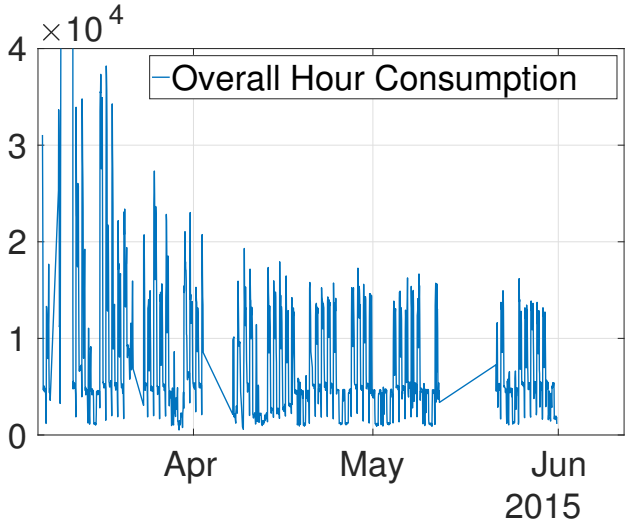


Figure 5.15: Power demand of customer commercial 1 aggregated by hours

5.3.2.1 Page Hinkley Analysis

With a threshold of 50000, a change allowed of 3000 and a fading factor of 0.9 the algorithm detects almost all the changes (16 out of 17) in the first part of the data-set.

The only alerted change in the second part is on April 12th. This detection is linked

to the aforementioned reduction of power consumption. After April 12 the algorithm does not alert any other changes in all the remaining part of the data-set. Although during the weekends the negative cumulative sum decreases, it is not enough to trigger a detection.

On the other hand, lowering the threshold to 30000 makes the algorithm detect also the weekends. However, in this case, the algorithm would alert a change for each day of the first part of the stream. Indeed, it is not possible to have the same parameter setting for two different stream´s behavior (unless they have a similar values range). Therefore, if we want to detect the same kind of anomalies in both parts of the stream we would need a different parameter setting. Also the analysis of the "customer office 1" data-set showed that the Page-Hinkley was not able to adapt to the anomaly. So it detected multiple changes. However, in that case we were dealing with an anomaly (i.e. limited in time), so it was acceptable that the algorithm continuously alerts an abnormality.

5.3.2.2 Adwin Analysis

The length of the stream after the aggregation by hour is only of 1565 input tuples. So, in order to have a bigger amount of checks the *Adwin clock* has been reduced to 12. Also, a smaller (with respect to the previous data-set) minimum size of the sub-window (30) and a bigger δ (0.05) have been set.

The two graphs of figure 5.16 show the evolution of the Adwin´s window size (left one) and of its average (right one). The last one exhibits an abrupt increment of the average on March 11^{th}. This abrupt increment is caused by the outlier.

More specifically, the outlier alone changes the average from 10769 to 27563 (kW/h). We have previously seen that these kinds of abrupt changes of the window´s average are related to a change detection (and a shrink of the Adwin window). However, this case is different: from the left graph, it is possible to see that this increment does not cause a shrink of the window.

Indeed, the Adwin shrinks its window for the first time at 22:00 of April 19^{th}, after more than one month with respect to the outlier.

If we remove the outlier and run the Adwin with the same parameters the algorithm would detect more changes (18 against 12). Of these changes, 2 are related to the decrease of power after the first part, at the end of March. The other 16 are linked to the lower power consumption during the weekends.

Unlike the Page Hinkley algorithm, the Adwin´s more elaborate structure is able to adapt to the stream´s behavior variation and detect changes in both parts of the data-set. However, despite the algorithm successfully identifies the intervals with low power consumption, the meaning of the obtained results is arguable. We already

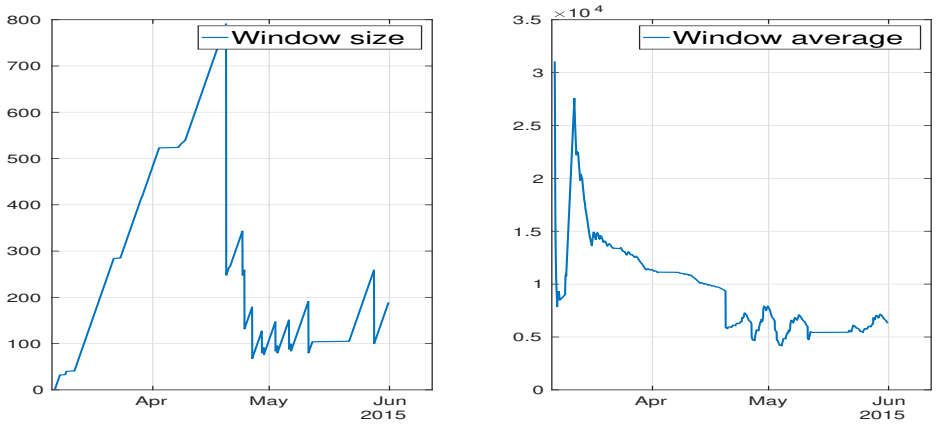


Figure 5.16: Size and average of the Adwin window for the customer commercial 1 data-set

expect a less consumption during the weekends, there is no point in detecting it as an anomaly.

More interesting results may be obtained by considering and comparing separately the day of the weeks (i.e. the power demand of the Mondays, the power demand of the Tuesdays and so on). Similarly to what already done for the data-sets about electricity consumption a manufacturing environment with the PH time-slots.

5.4 Time performances

This section evaluates and compares the time performances, as latency and job execution time, of the two algorithms. The evaluation chapter described how these metrics have been measured.

In these tests, the job execution time showed some variance. We do not know whether the reason is due to Apache Flink or to the computer used for the tests. However, the average job execution time for the Adwin is 665 ms. Its latency is of 113.41 ms. Instead, the Page-Hinkley´s job needs 615 ms and the latency introduced by the algorithm is of 68.13 ms. Since the analyzed data-set contains 132628 input tuples the throughputs are: 215.66 tuple/ms for the Page-Hinkley and 198.25 tuple/ms for the Adwin. The two jobs have the same structure, the only difference is the algorithm. So we expect that in terms of execution time the only variation is the one introduced by the algorithm.

The difference between the two job execution times of the algorithms is of 54 ms. The variation between their latencies is of 45.28 ms. As expected the values are close. The discrepancy may be caused by the variance of the job execution time.

The latency of each tuple is higher in the first parts of the stream. Indeed, at the beginning Flink it is still initializing some of the structures required by the program.

After this initial setup, the latency assumes a regular behavior. Once every *Adwin clock* tuples, the algorithm checks all the possible splits. So, for each 127 consecutive tuples with a similar low latency (from 0.0004 to 0.001 ms), there is a tuple with a considerably higher latency (0.005-0.01 ms).

Every time the algorithm drops a bucket it starts again searching for all the possible splits. So the latency of the 128^{th} tuple may show some variation depending on the number of dropped buckets.

There are also some latency values that are not correlated with the algorithm checking the splits but are very high (usually around 0.1, but they can reach also 0.5 ms). It is possible to suppose that these values are related to the variance of the job execution time and that they depend on how Flink internally organize the job.

Also, the first tuples analyzed with Page-Hinkley shows higher latencies. After that, the latencies assume values from $3.5 \cdot 10^{-4}$ to $5.5 \cdot 10^{-4}$. In this case, the tuples related to a change detection does not show a bigger latency. Indeed, the operations performed by the PH when detecting the change are very simple. However, as in the Adwin, there are higher and not regular values (from 0.008 to 0.0015).

6

Discussion

The results chapter provided several test cases while giving an explanation for some behaviors that could be observed. This chapter provides some additional thoughts about the results, giving suggestion on different approaches that may be used in future works.

6.1 Comparing the results

The results obtained on the data-sets about electricity consumption in a manufacturing environment show that with an adequate parameter setting the Adwin is able to identify the most meaningful situations.

On the other hand, the Page-Hinkley shows good performances only for some kinds of abnormal behaviors. Table 5.8 clearly outlines the limitation of this approach. The TP% of the PH is way lower than the one obtained with the Adwin (table 5.7). Moreover, setting the parameters is often troublesome and possible only with an in-depth knowledge of the data-set. Think, for example, of the accurate choice of the fading factor performed in section 5.2.1.1.

On the other hand, setting the δ of the Adwin was an easier task and did not require any knowledge of the data. However, we have seen that the setting of the minimum length of the sub-window may have a relevant impact on the number of detection and in the algorithm behavior. In this case, knowing the data may be of help when setting this parameter, as illustrated in section 5.2.1.2.

One advantage of the PH with respect to the Adwin is the smaller delay in detection. Moreover, after the end of the anomaly, the Adwin needs more time to re-adapt to the normal behavior. For this reason, its performances drop when the interval between the anomalies is not big enough (table 5.7). Additionally, this delay may make harder the interpretation of some changes. Why not cite as a proper example the situation described in section 5.3.1. In that case, the algorithm spotted changes linked to the anomaly long time after it was over.

On the other hand, the Page-Hinkley performs better on the anomalies of the second group. These anomalies differ greatly from the data. So with a correct parameter setting it is possible to separate normal from abnormal behavior (e.g. setting the allowed change to 0.08).

The Adwin manages to obtain quite good results with the anomalies of the second group. The precision and the recall are often close to 1. However, these results are worse than the PH´s ones. The worst results are obtained when dealing with anomalies of type F (on table 5.14). This behavior can be explained by looking at the dependency between ϵ_{cut} and the variance. If the variance is high also ϵ_{cut} will be bigger. Then there must be a considerable difference between the sub-windows to trigger a shrink. The anomalies of type F are the oscillating ones, so the variance is higher and detecting an anomaly may be harder.

Also the Page-Hinkley witnesses a worsening of its performances (table 5.13). Its detection delay almost double for all the different parameter setting (with the exception of the test with $\lambda = 50$ that is less meaningful given the high percentage of FNs). The sign of the anomaly is not constant so the algorithm needs to process more tuples before one of the two cumulative sums goes above or below the threshold. This kind of anomaly is hard to detect when considering the implementation with just one cumulative sum. In that case, the increment caused by the positive values of the anomaly would be compensated by the decrease caused by the negative values.

The Adwin results on the anomalies of type F are correlated to its behavior on the Customer Commercial 1 data-set (section 5.3.2. In that case, the presence of an outlier influences deeply the behavior of the algorithm. After having processed the outlier, the Adwin is not able to detect the decrease of the power consumption after the first part of the data-set. Once again, this behavior can be explained by looking at the dependency between ϵ_{cut} and the variance (σ^2). As soon as the algorithm introduces the outlier in its window, σ^2 rises. Only after a month, the variance normalizes and the Adwin detects a shrink. In this case, the Page-Hinkley proved more robust. It was able to detect the outlier instantaneously. Furthermore, after the detection the algorithm is not influenced anymore by the outlier. Section 5.2.4.1 shows another example of this behavior.

The results obtained on the two data-sets about power demand profiles underline an important limitation of the Page-Hinkley algorithm. We have seen that while dealing with the customer office 1 data-set almost all the detected changes are related to the anomaly (table 5.15) Since we are dealing with an anomaly it is acceptable that the algorithm continuously detects it. However, consider what would happen if we had a change in behavior. The algorithm would continue to detect the new behavior as anomaly. This is not acceptable.

We have seen in section 5.3.2 that it was not possible to find parameters valid for both parts of the stream. So it is possible to say that if the stream changes its

behavior the parameter setting will not be adequate anymore. This is not always true. For example, if the change in behavior just translate the stream´s values then the old parameter setting will be still valid (i.e. the difference between the input tuples and the average is the same of the old behavior).

On the other hand, when dealing with the anomaly, the Adwin was able to adapt to it. So in table 5.15 it is possible to see that the changes related to the anomaly are just three out of 16. Also in 5.3.2 the Adwin managed to detect changes in both part of the data-set.

6.2 Time performances

In section 5.4 we have seen that the PH has a higher throughput than the Adwin. Moreover, with the exception of the random delay introduced by the system, the latency is constant.

On the other hand, the Adwin shows a higher and not constant latency. Once each *adwin clock* records the Adwin checks all the possible split of the windows. Then if a change is detected the last bucket is dropped and the algorithm starts again searching for all the possible splits. This cause another variable increase of the latency (that depends on the number of dropped buckets).

The number of possible splits is related to the window size. Thus, it is possible to expect a dependency between the latency and the length of the window. However, the analysis of the time performances did not show this behavior. Indeed, the variation introduced by checking multiple times the splits is more relevant than the one introduced by the window size.

6.3 Time-slots implementation

We have seen that the Page-Hinkley approach is quite simple and has many limitations. The standard implementation looks at the stream and compares the new input tuple with the current average of the stream.

By contrast, the PH with time-slots compares an input tuple against a portion of the stream's statistics (the one that shares the same time-slot). This approach provides a clear advantage in the analysis For example, suppose that instead of consuming power only during the working hours, the machine starts consuming a stable amount of power through all the day. This amount correspond to the average consumption of the previous behavior. The first implementation of the Page-Hinkley would compare the new values against the old average and, since they are equal,

would not detect any change. On the other hand, the PH time-slot would be able to recognize that the time-slots averages have changed. It is possible to say that by using this implementation we are essentially introducing a new information in our algorithms: the time-slots subdivision.

This implementation has been designed specifically for the case of electricity consumption in a manufacturing environment. However, it is possible to generalize this situation. In all the cases showing a periodic behavior, it may be too hard for the PH to check for changes over the whole period. So it would be better to check only for changes of chunks of the period. In this variant of the PH, these portions were the time-slots. It is important to define these chunks such that it is possible to assume a constant average inside them. In this case, the algorithm is clearly favored (since the PH has been designed for detecting changes of the average).

An easier and perhaps more intuitive way would be to pre-aggregate our stream for days. The power consumption during a day is expected to be constant. Thus, it would be easier to monitor for days with a different power consumption. Additionally, the algorithm would not need any structure to store statistics such as the time-slots averages. In this case the algorithm is comparing each value with all the previous ones, just as in the 'standard' implementation, but on a different aggregation level.

We would obtain results, such as 6.1, which interpretation is straightforward. A major drawback of this implementation would be that the PH would check only the overall power consumption of a day. If instead of using energy during the working hours the machine consumed power overnight, the PH would not notice any difference.

Table 6.1: Extract of Page-Hinkley results on robot welter data set after per-day aggregation

	Cumulative Sum	**Average Stream**	**Average Day**	**Change**
Day 1	0.00062	0.019622	0.01965	false
Day 2	0.001367	0.019982	0.22759	false
Day 3	0.007928	0.020700	0.02933	true
Day 4	0	0.20549	0.019131	false

7

Conclusions

In this thesis, we have analyzed two methods able to monitor for changes in a streaming environment. These methods have been tested over several data-sets representing different contexts. With the help of ad-hoc modifications of the data-sets, it has been possible to provide a more rigorous evaluation of the performances. The definition of an anomaly is linked with the context. So measurements such as true positives, false negatives and false positive are dependent on the specific case under test. However, the inferred characteristics of the tested algorithms are general and do not depend on the context.

The analysis of the results highlighted that using the Page-Hinkley test requires the knowledge of the data to correctly set its parameters. Still, some kinds of changes, such as the anomalies of type B, may not be detected. Whenever is possible for the stream to change its behavior the PH test should not be used, unless it is possible to specify a new parameter setting during the analysis of the stream. We have seen, with the PH time-slots, that it is possible to build a domain specific implementation that overcomes most of the PH limitation.

The Adwin is more expensive in terms of memory and computational cost. It can be used without any knowledge of the data and handles all types of changes. However, it should be coupled with an algorithm that filters the outliers since it is not able to handle them. Finally, its high delay in detection may be a problem in situations where it is critical to recognize quickly the anomaly.

The detection of the anomaly in a streaming context is an active field of study. Moreover, we have seen that the definition of an anomaly is context specific. For these reasons, it is possible to extend this work by considering different context and other methods.

Appendices

A

A.1 Composition of the Adwin window in the smart meter example

This appendix presents a way to derive the composition of the Adwin window. It refers to the case of the smart meter data-set where the window stores the statistics of 16255 records.

In our implementation, the Adwin window is composed by a list where each element i has five buckets and each bucket contains statistic for 2^{i-1} input tuples. With 16255 values the Adwin list has 12 element ($Lsize$). This mean that the buckets of the last element contain $2^{12-1} = 2048$ input tuples.

The maximum size ($Asize$) of our window with $Lsize - 1 = 11$ element is given by A.1.

$$MaxA\_size = \sum_{t=1}^{11} 5 * (2^i) = 10235 \tag{A.1}$$

When a new input tuple is added the $Asize$ is incremented by just 1. However, the Adwin window structure looks completely different: the firsts 11 elements have four buckets. The 12^{th} element has just one bucket.

In order to calculate the composition it is possible to start from a situation were the 12^{th} element has two buckets and all the others have four. Our window can now contain $10236 + 4096$ records. Therefore, the remaining $16255 - 14332 = 1923$ input tuples have to be stored in the first 11 elements.

Each element has one free bucket of size 2^i. Then it is enough to convert 1923 in binary to derive the Adwin´s window composition. At the positions where there is a one in the binary representation the corresponding element in the Adwin window has 5 bucket. A zero means four bucket in that element.

In our example : 1923 = 11110000011. Thus the items number 0-1-7-8-9 and 10 will have 5 buckets, the ones number 2-3-4-5-6 will have 4 buckets. Finally, the element number 11 will have 2 buckets.

With this configuration the only possible division of the list in two sub-windows (such that both represent more than 8000 input tuples) is: the first sub-window contains 3 bucket of element 12 and 2 of element 11 (for a total of $2^{11} \cdot 3 + 2^{10} \cdot 2 = 8192$ input tuples). The second one contains all the remaining buckets (8063 input tuples).

Bibliography

[1] Apache flink: Introduction to apache flink. URL `https://flink.apache.org/introduction.html`.

[2] Adwin.java. URL `https://github.com/abifet/adwin/blob/master/Java/ADWIN.java`.

[3] Finescee. URL `http://www.finesce.eu/Trial_Site_Aachen.html`.

[4] Jira issue about latency. URL `https://issues.apache.org/jira/browse/FLINK-7608`.

[5] S. Ben-David, J. Gehrke, and D. Kifer. Detecting change in data streams. *Cornell University*.

[6] A. Bifet. Adaptive learning and mining for data streams and frequent patterns. *Doctoral Thesis presented to the Departament de Llenguatges i Sistemes Informàtics a Universitat Politècnica de Catalunya*, April 2009.

[7] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection : A survey. *ACM Computing Surveys*, September 2009.

[8] E. Cohen and M. Strauss. Maintaining time-decaying stream aggregates. In *In Proc. 22nd ACM Symposium on Princi- ples of Database Systems*, 2002.

[9] M. Datar, A.Gionis, P. Indyk, and R.Motwani. Maintaining stream statistics over sliding windows. *SIAM Journal on Computing*, pages 27–45, 2002.

[10] J. Gama. Knowledge discovery from data streams.

[11] C. Hartland, N. Baskiotis, S. Gelly, M. Sebag, and O. Teytaud. Change point detection and meta-bandits for online learning in dynamic environments. pages 237–250, 2007.

[12] E. Knorr and R. Ng. Finding intensional knowledge of distance-based outliers. In *Proc. Int. Conf. on Very Large Databases (VLDB99)*, pages 211–222.

[13] H. Mouss, D. Mouss, N. Mouss, and L. Sefouhi. Test of page-hinckley, an approach for fault detection in an agro-alimentary production system. 2004.

[14] D.-S. Pham, S. Venkatesh, M. Lazarescu, and S. Budhaditya. Anomaly detection in large-scale data stream networks. 2012.

[15] S. Sakthithasan, R. Pears, and Y. S. Koh. One pass concept change detection for data streams. In *School of Computing and Mathematical Sciences, Auckland University of Technology*.

[16] S. Subramaniam, T.Palpanas, D. Papadopoulos, V.Kalogeraki, and D.Gunopulos. Online outlier detection in sensor data using non-parametric models. *Proceedings of the 32nd international conference on Very large data bases*, 2006.

[17] T. Zhang, D. Yue, Y. W. Yu Gu, and G. Yu. Adaptive correlation analysis in stream time series with sliding windows. In *School of Information Science and Engineering, Northeastern University, Shenyang 11004, PR China*, 2008.