Delft University of Technology
Master of Science Thesis in Embedded Systems

Change Point Detection In Continuous Integration Performance Tests

Tim Arie Iskander van der Horst

PHILIPS

Embedded Networked Systems

TU Delft Delft University of Technology

Change Point Detection In Continuous Integration Performance Tests

Master of Science Thesis in Embedded Systems

Embedded and Networked Systems Group
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands

Tim Arie Iskander van der Horst
T.A.I.vanderHorst@student.tudelft.nl
timvanderhorst@gmail.com

28/06/2021

Author
Tim Arie Iskander van der Horst (T.A.I.vanderHorst@student.tudelft.nl)
(timvanderhorst@gmail.com)
Title
Change Point Detection In Continuous Integration Performance Tests
MSc Presentation Date
12/07/2021

Graduation Committee

prof. dr. ir. Koen Langendoen (chairman)	Delft University of Technology
dr. Maurício Aniche	Delft University of Technology
Remy Böhmer	Philips Medical Systems

Abstract

Software testing is an integral part of the development of embedded systems. Among other reasons, tests are frequently used to ensure that a system meets all the specifications, which is especially important when designing systems for the medical industry. Software changes that have a detrimental impact on a real-time system's performance can accumulate until the systems no longer meet the required performance levels. At this point, fixing accumulated defects can become a costly and complex endeavor. The goal of this thesis is to create a method that can detect changes in performance metrics from a continuous integration pipeline with minimal manual intervention.

To achieve this goal, we have created a novel online, univariate change detection method, which consists of a diverse ensemble of more than 10 existing change detection algorithms. The ensemble is robust against changes in distribution and requires little tuning. The contributions of this work include the proposal of a generic architecture to combine both statistics and decisions from individual algorithms. Related work uses simple majority voting for decision fusion in the ensemble - we demonstrate that an ensemble can benefit from more complex decision fusion, for example using a Random Forest. Synthetic data and a case study, using a dataset provided by Philips, are used to demonstrate that the overall ensemble is consistently able to outperform the individual algorithms in the ensemble. In addition, unlike the individual algorithms, the ensemble generalizes well to data that is not normally distributed and have not been encountered during training. Compared to the monitoring system for performance metrics that is currently being used by Philips, the ensemble is able to detect 75% more changes with a 50% lower false positive rate.

Preface

The ability to detect changes in an environment is essential to human survival [93]. Fortunately, the modern digital environment is much less life-or-death, although the ability to detect changes in these environments is still important. In this work, we highlight how change detection can be used in the context of software development when system performance tests produce too much data for a human to monitor for changes manually.

I would firstly like to thank Remy Böhmer and Gertjan Keesman from Philips Medical Systems for their expertise, extensive feedback and guidance during this thesis. I look forward to working with you as colleagues in the future.

To prof. dr. K.G. Langendoen, thank you for your valuable advice, and for challenging me to continue asking questions to gain a deeper understanding of the topic I was researching.

I would also like to extend my gratitude to dr. Maurício Aniche for being part of my thesis committee.

Last but not least, I could not have completed this without the support and encouragement of my family and friends.

Tim Arie Iskander van der Horst

Delft, The Netherlands
13th July 2021

Contents

Chapter 1

Introduction

Embedded systems depend heavily on the quality and reliability of both their hardware and software [16]. Maintaining the quality of a system can be challenging while allowing for a degree of flexibility to modify the hardware and software aspects of a system. Modifying one component of a complex system could affect the overall system operation in unpredictable, potentially detrimental ways.

Fortunately, software and hardware tests provide a way to assess the effect of those modifications. Testing is an essential part of any engineering process, used to maintain the quality of a system and to verify that a system fulfills all specified requirements.

Embedded systems that perform safety-critical operations, such as those in the medical industry, have strict functional and non-functional requirements. These requirements are important to ensure the safety and effectiveness of medical devices. Companies must prove that these requirements are met before they can bring a device to market [25].

The software development process for medical devices involves frequently running a system-level test suite to ensure that system requirements are always met. Due to the interaction between hardware and software, which characterizes embedded systems, these tests can no longer be performed on generic hardware but must instead run on the medical device's embedded hardware.

An increase in the complexity and scope of an embedded system will lead to an increment in the size of the code base, frequency of code changes, and size of the development team. Consequently, manual building, integrating, and testing of the embedded systems eventually becomes infeasible. The automation of these processes is key to achieving efficient software and hardware development. To this end, continuous integration can be used to automate and improve the process of software development [40].

Continuous integration is a set of practices that involves automating the building and testing process and running this many times per day. The functionality of the system under development is verified after every code change by running a test suite that includes different types of tests.

A test suite in the continuous integration build process may contain non-functional performance tests that aim to quantify the performance of a device on the system or component level. Once a baseline performance has been established for a specific software and hardware configuration, the impact of software and hardware changes on system performance can be determined. A change that is detrimental to a system's performance should be investigated and fixed as soon as possible to prevent costs in later stages of the development process. Figure 1.1 shows the cost to repair defects in various phases of an embedded system's development process. Fixing defects after the product has been released is approximately six times as expensive as repairing the same defects during the testing phase.

Non-functional performance requirements can define thresholds that must never be exceeded. For example, the statement *"completing this calculation should take no longer than 40 milliseconds"* can be used to create a regression test that proves this requirement is met. A test failure when the calculation takes longer than 40ms is a necessity, as this indicates a potentially serious problem that needs to be fixed. However, additional (early) warnings before this threshold is reached could also be useful. For example, if the calculation usu-

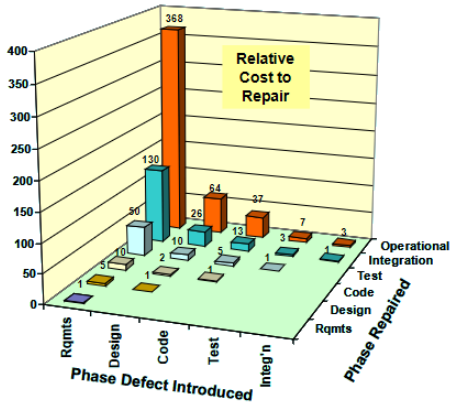


Figure 1.1: **The relative cost to repair defects in embedded systems project, from Wennberg and Bennett [11]. The graph shows that defects are more expensive to repair in later phases.**

ally only takes 30ms, but a software update unintentionally causes an increase to 35ms, the system would still be within specification. Despite this, engineers responsible for the performance may still want to be notified of this degradation to decide whether this is a problem that needs to be fixed.

Manually choosing early warning thresholds is feasible if there are only a handful of these tests and requirements. If there are hundreds, or thousands, of tests and metrics which need to be monitored this task becomes too time-consuming and thus costly. Approaching this software engineering problem as a mathematical problem allows us to analyze which changes in test outputs are significantly different from previous test results. The amount of change is now arbitrary and manually determining thresholds for each test is no longer required as this is done automatically. Nevertheless, some tuning is required to determine the desired sensitivity. A monitoring system that is too sensitive will detect a majority of changes at the cost of more false positives, which is detecting non-existent or insignificant changes. Likewise, too insensitive means some changes will be missed.

In this thesis, we will try to answer the following research question: how can changes in the performance benchmarks of an embedded system be detected automatically with minimal delay? To answer this question, we will describe the process of developing a change detection system for a specific medical device. This device has performance benchmarks that run as part of a continuous integration pipeline. The change detection system will be integrated with the same continuous integration pipeline and will be used to signal whether software or hardware changes negatively impact the overall system performance.

1.1 Context

Figure 1.2: **Philips Azurion Platform, from Philips [57]. The Philips Azurion platform is a medical X-Ray device used for image-guided therapy. This thesis will focus on creating a system to detect changes in performance benchmarks for this device.**

This thesis is done in collaboration with Philips Image Guided Therapy Systems. The medical device that is referred to in the thesis is the Azurion platform (see Figure 1.2). This is an X-Ray device designed for minimally invasive image-guided therapy. An example of an image-guided procedure is the placement of a stent inside a coronary artery. A stent is a small mesh metal tube designed to prevent arteries from narrowing, frequently used in the treatment of coronary artery disease [80]. During this procedure, X-Ray images are displayed in real-time in the operating room. To aid the hand-eye coordination of the surgeon performing the procedure, the delays between a surgeon's movement in real life and the same movement on the monitors must be as low as possible. The low-latency aspect of the medical device is critical and must be reliable.

The thesis will focus on the detection of changes in performance benchmarks for the image processing subsystem of the Azurion platform. The device has different hardware and software configurations that must be validated against strict functional and non-functional requirements. An existing set of performance tests quantifies the performance of the medical device during various medical imaging procedures and validates the non-functional requirements.

Philips has made multiple test benches dedicated to running performance benchmarks and software tests. Each test bench may have different operating systems or hardware configurations, which match configurations of devices sold to customers. A full performance benchmark is run on each test bench approximately twelve times per day. Each run generates 900-1000 different metrics which characterize system performance.

Changes in hardware and software can cause the overall system performance to change along with the performance metrics. Major changes need to be investigated and addressed by engineers and therefore the 900-1000 metrics require constant monitoring. This time-consuming task needs an automated solution.

1.2 Problem Statement

The main objective of this work is to create an automated system to monitor and detect changes in a large set of system performance metrics. The performance of the monitoring system should be optimized to reduce the operating costs for the company. Although a quantitative cost model for the detection system falls outside the scope of this thesis, there are a few important metrics that can be used to evaluate the monitoring system.

Firstly, the delay between the introduction of a software defect that affects performance and the detection of this defect by the monitoring system should be kept as low as possible. Otherwise, new software changes may be introduced which are dependent on the code containing the defect. Fixing this defect will then become more time-consuming. Detecting a defect quickly means that changes can be reverted without impacting other areas of the system. A maximum delay of 2 days has been specified from the moment the defect is introduced until it is detected.

Secondly, the number of false positives should be kept as low as possible. According to Zabyshny and Ragland [112], the perceived reliability of the automation often influences the value people place on the system warnings. A high false-positive rate may therefore reduce the perceived reliability of a detection system. Axelsson [8] suggests that most human operators will lose faith in detection systems with a false positive rate higher than 50%. The cost of fixing problems that have been ignored due the perceived unreliability is higher than fixing them immediately.

Another argument to reduce the number of false positives is the cost of a false alarm. Whenever the monitoring system signals that a change has occurred, a software developer should investigate whether the system has correctly signaled a defect. This investigation takes time that could better be spent elsewhere if it is a false alarm.

Finally, the sensitivity (true positive rate) of the change detection system should be as high as possible. This is related to the cost of a detection delay - a defect that is not detected in the early phases of the development process, but in the operational phase has a very high associated cost, as shown in Figure 1.1.

Although there are no hard requirements for the maximum false positive rate or minimum true positive rate of the detection system, we will strive to create a system that can detect

80% of changes that have occurred in the performance metrics with a maximum false positive rate of 50%.

1.3 Contributions

In this thesis, we create a novel change detection method that combines predictions and statistics from an ensemble of existing change detection methods to arrive at a final prediction for each metric. The method - evaluated using both synthetic data and performance metric data provided by Philips - outperforms multiple existing change detection methods. In addition, it is robust to different types of distributions and requires little tuning to achieve the desired performance.

Although this change detection system has been designed for a specific embedded system (Azurion), the system can be generically used to detect changes in time series for many different applications, ranging from IoT sensor outputs [39] to network intrusion detection systems [98].

1.4 Structure

The thesis is structured as follows: in Chapter 2 background information about change detection is presented. We identify a research gap, which will be addressed in Chapter 3 by presenting the design of the novel change detection method. Chapter 4 will be used to summarize the real dataset and any synthetic datasets used to evaluate change point detection algorithms. The performance of the currently implemented trend change detection system will be evaluated in Chapter 5. The algorithm used to combine the results of individual change detection algorithms will be presented in Chapter 6. Finally, the results will be summarized in Chapter 7 with potential areas for future research.

Chapter 2

Problem background

This chapter provides background information about the general problem of change point detection in time series. This chapter serves as a theoretical basis for the following chapters. Section 2.1 provides an analysis of properties related to time series. Section 2.2 provides an overview of theory related to change detection.

2.1 Time series properties

In this section, we will introduce concepts and definitions related to change point detection in time series. Finally, in Section 2.1.1 we will analyze the dataset used throughout the thesis using these properties.

We will use the following notation from Truong, Oudre & Vayatis [102]. For a one-dimensional signal or time series, $y = \{y_t\}_{t=0}^{T-1}$ the subsample with indices in the range $[a, b]$ of length $(b - a + 1)$ is represented as $y_{a..b}$. The complete signal is therefore $y_{0..T-1}$. A multi-dimensional time series is referred to as $Y_{0..T-1}$. To denote the i^{th} coordinate or dimension of a multidimensional time series, we will use $Y^i_{0..T-1}$ or $y^i_{0..T-1}$.

A time series is a set of observations of a process, each recorded at a specific time. Time series analysis and forecasting often involves describing the underlying process using a probabilistic model. For example, a simple time series model may be that individual time samples are statistically independent and that samples are observations of a random variable with a Gaussian probability density function with zero mean, or $\mathcal{N}(0, \sigma^2)$. A probabilistic model which accurately describes a process can be a very useful tool in anomaly detection, forecasting and change detection.

An important property of time series is stationarity, which implies that a time series $y_{a..b}$ has the same statistical properties as the shifted time series $y_{a+h...b+h}$ [90, Chapter 12]. A stationary time series has no trend or seasonality as the model parameters will not vary over time.

Another desired property of time series is that the observations are identically and independently distributed (i.i.d.). Independently distributed implies all observations are independent from each other given the model while identically distributed means that the observations come from the same generative distribution. This assumption is important for many machine learning applications and statistical learning methods.

2.1.1 Real dataset

The dataset which is used in the thesis can be evaluated according to the properties of time series listed above. The dataset is a multidimensional time series with 909 coordinates. Each row in the dataset represents the results of a performance benchmark which is run as part of a test suite for an embedded system. The coordinates represent different metrics calculated for this performance benchmark. The combination of all signals reflects the state of the system, which changes with every modification to the source code and hardware. In Chapter 3 an analysis of the data shows that these modifications frequently result in changes in the distributions of observations. The model parameters for each of the 909

signals may therefore change over time, which means the dataset is not stationary. These changes, however, do not occur continuously. In between these model parameter changes, each of the signals can be considered piecewise-stationary.

Using the same reasoning as above, the signals are not identically distributed due to changes made to the system which cause changes in model parameters. Between these model parameter changes, each of the signals can be considered piecewise i.i.d.

2.2 Change point detection

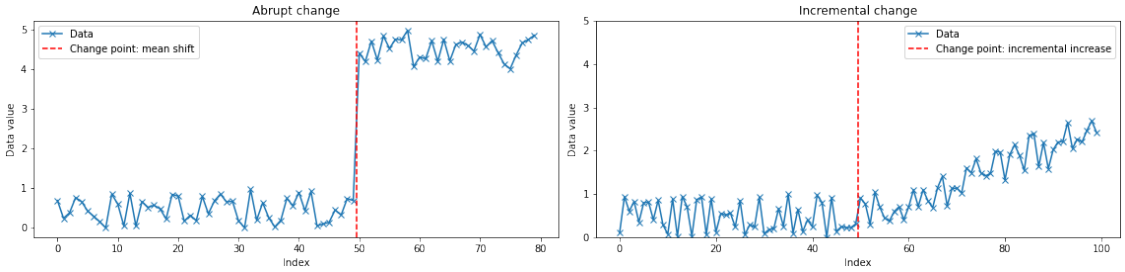


Figure 2.1: **An abrupt change compared to a gradual change in a process. The left figure shows an abrupt increase in the mean at index** 50**, while the figure on the right shows a gradual increase in mean after index** 50**.**

The concept of "change" is difficult to define and is often heavily context-dependent [37]. Figure 2.1 shows two types of change which may occur in a process. Faithfull [37] identifies four types of changes which are regularly discussed in the literature: abrupt, incremental, gradual changes and reocurring concepts. Abrupt and incremental changes are most relevant for the specific change detection application in this thesis and are highlighted below.

The left hand graph shows an abrupt change, more specifically an increase in the mean between index 49-50. In the context of embedded devices, this graph may show the average time required to perform some calculation on the device. After the 49th run of the test suite, a firmware update is performed which affects a performance critical component of the code. The average time required for the calculation increases in future measurements.

In some cases, changes may not be so abrupt and may occur more gradually, such as in the second figure. These incremental changes could, for example, be caused by performance degradation of microprocessors due to transistor aging [64]. Although these incremental changes may also occur in the performance benchmarks, these types of changes fall outside the scope of the thesis.

The problem of change detection was first described by Page [87, 88]in the context of quality control for an industrial application. Change detection usually involves monitoring observations of a stochastic process for deviation from expected or previously observed output.

Change detection algorithms have a number of important characteristics that allow us to analyze and categorize them. Below, we list the different categories of change detection algorithms with a brief explanation of the characteristics, followed by a more in-depth explanation of the important concepts from Section 2.2.1 onward.

The first characteristic used to analyze the algorithm is whether the algorithm analyzes a single/batch of observations or whether the algorithm considers the entire history of observations. The former is known as online change detection, the latter is known as offline change detection. Chen et al. [24] define online algorithms as those which have complexity bounded by the number of bits required to represent a single/batch of observations. The complexity is therefore independent of number of previously observed data points.

A second characteristic is the number of time series which are analyzed simultaneously. Some algorithms are designed to operate on a single-dimensional time series $y_{0..T}$. These algorithms are known as univariate change detection algorithms. Most complex systems have many different observable features and therefore produce multi-dimensional time series when

observed. Combining relevant information from multiple time series may enable the detection of smaller changes compared to the univariate analysis of the individual components. This approach is known as multivariate analysis.

Another characteristic is whether the algorithm makes assumptions about the underlying distribution of the data. A typical assumption is that the underlying distribution of a process is Gaussian. This assumption allows for the description of a time series using only two parameters: μ, σ^2 or the mean and variance respectively. These assumptions may be used to define worst-case bounds and expected performance of a change detection algorithm. Algorithms which make these assumptions are known as parametric. If no such assumptions are made, the algorithm is categorized as non-parametric. Non-parametric algorithms are typically more robust but often require a large number of observations before being able to signal a change.

2.2.1 Offline change detection

In offline change detection, the entire history of realizations $y_{0..T}$ is considered to contain an arbitrary number of change points which are to be detected. The samples between two change points at indices t_a, t_b are known as a segment of the time series. Offline change detection is therefore also known as time series segmentation, where the goal is to determine which collection of indices $\mathcal{T} = t_1, ..., t_n$ will result in an optimal segmentation.

In a comprehensive analysis of offline change point detection methods, Truong, Oudre & Vayatis [102] describe change detection methods as the combination of the following three elements

1. **Cost function** - a measure of homogeneity of a range of samples $y_{a..b}$. If there are no changes in the signal, the cost function is expected to be low. If there are one or more change points in the range of samples, the cost function is expected to be high.

2. **Search method** - the method used to solve the problem of detecting an unknown number, which can be formulated as a discrete optimization problem

$$\min_{\mathcal{T}} V(\mathcal{T}) + pen(\mathcal{T}) \tag{2.1}$$

   where $pen(\mathcal{T})$, short for penalty, is a measure of complexity for segmentation $\mathcal{T}$ and $V(\mathcal{T})$, short for value of T, is a criterion that must be minimized, such as the sum of costs of each segment.

3. **Constraint** - a complexity penalty related to the amplitude of the changes that are detected, used to prevent too many changes from being detected.

Truong et. al. [102] categorize cost functions as either parametric or non-parameteric. Parametric methods are focused on detecting changes in parameters such as the mean, scale and require certain assumptions to be made about the data such as the type of distribution. Non-parametric cost functions are based on non-parametric statistics such as kernel estimation or empirical cumulative distribution functions.

2.2.2 Online/Quickest change detection

In online change detection the algorithm runs in parallel with the process that is being monitored. The goal is to detect changes as quickly as possible, ideally as soon as they occur. Online change detection is also known as quickest change detection.

A considerable amount of research exists into the problem of online detection of change-points in time series [5, 10, 103]. Basseville and Nikiforov [10] formulate online detection as follows: let $y_{0..k}$ be a sequence of observed random variables with conditional density $p_\theta(y_k|y_{0..k-1})$. Before the change occurs at unknown time t_0, the conditional density is constant and equal to θ_0. After the change occurs, the conditional density is again constant and equal to $\theta_1, \theta_0 \neq \theta_1$.

There are a few different variants of the problem:

1. θ_0 is known, θ_1, t_0 are unknown. The model parameters belonging to the system in a normal state are known and we wish to measure deviations from this normal state to an abnormal state. This normal state is known as the in-control state while the abnormal state is known as out-of-control. This application is useful in industrial quality control settings [87], among others.

2. θ_0, θ_1 are known, t_0 is unknown. Model parameters before and after the change point are known, but the exact moment at which the change will occur is unknown. An example application is monitoring a control system which has multiple known states, except it is unknown when a state transition occurs.

3. θ_0, θ_1, t_0 are all unknown. Model parameters before and after the change point are unknown. Solutions often involve these unknown estimating model parameters using the observed data. There are more unknowns which need to be considered which makes this problem more general and more difficult.

The application of the thesis is most closely related to the third problem. The in-control state of the system, θ_0, is not known as the system is constantly being modified. We will therefore focus on solutions to this variant of the online change detection problem.

CUSUM

An important, frequently used algorithm in online change detection is the CUSUM (CUmulative SUM) algorithm by Page [87]. The algorithm is based on hypothesis testing using the following two hypotheses:

$$\begin{aligned} \mathbf{H}_0 &: \theta = \theta_0 \\ \mathbf{H}_1 &: \theta = \theta_1 \end{aligned} \tag{2.2}$$

The log-likelihood ratio s_i (Equation 2.3) is calculated for each new observation y_i and added to a cumulative sum of likelihood ratios, S_k (Equation 2.4). This cumulative sum is used to define a decision function G_k (Equation 2.5). If this decision function is greater than some threshold h, the null hypothesis can be rejected.

$$s_i = ln \frac{p_{\theta_1}(y_i)}{p_{\theta_0}(y_i)} \tag{2.3}$$

$$S_k = \sum_{i=0}^{k} s_i \tag{2.4}$$

$$G_k = S_k - \min_{0 \leq i \leq k-1} S_i \tag{2.5}$$

This original version of the algorithm relies entirely on the log likelihood ratio defined in Equation 2.3, which assumes that the the probability density function both before and after the change is known. As implied above, this is not the case for the application under consideration. Various extensions have been made to the original CUSUM algorithm in the cases when θ_0, θ_1 are unknown. Granjon [43] states that a value of θ_1 is typically assumed based on previous observed changes and that θ_0 can be estimated using maximum likelihood estimates. This leads to a frequently observed formulation of the CUSUM statistic, used to detect changes in the mean of an i.i.d. Gaussian signal [83]:

$$S_k = \max(0,\ S_{k-1} + \frac{\delta}{\hat{\sigma}_0^2} \left(y_k - \hat{\mu_0} - \frac{\delta}{2} \right)) \tag{2.6}$$

In this formulation of the CUSUM problem there are two parameters which can be adjusted to modify the performance. The first is δ, which is related to the expected magnitude of the change that is to be detected and h, which is a threshold used to signal once G_k crosses this value.

Many different extensions have been made to the CUSUM algorithm, such as estimating probability density functions $p_{\theta_0}, p_{\theta_1}$ using kernel density estimates [86], online monitoring

for changes in variance [50], adding transformation rules for increased robustness against outliers [78], multivariate extensions [28, 53, 60, 108] and adding methods to improve the initial response of the algorithm [77].

2.2.3 Univariate

Univariate change detection algorithms, such as the CUSUM algorithm, are limited to analyzing a single coordinate of a data stream. There are many examples of univariate algorithms, however an important category includes those which are based on geometric moving averages, such as the exponentially weighted moving average (EWMA) chart [56]. This control chart is used to monitor small changes in the mean by establishing control limits based on estimations of the standard deviation and an exponentially weighted moving average: $\text{EWMA}_k = \lambda y_k + (1 - \lambda)\text{EWMA}_{k-1}$. Control limits indicate a range of acceptable, in control values for a statistic such as the EWMA_k. If the statistic falls outside the control limits a change can be signaled. Similar algorithms exist for monitoring changes in the variance [21].

2.2.4 Multivariate

The disadvantage of limiting change detection to a single coordinate of a data stream of a complex system whose state can be observed using a number of different features is the potential loss of information due to interactions between features. A device may, for example, have different types of sensors measuring the same physical process. The analysis of all data streams at once instead of analyzing each sensor output separately may allow for the filtering of outliers or identifying faulty sensors more quickly.

Different multivariate change detection algorithms have been created, many of which are extensions of univariate algorithms such as the multivariate EWMA chart [76] and multivariate CUSUM [28, 60]. Another frequently used method is Hotelling's T^2 test [54], which can detect changes in the mean of correlated, normally distributed time series.

The problem with these "traditional" methods is that they do not scale very well to high-dimensional data. According to Wang et al. [106], these methods tend to suffer from the "curse of dimensionality" which makes parameter estimation and detection of abnormality in high-dimensional data difficult due to the huge amount of reference data required. Another problem is that for complex systems, changes may not affect all of the coordinates of the data stream. These types of changes are known as sparse changes, and are generally more difficult to detect than dense changes which affect all of the coordinates at once.

Multiple offline methods exist to detect sparse changes in multivariate time series [27, 44, 89, 105], with significantly less literature available on online multivariate change detection methods able to detect sparse changes. An exception is OCD by Chen et al. [24], although this method is limited to detecting mean shifts in Gaussian distributions.

2.2.5 Parametric

Parametric change detection are methods which make assumptions about the underlying distributions of the data. CUSUM is an example of a parametric change detection method due to the use of the probability density functions in Equation 2.3.

If, for example, we are observing count data of a Poisson Process as opposed to a Gaussian process, detecting changes in the estimated rate parameter may be more accurate than detecting changes in the mean.

2.2.6 Non-parametric

Another class of change detection methods are those which do not make any assumptions about the type of underlying distributions of the data, also known as non-parametric methods.

One approach is to estimate the probability density functions (pdfs) $p_{\theta_0}, p_{\theta_1}$ used in the CUSUM approach using nonparametric methods. This can be done, for example, using kernel density estimation [15] [30]. An estimated pdf can be used in a number of ways: to calcu-

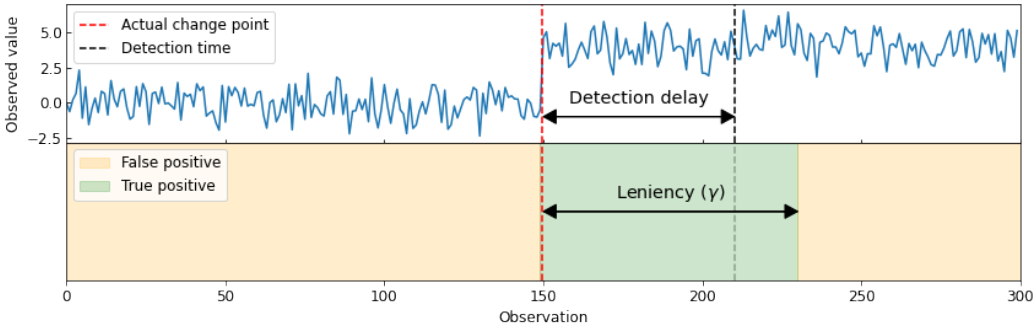


Figure 2.2: **Definition of detection delay and leniency γ. The leniency is used to decide whether a predicted change point is a false positive or a true positive.**

late dissimilarity measures between windows of past observations $y_{t-(w_1+w_2)..t-w_1-1}, y_{t-w_1..t-1}$ using the Kullback-Leibler divergence [67] (KL-divergence) or directly in the CUSUM Equation 2.3 above.

According to Liu et. al. [75] estimation of pdfs may not be accurate when scaling the number of dimensions due to the curse of dimensionality. They propose estimating the direct density ratio $\frac{p_{\theta_1}(y_k)}{p_{\theta_0}(y_k)}$ instead to detect change points, which is simpler to calculate. This method is used to develop an offline change detection algorithm called KLIEP. Kawahara and Sugiyama [63] propose a modification to KLIEP which allows for online change point detection with direct density ratio estimation.

These methods fall into a category of change detection algorithms which monitor and compare the distributions of two windows of data. A reference window with older samples $y_{ref} = y_{t-(w_1+w_2)..t-w_1-1}$ is compared to a more recent test window $y_{test} = y_{t-w_1..t-1}$. Different dissimilarity measures may be used to compare these windows. Statistical measures include the Wilcoxin distance and Kolmogorov-Smirnov distance for univariate data, the KL-divergence mentioned above or geometric measures such as the Manhattan or Euclidean distance [101]. The higher the dissimilarity, the lower the probability both are from the same distribution.

Comparing two windows of data presents additional problems which are related to the size of the windows. The larger the test and reference window sizes, w_1, w_2 respectively, the more accurate the decision. However, large window sizes require more initialization and therefore may not be suited to analyze changes which occur soon after each other. Smaller windows, on the other hand are more sensitive to outliers. To remedy this problem, methods have been developed to adaptively resize windows [12].

2.3 Evaluation of change detection methods

In this section, we will define different metrics used to analyze the performance of change detection methods and explain how the algorithms can be evaluated.

2.3.1 Metrics

Confusion matrix and derivations

Given a univariate time series $y_{0..T}$ we can evaluate the performance of an online change detection method by sequentially analyzing observations of the time series, $y_1, y_2, ..., y_T$. Whenever the change detection algorithm signals that a change occurs, this time $\tau = k$ is recorded and added to a set of predictions, $\hat{T} = \{\hat{\tau}_1, \hat{\tau}_2, ..., \hat{\tau}_M\}$.

This set of predictions can then be compared to a ground truth, T_{gt}. It is unlikely that a change point detection method will signal a change point at the exact moment a change occurred. We define a detection delay as the difference between the moment the change occurred and the moment it was detected:

$$DD = \tau - \hat{\tau} \tag{2.7}$$

A leniency parameter, γ, is used to limit the detection delay to a maximum after which a signal is considered a false positive. See Figure 2.2 for an illustration of the detection delay and the leniency parameter. Signals which fall within the range $(\tau, \tau + \gamma)$ are considered true positives.

The two sets, $\hat{T}, T_{gt}$ can be compared to calculate the number of true positives, false positives and false negatives. False negatives are change points which have been missed by the detection algorithm. True negatives can also be calculated, although this metric is less meaningful as there are relatively few change points compared to the total length of a time series.

The following metrics, derived from the set of true positives TP $\subseteq \hat{T}$, the set of false positives FP $\subseteq \hat{T}$ and the set of false negatives FN $\subseteq T_{gt}$ are frequently used to evaluate the performance of a change detection method:

$$\begin{aligned} \text{True positive rate: } TPR &= \frac{|\text{TP}|}{|\hat{T}|} \\ \text{False positive rate: } FPR &= \frac{|\text{FP}|}{|\hat{T}|} \\ \text{F1-score} &= \frac{2|\text{TP}|}{2|\text{TP}| + |\text{FP}| + |\text{FN}|} \\ \text{Expected Detection Delay: } EDD &= \frac{\sum_{i \in \text{TP}} DD_i}{|\text{TP}|} \end{aligned} \tag{2.8}$$

ARL

A change detection method often has a number of tunable parameters which can be used to modify the sensitivity of the algorithm. For example, the CUSUM has an expected magnitude of change b and a signaling threshold h. Deciding which parameter values to use can be difficult, as the impact on the sensitivity (true positive rate) or false positive rate can be significant. To help with the tuning of these parameters, Page [87] defines the average run length (ARL), also called the ARL_0. The ARL is defined as the average number of observations between two false alarms while the process under observation is stationary and is highly correlated with the false positive rate. By calculating the ARL for different combinations of tunable parameters we can reject combinations which will lead to an unacceptable false positive rate.

The ARL can be calculated analytically for some algorithms, for more complex algorithms the ARL is usually estimated by simulating an arbitrary length time series, $y_{0..T}$ where each y_k is a realization of a random variable with probability density function f. The most common probability density function is the standard normal distribution $\mathcal{N}(0, 1)$. This simulation is continued until the desired number of change points has been obtained.

2.3.2 Synthetic datasets

A common procedure for evaluating a change detection method is by generating a time series $y_{0..T-1}$ which is made up of two parts: the pre-changepoint portion $A = y_{0..\tau-1}$ and the post-changepoint portion $B = y_{\tau..T-1}$. A is used to estimate model parameters for parameterized methods or build up a window of past observations for non-parameterized methods. Each $y_a \in A$ and $y_b \in B$ are realizations of random variables with pdfs f_A, f_B, i.e. $X_A \sim f_A, X_B \sim f_B$ where $f_A \neq f_B$. See Table 2.1 for typical choices for f_A, f_B.

Multivariate

Multivariate change detection methods can be evaluated in two ways: by simulating dense changes or sparse changes. Dense changes are changes which affect a majority of the coordinates, sparse changes affect fewer coordinates.

Description	f_A	f_B
Mean shift	$\mathcal{N}(0,1)$	$\mathcal{N}(\theta,1)$
Variance change	$\mathcal{N}(0,1)$	$\mathcal{N}(0,\sigma^2)$
Poisson rate change	Poiss(λ_1)	Poiss(λ_2)

Table 2.1: **Examples of probability density functions for univariate synthetic datasets.**

Simulating dense changes can be as simple as replacing a univariate pdf with a multivariate pdf: $\mathcal{N}(\theta,\sigma^2)$ becomes $\mathcal{N}(\boldsymbol{\theta},\Sigma)$ with a p dimensional vector describing the magnitude of the mean shift $\boldsymbol{\theta}$ and post-change covariance matrix Σ.

Different approaches are used in the literature for simulating sparse changes, some of which are highlighted below.

Chen et al. [24] simulate sparse mean changes in a p-dimensional Gaussian time series. Their numerical study is limited to a single change point and requires the in-control parameters of the distribution to be known. Different magnitudes of mean changes are simulated, $\vartheta \in \{2, 1, 0.5, 0.25\}$. Their mean change vector is defined as $\boldsymbol{\theta} = \vartheta \frac{Z}{||Z||_2}$, with components of $Z = (Z^1, ..., Z^p)$. s components of Z are sampled from a standard normal distribution, the other $p - s$ components are equal to 0. By modifying s the impact of different levels of sparsity can be investigated.

Xie and Sigmund [110] simulate sparse mean changes in a p-dimensional Gaussian time series. Their numerical study is limited to a single change point and requires the in-control parameters of the distribution to be known. The components of the mean shift vector $\boldsymbol{\theta} = (\theta^1, ..., \theta^p)$ are defined as

$$\theta^i = \begin{cases} \mu_n, & \text{if } i \in S \\ 0 & \text{if } i \notin S \end{cases} \tag{2.9}$$

where S is a random selection of s coordinates, for some sparsity level s.

Kuncheva [69] use 30 datasets from the UCI machine learning repository [34]. The datasets have a dimensionality in the range $n = (4, 60)$. Two sets of samples W_1, W_2 are selected without replacement from a dataset. Change is simulated by selecting two features randomly and swapping these features in W_2. Faithfull et al. [38] also use datasets from the UCI repository which have class labels. A random subset of classes is selected. Samples with these class labels are selected without replacement to populate W_1. The remaining samples, belonging to the other subset of classes are used to populate W_2.

Multiple change points

The synthetic datasets above are all limited to simulating a single change point. In practice, change detection methods are used to continuously monitor a time series which could contain multiple change points.

Bodenham [13] presents a relevant evaluation method for a continuous monitoring scheme. Change points are spaced randomly by sampling from a Poisson distribution with parameter λ, $\xi_1, ..., \xi_n \sim Pois(\lambda)$. These values are then padded with additional variables G and D, where G is a parameter to control the grace period between change points, D is a period which allows an algorithm to detect a change. The time or index at which a change occurs is defined using the following equation

$$\begin{aligned} \tau_1 &= G + \xi_1 \\ \tau_k &= \tau_{k-1} + D + G + \xi_k \end{aligned} \tag{2.10}$$

Changes are shifts in the mean with magnitudes defined by $\mu_k = \mu_{k-1} + \delta_k$, with δ_k a random selection from the set $S = \{\pm 0.25, \pm 0.5, \pm 1, \pm 3, \}$. The time series is defined as $\mathcal{N}(\mu, 1)$ with μ changing only at change points $\tau_1, ..., \tau_M$.

Multivariate extensions include replacing μ with a vector of mean changes $\boldsymbol{\mu}$ defined in Equation 2.11. The covariance matrix is either the identity matrix or a randomly generated, positive semi-definite covariance matrix where the upper triangle was generated from

$\mathcal{N}(0.5, 0.2^2)$.

$$\boldsymbol{\mu} = \begin{pmatrix} \mu \\ 0 \\ 0 \\ 0 \end{pmatrix} \qquad \Sigma_1 = \mathbb{I} \tag{2.11}$$

2.4 Research Gap

In this Chapter, so far we have presented the necessary background information related to continuous change detection in time series and explained how to evaluate change detection methods. In this section, we will identify a research gap in online change detection and we will explain how we will address this research gap.

Change detection methods can be categorized according to three important characteristics: whether the method runs offline or on a stream of observations, the dimensionality of the observations, and whether the method makes assumptions about the distribution of the data. Each of these characteristics can again be divided into two or more subcategories: offline/online methods, univariate/multivariate methods, and parametric/non-parametric methods.

Recent research by Chen et al. [24] indicates that there is a scarcity of prior literature of the online, multivariate type. The authors claim that the literature that does exist in this category mainly considers the problem where the sign and magnitude of the changes are known a priori.

Another area with limited prior research is change detection using ensembles of distinct univariate change detection algorithms. Woźniak et al. [109] present a relevant literature survey about ensemble methods, also known as Multiple Classifier Systems (MCS). The authors highlight that MCS have been used for many different applications, for example, land cover monitoring and anomaly detection in time series. The outputs of large sets of classifiers are combined using techniques such as majority voting schemes or Naïve Bayes. Although there is ample research about the general use of ensemble methods for classification, applying this method to univariate detection of change points in time series in a continuous monitoring setting seems to be an open field of research.

Existing Ensemble methods

Kuncheva [68] introduces the general concept of ensemble methods for change detection in univariate data. The author demonstrates that by using a process quality control (control chart) algorithm with different sensitivity settings in combination with a simple voting scheme, more changes can be detected than by using a single control chart algorithm. Ensembles with distinct control chart algorithms are suggested as an area for future research.

More recently, a method has been proposed to combine multiple algorithms sequentially [18] for univariate change detection in land change monitoring data. Two change detection methods are used sequentially followed by a validation test to reduce false positives. The first change detection method measures the deviation of the data from a pre-defined model and makes assumptions about the seasonality of the data. The second method involves monitoring the mean deviation from model using the CUSUM to detect any changes which have been missed by the first method.

Faithfull et al. [38] demonstrates that ensembles can also be applied to multivariate data. The authors apply univariate change detection methods, such as ADWIN, EWMA and CUSUM, to each coordinate of a high-dimension time series and the results are combined to outperform more classical multivariate change detection techniques such as Hotelling's T^2 test [54] and SPLL/LL-KL [69]. The ensembles evaluated by the authors are comprised of identical univariate change detection methods, as opposed to a heterogeneous mixture of algorithms. The decision function used is a simple threshold: the decision function will only signal a change at a given time if a fraction $> p$ detectors is currently signaling a change.

In conclusion, despite ample research on ensemble methods in other applications, there appears to be comparatively little research on using ensemble methods for change detection

in a continuous monitoring setting. The related works which do exist are either model-based or make use of homogeneous ensembles with identical algorithms. We have been unable to find prior research which uses a diverse ensemble of change detection algorithms in a continuous monitoring setting.

This thesis will address the identified research gap by creating a change detection method of the online, univariate type that combines multiple distinct univariate process quality control (control chart) algorithms in an ensemble. This new method will be model-free and will make few assumptions regarding the types of changes which occur in the dataset.

Chapter 3

Contributions

This Chapter first proposes several approaches to improve the state of the art in ensemble change detection methods, followed by the proposal of a design that incorporates these improvements in a new ensemble method. The new ensemble method is the main contribution of this thesis to the state of the art.

3.1 Improvements

In this section, we present how related works can be combined and improved to create a novel change detection method.

We hypothesize that the ensemble methods by Faithfull et al. [38] can be improved in several ways. Firstly, by increasing the ensemble diversity and secondly by improving the decision fusion method.

Ensemble diversity

Brzezinski [17] defines ensemble diversity as the degree by which component classifiers make different decisions for a single case. If, for example, the ensemble is solely comprised of univariate change detectors to detect mean shifts in Gaussian data streams, the ensemble may be unable to detect changes in variance or may perform poorly when the underlying process is not Gaussian in nature. By incorporating different types of change detectors we can improve the overall robustness of the ensemble. As we will see in Chapter 4, over 50% of the time series in the dataset provided by Philips are non-Gaussian in nature, which makes ensemble diversity especially important.

Decision fusion

The decision fusion method proposed by Faithfull et al. [38] is a majority vote with equal weights. This decision fusion method could be improved by considering a window of past signals from the individual change detection algorithms. Consider a hypothetical situation where a multivariate data stream undergoes a mean shift defined as the vector $\boldsymbol{\theta}$, where each element of the vector has a different magnitude, $|\theta_i| \neq |\theta_j|$. The detection delay for algorithms such as CUSUM depends on the magnitude of the change and the threshold value. This mean shift may therefore not be detected by an ensemble of univariate CUSUM detectors with identical threshold values as the detectors will signal at different timestamps.

To fix this, we could use majority voting in an interval as follows. We define W^i, the window of the past N predictions of a change detection for the i^{th} coordinate of a data stream. A prediction is encoded as a binary number, 1 if a change has occurred and 0 otherwise. We apply the indicator function (Equation 3.1) to each coordinate, which tests for the presence of value x in set S.

$$\mathbf{I}_S(x) = \begin{cases} 1 & \text{if } x \in S \\ 0 & \text{if } otherwise \end{cases} \tag{3.1}$$

The number of coordinates of the data stream which have changed is defined as

$$changed = \sum_i \mathbf{I}_{W^i}(1) \tag{3.2}$$

By considering a sufficiently large window size of past predictions, a change affecting at least p signals can be detected by checking if $changed >= p$.

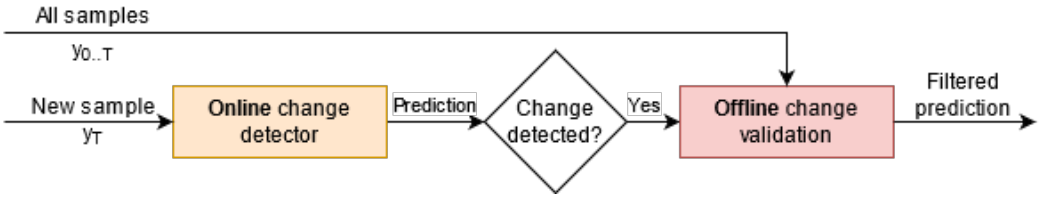


Figure 3.1: **Hierarchical change detection, a method proposed by Alippi et al. [3]. In this example, the offline change detection layer uses the entire history of samples to to filter false positives from the online change detection layer's output.**

The decision fusion method could also be improved by using the hierarchical approach suggested by Alippi et al. [3]. Figure 3.1 shows an example of this architecture. An offline change validation layer, which uses more complex and time-consuming offline hypothesis tests than online algorithms, considers the entire history of samples to validate changes and remove false positives from the output of an online algorithm.

Another option is to replace the threshold decision function altogether with a more sophisticated algorithm. Weighted majority voting could be used to reduce false positives caused by over-eager detectors and provide detectors that perform well with more decision power. If there is sufficient labeled training data, machine-learning-based binary classifiers may be used instead of weighted majority voting.

Incorporating statistics

Another method to improve the performance of the ensemble is to incorporate statistics from the algorithms in the ensemble. In addition to a prediction of whether a change has occurred, some algorithms may also produce statistics such as S_k in CUSUM Equation 2.4. These statistics contain additional information about the state of the algorithm, which is lost if only the final prediction is considered. Statistics that are compared against threshold values to determine if a change has occurred could be used in an ensemble to signal changes before the threshold has been reached.

3.2 Univariate vs. Multivariate detection

As we will highlight in Chapter 4, the dataset contains changes that affect multiple performance metrics simultaneously. Creating or using a multivariate change detection method would therefore seem the obvious choice compared to using a univariate method for individual performance metrics. As mentioned in Chapter 2, multivariate methods have more information available and may be able to detect changes more quickly. In practice, initial experiments with multivariate methods indicated that these methods perform poorly on the highly-dimensional dataset. Selecting subsets of metrics that frequently change simultaneously has also proved to be difficult due to the required domain knowledge and the fact that the sparsity of changes in the subsets also changes over time.

An important requirement is that the system is able to detect changes in individual performance metrics. A multivariate method with a high-dimensionality is unlikely to detect a change in a single coordinate. Therefore, to increase the sensitivity of the detection system we have decided to only make use of univariate change detection methods.

3.3 Ensemble algorithms

Woźniak et al. [109] mention that ensembles should "aim to include mutually complementary individual classifiers which are characterized by high diversity and accuracy". In this section, we will explain which algorithms were included in the ensemble and how they contribute to the overall diversity.

There is no clear specification of the types of changes that must be detected. Likewise, the distribution of data is either unknown or may change over time. Therefore, it is important to create an ensemble that is as diverse as possible. To maximize the diversity of the ensemble, we have selected both parametric and non-parametric methods, methods that can monitor process location, and others for monitoring process dispersion.

In addition to the characteristics mentioned in Section 2.4, we define an additional important characteristic: "Phase 1 estimation required". Control charting is defined in terms of distinct phases, Phase 1 and Phase 2. Phase 1 is used to define the in-control parameters of a process, for example by estimating the population standard deviation or mean during a period without changes. Phase 2 is when the actual monitoring occurs, using parameters which have been estimated during Phase 1 [104].

Many parametric control chart algorithms require estimations of the process mean and variance to work. Although we can estimate these parameters from the data, an assumption needs to be made that no change has occurred during this initialization period. Similarly, some non-parametric methods may require a buffer of reference samples estimated during Phase 1.

Algorithms that require Phase 1 initialization pose a problem for unsupervised change detection in a continuous setting, as the assumption that no changes occur during the initialization period may not always hold and manual validating this defeats the purpose of an unsupervised change detection algorithm. This initialization period could be compensated for by combining algorithms that require Phase 1 estimates with algorithms that do not in an ensemble. For this reason, both types are included in Table 3.1.

Algorithm	Parameteric	Multivariate	Phase 1 estimation required
CUSUM [87]	**x**		x
EWMA [56]	x		x
Robust AEWMA [20, 84]	x		x
AFF [13]	x		
Dispersion CUSUM [85]	x		x
ADWIN [12]			
KSWIN [92]			
ScanB [72]			x
NEWMA [65]			
KernelCPD [48]		x	
EMW [45]		x	x
DMW [45]		x	x

Table 3.1: **A categorization of algorithms used in the univariate ensemble.**

3.3.1 Algorithms

In this section, we briefly explain how the chosen algorithms work and why they are included in the ensemble. Some algorithms, such as CUSUM and EWMA are not included in this Section as they have already been mentioned in Sections 2.2.2 and 2.2.3 respectively.

RobustAEWMA

The Adaptive EWMA control scheme [20] was proposed to overcome problems with the normal EWMA control chart which cannot perform well for both large and small changes

simultaneously. By adapting the weight of new observations, this problem can be overcome. The Robust Adaptive EWMA [84] is a modified version of the Adaptive EWMA scheme that uses robust estimators of the process mean and variance and therefore performs better when the assumption of normality is violated.

AFF

The Adaptive Forgetting Factor [13] is a continuous monitoring scheme for changes in location (central value of a signal) that does not require Phase 1 estimation, which makes it a potentially useful algorithm in the ensemble. However, initial tests using synthetic data indicate that the algorithm is slow to detect changes in contaminated Gaussian distributions with anomalies.

Dispersion CUSUM

Nazir et al. [85] propose multiple variants robust CUSUM control charts for monitoring the process dispersion parameter. Most other methods in the ensemble are only able to detect changes in the location. By including this method we strive to make the ensemble more sensitive to changes in variance.

ADWIN

Adaptive Windowing [12] uses dissimilarity measures between adaptively resizable windows of past observations to detect changes in both location and scale. This method has already been shown to work in change detection ensembles by Faithfull et al. [38] in a multivariate setting. We use an existing implementation of this method [82].

KSWIN

Kolmogorov-Smirnov Windows [92] is a method used in concept drift detection that uses the Kolmogorov-Smirnov dissimilarity measure between a window of recent samples and a random sample of older samples. This method makes no assumptions about the distribution of the data and can be used to detect changes in both process location and dispersion. We use an existing implementation of this method [82].

ScanB

ScanB [72] is a non-parametric, kernel-based method suited for both offline and online change detection. The authors demonstrate that this method is robust to different types of distributions and has a low detection delay, which is why we include this method in the ensemble.

NEWMA

"No-prior-knowledge" EWMA [65] is a non-parametric, memory-efficient change detection algorithm that works by computing the difference between multiple EWMA statistics without having to store any samples in memory. The authors demonstrate that this method is suitable for continuous monitoring purposes, which makes it a good addition to the ensemble.

EMW & DMW

Energy Multivariate Window & Depth Multivariate Window are two multivariate methods that compare reference windows to test windows, similar to KSWIN and ADWIN, using either the energy statistic [97] or the Mahalanobis depth [74]. These methods were included as they can be used to extend the ensemble to a multivariate input.

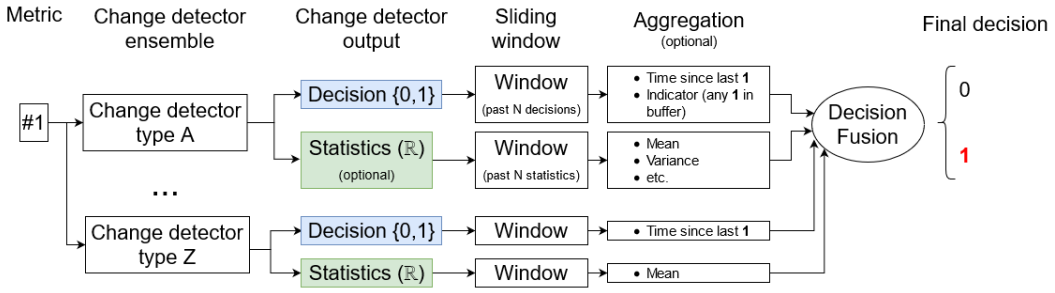


Figure 3.2: **Design of an ensemble change detection method with multiple different types of change detectors and a window of past signals.**

KernelCPD

This is an offline detection method, proposed by [48], which we adapted for use in an online setting by maintaining a FIFO buffer of samples and returning only the latest detected change in the buffer. According to [24], offline algorithms are unlikely to work when used in an online setting partly due to restrictive computational costs. However, we use a highly optimized implementation [102] that is faster than most evaluated implementations of online methods. We include KernelCPD as it can be used for multivariate data and because the online implementation performs very well on synthetic data, see Chapter 6.

3.4 Contributions

This section presents a novel design for an online univariate change detection method. This method will be evaluated in Chapter 5. Figure 3.2 illustrates the design of the ensemble with a single change detection algorithm. It is consists of the following steps:

1. A sample arrives from a time stream and is analyzed by each of the algorithms in the ensemble in parallel. This results in a decision: a change has occurred, **1** or no change has occurred, **0**. Some algorithms may also produce some statistic such as S_k in CUSUM Equation 2.4.

2. The decisions and statistics, if applicable, are appended to a fixed-length FIFO buffer.

3. The values in the buffers are either passed directly to the decision fusion algorithm or aggregated and then passed to the decision fusion algorithm. One or more aggregations may be applied to each sliding window in parallel. For example, the mean, variance, and the 75^{th} quantile (Q75) may all be passed to the decision fusion algorithm.

4. The decision fusion algorithm uses the aggregated or unprocessed values from each of the algorithms to come to a final decision.

These steps can be extended to an arbitrarily sized ensemble of change detection algorithms. Different combinations of aggregations and decision fusion algorithms result in different performance characteristics. To demonstrate this, three different configurations are elaborated on below and evaluated in Chapter 6.

3.5 Implementation details

We will proceed with more specific details about how the implementation of the ensemble, including an explanation of the three different configurations used in the ensemble.

The algorithms that require Phase 1 initialization were modified for use in a continuous monitoring setting by including a reset procedure. This procedure, shown in Figure 3.3, is an adaptation of the hierarchical method proposed by Alippi et al. [3]. Once a univariate algorithm detects a change, an offline change detection algorithm is used to estimate the

time when the change occurred in a sliding window of samples from the data stream. The post-change samples in the sliding window can then be used to quickly restart an algorithm, instead of having to wait until sufficient samples have been accumulated to estimate parameters such as mean and variance of a data stream.

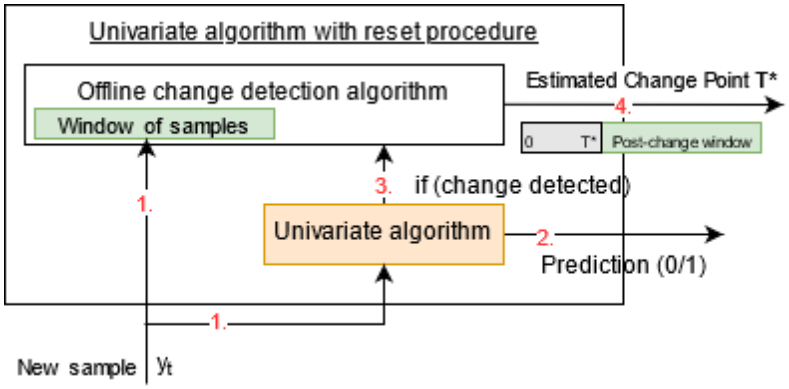


Figure 3.3: **Reset procedure used for faster start-up of univariate algorithms.** ***1.*** **A sample arrives from the time series and is appended to the end of a sliding window of samples and evaluated by the univariate algorithm. The new sample is used to update internal statistics required for its operation, e.g. estimations of the mean or variance.** ***2.*** **The algorithm predicts whether a change has occurred** ***3.*** **If the algorithm has detected a change, an offline change detection algorithm is used to extract the time the change occurred in the window of samples.** ***4.*** **The window is split into two parts: pre-change and post-change. The pre-change samples in the window are discarded. The post-change samples can be used to re-initialize the statistics used by the algorithms.**

For a consistent comparison between decision fusion and aggregation methods, the composition of the ensemble was kept the same throughout training and testing.

3.5.1 Configuration 1: Weighted majority voting

Weighted majority voting was implemented by considering only the decisions of the change detection algorithms and ignoring the output statistics. The window of past $\mathbf{N}$ decisions is first aggregated using the indicator function in Equation 3.1.

This results in a M-dimensional binary vector, denoted as $\mathbf{D}$. M is the total number of univariate algorithms used in the ensemble. Each univariate algorithm will be assigned a weight, w_i, which can be expressed as weight vector $\mathbf{w}$.

A change has occurred if the weighted sum of components of $\mathbf{D}$ are greater than some predetermined threshold p:

$$\mathbf{w}^T \cdot \mathbf{D} > p \tag{3.3}$$

Decision refinement

Due to the indicator function, which considers the past N decisions of each algorithm, a change that is detected by all algorithms should result in a sequence of positive decisions. A single positive decision may be an anomaly and is therefore filtered. This is achieved by applying a refinement to the output of the decision fusion algorithm. See Figure 3.4 for a visual representation of this refinement. Each decision is appended to a sliding window of decisions with size $D_{sliding}$. This sliding window is converted to a binary variable using the following equation, which calculates the sum of binary values in the sliding window and returns 1 if the sum is above some threshold value D_{min}.

$$f(\mathbf{S}_{win}) = \begin{cases} 1 & \text{if } \sum S_{win} \geq D_{min} \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

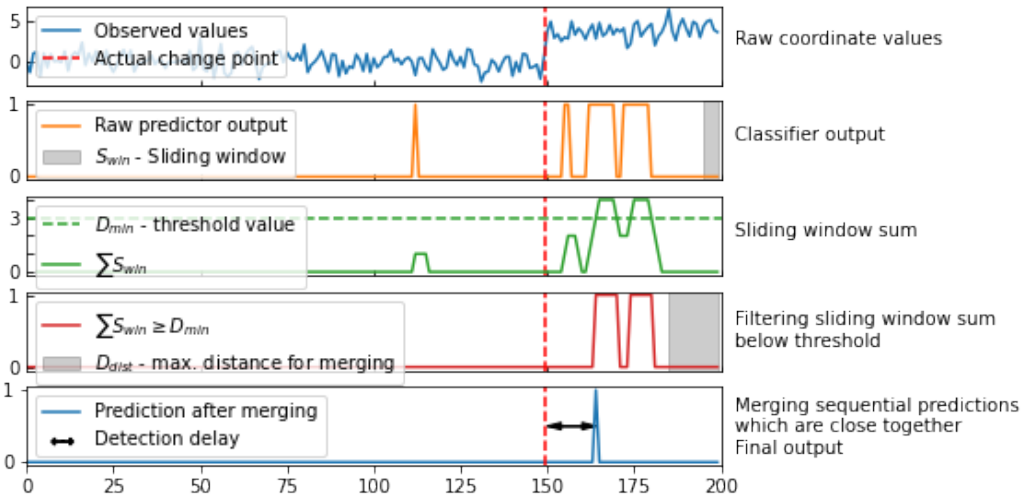


Figure 3.4: **Refinement of sequential classifier outputs using sliding windows. Sporadic positive outputs are filtered and sequential predictions closer than D_{dist} are merged to obtain the final output.**

To ensure that a user only receives a single notification that a change has occurred, any resulting signals that are closer than D_{dist} samples apart are considered to be caused by the same change and therefore merged. Only the (filtered) signal will remain.

Parameter optimization

The weighted majority voting and subsequent decision refinement procedure result in $M+5$ different parameters that need to be determined (see Table 3.2). The number of feasible parameter combinations is very large and different parameter combinations may result in very different performance characteristics. The search for an optimal set of parameters was approached as a multi-objective optimization problem. Given a set of training data with labeled change points, different parameter combinations are applied to the dataset and the subsequent FPR and TPR are calculated using Equation 2.8 with a fixed leniency (maximum detection delay) of 25 samples. NSGA-II [33], a multi-objective evolutionary algorithm, was used to find different combinations of parameters that maximize the TPR while minimizing the FPR. This results in a Pareto front of solutions, see Figure 3.5. For ease of comparison with other decision fusion algorithms, a single solution was chosen from the Pareto front that maximized the F1-score on the given training dataset.

Parameter	Description
N	Size of first sliding decision window
w	Weights, one for each algorithm
p	Majority vote threshold
$D_{sliding}$	Size of second sliding decision window
D_{min}	Minimum number of **1**s in second sliding window, i.e. threshold for final signal
D_{dist}	Minimum distance between subsequent changes

Table 3.2: **An overview of parameters required for decision fusion using weighted majority voting**

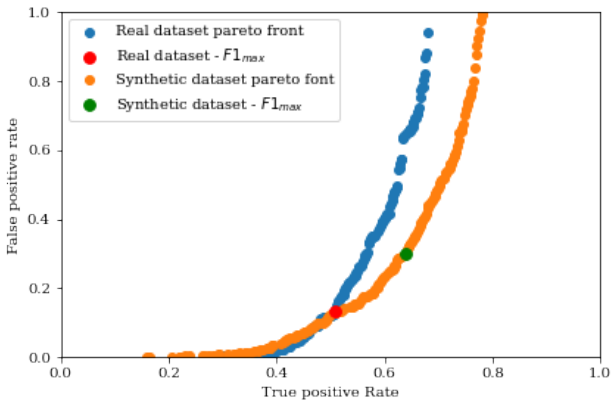


Figure 3.5: **Pareto front of parameters found using NSGA-II for the weighted majority voting decision fusion algorithm run on the synthetic dataset and the real dataset. The $F1_{max}$ for the synthetic dataset lies on the intersection between the two Pareto fronts, which is by coincidence.**

3.5.2 Configuration 2: Random Forest with aggregated features

The disadvantage of weighted majority voting is that it only considers the final decisions of each of the algorithms and discards the statistics used to make this decision. These statistics may contain more information about the state of each algorithm. Consider an algorithm for which a threshold has been set too high - the statistic may increase to some lower value but will never reach the threshold value. A classifier provided with informative features, which aggregate information from a sliding window of statistics, may be able to detect a change despite the threshold never being reached.

This hypothesis is evaluated by transforming the window of decisions and statistics into multiple features and using a Random Forest classifier to predict whether a change had occurred. Random Forest is an ensemble machine learning method that adds additional randomness to ensembles of decision trees created using bagging [14, 73]. An advantage of Random Forests is that they are relatively insensitive to outliers in the training data sets [2]. The dataset used in this thesis is labeled using an offline change detection method (see Chapter 4) which may incorrectly label some changes, therefore outliers can occur in the training dataset. Other advantages include their robustness against noise and reduced overfitting [1]. The decision trees used in the ensemble tend to select only informative features at each split and therefore avoid using noisy features. These properties allow us to use many different aggregations of statistics without performing feature selection, as the random forest will partly do this for us.

For each of the algorithms, the following features were extracted for the windows of decisions and statistics:

- **Decisions: Time since the latest signal** - The detection delay for some algorithms is higher than others. A classifier may be able to use this information to remove false positives caused by late signaling.
- **Statistics: Variance estimators** - The variance of the sliding window of statistics was calculated using the median absolute deviation (MAD), interquartile range (IQR), and the standard deviation. An increase in variance could indicate that an algorithm's statistic is increasing or decreasing.
- **Statistics: Location estimation** - The mean, maximum, median, and 75^{th} quantiles are used to estimate the location of the statistic relative to the threshold. This can be used for the early signaling of changes.
- **Statistics: Trend** - The mean of the first half of the window of statistics is subtracted from the mean of the second half to provide information about the change of the statistic within the window.

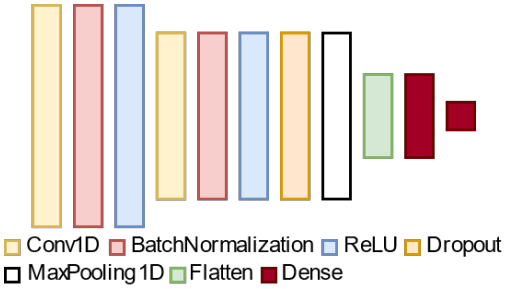


Figure 3.6: **Fully Convolutional Network architecture. Hyperparameters were chosen using cross-validation and are listed in Appendix Table B.1.**

The features for each of the algorithms are combined and used by the classifier to predict whether a change has occurred. Finally, these decisions are refined using the same method as described in Section 3.5.1. The parameters used in the refinement were selected using cross-validation on a validation training set.

Hyperparameter optimization

A frequently used approach to improve the performance of machine learning models is through the tuning of model parameters. Random Forests are typically assumed to be relatively insensitive to changes compared to default model parameter settings [96], although some studies [55] suggest that well-tuned models can outperform models with default settings. Random Forests have several different hyperparameters which can be tuned such as the maximum depth of each decision tree, the number of features randomly selected at each split, and the total number of trees. A grid search was used to explore the performance of the model given different combinations of hyperparameters. The model with the best performance was selected using cross-validation and will be used for future evaluation.

3.5.3 Configuration 3: Fully Convolutional Network with non-aggregated features

The final decision fusion algorithm that was evaluated is a fully convolutional neural network (FCN). Fully convolutional networks are examples of deep neural networks which have been successfully applied to different time series classification problems [107]. An advantage of FCNs is that they require very little data preprocessing compared to other network architectures [61]. Instead of aggregating features, the FCN can therefore accept a sliding window of feature values as input.

An FCN is made up of convolutional blocks which include a convolutional layer, followed by a batch normalization layer and an activation function. The first layer in the block extracts features by convolving a convolution kernel with the input vector. The weights of the filters define the types of features that can be extracted. For example, a filter with equal weights can be used to perform smoothing of an input vector, while others may be used to detect spikes or abrupt changes. By combining different filters an FCN is, therefore, able to extract time-dependent features.

The network architecture which is used is visualized in Figure 3.6 and is adapted from [107]. This original architecture is designed for univariate time series. Since there are multiple features, the FCN is extended to accept multiple input series, one for each feature. The convolutional layers act as feature extractors for the fully connected layer that follows.

Parameter optimization

Hyperparameters such as the number of convolutional blocks, the number of filters in each convolutional layer, and the number of neurons in the fully connected layer were found using hyperparameter optimization [71] performed on a validation training set. In addition, feature selection was performed by using only the most important features in the Random

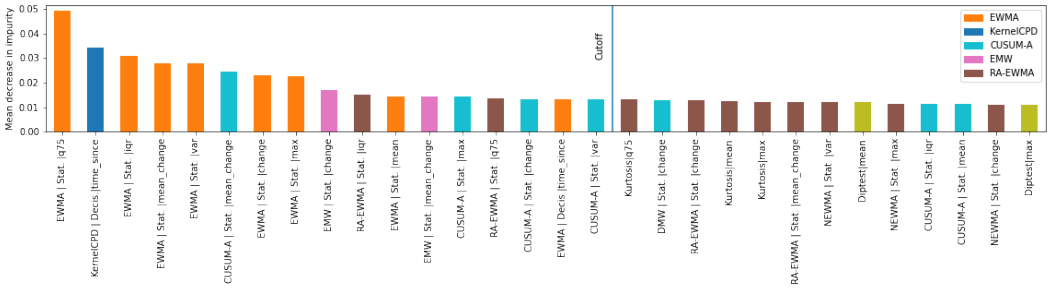


Figure 3.7: **Relative feature importance for a Random Forest classifier trained on the real training dataset using all aggregated statistics and decision windows, sorted by decreasing feature importance. The labels on the x-axis are formatted as follows: [Name of algorithm] ‖ [Statistic OR Decision aggregation] ‖ [Type of aggregation].**

Forest classifier trained in Section 3.5.2. The feature importance was measured using the mean accumulation of impurity decrease in the forest's trees for a forest trained on the real training dataset. From Figure 3.7 it is clear that statistics from EWMA, CUSUM-A, EMW and RA-EWMA are important in addition to decisions from KernelCPD. These features are therefore selected as features for the FCN.

Chapter 4

Dataset

This Chapter describes the dataset provided by Philips and any synthetic datasets used to evaluate change point detection algorithms.

4.1 Data analysis

In this Section, we we will give a short description of the dataset which was provided by Philips.

A performance benchmark runs both periodically and whenever new code is pushed to the main branch of the version control system. On average, this occurs approximately 13 times per day. The same performance benchmark can run on multiple test devices in parallel. To simplify the problem and amount of data that needed to be analyzed, we select a single test bench for analysis.

In this dataset, a single run of a performance benchmark produces 909 different metrics. These metrics provide information about the state of the system during a performance benchmark. A few example metrics are:

1. **Number of images dropped during performance benchmark** - images are received via a network protocol and buffered before processing. Dropped images may indicate the buffer is filling up and that images are being processed too slowly to handle the incoming data stream. In a medical device, this may result in injury to the patient.
2. **Mean image processing time** - the time required to process a raw image, excluding time spent in the image queue.
3. **Mean time required to write the processed images to disk** - images must eventually be stored on disk after processing.
4. **Maximum processor utilization during test**

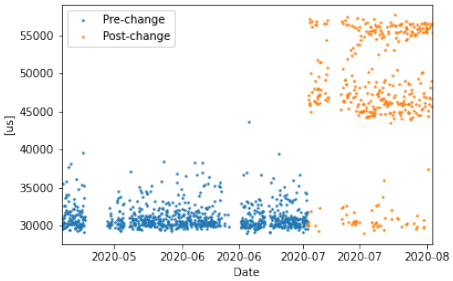
Some of these metrics are discrete, such as counts, and some are continuous, such as mean image processing time.

A history of approximately two years worth of data (or 8,000 samples) is available for most metrics. Some metrics or test cases have been added to the performance benchmark at a later stage, which means data may be incomplete. The history for each metric is considered a time series.

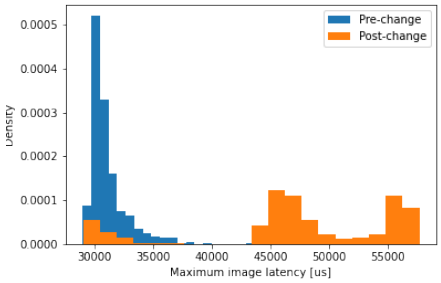
4.2 Assumptions

A number of assumptions are made to make the analysis of the data easier. These assumptions are justified and described in this Section.

Firstly, we assume that the data contains only abrupt changes and no incremental changes. On average, software changes occur less frequently than the sampling rate of the performance



(a) Time series (maximum image latency) from dataset.



(b) Histograms with pre and post change distributions.

Figure 4.1: **Example time series and histograms from dataset illustrating change in distribution from a skewed, unimodal to a multimodal distribution.**

benchmarks. In between software changes, the performance metrics are assumed to have a stationary distribution, which means incremental changes are unlikely to be observed.

Due to the implementation of the continuous integration system, situations may occur where performance benchmarks do not run in chronological order. Benchmarks that are started manually are considered high-priority, and may be executed before other automatically queued benchmarks. Manual benchmarks can contain software changes that occurred after the other benchmarks were queued. This means that the samples in the dataset are not necessarily in chronological order. However, we assume that this occurs so infrequently that these samples can be considered anomalies instead of change points.

4.3 Data distributions

In this Section, we will categorize the types of distributions that are present in the dataset.

A common assumption made in statistical control charting procedures is that the data is normally distributed. In an analysis of 235 different control charting applications, Alwan and Roberts [4] showed that this assumption frequently does not hold, which has impact on the effectiveness of change detection algorithms [91].

We categorize each of the 909 time series by the type of distribution that best fits the data to find out to what extent the assumption of normality hold.

Since the data is known to contain change points, the assumption that each time series is stationary does not hold. As illustrated in Figures 4.1(a), 4.1(b) the underlying distribution may also change over time. Mapping each time series to a single distribution therefore becomes more challenging, as this mapping is also dependent on the time of analysis.

The method used to overcome this problem involved categorizing the time series into different distributions by applying a set of statistical tests to a sliding window of the time series and choosing the distribution which best fits the majority of the sliding windows. In the Sections below, we will describe which distributions we identified and the statistical tests used to categorize the time series.

4.3.1 Constant/near-constant time series

As stated before, a number of the time series have discrete values. Many discrete time series in the dataset frequently assume the same value. For example, the metric that counts the number of images dropped during a test case is almost always 0, as dropping images is an event that rarely occurs in the system.

Let $S \subset \mathbb{Z}$ be the set of elements that appear in a discrete time series $y_{0..n-1}$ of length n, and let $C(x)$ be the number of elements of $y_{0..n-1}$ equal to x. The value that occurs most frequently is denoted as

$$freq(y) = \underset{s \in S}{\operatorname{argmax}}\, C(s). \tag{4.1}$$

A time series is classified as **constant** if

$$C(freq(y)) \geq 0.9 \cdot n \tag{4.2}$$

and **near constant** if

$$0.2 \cdot n \leq C(freq(y)) < 0.9 \cdot n \tag{4.3}$$

Classification	Number of time series	Percentage of total
Constant	109	12%
Near Constant	37	4%
Normal	439	48%
Multimodal	137	15%
Skewed	156	17%
High kurtosis	25	3%
Other	6	< 1%

Table 4.1: **The number of time series assigned to each category after applying statistical tests.**

4.3.2 Multimodality

For unknown reasons, metrics in the performance benchmarks occasionally exhibit multi-modal distributions. Data sampled from multi-modal distributions consist of multiple sub-populations corresponding to each mode. Change points may affect the entire population or a single mode in the distribution. An example of a multi-modal distribution can be seen in the post-change samples in Figure 4.1.

Hartigan & Hartigan's dip test for unimodality [49] is used to determine whether a time series is uni-modal or multi-modal. According to the authors, the test "measures multimodality in a sample by the maximum difference, over all sample points, between the empirical distribution function, and the unimodal distribution function that minimizes that maximum difference". The test is applied to a sliding window $y_{i..i+w-1}$ of samples. If the null hypothesis that the samples are from a unimodal distribution is rejected for a 50% majority of sliding windows, the time series is assumed to be multi-modal.

4.3.3 Normality

Different normality tests exist [41] for testing a population sample. A few examples are D'Agostino and Pearson's Omnibus Test for normality [31], the Kolmogorov-Smirnov Test for Normality [59]. D'Agostino and Pearson's Omnibus Test was chosen due to the robustness to outliers and grouping of values which may occur in discrete time series.

The same sliding window method was applied as before: the test is applied to a sliding window $y_{i..i+w-1}$ of samples. If the null hypothesis that the data is from a normal distribution is rejected for a majority of the time series, the data is assumed not to be normal.

This test detects a deviation from the normal distribution in terms of skewness and kurtosis. Time series which are classified as non-normal are tested for high skewness [32] and kurtosis [6] and classified as such using the same sliding window technique.

4.3.4 Results

Table 4.1 shows the results of applying the statistical tests described above to the dataset. Clearly, the assumption of normality does not always hold in this dataset, highlighting the need for a change detection algorithm which is robust to different types of distributions.

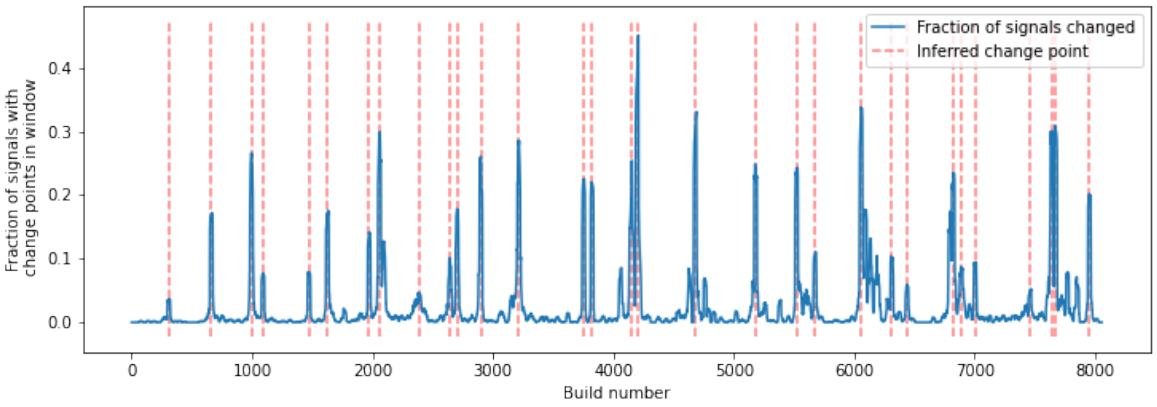


Figure 4.2: **Fraction of time series with change points in sliding window. Change points were detected by applying kernel-based offline change detection to the dataset.**

4.4 Change points

This Section describes how the change points in the dataset are extracted and analyzed.

Unfortunately, as mentioned in Chapter 2 the dataset does not contain any ground truth. This makes evaluating change detection algorithms more difficult, as there is no certain way of knowing whether a detected change is a true positive and when the exact moment is that a change occurred. Many related works in the literature face the same problem, and use simulation studies to generate synthetic datasets to evaluate their algorithms.

Since the change detection system created during our research is designed for this specific application, a generic evaluation using synthetic datasets will not necessarily mean that the system will work for this dataset. An evaluation using the real dataset requires labels in this dataset. Manual labeling is not an option due to the size of the dataset, therefore offline change detection methods were used to detect change points in the dataset.

As mentioned in Section 2.2.1, parametric offline detection methods may make assumptions about the underlying distributions of the sequential datasets. As shown above, these assumptions do not hold across all time series which is why a choice was made to use non-parametric change offline change detection methods to analyze the dataset. The kernel-based detection method proposed by Harchaoui and Cappe [7] was used to detect changes in each individual time series in the dataset.

Under the assumption that major software or hardware changes may affect multiple time series simultaneously, the detected change points were aggregated by counting the number of time series with at least one change point within a sliding window. Although several time series may change simultaneously the change detection method may introduce some noise in the predicted change point location. The sliding window is used to compensate for this noise. Peaks are extracted from the rolling count by analyzing the peaks using a wavelet transformation [35]. The extracted change points in the dataset are equal to the locations of these peaks.

The rolling count and extracted peaks are plotted in Figure 4.2. The number of peaks that are extracted depends on the widths of the wavelet transformation, window size, the type of kernel used in the change detection method and the complexity penalty mentioned in Section 2.2.1. The settings used for this analysis can be found in Appendix B.1 and resulted in 31 distinct change points.

Figure 4.2 shows that a change point may affect up to 40% of the time series in the dataset, which validates the assumption that major software or hardware changes may affect multiple time series simultaneously.

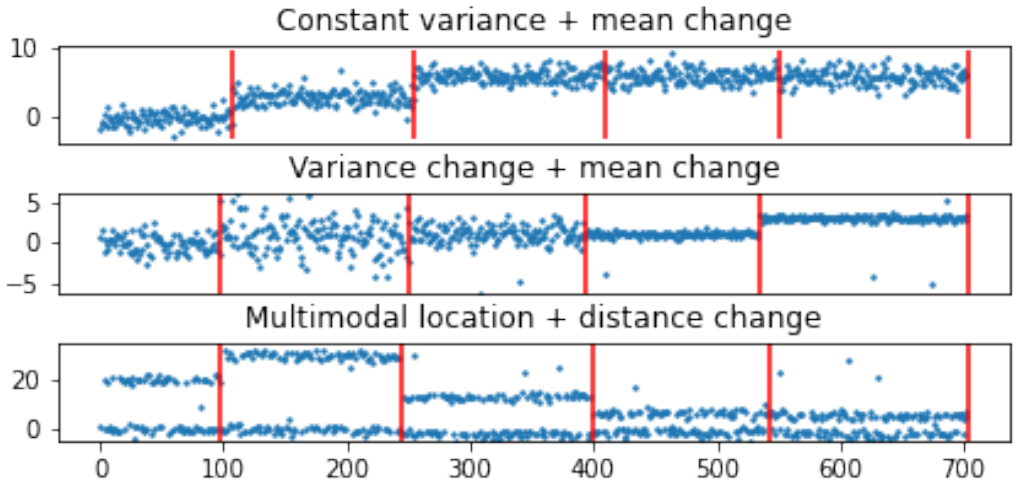


Figure 4.3: **Example synthetic datasets with five change points, which are denoted by the red vertical line.**

4.4.1 Change point analysis

A change point can be caused by a change in mean, variance, distribution, etc. The change points detected using the method above were analyzed to find out if there is a pattern in the types of changes which occur in the dataset. The analysis is limited to two parameters: scale and location. Each of the aggregated change points $\tau_i \in \{\tau_1, ..., \tau_n\}$ detected by the method described above affects one or more time series. The pre-change scale and location of these time series were compared to the post-change parameters.

In the absence of outliers, sample mean and standard deviation would normally be used to estimate location and scale respectively. However, the dataset contains many outliers, which can distort these estimators heavily [29]. Robust estimators are therefore used for both: median to estimate location and median absolute deviation to estimate the scale. To make the MAD consistent with the standard deviation, the MAD must be divided by a scale factor which depends on the type of distribution [94]. For a Gaussian distribution, this scale factor is equal to 1.48.

Table C.1 shows the distribution of absolute changes in location and scale. The absolute change in median, normalized by dividing by the pre-change MAD has a mean value of 1.40 ± 0.21 while the absolute change in scale has a mean value of 1.74 ± 0.88.

4.5 Synthetic data

In this section, we will explain why synthetic data is required to evaluate a change detection algorithm. We will also present the synthetic datasets that were used in the remainder of this work.

The labels inferred from the dataset using offline detection methods are not necessarily correct, and depend on different parameters such as the sensitivity of the offline detection method and the distribution of the data which is being analyzed. As mentioned in Section 2.3.2 a common method to approach this problem involves the use of synthetic datasets.

Ideally, an algorithm which performs well on a synthetic dataset would also perform well on a dataset with real data. To make this possible the synthetic dataset should be similar to the real dataset in terms of the distribution of data, the types of changes that occur, and the amount of samples between the changes.

Three different univariate synthetic datasets have been created based on the method described in Section 2.3.2. These datasets will be described below, and are visualized in Figure 4.3. The settings used to generate the datasets can be found in Appendix Section B.4.

Contaminated Gaussian with constant variance and changes in location

Samples are drawn from a contaminated Gaussian distribution. This is a two-component mixture of Gaussians with equal mean but different variances. A sample is drawn from either the first component, which has unit variance, with probability p or the second component, with much larger variance, with probability $1 - p$. This is used to simulate outliers in the data. Change detection algorithms should be robust to outliers, as a single outlier does not necessarily mean that a change has occurred.

$$X(k) \sim \begin{cases} \mathcal{N}(\mu_k, 1) & \text{if } U(0F, 1) < p \\ \mathcal{N}(\mu_k, 20) & \text{otherwise} \end{cases} \tag{4.4}$$

The times of the changes are defined using Equation 2.10. At each change point, the mean changes with some value which is randomly sampled from a set of values S.

Contaminated Gaussian with changes in both location and scale

Unlike the previous synthetic dataset, which keeps the variance constant, we now simulate a dataset with both changes in scale and location.

Each change can be expressed using the two equations below:

$$\mu_{k+1} = \mu_k + s_1 \tag{4.5}$$

$$\sigma_{k+1} = \sigma_k \cdot s_2 \tag{4.6}$$

where s_1 and s_2 are sampled from two sets of values S_1, S_2. If s_1 and s_2 are equal to $0, 1$ respectively the values are resampled to make sure that a change does actually occur. We now define the contaminated Gaussian as follows:

$$X(k) \sim \begin{cases} \mathcal{N}(\mu_k, \sigma_k) & \text{if } U(0, 1) < p \\ \mathcal{N}(\mu_k, 20 \cdot \sigma_k) & \text{otherwise} \end{cases} \tag{4.7}$$

Contaminated Two-Component Gaussian with constant variance and changes in location

As shown in Figure 4.1(a), the real dataset may contain time series with more than one mode. To simulate changes in multimodal distributions, we define a two-component, contaminated Gaussian mixture model. We start by simulating a single mode with constant variance using Equation 4.4. We define a second mode in terms of a distance γ to Mode 1. This distance scales at every change point according to Equation 4.8 below. The dataset is created by sampling from random variables $X(k)$ or $X_2(k)$ with equal probability.

$$\gamma_{k+1} = \gamma_k \cdot s_3 \tag{4.8}$$

$$X_2(k) = X(k) + \gamma_k \tag{4.9}$$

4.5.1 Training & Testing datasets

To evaluate the performance of the new change detection algorithm, different training and testing datasets were created. To make the evaluation as fair as possible, the testing datasets were only used for the final evaluation of the algorithms and not for hyperparameter tuning.

Dataset 1 - ARL tuning dataset

The first dataset was used to tune the ARL of different algorithms using the settings in Appendix Section B.4, dataset S2. This dataset consists of 100,000 samples of a contaminated Gaussian distribution.

Dataset 2 - Synthetic datasets

The second dataset is a synthetic dataset that simulates changes in location with constant variance, changes in both location and variance and a multimodal distribution. The training dataset consists of the first 80% of datasets $S1, S3, S4$ (see Appendix Section B.4) concatenated one after the other. The testing dataset consists of the remaining 20% of the datasets in the same order.

Labels are generated for the ensemble algorithms. These labels are defined using Equation 4.10, with τ_k being the index of a change point in the dataset. These labels are binary values that are equal to 1 for the 25 samples after a change has occurred and are 0 otherwise. This value of 25 is equal to approximately two days of samples, which would be an acceptable detection delay for most changes according to Philips' specifications in Section 1.2.

$$y(t) = \begin{cases} 1 & \text{if } \exists\ \tau_k \ \mid 0 < (t - \tau_k) \leq 25 \\ 0 & \text{otherwise} \end{cases} \tag{4.10}$$

Dataset 3 - Real dataset: normally distributed data

The third dataset consists of a randomly selected subset of time series from the real dataset that were classified as normally distributed in Table 4.1. 82 time series were selected. The first 50% of the time series were concatenated and used for training, the second 50% were used for testing. This amounts to approximately 300,000 samples, which include 577 change points for training and 300,000 samples with 799 change points for testing. The dataset was labeled using KernelCPD [102], an implementation of the offline kernel multiple change-point algorithm proposed by Harchaoui and Cappe [7]. The settings used for this algorithm can be found in Appendix B.1.

The selection of a subset of time series is done due to memory constraints on the PC that was used for evaluation and to reduce the time required for evaluation.

Dataset 4 - Real dataset: normally distributed data, PELT labeled

A disadvantage of labeling an unlabeled dataset using an offline detection method is that the change points found by the offline detection method may be incorrect.

To evaluate whether the assumption that the change points found by the KernelCPD method in Dataset 3 are correct, we will use a different offline detection method to detect change points for the same dataset and compare the performance of an algorithm evaluated using these labels.

The same time series are selected and the same test/train split was used to create a training and testing dataset. The only difference are the set of change points used to generate the labels for training and testing: $y(t)$, defined in Equation 4.10. We use the parametric, normal univariate cost function for changes mean and variance [23] using PELT as a search function [66] to find these change points, as opposed to the non-parametric kernel based method used for Dataset 3.

This cost function is only suited for detecting changes in distributions that are normally distributed, which is why this method is not applied to the non-normal time series in the dataset.

Dataset 5 - Real dataset: non-normal data

The third dataset consists of a randomly selected subset of 82 time series from the real dataset that were classified as Skewed, Multimodal, High kurtosis or "Other" in Table 4.1. We used KernelCPD to label the change points and created training and testing datasets using the same approach as Dataset 3.

Chapter 5

Baseline Performance

In this Chapter, we will describe the currently implemented trend change detection system and will evaluate the performance of this system. The performance will be compared to the ensemble methods which are described in Chapter 3.

5.1 Current change detection implementation

In this section, we will describe how the currently implemented algorithm works, followed by an evaluation of the performance using synthetic data and an evaluation using the real dataset.

Philips has implemented a simple change detection algorithm to detect increases in the mean value of performance tests. In the remainder of this work, we will refer to this method as *CurrentCP*. The condition used to signal a change is: $S_t \geq s$ (Equation 5.5).

$$\bar{x_t} = \frac{1}{w} \sum_{i=0}^{w-1} x_{t-i} \tag{5.1}$$

$$\sigma_t = \sqrt{\frac{\sum_{i=0}^{w-1} (x_{t-i} - \bar{x_t})^2}{w-1}} \tag{5.2}$$

$$\bar{\sigma_t} = \frac{1}{w} \sum_{i=0}^{w-1} \sigma_{t-i} \tag{5.3}$$

$$UCL_t = \bar{x_t} + c \cdot \bar{\sigma_t} \tag{5.4}$$

$$S_t = \sum_{i=0}^{w_2-1} \left(\begin{cases} 0, & \text{if } x_{t-i} \leq UCL_{t-i} \\ 1, & \text{otherwise} \end{cases} \right) \tag{5.5}$$

CurrentCP works by defining a centerline using a simple moving average and a control limit which is $c \cdot \bar{\sigma_t}$ above the centerline. $\bar{\sigma_t}$ is an estimator for the standard deviation and c is a factor that controls the distance between the centerline and the control limit. A change is detected if at least s out of the last w_2 samples are strictly greater than the UCL. The sensitivity can be configured by modifying the values of c, s, w_2, w. The settings currently in use are listed in Table B.2 and will be used to evaluate the baseline performance of the trend detection method.

5.1.1 Performance evaluation - synthetic data

Simulated data were used to establish a baseline performance for this method. The performance of the method can be characterized in terms of the average run length (ARL), sensitivity/true positive rate (TPR), and the expected detection delay (EDD).

The ARL is the average time between each false positive. It can be estimated by simulating an arbitrary length, in-control time series and recording when the change point algorithm

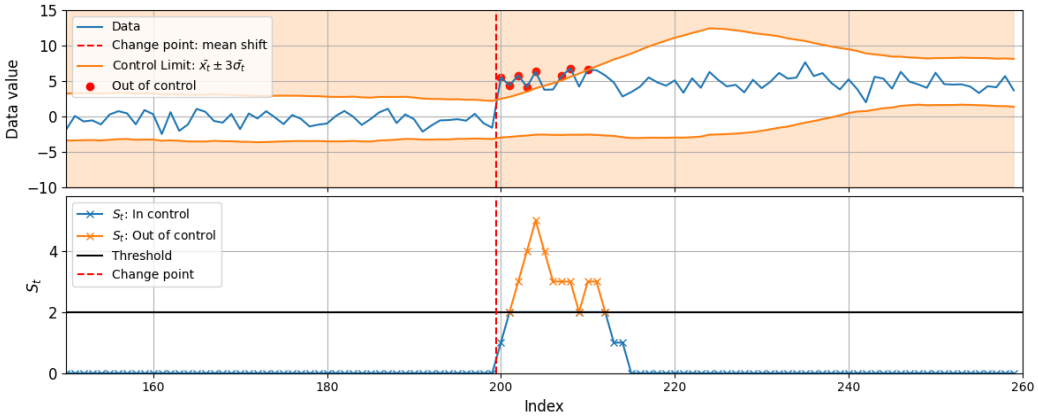


Figure 5.1: **Example of currently implemented change detection method detecting an increase in the mean of a time series.**

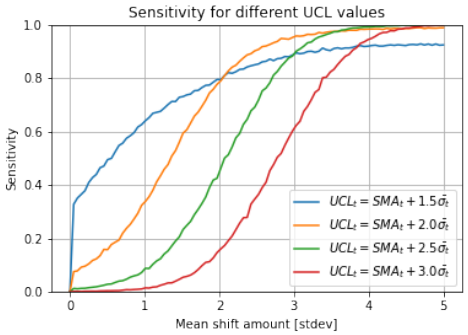


Figure 5.2: **Sensitivity of the current change detection algorithm, CurrentCP, for different mean shift magnitudes in normally distributed data** $(N(0,1) \rightarrow N(\theta,1))$ **and values of** c**, with 10,000 runs performed at every magnitude.**

signals. By repeating this procedure often enough and calculating the mean time between false positives we can obtain an estimate for the ARL. The parameters that are currently in use give an ARL of approximately 15,000 samples when sampling from a standard $N(0,1)$ Gaussian distribution. Table D.1 shows an overview of ARL values with different parameters. If we use the contaminated Gaussian distribution defined in 4.5.1 instead of a standard $N(0,1)$ distribution, the ARL decreases to 900. This suggests that the current method is not robust to anomalies.

Datasets S1, S2, and S3 were used to evaluate the performance of CurrentCP on synthetic data. The results are summarized in Table 5.1. This table shows that CurrentCP has a sensitivity of 0.2 for contaminated Gaussian datasets S1 and S2. The reason for this low sensitivity can be explained using a simulation study involving mean shifts with varying magnitudes in normally distributed data. Figure 5.2 shows that with the currently used settings ($c = 3.0$), the sensitivity of CurrentCP is less than 0.6 for mean shifts with a magnitude of less than 3σ. The algorithm therefore has a low sensitivity for small changes in location and scale. The sensitivity decreases even further for contaminated Gaussian distributions due to the presence of outliers and the resulting over-estimation of the standard deviation. The sensitivity is even lower (0.02) for the multimodal dataset S3. This is because the standard deviation of a population with multiple modes is higher than the standard deviation of the individual modes, which increases the distance of the control limits from the centerline.

	TPR			FPR			F1			EDD		
Dataset	S1	S2	S3	S1	S2	S3	S1	S2	S3	S1	S2	S3
	0.2	0.19	0.02	0.46	0.41	0.15	0.24	0.24	0.04	7	5	10
Mean	0.14			0.34			0.19			8		

Table 5.1: **Synthetic data results for current detection method.**

Type	TPR	FPR	F1	TP	FP	FN	EDD
Constant	0.13	11.60	0.02	9	777	58	1.78
Near Constant	0.21	5.09	0.07	36	890	139	5.64
Multimodal	0.20	1.63	0.14	267	2,150	1,049	3.81
Skewed	0.17	2.78	0.08	216	3,614	1,083	5.92
Heavy Tailed	0.20	0.98	0.18	73	362	295	4.73
Normal	0.19	0.61	0.21	1,002	3,257	4,296	4.51
Other	0.20	1.55	0.15	11	85	44	5.27
Total	0.19	1.30	0.15	1,614	11,135	6,964	4.61

Table 5.2: **Performance of current trend detection method when applied to the real dataset, categorized by signal type.**

5.1.2 Performance evaluation - real data

The implementation of CurrentCP which is currently in use can only detect increases in the mean and scale. In order to compare the performance with the offline kernel detection method, which is also able to detect decreases in the mean, the algorithm was modified to add a lower control limit defined as $LCL_t = \bar{x_t} - c \cdot \bar{\sigma_t}$. CurrentCP was applied to each of the 909 time series and change points were recorded and compared to those detected by the offline method. The results in Table 5.2 show a false positive rate which is significantly higher than expected (0.6-12), which may be caused by anomalous values in the dataset. The false positive rate is especially high for constant and near-constant signals. Estimating the population variance is not possible when all values are the same within a window. Any departure from this constant value will be considered a change. As expected, CurrentCP performs better when the underlying distribution of the time series is Gaussian. The false positive rate is significantly lower for time series which have been classified as normal.

Chapter 6

Ensemble methods

In this Chapter, we will evaluate the different configurations of ensemble methods presented in Chapter 3.

6.1 Evaluation

In this section, we will explain how the algorithms were evaluated followed by a summary of the results. Unless otherwise specified, the synthetic dataset will refer to dataset 1 and the real dataset will refer to dataset 3. The composition of these datasets are explained in Section 4.5.1.

6.1.1 Algorithm initialization and training

We begin by tuning the ARL of univariate algorithms using Dataset 2 [1]. If the algorithms have additional parameters these are tuned manually using synthetic validation datasets. These algorithms are then added to the ensemble and the decision fusion algorithm is trained on the specified training dataset. Any hyperparameters are tuned using a validation dataset with the procedures described in Chapter 3.

We create two versions of each ensemble by training the algorithm on either the synthetic dataset or the real dataset. We can use these different versions to evaluate how well the algorithms generalize to types of data that they have not seen yet. In addition, the results can be used to evaluate whether the synthetic dataset accurately represents the real dataset. Ideally, an algorithm that is trained on the synthetic dataset would also perform well on the real dataset.

6.1.2 Algorithm Evaluation

We evaluate each of the algorithms on the different testing datasets, which are defined in 4.5.1. A summary of the results is presented below, with more complete tables and figures available in Appendix Section E.1.

Dataset 1, a synthetic dataset composed of different types of synthetic changes, was used to evaluate the performance of the algorithms on synthetic data. The results are summarized in Table 6.1.

Datasets 3, 4 and 5 [2] were separately used to evaluate the performance of the algorithms on the real dataset. The results are summarized in Table 6.2 and Figure 6.1.

[1] Dataset 2 is a contaminated Gaussian with constant mean and variance

[2] Normally distributed data labeled by the offline version of KernelCPD, non-normally distributed data labeled by the offline version of KernelCPD, and normally distributed data labeled by PELT respectively

6.1.3 Synthetic data test results

Method	Training Dataset	TPR	FPR	F1	EDD
Random Forest	Synthetic	0.63	0.07	0.74	13
Conv1D	Synthetic	0.51	0.04	0.66	16
KernelCPD		0.61	0.28	0.64	9
Majority vote	Synthetic	0.60	0.33	0.62	10
KSWIN		0.55	0.47	0.54	17

Table 6.1: **Top 5 results based on F1-scores for combined synthetic datasets. The rest of the results can be found in Appendix Tables E.1, E.2 and Figure E.1.**

6.1.4 Real data results

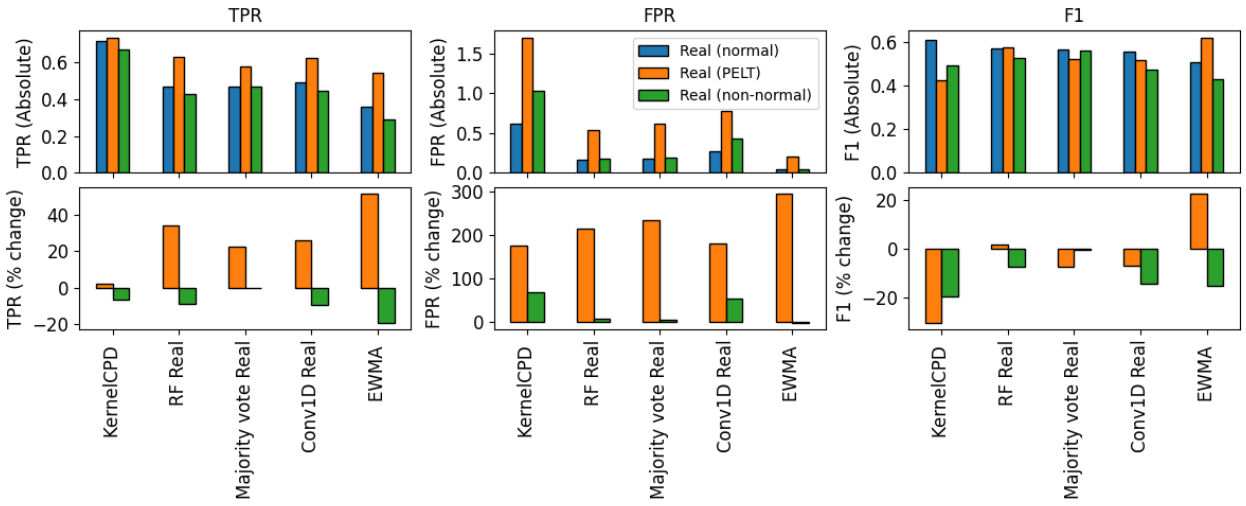


Figure 6.1: **Top change detection 5 algorithms based on F1-scores for algorithms evaluated on the real dataset. The second row of figures show relative changes in scores compared to the evaluation of the same algorithm on the real, normal dataset. The same data is tabulated in Table 6.2.**

	TPR			FPR			F1			EDD		
Method ‖ Training	3 Normal KernelCPD	4 Normal PELT	5 Non-Normal KernelCPD	3	4	5	3	4	5	3	4	5
KernelCPD	0.72	0.73	0.67	0.62	1.70	1.04	0.61	0.43	0.49	9	7	8
RF ‖ 3	0.47	0.63	0.43	0.17	0.54	0.18	0.57	0.58	0.53	13	12	12
Majority vote ‖ 3	0.47	0.57	0.47	0.18	0.61	0.19	0.57	0.53	0.56	11	10	11
Conv1D ‖ 3	0.49	0.62	0.45	0.28	0.79	0.43	0.56	0.52	0.48	14	14	15
EWMA	0.36	0.54	0.29	0.05	0.20	0.05	0.51	0.62	0.43	11	11	10

Table 6.2: **Top change detection 5 algorithms based on F1-scores for algorithms evaluated on the real dataset. The first column defines the change detectoin method followed by the dataset it was trained on, if applicable. The same data is visualized in Figure 6.1.**

6.2 Discussion

In this section, we will summarize and discuss the results of our evaluation process.

All three decision fusion methods that were evaluated are promising and can outperform the individual algorithms that compose the ensemble. KernelCPD is one of the methods in the ensemble that has good performance in both the synthetic dataset and the real dataset. However, the KernelCPD method's performance at the current sensitivity settings decreases dramatically when switching to labels generated by a different algorithm. In addition, the false positive rate for this algorithm is high, more than 60% on the real dataset regardless of the type of labels used. This is higher than the maximum FPR of 50% suggested by [8] before human operators start to lose faith in the system. Although the accuracy of these real-data results is questionable due to the lack of a ground truth, the FPR is also higher than the Random Forest and Conv1D algorithms evaluated on the synthetic dataset.

Table 6.1 shows that the Random Forest algorithm trained using synthetic data outperforms the best univariate algorithms (KernelCPD and KSWIN) when evaluated on the synthetic dataset, in terms of a higher TPR and lower FPR. The higher TPR shows that the algorithm can combine results from different univariate algorithms and does not simply copy the predictions of the best univariate algorithm. This comes at the cost of a higher detection delay than the univariate algorithms, which is likely caused by the refinement steps added to the final prediction. On average, the random forest algorithm outperforms all other algorithms in terms of F1-scores by a small margin (see Appendix Figure E.2. For this reason, we would opt for this algorithm out of all three ensemble methods although the differences are very small. Further testing and an evaluation by the end-users of the system are required to decide which method to use in practice.

6.2.1 Comparison to current method

All methods are a substantial improvement compared to the method that is currently in use, CurrentCP. Table E.4 contains the performance evaluation results for the entire dataset split by distribution type. Except for constant and near-constant data types, the Random Forest algorithm is consistently able to detect changes with a 50-100% higher detection true positive rate and a 75% lower false positive rate compared to the results for the current detection method in Table 5.2.

Although the algorithm that is currently in detects changes very quickly (within 2-10 samples) the TPR is lower and the FPR is higher than all ensemble methods across all datasets (except for the majority voting ensemble trained on the synthetic dataset). Based on the most conservative estimates using Table E.3 if Philips switches to the Random Forest algorithm trained on real data, we predict that at least 60% more change points will be detected with a 35% lower false positive rate. The difference in performance is especially large for non-normal data, with an 86% higher TPR and an 87% lower FPR.

6.2.2 Offline algorithms for labeling

Figure 6.1 shows that there are significant changes in performance when switching from the KernelCPD labels to the PELT labels for the same normally distributed data. The online KernelCPD method was expected to perform well on the dataset labeled by the offline version of the same algorithm, as the same sensitivity settings were used. The FPR for all algorithms increases significantly, although the difference is especially large for the online KernelCPD method (from 0.62 to 1.7). This increase in the FPR suggests that the offline PELT method is less sensitive than the offline KernelCPD method.

The difference in performance highlights one of the major problems with unsupervised labeling of the dataset and training algorithms on these labels as it is not known which offline detection method and sensitivity settings are more correct. To improve the quality of the training data, we suggest a manual review of the change points and sensitivity settings of the offline algorithms.

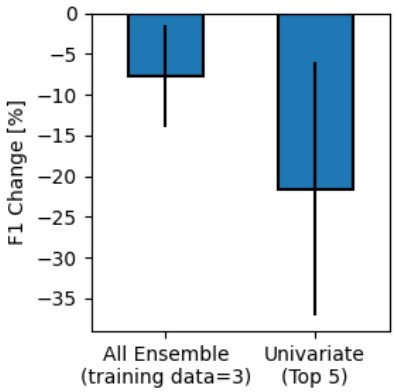


Figure 6.2: **Change in F1 score for algorithms evaluated on dataset 3 vs. 4. The 5 univariate algorithms that have the highest performance on dataset 3 are compared to all three ensemble methods that were trained on dataset 3.**

6.2.3 Generalization

In this context, the ability to generalize means that an algorithm can detect changes in distributions or types of data that have not been encountered during training. Generalization is important in a software development context as adding a new test to a performance test suite may produce data that has not been encountered before.

Baseline performance for each ensemble algorithm is first established by training and evaluating the algorithm on dataset 3. The relative change in performance when evaluating the trained algorithms on dataset 4 is shown In Figure 6.2. What can be clearly seen in this figure is that on average, all algorithms perform worse on dataset 4. One reason for this is that the ensemble contains several parametric methods that assume the time series are normally distributed. The figure shows that ensemble methods seem more robust to situations when this assumption is violated, even when this type of data has not been encountered before.

6.2.4 Synthetic data accuracy

Ideally, an algorithm that is trained on the synthetic dataset would perform equally well on the real dataset. This would suggest that the changes in the synthetic dataset accurately represent the changes in the real dataset.

However, this does not seem to be the case. Figure 6.3 shows that ensemble algorithms that have been trained on synthetic data are better at detecting changes in a synthetic dataset than the same algorithm trained using real data.

The data in Table 6.3 and E.2 suggest that the algorithms that are trained on the real data are less sensitive to changes in the multimodal dataset. One potential explanation for these findings is that the types of simulated changes occur less frequently in the real training dataset. Changes that are not simulated in the synthetic data, for example, are changes in the number of modes and the weight of each mode although these changes do occur in the real dataset, for example in Figure 4.1(a).

To remedy this and improve the quality of the synthetic dataset, one could perform an exhaustive analysis and categorization of the types of changes that occur in the real dataset. This analysis could be used to create a new synthetic dataset that is more similar to the real dataset. Unfortunately, due to the volume of data and time constraints, this exhaustive analysis could not be completed.

6.2.5 Missed change points

To gain insight into the limitations of the ensemble algorithm, we analyzed the change points in the real dataset that were not detected by the ensemble algorithm. Figure 6.4 shows the

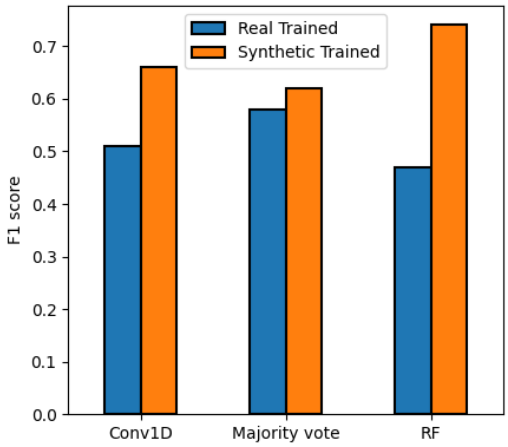


Figure 6.3: **F1 scores for ensemble algorithms evaluated on the synthetic dataset with different datasets used for training.**

Method	Training Dataset	F1		
		S1	S3	S4
RF	Synthetic	0.84	0.76	0.59
	Real	0.75	0.44	**0.04**
Conv1D	Synthetic	0.83	0.71	0.33
	Real	0.72	0.6	**0.08**
Majority vote	Synthetic	0.77	0.78	0.25
	Real	0.82	0.64	**0.11**

Table 6.3: **F1 scores per synthetic dataset for ensemble algorithms evaluated on the synthetic dataset with different datasets used for training. S1=Mean change with constant variance, S2=Mean & Variance change, S4=Multimodal changes**

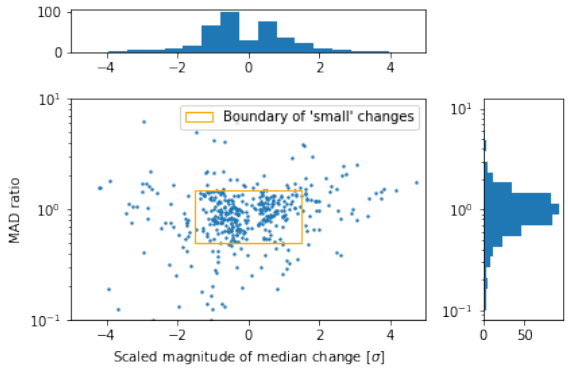
distribution of scale and location of these missed changes. The majority of the missed change points (282/427) can be considered small changes, which fall inside the orange box in Figure 6.4(a). Smaller changes generally take longer to detect using algorithms such as CUSUM [87] and EWMA. When combined with the additional delay caused by the ensemble decision refinement procedure the maximum detection delay of 25 samples can be exceeded for these smaller changes.

This reasoning does not apply to changes with large magnitudes, as these should be detectable within 25 samples. Figure 6.4(b) may give a partial explanation for these missed change points: a majority of these missed change points occur less than 50 samples after another change point. Detecting multiple change points in quick succession can be difficult for parametric algorithms that require an estimation of in-control parameters from the data stream or algorithms that compare new samples with a window of past samples such as ScanB [72] or EMW [45].

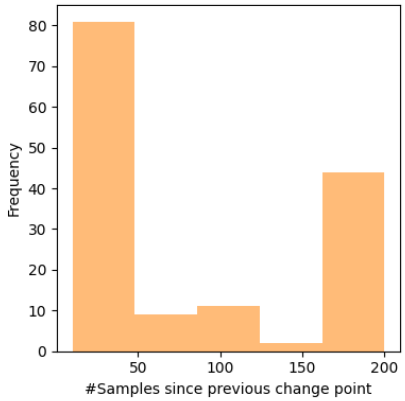
This leaves the change points with a high magnitude of change and a sufficient number of samples since the previous change point. Unfortunately, the Random Forest algorithm used in the ensemble is a black-box algorithm. This makes it difficult to find out how the algorithm arrived at its prediction. All missed change points are visualized in Appendix Figure E.4. Although some change points can be grouped into similar types (e.g. changes from a unimodal to a bimodal signal or changes in variance with constant mean) we are unable to find a common trend between all missed change points. Upon further investigation into the data, 38/44 of these change points were also missed by the EWMA algorithm in the ensemble. The Random Forest ensemble assigns a very high overall feature importance to EWMA statistics, as shown in Figure 3.7. This could be a contributing factor, however, further investigation is required to conclusively determine why these change points were missed.

6.3 Future Work

In this section, we present a number of methods which can be used to improve the performance of the ensemble.



(a) Distribution of magnitudes of scale and location for change points missed by the Random Forest ensemble. The magnitude of scale change is defined as the ratio between the post-change MAD and the pre-change MAD, while the location change magnitude is defined as the change in median divided by the pre-change MAD.



(b) Samples since the previous change point for missed change points which have large magnitudes, defined as a change in the median that falls outside the range $(-1.5, 1.5)$ or a change in scale outside the range $(-0.5, 2)$.

Figure 6.4: **Distributions of change points missed by the random forest algorithm in the real dataset.**

6.3.1 Training data

Data imbalance

The current training datasets with real data are heavily imbalanced, as there are approximately 800 changes in a dataset with 300,000 samples. The training data therefore contains more in-control samples than samples with change points. Reducing the amount of redundant information in imbalanced training datasets is known to improve the performance of classifiers [58]. This problem is currently approached by randomly undersampling the in-control samples, which has the disadvantage of also removing informative samples from the training set. Other approaches such as the Condensed Nearest Neighbours method [42] and removing Tomek links [100] could be used to retain informative samples in the training data while removing redundant samples and may improve the performance of the classifiers in the ensemble.

Incremental training

Due to the volume of data and practical constraints regarding memory and time, the dataset used for training the decision fusion algorithms was limited to a fraction of the available data. Using more data for training, either through incremental training, increasing the amount of available memory or by removing redundant features could result in improved performance.

Mixing synthetic and real data

Ensembles trained on synthetic data do not perform well on real data and vice versa. Although the performance on real data is more important, an ensemble that performs equally well on both datasets may be able to generalize well to other datasets. Through the combination of synthetic and real data during training, we may be able to improve the performance on both datasets. Initial experiments are encouraging and show that a Random Forest trained on both real and synthetic data improves performance on the synthetic dataset significantly, while only causing a slight decrease in the performance on the real datasets (see Tables E.1 and E.3).

6.3.2 Ensemble improvement

Thresholds and tuning

The sensitivity of each algorithm in the ensemble was tuned using the ARL on a synthetic dataset. Table E.3 shows that some algorithms (e.g. EWMA) have thresholds that may be too low, while others (e.g. KernelCPD) may be too high. Other methods for tuning the sensitivity, or including multiple algorithms with different sensitivity settings could be investigated.

Methods for constant & near constant signals

Table E.4 shows that the performance of the Random Forest ensemble on the metrics classified as constant or near-constant is quite poor. These metrics, which are often types of count data, are more likely to be Poisson than Gaussian distributions. By including algorithms in the ensemble that are suited specifically for these types of signals we could improve the overall performance of the ensemble.

6.3.3 Decision Fusion improvements

Improved FCN design

Although the overall performance of the convolutional neural network is comparable with the Random Forest algorithm, FCNs should be able to detect patterns in time series data that may be lost when aggregating the data for the Random Forest algorithm. The architecture used for the evaluation was not designed with multivariate inputs in mind. Other architectures may be more suited for this application, such as a Multivariate LSTM-FCN [62] architecture which uses a Long Short-Term Recursive Neural Network in combination with an FCN to learn temporal dependencies in the data which an FCN alone cannot. In an evaluation with 35 different multivariate benchmark datasets, the architecture is consistently able to outperform univariate architectures.

Reset procedures

Each of the algorithms in the ensemble run independently from the other algorithms in the ensemble. Some of the algorithms require a reset after a change has been detected to reinitialize parameters with new post-change statistics estimated from the time series. An algorithm which is too sensitive may prematurely trigger this reset procedure, which is detrimental to the performance of the algorithm. The results show that the decision fusion algorithm can reduce the false positive rate significantly compared to the individual algorithms. Therefore, the performance could be improved by waiting to reset algorithms until the decision fusion algorithm detects a change.

Filtering warnings

The disadvantage of performing univariate change detection is that a change that affects multiple performance metrics simultaneously can result in multiple warnings if the changes are not detected simultaneously. This could result in the same reduction in a user's perceived reliability of a detection as a false positive if it occurs very frequently. A change validation layer which checks if the same change has already been detected in a different performance metric can be used to filter these warnings.

Chapter 7

Conclusions

In this final Chapter, we will summarize the problem and the research gap that this work addresses. We will describe the method used to solve this problem, discuss our results and answer the main research question. A summary of the contributions to the state of the art will be presented, followed by a discussion of the limitations of this work. Finally, we will make suggestions for future research.

Philips has created an automatic system, called *CurrentCP*, that monitors metrics from performance benchmarks for changes that are detrimental to an embedded system's performance. Unfortunately, *CurrentCP* leaves much to be desired as it frequently produces false positives and misses changes. Costly manual intervention is therefore required to monitor for missed changes and validate the changes flagged by *CurrentCP*.

The goal of this thesis was to improve on *CurrentCP* and answer the following research question: *How can changes in the performance benchmarks of an embedded system be detected automatically with minimal delay?* Measurable goals for a new monitoring system include a minimum true positive rate (TPR) of 80%, maximum false positive rate (FPR) of 50%, and a maximum detection delay of 48 hours.

7.1 Method & Results

In this Section, we will answer the research question and present the main results of this work.

An initial analysis of the dataset of performance metrics, provided by Philips, showed that there were several complicating factors in the problem that we were trying to solve. Firstly, the large number of distinct performance metrics eliminates the ability to manually select and tune a change detection algorithm for each performance metric. In addition, there is no one-size-fits-all algorithm that can detect changes in both process location and dispersion for all types of distributions encountered in the dataset, while minimizing the detection delay.

Ensemble methods were identified as a promising approach to solve this problem due to their ability to combine the unique advantages of individual change detection methods and compensate for biases through ensemble diversity. In related work, ensemble methods are used for both univariate and multivariate online change detection. These approaches are characterized by the use of homogeneous ensembles and simple majority voting schemes as a decision fusion method. Aspects that have not been addressed in prior literature are how ensemble diversity and more complex decision fusion algorithms can contribute to the overall performance of the ensemble. Therefore, this work focuses on addressing this gap in previous research.

To address the identified research gap and improve on Philips' existing change detection system, we propose a generic architecture that combines multiple algorithms in an ensemble and aggregates decisions and statistics for a decision fusion algorithm, which then decides if a change has occurred or not.

To demonstrate the applicability of this architecture, we compared three different decision fusion algorithms: Random Forest, Fully Convolutional Neural Network, and Weighted

Majority Voting. All three can outperform the best-performing algorithms in the ensemble on real and synthetic data in terms of F1-score.

Our overall recommendation to Philips would be to use the ensemble with a Random Forest for decision fusion. We predict this will achieve a TPR between 43-63%, an FPR between 17-54%, and an expected detection delay of 24 hours. Despite not achieving all of the goals that were specified in Section 1.2, this is a huge improvement compared to the *CurrentCP*, which has an estimated TPR of 21-38%, and an estimated FPR of 46-140%.

7.2 Contributions

We have made several contributions to the state of the art, which we will detail below.

Our solution firstly addresses the research gap by demonstrating the benefits of a more diverse ensemble. A selection of more than 10 different algorithms was included in the ensemble based on an analysis of changes and distributions observed in the dataset, with the intent to maximize diversity and thereby increase the overall robustness. We have succeeded in this goal: compared to individual methods in the ensemble, an ensemble that is only trained on real data with a normal distribution can generalize well to data with skewed, multimodal, and heavy-tailed distributions. On average, there is a slight decrease in the F1-score of $6\% \pm 4\%$ compared to a considerably larger decrease of $20\% \pm 15\%$ for the individual algorithms in the ensemble.

Another innovation is the inclusion of aggregated statistics in the decision fusion process, unlike related work, which only considers the final decisions of each algorithm in the ensemble. These statistics can be used to compensate for sensitivity thresholds that have been set too high. Even with minimal tuning of these sensitivity thresholds, the ensemble is still able to outperform the individual algorithms.

In this work, we also show that ensemble methods for change detection can profit from more sophisticated decision fusion than simple majority voting. We demonstrate that decision fusion using a Random Forest outperforms majority voting, consistently achieving 10-20% higher F1-scores on both synthetic and real data.

7.3 Limitations of this work

Some assumptions, simplifications and choices have been made during this thesis, which we will address below.

The main limitation of our work is that we have been unable to manually verify the change points found by the offline change detection method in the dataset provided by Philips. These change points have been used as the ground truth to generate labels for training and evaluation. It is possible that change points have been missed or that there are false positives. The quality of our results could be improved by manually validating these labels and repeating the processing of tuning, training and evaluating all algorithms.

Another, more practical limitation is that the ensemble detects all types of changes in performance metrics, including changes that indicate the overall embedded system has improved. To make the monitoring system more usable, the warnings sent to developers could be limited to those related to performance degradation. This could be done by including an additional validation step that analyzes what type of change has occurred using an offline change detection method.

7.4 Recommendations for Philips

This work has resulted in the creation of a new change detection algorithm, which is an improvement compared Philips' existing monitoring system. Before *CurrentCP* is replaced entirely, we would like to make a few practical recommendations.

We propose allowing the ensemble to run in parallel with the current method for a predetermined evaluation period. During this period, offline detection methods should be used periodically to find changes that have been missed. Any changes signaled by the ensemble

should be validated, which will result in a new, correctly labeled dataset. At the end of the evaluation period, Philips can choose whether to continue with the new ensemble or a different algorithm.

To tune the sensitivity of the ensemble, we suggest estimating the true cost of a false positive, false negative, and detection delay for the company. The overall effectiveness of a detection system is always a balance between the TPR and FPR. We can simplify this problem by modeling the overall cost of operating the monitoring system as a (non-)linear function of the false positive rate, false negative rate and detection delay. Even if our system does not meet the original goals, we can create a more cost-effective system by selecting the sensitivity settings which minimize the cost for Philips.

Finally, the performance of any change detection algorithm Philips decides to use could be improved by filtering and preprocessing the data. Firstly, by ensuring that performance metrics always arrive in chronological order. Performance benchmarks that evaluate older versions of the code are not representative for the current system performance and should be ignored by the change detection algorithm. Although a change detection algorithm should be robust against these low-frequency occurrences, the overall performance of a change detection algorithm will improve if these anomalous samples are excluded.

7.5 Future Research

We have identified several directions for future research, which are summarized below.

Multivariate extensions Future research is needed to extend the univariate, online method presented in this work to a multivariate method. Multivariate change detection could reduce the detection delay and increase the sensitivity of the monitoring system. Initial experiments have shown that a multivariate approach is complex due to the sparsity of the changes and the high dimensionality of the dataset. One approach could be to consider only clusters of performance metrics that are known to change simultaneously. These clusters could be manually selected or discovered automatically using methods such as association learning [47].

Sensitivity settings Currently, the sensitivity settings of the algorithms in the ensemble are tuned manually by specifying a minimum average run length on a synthetic dataset. This method is simple and effective, however, may result in some algorithms with threshold settings that are too low or too high. The overall performance of the ensemble is dependent on these settings, which makes this a complex optimization problem with many independent variables that need to be tuned simultaneously to obtain the best performance. Additional research is required to find an effective method to approach this problem.

Reset procedure Some of the algorithms in the ensemble that require Phase 1 initialization have been modified for use in this continuous monitoring setting by including a hierarchical reset procedure, which is triggered whenever a change is detected. Whether this reset procedure should be used at all, and whether it should be triggered by the decision fusion algorithm or by the individual algorithm are both factors that affect the overall performance of the ensemble. Future study could address these factors, for example by including them in the optimization procedure along with the sensitivity settings of the algorithm, as mentioned in the paragraph above.

Bibliography

[1] Na'eem Hoosen Agjee, Onisimo Mutanga, Kabir Peerbhay, and Riyad Ismail. The Impact of Simulated Spectral Noise on Random Forest and Oblique Random Forest Classification Performance. *Journal of Spectroscopy*, 2018, March 2018.

[2] Jehad Ali, Rehanullah Khan, Nasir Ahmad, and Imran Maqsood. Random Forests and Decision Trees. *International Journal of Computer Science Issues(IJCSI)*, 9(5):272, September 2012.

[3] Cesare Alippi, Giacomo Boracchi, and Manuel Roveri. Hierarchical Change-Detection Tests. *IEEE Transactions on Neural Networks and Learning Systems*, 28(2):246–258, February 2017.

[4] Layth C. Alwan. The Problem of Misplaced Control Limits. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 44(3):269–278, 1995.

[5] Samaneh Aminikhanghahi and Diane J. Cook. A survey of methods for time series change point detection. *Knowledge and Information Systems*, 51(2):339–367, May 2017.

[6] F. J. Anscombe and William J. Glynn. Distribution of the Kurtosis Statistic b2 for Normal Samples. *Biometrika*, 70(1):227–234, 1983.

[7] Sylvain Arlot, Alain Celisse, and Zaid Harchaoui. A Kernel Multiple Change-point Algorithm via Model Selection. *Journal of Machine Learning Research*, 20(162):1–56, 2019.

[8] Stefan Axelsson. The Base-Rate Fallacy and the Difficulty of Intrusion Detection. *ACM Transactions on Information and System Security*, 3(3):20, 2000.

[9] Rafal Baranowski, Yining Chen, and Piotr Fryzlewicz. Narrowest-Over-Threshold Detection of Multiple Change-points and Change-point-like Features. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 81(3):649–672, 2019.

[10] Michèle Basseville and Igor Nikiforov. *Detection of Abrupt Change Theory and Application*, volume 15. Prentice-Hall, Inc., April 1993.

[11] Ted Bennett and Paul Wennberg. Eliminating Embedded Software Defects Prior to Integration Test. Technical report, Triakis Corporation, 2005.

[12] Albert Bifet and Ricard Gavaldà. Learning from Time-Changing Data with Adaptive Windowing. In *Proceedings of the 2007 SIAM International Conference on Data Mining*, pages 443–448. Society for Industrial and Applied Mathematics, April 2007.

[13] Dean Adam Bodenham. *Adaptive estimation with change detection for streaming data*. PhD thesis, Imperial College London, London, 2014.

[14] Leo Breiman. Random Forests. *Machine Learning*, 45(1):5–32, October 2001.

[15] B. E. Brodsky and B. S. Darkhovsky. A Posteriori Change-Point Problems. In *Nonparametric Methods in Change-Point Problems*, Mathematics and Its Applications, pages 36–94. Springer Netherlands, Dordrecht, 1993.

[16] Bart Broekman and Edwin Notenboom. *Testing Embedded Software*. Addison Wesley, Boston, 2003.

[17] Dariusz Brzezinski and Jerzy Stefanowski. Ensemble Diversity in Evolving Data Streams. In Toon Calders, Michelangelo Ceci, and Donato Malerba, editors, *Discovery Science*, Lecture Notes in Computer Science, pages 229–244, Cham, 2016. Springer International Publishing.

[18] Eric L. Bullock, Curtis E. Woodcock, and Christopher E. Holden. Improved change monitoring using an ensemble of time series algorithms. *Remote Sensing of Environment*, 238:111165, March 2020.

[19] Meshack Linda Bulunga. Change-point detection in dynamical systems using auto-associative neural networks. Master's thesis, Stellenbosch University, 2012.

[20] Giovanna Capizzi and Guido Masarotto. An Adaptive Exponentially Weighted Moving Average Control Chart. *Technometrics*, 45(3):199–207, 2003.

[21] Philippe Castagliola. A New S2-EWMA Control Chart for Monitoring the Process Variance. *Quality and Reliability Engineering International*, 21(8):781–794, 2005.

[22] Hock Peng Chan. Optimal sequential detection in multi-stream data. *The Annals of Statistics*, 45(6):2736–2763, 2017.

[23] Jie Chen and Arjun K. Gupta. Mean and Variance Change. In *Parametric Statistical Change Point Analysis*, pages 57–88. Springer, Boston, 2012.

[24] Yudong Chen, Tengyao Wang, and Richard J. Samworth. High-dimensional, multiscale online changepoint detection. *arXiv:2003.03668 [math, stat]*, October 2020.

[25] Michael Cheng. *Medical device regulations: global overview and guiding principles*. World Health Organization, Geneva, 2003.

[26] Haeran Cho. Change-point detection in panel data via double CUSUM statistic. *Electronic Journal of Statistics*, 10(2), January 2016.

[27] Haeran Cho and Piotr Fryzlewicz. Multiple-change-point detection for high dimensional time series via sparsified binary segmentation. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 77(2):475–507, March 2015.

[28] Ronald B. Crosier. Multivariate Generalizations of Cumulative Sum Quality-Control Schemes. *Technometrics*, 30(3):291–303, 1988.

[29] Christophe Croux and Catherine Dehon. Robust Estimation of Location and Scale. In *Wiley StatsRef: Statistics Reference Online*. American Cancer Society, 2014. [Online]. Available: https://onlinelibrary.wiley.com/doi/full/10.1002/9781118445112.stat07416.

[30] Miklós Csörgő and Lajos Horváth. Nonparametric methods for changepoint problems. In *Quality Control and Reliability. Series 'Handbook of Statistics'*, volume 7 of *Handbook of Statistics*, pages 403–425. Elsevier, Amsterdam, January 1988.

[31] Ralph D'Agostino and E. S. Pearson. Tests for Departure from Normality. Empirical Results for the Distributions of b2 and $\sqrt{}$ b1. *Biometrika*, 60(3):613–622, 1973.

[32] Ralph B. D'Agostino and Albert Belanger. A Suggestion for Using Powerful and Informative Tests of Normality. *The American Statistician*, 44(4):316–321, 1990.

[33] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, April 2002.

[34] Dua Dheeru and Casey Graff. UCI Machine Learning Repository, 2017. Accessed on: May 7, 2021. [Online]. Available: https://archive.ics.uci.edu/ml.

[35] Pan Du, Warren A. Kibbe, and Simon M. Lin. Improved peak detection in mass spectrum by incorporating continuous wavelet transform-based pattern matching. *Bioinformatics*, 22(17):2059–2065, September 2006.

[36] Ryan Elwell and Robi Polikar. Incremental Learning of Concept Drift in Nonstationary Environments. *IEEE Transactions on Neural Networks*, 22(10):1517–1531, October 2011.

[37] Will Faithfull. *Unsupervised Change Detection in Multivariate Streaming Data.* PhD thesis, Bangor University, November 2018.

[38] Will Faithfull, Juan Rodríguez, and Ludmila Kuncheva. Combining Univariate Approaches for Ensemble Change Detection in Multivariate Data. *Information Fusion*, 45, February 2018.

[39] Lev Faivishevsky. Information theoretic multivariate change detection for multisensory information processing in Internet of Things. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6250–6254, March 2016.

[40] Martin Fowler and Matthew Foemmel. Continuous Integration, September 2000. Accessed on: Apr 22, 2021. [Online]. Available: https://martinfowler.com/articles/originalContinuousIntegration.html.

[41] Asghar Ghasemi and Saleh Zahediasl. Normality Tests for Statistical Analysis: A Guide for Non-Statisticians. *International Journal of Endocrinology and Metabolism*, 10(2):486–489, 2012.

[42] K. Gowda and G. Krishna. The condensed nearest neighbor rule using the concept of mutual nearest neighborhood. *IEEE Transactions on Information Theory*, 25(4):488–490, July 1979.

[43] Pierre Granjon. The CuSum algorithm - a small review, June 2013.

[44] Thomas Grundy, Rebecca Killick, and Gueorgui Mihaylov. High-Dimensional Changepoint Detection via a Geometrically Inspired Mapping. *Statistics and Computing*, 30(4):1155–1166, July 2020.

[45] Lingzhe Guo and Reza Modarres. Two multivariate online change detection models. *Journal of Applied Statistics*, 0(0):1–22, September 2020.

[46] Georg Hahn, Paul Fearnhead, and Idris A. Eckley. BayesProject: Fast computation of a projection direction for multivariate changepoint detection. *Statistics and Computing*, 30(6):1691–1705, November 2020.

[47] Jiawei Han, Jian Pei, Yiwen Yin, and Runying Mao. Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach. *Data Mining and Knowledge Discovery*, 8(1):53–87, January 2004.

[48] Zaid Harchaoui and Olivier Cappe. Retrospective Mutiple Change-Point Estimation with Kernels. In *IEEE/SP 14th Workshop on Statistical Signal Processing*, pages 768–772, August 2007.

[49] J. A. Hartigan and P. M. Hartigan. The Dip Test of Unimodality. *The Annals of Statistics*, 13(1):70–84, March 1985.

[50] Douglas M. Hawkins and K. D. Zamba. A Change-Point Model for a Shift in Variance. *Journal of Quality Technology*, 37(1):21–31, January 2005.

[51] Kaylea Haynes, Idris A. Eckley, and Paul Fearnhead. Computationally Efficient Changepoint Detection for a Range of Penalties. *Journal of Computational and Graphical Statistics*, 26(1):134–143, January 2017.

[52] Kaylea Haynes, Paul Fearnhead, and Idris A. Eckley. A computationally efficient nonparametric approach for changepoint detection. *Statistics and Computing*, 27(5):1293–1305, February 2016.

[53] John D. Healy. A Note on Multivariate CUSUM Procedures. *Technometrics*, 29(4):409–412, 1987.

[54] Harold Hotelling. Multivariate Quality Control - Illustrated by Air Testing of Sample Bombsights. *Techniques of Statistical Analysis*, pages 111–184, 1947.

[55] Barbara F.F. Huang and Paul C. Boutros. The parameter sensitivity of random forests. *BMC Bioinformatics*, 17(1):331, September 2016.

[56] J. Stuart Hunter. The Exponentially Weighted Moving Average. *Journal of Quality Technology*, 18(4):203–210, October 1986.

[57] Philips Medical Imaging. Philips Azurion Platform, February 2017. Accessed on: Apr 22, 2021. [Online]. Available: https://www.philips.com/a-w/about/news/archive/standard/news/press/2017/20170222-philips-reinforces-leadership-in-image-guided-therapy-solutions-with-global-launch-of-next-generation-azurion-platform.html.

[58] Nathalie Japkowicz and Shaju Stephen. The class imbalance problem: A systematic study. *Intelligent Data Analysis*, 6(5):429–449, January 2002.

[59] Frank J. Massey Jr. The Kolmogorov-Smirnov Test for Goodness of Fit. *Journal of the American Statistical Association*, 46(253):68–78, March 1951.

[60] Joseph J. Pignatiello Jr and George C. Runger. Comparisons of Multivariate CUSUM Charts. *Journal of Quality Technology*, 22(3):173–186, July 1990.

[61] Fazle Karim, Somshubra Majumdar, Houshang Darabi, and Shun Chen. LSTM Fully Convolutional Networks for Time Series Classification. *IEEE Access*, 6:1662–1669, 2018.

[62] Fazle Karim, Somshubra Majumdar, Houshang Darabi, and Samuel Harford. Multivariate LSTM-FCNs for Time Series Classification. *Neural Networks*, 116:237–245, August 2019.

[63] Yoshinobu Kawahara and Masashi Sugiyama. Sequential change-point detection based on direct density-ratio estimation. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 5(2):114–127, 2012.

[64] John Keane and Chris H. Kim. An odomoeter for CPUs. *IEEE Spectrum*, 48(5):28–33, May 2011.

[65] Nicolas Keriven, Damien Garreau, and Iacopo Poli. NEWMA: A New Method for Scalable Model-Free Online Change-Point Detection. *IEEE Transactions on Signal Processing*, 68:3515–3528, 2020.

[66] R. Killick, P. Fearnhead, and I. A. Eckley. Optimal detection of changepoints with a linear computational cost. *Journal of the American Statistical Association*, 107(500):1590–1598, December 2012.

[67] S. Kullback and R. A. Leibler. On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, March 1951.

[68] Ludmila Kuncheva. Classifier ensembles for detecting concept change in streaming data: Overview and perspectives. *Proc. Eur. Conf. Artif. Intell.*, January 2008.

[69] Ludmila I. Kuncheva. Change Detection in Streaming Multivariate Data Using Likelihood Detectors. *IEEE Transactions on Knowledge and Data Engineering*, 25(5):1175–1180, May 2013.

[70] Mehmet Necip Kurt. *Data-Driven Quickest Change Detection*. PhD thesis, Columbia University, 2020.

[71] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018.

[72] Shuang Li, Yao Xie, Hanjun Dai, and Le Song. Scan B-Statistic for Kernel Change-Point Detection. *arXiv:1507.01279 [cs, math, stat]*, November 2018.

[73] Andy Liaw and Matthew Wiener. Classification and Regression by randomForest. *R news*, 2:18–22, December 2002.

[74] Regina Y. Liu and Kesar Singh. A Quality Index Based on Data Depth and Multivariate Rank Tests. *Journal of the American Statistical Association*, 88(421):252–260, March 1993.

[75] Song Liu, Makoto Yamada, Nigel Collier, and Masashi Sugiyama. Change-Point Detection in Time-Series Data by Relative Density-Ratio Estimation. *Neural Networks*, 43:72–83, July 2013.

[76] Cynthia Lowry, William Woodall, Charles Champ, and Steven Rigdon. A Multivariate Exponentially Weighted Moving Average Control Chart. *Technometrics*, 34:46–53, March 2012.

[77] James M. Lucas and Ronald B. Crosier. Fast Initial Response for CUSUM Quality-Control Schemes: Give Your CUSUM A Head Start. *Technometrics*, 24(3):199–205, 1982.

[78] James M. Lucas and Ronald B. Crosier. Robust cusum: a robustness study for cusum quality control schemes. *Communications in Statistics - Theory and Methods*, 11(23):2669–2687, January 1982.

[79] David S. Matteson and Nicholas A. James. A Nonparametric Approach for Multiple Change Point Analysis of Multivariate Data. *Journal of the American Statistical Association*, 109(505):334–345, January 2014.

[80] MedlinePlus. Angioplasty and stent placement - heart, May 2021. Accessed on: May 6, 2021. [Online]. Available: https://medlineplus.gov/ency/article/007473.htm.

[81] Y. Mei. Efficient scalable schemes for monitoring a large number of data streams. *Biometrika*, 97(2):419–433, June 2010.

[82] Jacob Montiel, Max Halford, Saulo Martiello Mastelini, Geoffrey Bolmier, Raphael Sourty, Robin Vaysse, Adil Zouitine, Heitor Murilo Gomes, Jesse Read, Talel Abdessalem, and Albert Bifet. River: machine learning for streaming data in Python. *Journal of Machine Learning Research*, 22(110):1–8, 2021.

[83] Carlos Murguia and Justin Ruths. CUSUM and chi-squared attack detection of compromised sensors. In *2016 IEEE Conference on Control Applications (CCA)*, pages 474–480, Buenos Aires, Argentina, September 2016. IEEE.

[84] Hafiz Zafar Nazir, Tahir Hussain, Noureen Akhtar, Muhammad Abid, and Muhammad Riaz. Robust adaptive exponentially weighted moving average control charts with applications of manufacturing processes. *The International Journal of Advanced Manufacturing Technology*, 105(1):733–748, November 2019.

[85] Hafiz Zafar Nazir, Muhammad Riaz, and Ronald J. M. M. Does. Robust CUSUM Control Charting for Process Dispersion: Robust CUSUM Chart. *Quality and Reliability Engineering International*, 31(3):369–379, April 2015.

[86] Daniel Nikovski and Ankur Jain. Memory-Based Algorithms for Abrupt Change Detection in Sensor Data Streams. In *2007 5th IEEE International Conference on Industrial Informatics*, volume 1, pages 547–552, June 2007.

[87] E. S. Page. Continuous Inspection Schemes. *Biometrika*, 41(1/2):100–115, 1954.

[88] E. S. Page. A Test for a Change in a Parameter Occurring at an Unknown Point. *Biometrika*, 42(3/4):523–527, 1955.

[89] Emmanuel Pilliat, Alexandra Carpentier, and Nicolas Verzelen. Optimal multiple change-point detection for high-dimensional data. *arXiv:2011.07818 [math, stat]*, November 2020.

[90] Michel Piot. *Statistics In Climate Sciences I + II.* Universität Bern, Bern, 2013. Accessed on: April 30, 2021. [Online]. Available: `https://www.math-stat.unibe.ch/unibe/portal/fak_naturwis/a_dept_math/a_dept_ms/content/e237483/e237655/e243381/e281679/files281691/Chap12_ger.pdf`.

[91] Peihua Qiu and Jingnan Zhang. On Phase II SPC in Cases When Normality is Invalid. *Quality and Reliability Engineering International*, 31(1):27–35, 2015.

[92] Christoph Raab, Moritz Heusinger, and Frank-Michael Schleif. Reactive Soft Prototype Computing for Concept Drift Streams. *Neurocomputing*, 416:340–351, November 2020.

[93] Geraint Rees. Vision: the evolution of change detection. *Current biology: CB*, 18(1):R40–42, January 2008.

[94] Peter J. Rousseeuw and Christophe Croux. Alternatives to the Median Absolute Deviation. *Journal of the American Statistical Association*, 88(424):1273–1283, 1993.

[95] Yong Sheng Soh and Venkat Chandrasekaran. High-dimensional change-point estimation: Combining filtering with convex optimization. *Applied and Computational Harmonic Analysis*, 43(1):122–147, July 2017.

[96] Yan V. Sun, Lawrence F. Bielak, Patricia A. Peyser, Stephen T. Turner, Patrick F. Sheedy, Eric Boerwinkle, and Sharon L. R. Kardia. Application of machine learning algorithms to predict coronary artery calcification with a sibship-based design. *Genetic Epidemiology*, 32(4):350–360, 2008.

[97] Gábor J. Székely and Maria L. Rizzo. Energy statistics: A class of statistics based on distances. *Journal of Statistical Planning and Inference*, 143(8):1249–1272, August 2013.

[98] Alexander G. Tartakovsky, Boris L. Rozovskii, Rudolf B. Blažek, and Hongjoong Kim. Detection of intrusions in information systems by sequential change-point methods. *Statistical Methodology*, 3(3):252–293, July 2006.

[99] Alexander G. Tartakovsky and Venugopal V. Veeravalli. Asymptotically Optimal Quickest Change Detection in Distributed Sensor Systems. *Sequential Analysis*, 27(4):441–475, November 2008.

[100] Ivan Tomek. Two Modifications of CNN. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-6(11):769–772, November 1976.

[101] Dang Hoan Tran. *Change Detection in Streaming Data.* PhD thesis, Ilmenau University of Technology, Ilmenau, 2013.

[102] Charles Truong, Laurent Oudre, and Nicolas Vayatis. Selective review of offline change point detection methods. *Signal Processing*, 167:107299, February 2020.

[103] Venugopal V. Veeravalli and Taposh Banerjee. Chapter 6 - Quickest Change Detection. In Abdelhak M. Zoubir, Mats Viberg, Rama Chellappa, and Sergios Theodoridis, editors, *Academic Press Library in Signal Processing*, volume 3 of *Academic Press Library in Signal Processing: Volume 3*, pages 209–255. Elsevier, January 2014.

[104] Geoff Vining. Technical Advice: Phase I and Phase II Control Charts. *Quality Engineering*, 21(4):478–479, September 2009.

[105] Tengyao Wang and Richard J. Samworth. High-dimensional changepoint estimation via sparse projection. *arXiv:1606.06246 [math, stat]*, March 2017.

[106] Zezhong Wang and Inez Maria Zwetsloot. A Change-Point Based Control Chart for Detecting Sparse Changes in High-Dimensional Heteroscedastic Data. *arXiv:2101.09424 [stat]*, January 2021.

[107] Zhiguang Wang, Weizhong Yan, and Tim Oates. Time Series Classification from Scratch with Deep Neural Networks: A Strong Baseline. In *IJCNN 2017 : International Joint Conference on Neural Networks*, pages 1578–1585, 2017.

[108] William H. Woodall and Matoteng M. Ncube. Multivariate CUSUM Quality-Control Procedures. *Technometrics*, 27(3):285–292, 1985.

[109] M. Woźniak, M. Graña, and E. Corchado. A survey of multiple classifier systems as hybrid systems. *Information Fusion*, 16(1):3–17, 2014.

[110] Yao Xie, Jiaji Huang, and Rebecca Willett. Changepoint detection for high-dimensional time series with missing data. *IEEE Journal of Selected Topics in Signal Processing*, 7(1):12–27, February 2013.

[111] Yao Xie and David Siegmund. Sequential multi-sensor change-point detection. *The Annals of Statistics*, 41(2):670–692, April 2013.

[112] Aleksandr A Zabyshny and David R Ragland. False Alarms and Human-Machine Warning Systems. Technical Report 88w8b8g3, Institute of Transportation Studies, UC Berkeley, 2003.

Appendix A

Literature Review: Multivariate and offline change detection methods

In this Chapter, we present an overview of state-of-the-art methods in change detection. This literature review focuses specifically on the state of the art in online, multivariate change detection and offline change detection. Table A.1 categorizes a selection of these state-of-the-art methods. For a more complete overview of offline detection methods, see the recent literature survey by Truong et. al. [102].

Detection method	**Offline**	**Online**	**Univariate**	**Multivariate**	**Parametric**	**Non-Parametric**
CUSUM agg. [22] [81] [99]		x		x	x	
HCDT [3]	x (hybrid)	x	x		x	x (hybrid)
ADWIN [12]		x	x			x
OCD [24]		x		x	x	
MLR [111]		x		x	x	
SPLL [69]		x		x	x (semi)	
LL-KL [69]		x		x		x
Ensemble [38]		x		x		x
DeepQCD [70]		x		x		x
PELT [66]	x		x		x	
NP-PELT [52]	x		x			x
CROPS [51]	x		x			x
NOT [9]	x		x			x
SMUCE [95]	x		x			x
E-Divisive [79]	x			x		x
Double-CUSUM [26]	x			x	x	
inspect [105]	x			x		x
BayesProject [46]	x			x		x

Table A.1: **An overview of the most relevant state of the art change detection literature.**

CUSUM aggregation

A method which is frequently used to extend the univariate CUSUM to a multi-dimensional data stream is to perform an aggregation of CUSUM statistics. For example, Mei [81] sums the CUSUM statistics across all n coordinates and signals when the sum of statistics is greater than some threshold α, $\sum_i^n S_k^i > \alpha$ (S_k from Equation 2.4) while Tartakovsky and Veeravalli [99] suggest signaling when the maximum of the CUSUM statistic is greater than some threshold, $(\max_i S_k^i) > \alpha$.

Xie and Sigmund [111] use a similar approach with the mixture likelihood ratio (MLR) instead of the CUSUM statistic, which assumes that the change sparsely affects the coordinates of the data stream. Chan [22] proposes a transformation of the CUSUM statistic and shows that this transformation performs best out of all these methods to detect sparse changes.

OCD

Chen et. al. [24] present an online change detection method used to detect changes in high dimensional Gaussian data streams. Their method performs likelihood ratio tests for different scales of mean changes on each coordinate and aggregates these tests across coordinates. The method is limited to detecting sparse and dense changes in the mean vector.

Neural networks

Neural networks have also been successfully applied to change detection problems. For example, Bulunga [19] uses an autoencoder neural network to extract important features from a time series and reconstruct the time series using these features. Changes in the magnitude of the reconstruction error are used to detect changes.

DeepQCD [70] is another example of a (deep) neural network applied to online change detection problems. In a data-driven approach, features are extracted from high-dimensional time series and mapped onto an internal state using recurrent neural networks. A decision whether a change has occurred is made based on this internal state and new observations of the process. The author shows that the model-free deep learning approach is an improvement in terms of detection delay and false positive rate when compared to parametric change detection methods.

The disadvantage of neural network based methods is the training data necessary to pretrain the networks. Kurt (2020) and and Bulunga (2012) both require datasets with labeled in-control and out-of-control periods, making their methods unsuitable when the ground truth is not available. In addition, a large volume of training data may also be necessary to prevent over-fitting for more complex neural networks. This dependency on a large volume of data could be alleviated by using techniques such as incremental learning [36].

Appendix B

Settings and Hyperparameters

To make this work more reproducible, we present an overview of settings and hyperparameters which were used.

B.1 Settings used for offline analysis of change points

1. **Method used**: KernelCPD, implementation from the **Ruptures** package [102]
2. **Kernel Type**: RBF
3. **Minimum change point distance**: 50
4. **Penalty term**: 5
5. **Rolling window size**: 25
6. **Wavelet transform widths**: [15]

B.2 Hyperparameters for FCN network

Layer	**Hyperparameter**	**Value**
Conv1D layer 1	#Filters	35
Conv1D layer 1	Kernel size	4
Conv1D layer 2	#Filters	35
Conv1D layer 2	Kernel size	4
Dropout	Dropout	0.5
Max Pool	Pool size	4
Dense layer 1	Layer size	20
Dense layer 1	Activation	ReLu
Dense layer 2	Layer size	1
Dense layer 2	Activation	Sigmoid

Table B.1: **Hyperparameters for FCN network.**

B.3 Settings used for current trend detection method

Description	Parameter	Value
Distance of UCL to simple moving average (in standard deviations)	c	3.0
Sliding window size	w	25
Number of last samples compared to UCL	w_2	5
Signaling threshold value	s	2

Table B.2: **Settings used for current trend detection method.**

B.4 Settings for synthetic datasets

B.4.1 Dataset S1 - settings

- **Description**: Contaminated Gaussian with constant variance and changes in location
- **Length**: 100000 samples
- **Number of changepoints**: 540
- **Grace period**: G = 50
- **Detection period**: D = 50
- **Fraction of anomalies**: 0.05
- **Scale of anomalies**: 20
- **Mean change magnitudes**: $S = \{\pm 1, \pm 2, \pm 3, \pm 4\}$

B.4.2 Dataset S2 - settings

- **Description**: Gaussian without changes, for ARL evaluation
- **Length**: 100000 samples
- **Fraction of anomalies**: 0.05
- **Scale of anomalies**: 20

B.4.3 Dataset S3 - settings

- **Description**: Contaminated Gaussian with changes in both scale and location
- **Length**: 100000 samples
- **Number of changepoints**: 540
- **Grace period**: G = 50
- **Detection period**: D = 50
- **Fraction of anomalies**: 0.05
- **Scale of anomalies**: 20
- **Mean change magnitudes**: $S_1 = \{0, \pm 0.5, \pm 1, \pm 2, \pm 3\}$
- **Scale change magnitudes**: $S_2\{0.25, 0.5, 1, 2, 4\}$

B.4.4 Dataset S4 - settings

- **Description**: Contaminated Two-Component Gaussian with constant variance and changes in location
- **Length**: 100000 samples
- **Number of changepoints**: 540
- **Grace period**: G = 50
- **Detection period**: D = 50
- **Fraction of anomalies**: 0.05
- **Scale of anomalies**: 20
- **Mean change magnitudes**: $S_1 = \{0, \pm 1, \pm 2, \pm 3, \pm 4\}$
- **Distance between mode change magnitudes**: $S_3 = \{0.5, 1, 1.5\}$

Appendix C

Offline change point analysis

In this chapter, we provide extra information related to the types of change points which were encountered in the dataset.

C.1 Magnitude of mean and variance changes

Changepoint	Median absolute scale change	Median absolute location change
310	1.14	1.49
662	1.82	1.17
995	0.56	1.55
1091	3.22	1.38
1473	3.78	1.45
1626	1.85	1.15
1972	2.38	1.21
2058	1.67	1.53
2390	0.80	1.31
2642	1.10	1.43
2704	1.17	1.40
2899	1.43	1.17
3215	1.09	1.25
3754	3.15	1.28
3823	2.61	1.26
4147	3.15	2.04
4200	1.65	1.95
4681	1.73	1.77
5181	1.01	1.37
5521	2.65	1.61
5675	1.02	1.24
6057	2.20	1.45
6312	1.26	1.25
6439	1.00	1.34
6821	1.80	1.23
6891	0.70	1.44
7003	0.42	1.42
7461	1.11	1.18
7637	2.39	1.26
7673	2.80	1.30
7952	1.23	1.58
Mean	1.74	1.40
Std	0.88	0.21

Table C.1: **Absolute changes in scale and location for changepoints detected using offline detection method. Location change is normalized by the pre-change MAD and is defined as** $|\frac{Median(x_{post})-Median(x_{pre})}{MAD(x_{pre})}|$ **while the scale change is defined as** $exp(|log(\frac{MAD(x_{post})}{MAD(x_{pre})})|)$**.** x_{post} **are all samples from between the previous change point to the current change point while** x_{post} **are all samples from the current change point until the next change point.**

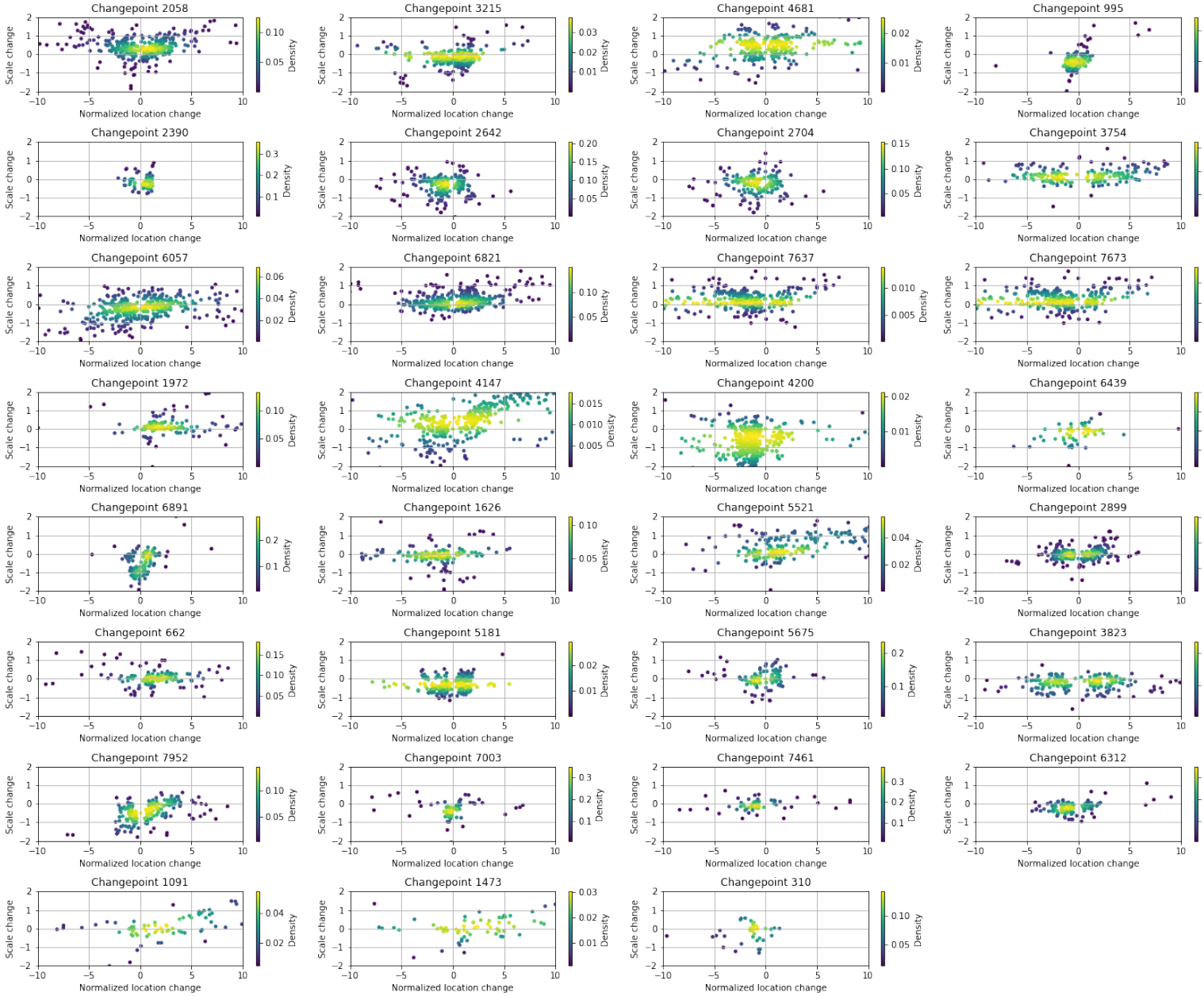


Figure C.1: **Scale change vs location change for all change points detected using offline method. Location change is normalized by the pre-change MAD and is defined as $\frac{Median(x_{post})-Median(x_{pre})}{MAD(x_{pre})}$ while the scale change is defined as $log(\frac{MAD(x_{post})}{MAD(x_{pre})})$. The colour indicates the density - lighter means there are more signals with changepoints that have similar parameters in terms of scale and location.**

Appendix D

Evaluation of current change detection method

In this chapter, we present additional figures related to the evaluation of the existing change detection method.

D.1 Average runlength current trend detection method

c	s	w	w2	ARL	Average log RL
1.5	2	25	5	47	3.20
2.0	2	25	5	248	4.91
2.5	2	25	5	1970	7.06
3.0	**2**	**25**	**5**	**14785**	**9.09**
1.5	3	25	5	418	5.48
2.0	3	25	5	4698	7.84

Table D.1: **Average run length (ARL) and average log run length of current trend detection method.**

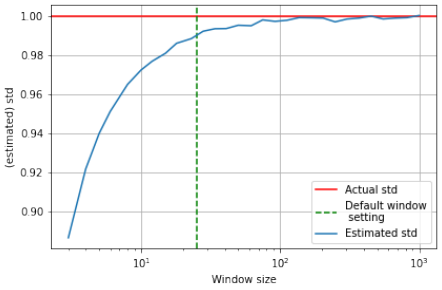
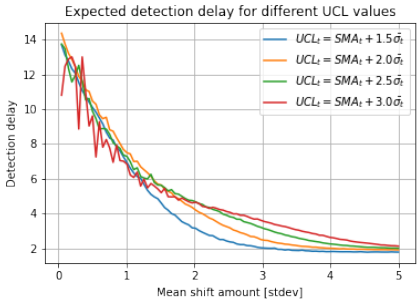
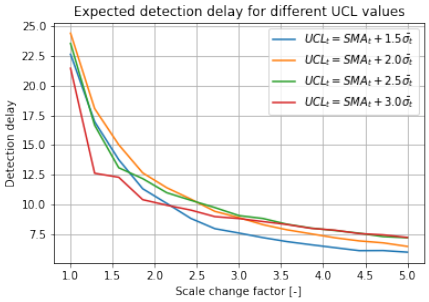


Figure D.1: **Mean of rolling standard deviation for various window sizes, with samples from an $N(0, 1)$ distribution. The mean of the rolling standard deviation underestimates the actual standard deviation by about 1.1% with a window size of 25, which causes a lower ARL than expected when compared to the analytical solution of the false positive rate.**



(a) Expected detection delay of the change detection algorithm for different mean shift magnitudes and values of c



(b) Expected detection delay of the change detection algorithm for different scale change magnitudes and values of c

Figure D.2: **Expected detection delay analysis of the current change detection algorithm.**

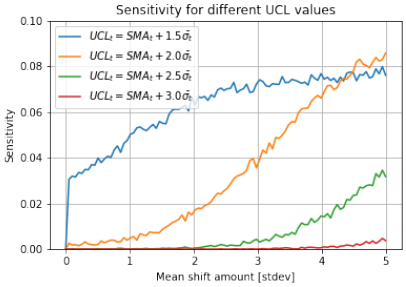


Figure D.3: **Sensitivity of the change detection algorithm for different mean shift magnitudes with an underlying multimodal distribution $N(-3,1), N(3,1)$.**

Appendix E

Results

In this chapter, we present an overview of the performance of each algorithm on the various datasets used for evaluation.

E.1 Results

Method	Train Dataset	TPR	FPR	F1	EDD
RF	Synthetic	0.63	0.07	0.74	13
RF	Real & Synthetic	0.64	0.10	0.74	13
Conv1D	Synthetic	0.51	0.04	0.66	16
KernelCPD		0.61	0.28	0.64	9
Majority vote	Synthetic	0.60	0.32	0.62	10
Majority vote	Real	0.44	0.08	0.58	13
KSWIN		0.55	0.47	0.54	17
EMW		0.46	0.26	0.53	14
Conv1D	Real	0.37	0.09	0.51	16
RF	Real	0.31	0.02	0.47	14
RF	Real (non-normal)	0.30	0.01	0.45	16
CUSUM-A		0.28	0.04	0.43	13
RobustAEWMA		0.45	0.69	0.42	10
DMW		0.34	0.58	0.35	9
EWMA		0.18	0.01	0.30	12
DispersionCUSUM		0.28	0.79	0.27	13
ADWIN		0.20	0.90	0.19	12
CurrentCP		0.14	0.34	0.19	8
NEWMA		0.11	0.08	0.19	11
CUSUM-R		0.10	0.07	0.16	17
ScanB		0.01	0.30	0.02	16

Table E.1: **Change detection algorithm: performance on combined synthetic datasets.**

		TPR			FPR			F1			EDD		
Method	Train Dataset	1	2	3	1	2	3	1	2	3	1	2	3
RF	Synthetic	0.78	0.66	0.45	0.07	0.08	0.07	0.84	0.76	0.59	11.0	12.0	16.0
RF	Real & Synthetic	0.8	0.68	0.45	0.08	0.09	0.12	0.85	0.77	0.57	11.0	12.0	15.0
Conv1D	Synthetic	0.74	0.58	0.2	0.04	0.05	0.02	0.83	0.71	0.33	13.0	14.0	19.0
KernelCPD		0.87	0.85	0.11	0.32	0.29	0.25	0.8	0.8	0.16	6.0	8.0	13.0
Majority vote	Synthetic	0.78	0.8	0.21	0.25	0.26	0.46	0.77	0.78	0.25	6.0	8.0	15.0
Majority vote	Real	0.75	0.5	0.07	0.07	0.06	0.13	0.82	0.64	0.11	10.0	11.0	17.0
KSWIN		0.72	0.54	0.38	0.43	0.44	0.55	0.67	0.55	0.39	15.0	16.0	19.0
EMW		0.77	0.5	0.11	0.15	0.37	0.28	0.8	0.53	0.16	11.0	12.0	18.0
Conv1D	Real	0.61	0.46	0.05	0.09	0.06	0.13	0.72	0.6	0.08	15.0	15.0	19.0
RF	Real	0.61	0.29	0.02	0.02	0.03	0.01	0.75	0.44	0.04	11.0	12.0	20.0
RF	Real (non-normal)	0.57	0.29	0.03	0.01	0.0	0.02	0.72	0.44	0.06	12.0	14.0	22.0
CUSUM-A		0.49	0.34	0.02	0.03	0.02	0.08	0.64	0.5	0.04	13.0	13.0	12.0
RobustAEWMA		0.72	0.54	0.1	0.5	0.92	0.66	0.65	0.44	0.11	9.0	9.0	12.0
DMW		0.51	0.22	0.29	0.19	0.06	1.5	0.61	0.34	0.2	8.0	10.0	9.0
EWMA		0.38	0.14	0.0	0.02	0.0	0.0	0.54	0.25	0.0	12.0	11.0	N/A
DispersionCUSUM		0.14	0.49	0.2	0.59	0.55	1.23	0.16	0.48	0.17	10.0	14.0	15.0
ADWIN		0.34	0.14	0.12	1.22	0.76	0.73	0.27	0.14	0.13	11.0	12.0	14.0
CurrentCP		0.2	0.19	0.02	0.46	0.41	0.15	0.24	0.24	0.04	7.0	5.0	10.0
NEWMA		0.14	0.11	0.08	0.04	0.02	0.18	0.24	0.19	0.13	11.0	8.0	15.0
CUSUM-R		0.18	0.09	0.02	0.13	0.04	0.05	0.27	0.16	0.03	17.0	14.0	19.0
ScanB		0.01	0.02	0.01	0.34	0.29	0.28	0.01	0.03	0.01	23.0	14.0	11.0

Table E.2: **Change detection algorithms: performance on individual synthetic datasets. 1=Dataset S1, 2=Dataset S3, 3=Dataset S4.**

		TPR			FPR			F1			EDD		
Method	Train Dataset	3	5	4	3	5	4	3	5	4	3	5	4
KernelCPD		0.72	0.73	0.67	0.62	1.70	1.04	0.61	0.43	0.49	9	7	8
RF	Real	0.47	0.63	0.43	0.17	0.54	0.18	0.57	0.58	0.53	13	12	12
Majority vote	Real	0.47	0.57	0.47	0.18	0.61	0.19	0.57	0.53	0.56	11	10	11
RF	Real (non-normal)	0.43	0.56	0.40	0.09	0.38	0.12	0.57	0.58	0.53	13	13	13
Conv1D	Real	0.49	0.62	0.45	0.28	0.79	0.43	0.56	0.52	0.48	14	14	15
RF	Real & Synthetic	0.42	0.59	0.42	0.10	0.36	0.23	0.55	0.61	0.51	13	12	12
EWMA		0.36	0.54	0.29	0.05	0.20	0.05	0.51	0.62	0.43	11	11	10
EMW		0.54	0.68	0.50	0.63	1.45	0.83	0.50	0.44	0.43	12	12	12
Conv1D	Synthetic	0.37	0.51	0.37	0.11	0.36	0.18	0.50	0.55	0.48	14	15	15
RF	Synthetic	0.41	0.55	0.44	0.26	0.69	0.62	0.49	0.49	0.43	12	12	12
Majority vote	Synthetic	0.50	0.56	0.49	0.79	1.78	1.15	0.43	0.34	0.37	9	8	9
CUSUM-A		0.26	0.43	0.25	0.01	0.06	0.12	0.41	0.57	0.37	12	13	13
RobustAEWMA		0.44	0.60	0.48	0.78	1.63	3.35	0.39	0.37	0.20	10	10	9
CUSUM-R		0.25	0.40	0.22	0.05	0.16	0.11	0.39	0.51	0.34	11	11	11
NEWMA		0.25	0.32	0.26	0.29	0.67	0.61	0.32	0.32	0.28	11	9	11
KSWIN		0.39	0.48	0.42	1.22	2.44	1.36	0.30	0.24	0.30	16	16	16
DMW		0.45	0.55	0.43	1.86	3.65	2.93	0.27	0.21	0.20	10	9	10
CurrentCP		0.21	0.38	0.23	0.46	0.84	1.40	0.25	0.34	0.18	2	3	3
DispersionCUSUM		0.24	0.29	0.31	1.45	2.78	2.98	0.18	0.14	0.14	13	14	15
ScanB		0.10	0.12	0.11	0.41	0.81	0.79	0.14	0.13	0.12	15	15	15
ADWIN		0.10	0.09	0.09	1.32	2.48	1.37	0.08	0.05	0.07	17	20	14

Table E.3: **Change detection algorithm: performance on real dataset. 1=Normally distributed time series in real dataset, labeled by KernelCPD. 2=Normally distributed time series in real dataset, labeled by PELT. 3=Non-normally distributed time series in real dataset, labeled by KernelCPD.**

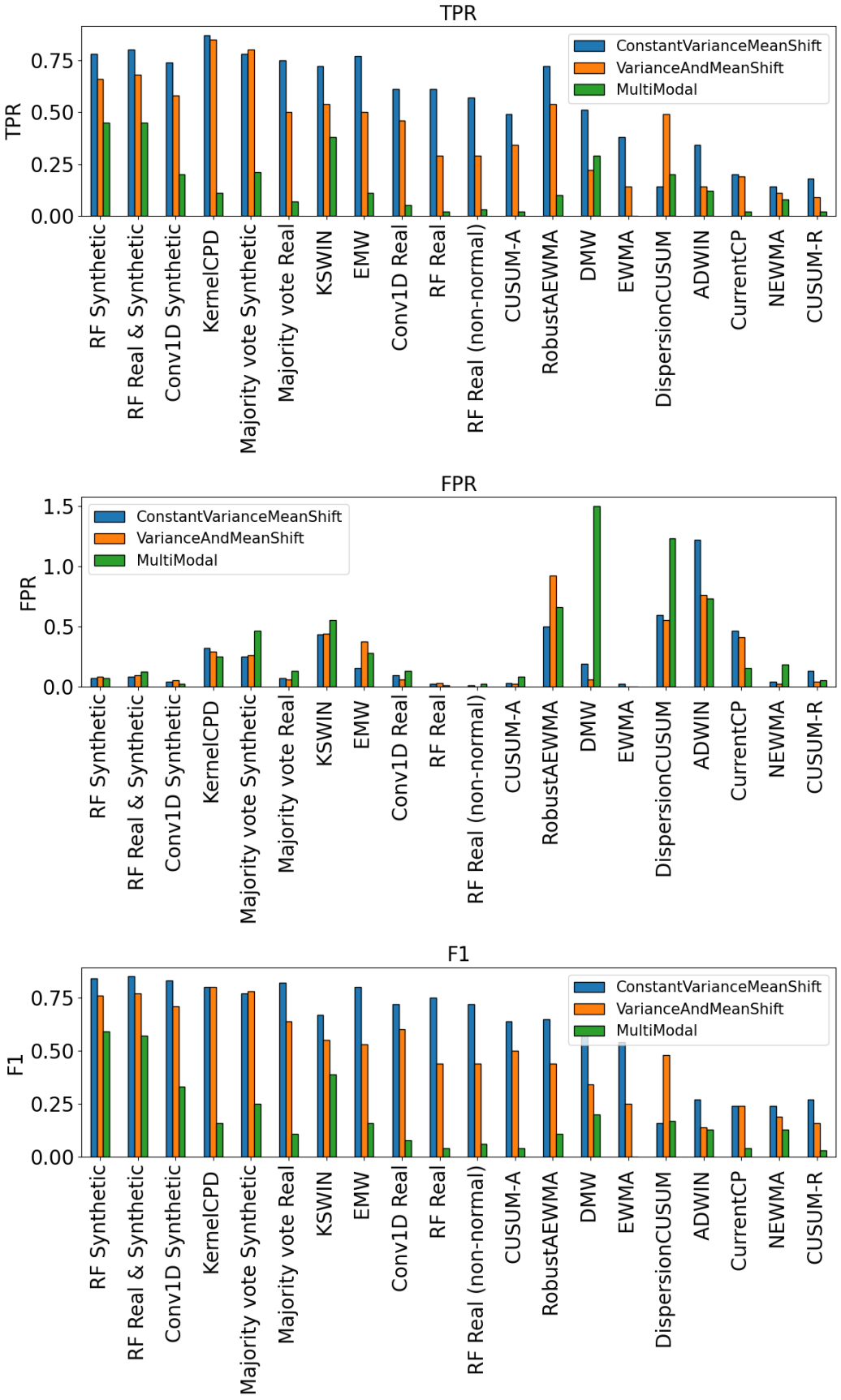


Figure E.1: **Change detection algorithms: performance on individual synthetic datasets.**

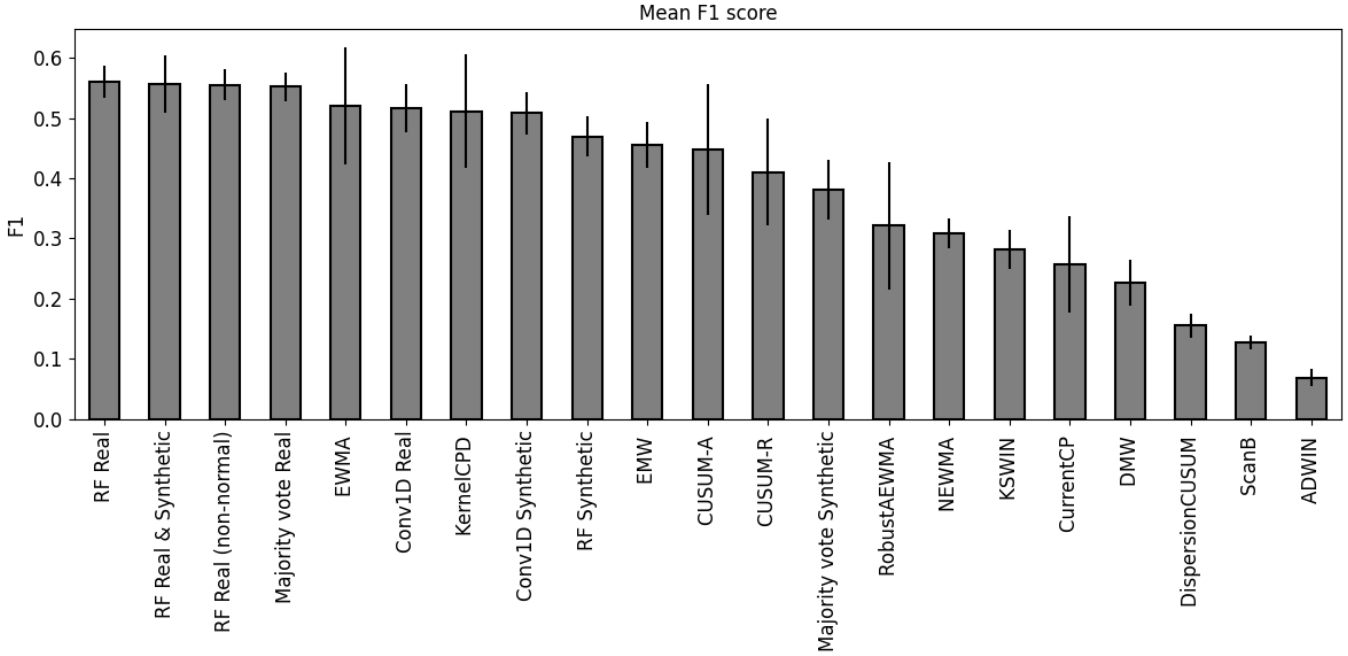


Figure E.2: **Mean F1 scores for real dataset with datasets 3,5,4. 3=Normally distributed time series in real dataset, labeled by KernelCPD. 5=Normally distributed time series in real dataset, labeled by PELT. 4=Non-normally distributed time series in real dataset, labeled by KernelCPD.**

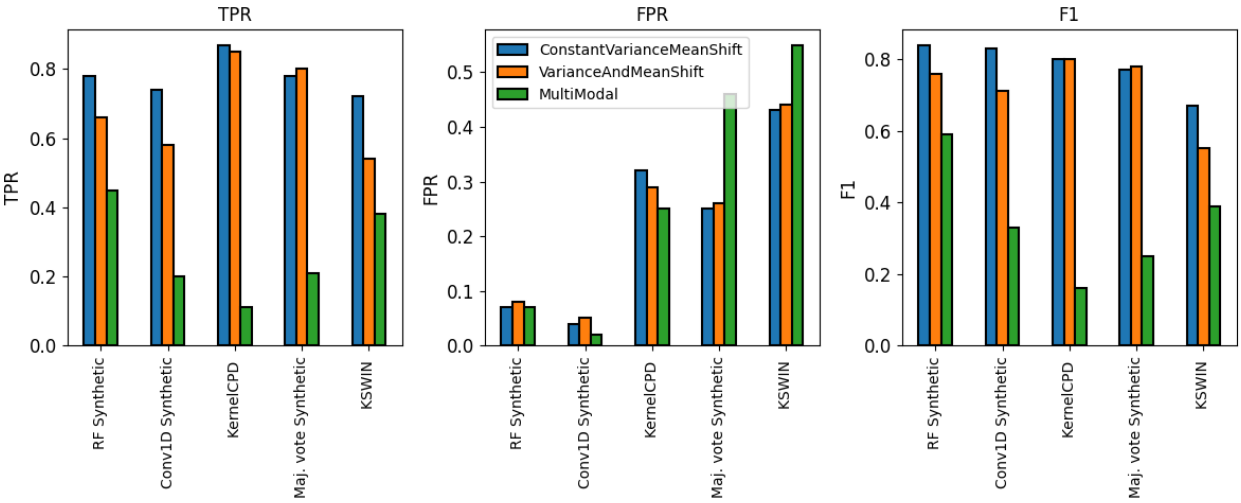


Figure E.3: **Top 5 results based on F1-scores for the separate synthetic datasets in dataset 2, including results for both ensemble and univariate algorithms.**

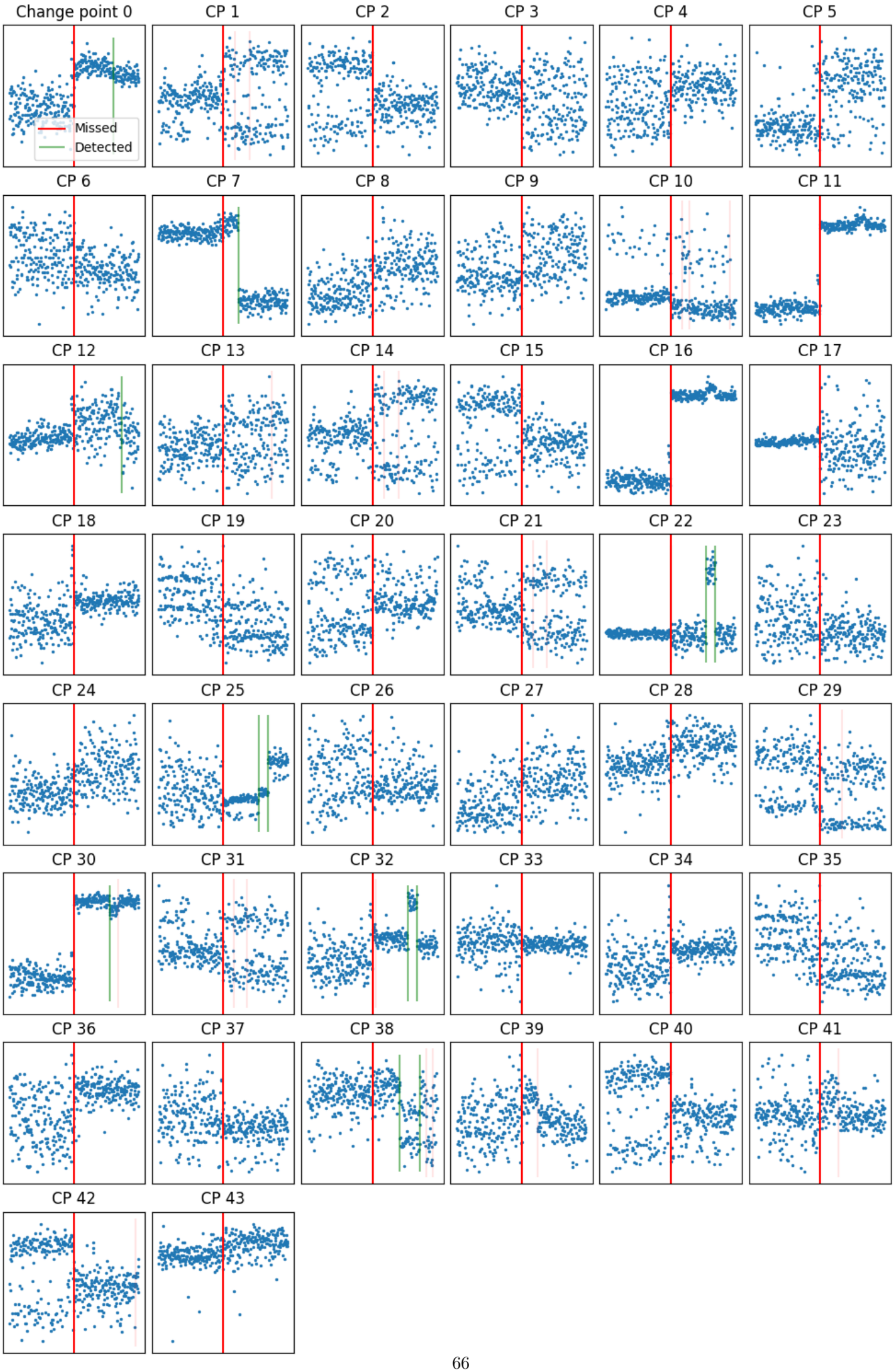


Figure E.4: **All change points with large location or scale change and sufficient samples since the previous change point which were missed by the Random Forest ensemble**

Type	TPR	FPR	F1	TP	FP	FN	EDD
Constant	0.09	0.09	0.15	2	2	20	13.50
Near constant	0.21	0.26	0.28	36	45	138	12.42
Multimodal	0.37	0.39	0.42	490	511	826	12.81
Skewed	0.31	0.46	0.35	405	603	894	13.66
Heavy tailed	0.39	0.31	0.46	144	115	224	12.53
Normal	0.47	0.41	0.50	2469	2156	2829	13.53
Other	0.33	0.51	0.36	18	28	37	14.39
Overall	0.42	0.41	0.46	3564	3460	4968	13.40

Table E.4: **Changes detected by Random Forest trained on real, normally distributed data.**

Type	TPR	FPR	F1	TP	FP	FN	EDD
Constant	0.25	0.62	0.27	6	15	18	7.17
Near Constant	0.53	2.78	0.24	92	487	83	10.35
Multimodal	0.68	2.98	0.29	900	3919	416	8.89
Skewed	0.62	2.47	0.30	803	3214	496	9.51
Heavy Tailed	0.64	1.63	0.39	237	600	131	9.23
Normal	0.69	2.08	0.37	3647	11025	1651	9.82
Other	0.64	2.78	0.29	35	153	20	10.31
Overall	0.67	2.27	0.34	5720	19413	2815	9.61

Table E.5: **Changes detect by KernelCPD.**

Type	TPR	FPR	F1	TP	FP	FN	EDD
Constant	0.10	608.90	0.00	7	40796	60	4.71
Near Constant	0.21	1.37	0.16	36	239	138	10.69
Multimodal	0.28	0.13	0.40	372	172	944	11.51
Skewed	0.19	0.13	0.28	244	173	1055	11.75
Heavy Tailed	0.27	0.10	0.40	100	35	268	11.72
Normal	0.29	0.12	0.41	1511	642	3787	11.80
Other	0.18	0.07	0.29	10	4	45	11.40
Overall	0.27	4.90	0.09	2280	42061	6297	11.71

Table E.6: **Changes detect by EWMA.**

Type	TPR	FPR	F1	TP	FP	FN	EDD
Constant	0.17	1.34	0.14	7	55	34	7.43
Near Constant	0.38	2.80	0.18	66	490	109	11.14
Multimodal	0.44	0.68	0.41	574	895	742	11.58
Skewed	0.42	0.69	0.40	540	894	759	12.98
Heavy Tailed	0.46	0.45	0.48	168	167	200	11.71
Normal	0.42	0.46	0.45	2229	2462	3069	12.78
Other	0.36	0.51	0.39	20	28	35	14.50
Overall	0.42	0.58	0.42	3604	4991	4948	12.54

Table E.7: **Changes detect by Majority voting trained on real data.**