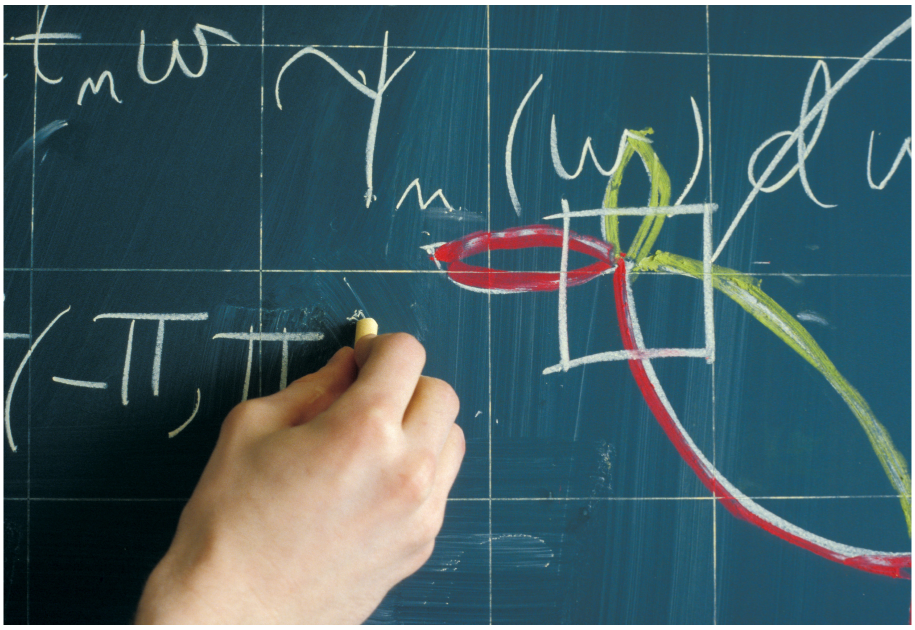
Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

Amirhossein Kazemi

# A Semi-Supervised Approach to the Application of Sensor-based Change-Point Detection for Failure Prediction in Industrial Instruments

Master's thesis in Applied Physics and Mathematics
Supervisor: Prof. Mette Langaas and Dr. Martin Høy
June 2019

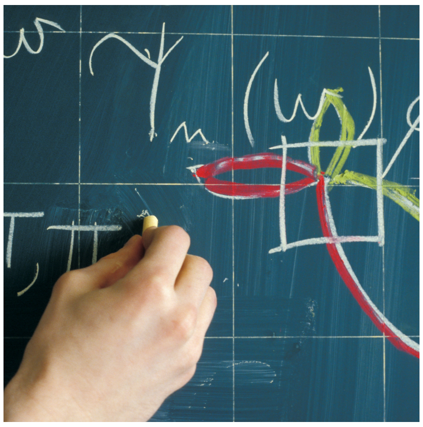


NTNU
Norwegian University of Science and Technology

DNV·GL

Amirhossein Kazemi

# A Semi-Supervised Approach to the Application of Sensor-based Change-Point Detection for Failure Prediction in Industrial Instruments



Master's thesis in Applied Physics and Mathematics
Supervisor: Prof. Mette Langaas and Dr. Martin Høy
June 2019

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

NTNU
Norwegian University of
Science and Technology

# Summary

In this work we analyze the possibility of applying statistical methods of change-point detection for predicting failures in industrial instruments. In addition, we provide an overview over various methods of anomaly detection that can be applied for this purpose. The solution proposed here is model-driven. We propose to use methods of change-point detection on historical sensor data from the operation of a machine to identify patterns that have led to unsafe conditions under which the machine has gone through an unplanned shut-down in the past. The solution proposed in this work has a semi-supervised nature. We assume that there exists a change in the distribution of the data in a time period of known length prior to failures. It is assumed that the time of the change is unknown. The data points prior to and after this change are labeled as normal and non-normal, respectively. We develop methods to find signals in the data the distributions of which are less affected by the normal variations in the data than by the non-normal changes that cause failures. Based on these signals, several univariate predictive models are developed. The parameters of the predictive models are determined with an ad hoc approach, using the semi-supervised labeling of the data points. The methods presented in this work are inspired by and applied to a data set provided by the Norwegian DNV GL. The data comes from the gas compressors and the gas turbines used by a European gas transmission system (GTS) operator. In this work, as an original work by the author, the theories of the vector autoregressive model, the offline change-point detection method, the principal and the Box-Tiao analyses, the Mann-Whitney U test, the signals discrepancy, leverage and influence, the CUSUM method and statistical performance metrics have been combined to create end-to-end solutions to the problem of sensor-based failure prediction in industrial instruments. The main contribution of the author is combining all the methods presented in this work into four possible end-to-end solutions and testing and evaluating the solutions on simulated and real-world data sets. Here we have underpinned the existence of predictive information in our real-world data and a potential in the presented methods for detecting the difference between normal and non-normal behaviour. Achievable improvements of the analyses performed are proposed in this work. Given the current results, we conclude that applying the methods presented here for failure prediction in our business case might be economically profitable if the economic value of a true positive prediction is much larger than the economic loss of a false positive prediction. We recommend further work on the subject by conducting a similar multivariate analysis.

# Preface

This dissertation is an original work of the author Amirhossein Kazemi, performed as a master's thesis at the Norwegian University of Science and Technology (NTNU), under the supervision of Prof. Mette Langaas, from the Department of Mathematical Sciences at this university and external supervision of Dr. Martin Høy from DNV GL. This work is submitted in fulfillment of the requirements of TMA4900 Industrial Mathematics, Master's Thesis. The methods presented in this work are inspired by and tested on a real-world data set. The business case and the real-world data set were provided to the author by the Norwegian DNV GL. Additional external support was provided by Dr. Huiming Zeng, Dr. Abdillah Suyuthi and Jørgen Christian Kadal from this company. Ian Peter Mather and Edward Paul Goulbourne from DNV GL in the UK provided useful insight into the business case.

I would like to take this opportunity to express my deepest gratitude to my main supervisor, Mette. Through many months of work, her unconditional support made the creation of this dissertation possible and the journey to complete it enjoyable for me. From replying to my e-mails within minutes after they were sent to writing encouraging notes in the margins of the drafts of this work, she spared no pains to support me through the whole journey. I would also like to express my appreciation to my external supervisor, Martin, whose ideas inspired much of the analysis performed in this work. In conclusion, I wish to convey my sincere thanks to Jørgen for his trust and support through the last year of my studies, including prior to and during the work on this dissertation.

# Table of Contents

# List of Tables

# List of Figures

# Abbreviations

| | | |
|---|---|---|
| AIC | = | Akaike Information Criterion |
| AR | = | Autoregressive |
| BIC | = | Bayesian Information Criterion |
| BTA | = | Box-Tiao Analysis |
| CDF | = | Cumulative Distribution Function |
| CLT | = | Central Limit Theorem |
| CROPS | = | Change-points for a Range of PenaltieS |
| CUSUM | = | Cumulative Sum |
| DNV GL | = | Det Norske Veritas & Germanischer Lloyd |
| EWMA | = | Exponentially Weighted Moving Average |
| FN | = | False Negative |
| FP | = | False Positive |
| GE | = | General Electric |
| GLS | = | Generalized Least Squares |
| GTS | = | Gas Transmission System |
| LCL | = | Lower Control Limit |
| LOCF | = | Last Observation Carried Forward |
| NA | = | Not Available |
| NTNU | = | Norwegian University of Science and Technology |
| OLS | = | Ordinary Least Squares |
| PCA | = | Principal Component Analysis |
| PCC | = | Principal Component Classifier |
| PELT | = | Pruned Exact Linear Time |
| RPM | = | Revolutions Per Minute |
| SVD | = | Singular Value Decomposition |
| TN | = | True Negative |
| TP | = | True Positive |
| UCL | = | Upper Control Limit |
| VAR | = | Vector Autoregressive |

Chapter 1

# Introduction

This study is motivated by the possibility of applying statistical methods of change-point detection in order to detect anomalies that cause a failure in industrial instruments. Machines in different industries often fail because of some undesired behaviour in their operation. Examples of such instruments are jet motors, wind turbines, gas compressors, etc. A failure in such instruments can cause a substantial economic cost and even loss of life. Therefore the operation is monitored using many sensors inside the machine, measuring different physical quantities such as temperature, pressure, etc. A control system is used to monitor the sensor values. In general the control system is in charge of the operation. The rules under which it operates are made by the unit producer. The control system normally tries to operate the machine in the most optimal and safe way for achieving the desired output, set by the engineers. However, often it performs a fast controlled shut-down whenever there is high risk of a costly failure in order to avoid damage to the machine or its surroundings. This is not optimal for the user of the machine because an unplanned shut-down can also cause economic loss. In this work we try to give a framework for applying statistical methods to predict the unsafe conditions before they occur, giving the engineers enough time to adjust the operation in order to avoid an unplanned shut-down performed by the control system.

## 1.1 Model-Driven Nature of the Solution

The solution proposed here is model-driven. We provide an end-to-end solution from historical data preparation, through identifying patterns that have led to unsafe conditions under which the machine has either failed or gone through an unplanned shut-down in the past, to a predictive model that can be used to predict and prevent such events in the future. Methods of change-point detection are applied in different stages of the solution. We will also discuss the challenges in such problem settings and propose solutions to overcome the challenges.

One challenge for a model-driven model is that most industrial machines do not actually fail very often, and this provides little data for a model-driven approach. However, this can be used as an advantage if one assumes that most of the time the machine is operating under safe circumstances. This is the so-called normal operation mode. The patterns that cause an unplanned shut-down are the anomalies in the data, meaning that they do not occur too often. For instance, if the data is clustered, anomalies are the observations that do not belong to the largest clusters. It can be assumed that failures are caused by anomalies. Therefore anomaly detection methods can be developed to predict occurrence of unsafe conditions in a machine before they occur. However, in our analysis the normal operation mode as well as the nature of the anomalies that cause a failure are unknown. In addition, it is not known how long it takes from the time a pattern leading to a failure starts until the control system shuts down the machine. Therefore it is assumed here that there exists a change-point in the data separating the normal operation mode from the anomalies that lead to failure. Then a predictive model can be developed to detect these changes in the distribution of the data and thereafter predict the failure.

## 1.2 Business Case

Methods presented in this work are applied to a data set provided by a European gas distribution network owner. This company is the owner and the operator of a gas transmission system (GTS) in a European country. A gas transmission system is a network of gas pipelines that supplies natural gas to different power stations, industrial users and gas distribution companies. Gas distribution companies then distribute the gas further, supplying smaller commercial and domestic natural gas users. It is desired to keep the gas pressure in the GTS between some lower and upper limit. For this the GTS operator uses several units of gas compressors stationed at different location points along the pipelines. The compressors are turned on and off on demand in order to regulate the gas pressure in the GTS. Gas turbines of the model LM2500 are the most common type of turbines that the GTS operator uses to produce power for the gas compressors installed along the GTS. The turbines are produced by General Electric (GE). One can see an illustration of this machine in Fig. 1.1. The data set analyzed in this work contains sensor-based data from the gas compressors and the gas turbines used by the GTS operator. An unplanned shut-down of one of the systems can be due to a failure either in the gas compressor or the gas turbine used to power it.

The data set provided to us is divided into different *running groups*. A running group refers to the data recorded between the time one compressor is *turned on* until the consecutive *shut-down*. A compressor is considered turned on as soon as its recorded value of speed goes above 500 revolutions per minute (RPM). This value is selected in consultation with the field experts. A compressor is considered shut down as soon as its recorded value of speed goes below

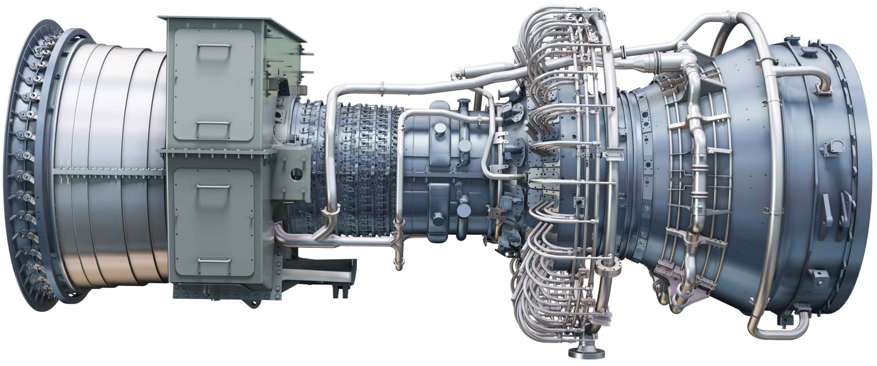


**Figure 1.1:** An illustration of the gas turbine GE LM2500 provided by GE Power (2018)

500 RPM.

An unplanned shut-down due to a failure in the system is referred to as a *trip*. The running groups are categorized based on their type of shut-down. The possible categories are *normal stop*, *running trip* and *starting trip*. This information is manually logged by engineers at the GTS operator company, and provided to us in a log file. We ignore the running groups ending in a starting trip and refer to the running groups ending in a normal stop and a running trip as a *normal group* and a *tripping group*, respectively.

Under mild conditions, an unplanned shut-down of one compressor does not have a substantial impact on the operation of the pipelines. This is because the load can easily be picked up by the other compressors since a large degree of redundancy is built into the operations. However, there can be a black out of gas in some places if several compressors undergo an unplanned shut-down nearly at the same time. At times, local maximum capacity is needed for maintaining enough pressure in the pipelines. In this case an unplanned shut-down of even one critical compressor can cause a black out in some places. Over time this can have substantial economic consequences for natural gas dependent industries and commercial and domestic users. The GTS operator company receives a fine from the state when a black out occurs. Therefore the company is interested in solving the issue.

The engineers at the GTS operator company already have a good understanding of possible issues in the system that can cause an unplanned shut-down in a compressor. We propose that it is possible to use model-driven solutions in order to predict failures, giving the engineers enough time to act on the provided information to avoid unplanned shut-downs in the future.

## 1.3 Work Overview

In Chapter 2 the concepts of anomaly detection and change-point detection are introduced along with the underlying assumptions of the different approaches. In addition, we present some of the literature on the subject. We also present arguments for why it is beneficial to use change-point detection methods in the way presented in this work for failure prediction in our business case. In Chapter 3, the mathematical theory needed for our proposed framework for sensor-based failure prediction is presented. The reader is taken step by step through the solution. In Chapter 4, we present different approaches to test the methods presented in Chapter 3. In this chapter a simulation study is designed to evaluate the framework developed in Chapter 3. In Chapter 4, we also describe the business case, the real-world data set and the simulated data sets in details and explain how the methods presented in Chapter 3 can be applied to these data sets. The details of the implementations of the methods is also discussed in this chapter. In Chapter 5, we present and analyze the results of the experiments designed in Chapter 4 and evaluate the different methods presented in Chapter 3. The conclusions are discussed in Chapter 6, where we also propose ways to improve the methods to obtain better performance by the predictive model.

# Chapter 2

# Concepts

Here the concepts of anomaly and change-point detection are introduced. An anomaly, as defined by Steinwart et al. (2005), is non-concentrated data. For a time series, a change-point, as defined in Section 3.3, is the time when a change occurs in the data generating process of the series. Anomaly detection and change-point detection are two different approaches often applied for the common purpose of failure prediction. However, they can be combined in order to create a new approach with few underlying assumptions, as explained in Subsection 2.5 and Kuncheva and Faithfull (2014). In this work, sensor-based data is of special interest. Sensor-based data consists of repeated measurements of some physical quantities recorded by the sensors in a system.

## 2.1 Nature of the Anomalies

In Chandola et al. (2009), data identified as an anomaly is categorized into three groups:

- Point anomaly: These are individual data points that can be considered as anomalies with respect to the rest of the data. A large transaction in the banking industry can be an example of this kind of anomaly. In a sensor-based setting this would mean an non-normal value recorded by one sensor.

- Contextual anomaly: These are data instances that are considered non-normal in specific contexts while normal in other contexts. For instance a large amount of rainfall would be non-normal in a desert while the same amount would be normal in a rain forest. This category is sometimes referred to as conditional anomaly. In a sensor-based setting, this happens when the value recorded by a sensor is non-normal given the value recorded by some other sensors at the same time-point.

- Collective anomaly: In this case several related data points make up an anomaly while each data point by itself might not be non-normal. In a sensor-based setting, this can be for instance the case if a peak and a valley happen in an non-normal order in the time series.

In order to detect point anomalies, one approach is to build a model which predicts the state of the system in the present, given the state of the system in the past. Then the prediction can be compared to the observed state of the system in the present, and one can calculate the residuals of the model. Then one can detect outliers using the residuals. This approach is discussed in detail in Steinwart et al. (2005), in which the authors present a classication approach to anomaly detection based on a framework of density-level detection. Following this approach, one should build a model that explains the normal variations in the data to a sufficient degree. In our case, the data can be generated under different normal operation modes, and there does not exist any meta-data about when the operation mode changes from one to another. Therefore it is specially challenging to construct a model that explains the normal variations in the data to a sufficient degree.

For contextual and collective anomaly detection, clustering and $K$ nearest neighbour techniques can be used. The underlying assumption for clustering is that normal data points will belong to large clusters while anomalies will belong to smaller ones. This is somehow similar to the definition of an anomaly in Steinwart et al. (2005). Following this approach, one assumes that there exists a clear cluster structure in the data set. However, in an initial analysis of our data set, conducted by DNV GL, no clear cluster structure could be found in the data. The main assumption behind $K$ nearest neighbour techniques is that new anomalies will be closer to known anomalies than to normal data points. In our case, since the number of normal data points is much larger than the number of anomalies and there exists no clear cluster structure in the data set, a new data point will almost always be surrounded by many other normal data points. Therefore a $K$ nearest neighbour approach may not be suitable for anomaly detection in our case. The clustering and the $K$ nearest neighbour techniques are presented with their applications in Choudhary (2017) and Perera (2015).

In this work, we will develop methods that can potentially detect all of the three types of anomalies presented in this section. Our approach is described in Section 2.5 and presented in Chapter 3.

## 2.2 Nature of the Methods

Reviewing the literature on anomaly detection, Chandola et al. (2009), the methods can be divided into three main categories, namely supervised, semi-supervised and unsupervised methods. In the following training data refers to the data used to fit the model used for anomaly detection.

- Supervised: In this approach the training data points are all labeled as normal or non-normal. Classification methods are used to predict if a previously unseen data point belongs to the normal class or the non-normal one. $K$ nearest neighbour techniques introduced in Section 2.1 can for instance be implemented in a supervised manner, if both normal data points and anomalies are labeled.

- Semi-supervised: In this approach only labeled training data for normal states is available. Every observation far away from the identified normal states should be predicted to be an anomaly. $K$ Nearest neighbour techniques can be implemented also in a semi-supervised manner, such that only the normal data points are labeled and a new data point with too few normal data points in their neighbourhood is predicted to be an anomaly.
- Unsupervised: In this approach the training data is not labeled at all. The very fundamental assumption here is that normal behaviour is more common than anomalies in the data. Therefore one is trying to identify data points that are different from the common behaviour in the data set in some sense. Clustering is an example of an unsupervised approach.

Supervised anomaly detection with labeled data can be as simple as a classification problem. For time series data, Kadous (2011) provides a list of appropriate classification methods with references. In our data set, the data points are not initially labeled as belonging to a normal or non-normal state. In order to take a semi-supervised or unsupervised approach there needs to exist clear borders between the normal and the non-normal states. For the clustering and the $K$ nearest neighbour techniques, this is explained in Section 2.1. In our data set no clear borders exist between the normal and the non-normal states. We apply a semi-supervised method of change-point detection, where one assumes that anomalies happen because of a change in the data generating process. This approach is described further in Section 2.5.

## 2.3 Nature of the Data

The nature of the input data is according to Chandola et al. (2009) an important factor for choosing a suitable technique. Data can be categorical or continuous. The number of features is another important aspect of the input data. The data can be:

- 1-dimensional: Consisting of only one feature measured in each data point.
- Multidimensional: Consisting of multiple features measured in each data point.

Here time series data is of special interest. In the case of multidimensional time series data, at any given time, values for multiple features are measured. This means the data consists of multiple so-called streams, that may depend on each other. Non-normal behaviour might happen in only one stream, or the behaviour of one stream might be non-normal given the values of some other streams. This should be taken into account when the method is being developed.

## 2.4 Online vs. Offline Detection

This categorization applies only to time series data, and depends on the ability of the model to detect anomalies or change-points based only on the past data.

- Online Detection: These methods only need data from the past.
- Offline Detection: These methods require data from both the past and the future of any given data point.

Offline detection methods can be used for descriptive analysis while online detection methods can be used to make predictive models used in real time.

## 2.5 Our Semi-Supervised Approach

Change-point detection methods detect changes in the distribution of the data. The definition of a change-point is provided in Section 3.3, where this method is discussed in detail. Aminikhanghahi and Cook (2017), in their survey article, enumerate, categorize, and compare many of the methods for change-point detection in time series. We apply change-point detection for failure prediction in this work. The method presented in Section 3.3 is an offline change-point detection method while the method presented in Subsection 3.5.1 is an online change-point detection method. Applying change-point detection for failure prediction falls under the category of unsupervised anomaly detection if the parameters of the model are determined independently of the historical data. This is the case for the offline change-point detection method presented in Section 3.3. In the online change-point detection method presented in Subsection 3.5.1, we determine the parameters of the model with an ad hoc approach using the historical data. In this approach, the normal data is labeled and the change from the normal state to the non-normal state is assumed to occur in a predetermined time-window. This falls under the category of semi-supervised anomaly detection. A change-point detection method should be used for failure prediction in the cases where it is hypothesized that the system is usually in a normal state and a change in the distribution of the data will cause a failure.

In our business case, there exist several *normal operation modes* for the gas compressors, described in Section 1.2. The operation mode can be changed without causing a failure in the system. A change in the operation mode causes a change in the distribution of the data. We refer to these changes as *normal changes*. This might be when some other types of changes in the distribution of the data in fact cause failures. These are referred to as *non-normal changes*. We assume that normal changes occur more often than non-normal changes. If an unsupervised change-point detection model is applied directly to the sensor measurements, it may raise an alarm when a normal change occurs as well as when a non-normal change occurs. In this case it might not be possible to determine the model parameters using the historical data to avoid this issue because the amplitude of the normal changes might be as large as or larger than the amplitude of the non-normal changes. Therefore in a change-point detection model applied for failure prediction the number of *false positive predictions* may be too high. This may be the case for other unsupervised anomaly detection models as well. In our case, a false positive prediction occurs when the model predicts a failure in the system when no failures are in fact going to happen. Therefore for the purpose of sensor-based failure prediction in business cases like ours, we propose not to develop an unsupervised change-point detection or other unsupervised anomaly detection models based on the sensor measurements directly.

In sensor-based failure prediction, applying supervised anomaly detection is also challenging. This is because there does not exist a trivial labeling of the data points. One does not know which data points are normal and which data points are non-normal. One could argue that the sensor measurements that are recorded shortly before a failure in the system should be labeled as non-normal, and the rest of the data set as normal. The question then becomes how close to a failure should measurements be to be labeled as non-normal.

In this work we present a semi-supervised approach to overcome the challenges presented in this section. We develop a method to find signals in the data the distributions of which are less affected by normal changes than by non-normal changes. The larger the difference is between the effect of non-normal and normal changes on the distribution of a signal, the more predictive information is contained in the signal. We propose to find the signal with *maximum predictive information* among some alternatives. This signal is referred to as the *signal of interest*. Finally, we present an online predictive model that can detect changes in the distribution of the signal of interest, and therefore predict failures. We also present some statistical metrics used to evaluate the predictive model.

For the selection of the signal of interest and the construction of the predictive model, a semi-supervised labeling of the data points is performed. In this approach, we assume that there exists a change in the distribution of the signal of interest in a time period of known length prior to failures. We do not assume that all the data points in this time period are non-normal, as one would have assumed in a fully supervised approach. Assuming the existence of a change in the distribution of the signal of interest in this time period only implies that the data points appearing after the change has occurred are non-normal. It is assumed that the time of the change is unknown.

Following our approach both point, contextual and collective anomalies can potentially be detected. We propose to use a signal of interest that is a linear combination of the dimensions in the data set. Therefore both point anomalies in each individual dimension and contextual anomalies in a group of dimensions should appear in the signal of interest. In the predictive model, we propose to use a test statistic that to some degree is affected by the observations of the past states of the system as well as the observation of the present state of the system at any given time. This approach enables the model to also detect some types of collective anomalies.

Chapter 3

# Theory

In this chapter, we will present the mathematical theory needed for our proposed framework that can be used for sensor-based failure prediction in industrial settings like ours. In Figure 3.1, one can see the analysis pipeline for creating a predictive model using sensor-based data.

The Cumulative Sum (CUSUM) is our proposed method for the final predictive model. This method is described in Section 3.5. Some statistical metrics for evaluation of this model are presented in Subsection 3.5.2. There exists variations of CUSUM for multidimensional data. We will however only use this method on a 1-dimensional signal.

We propose a semi-supervised Principal Component Analysis (PCA) or Box-Tiao Analysis (BTA) method to find directions in the data with maximum predictive information. These methods and their semi-supervised nature are described in Section 3.4. In order to perform a PCA or a BTA, one needs to estimate the covariance matrix of the data. The estimation of the covariance matrix is specially challenging in our business case since there does not exist one continuous stream of data. Instead, the data comes from many disconnected time intervals since the gas turbines are frequently turned on and off. A method for covariance matrix estimation in problem settings like our business case is discussed in Subsection 3.4.4. In order to perform a BTA, one also needs a model describing the data generation process, where the value measured by a sensor at any time point, $t$, is modelled as a function of the values recorded by all the sensors in the time prior to $t$ and some error term. For this purpose, we propose a linear function, called a Vector Autoregressive (VAR) model, described in Section 3.2.

After transforming the data, we pick a 1-dimensional signal which is used for predictive modelling. One is however not limited to picking only one dimension here and may select as many dimensions as one's available computational and implementation capabilities allow. We refer to this 1-dimensional signal as the *signal of interest* and the goal is to select the signal with maximum pre-

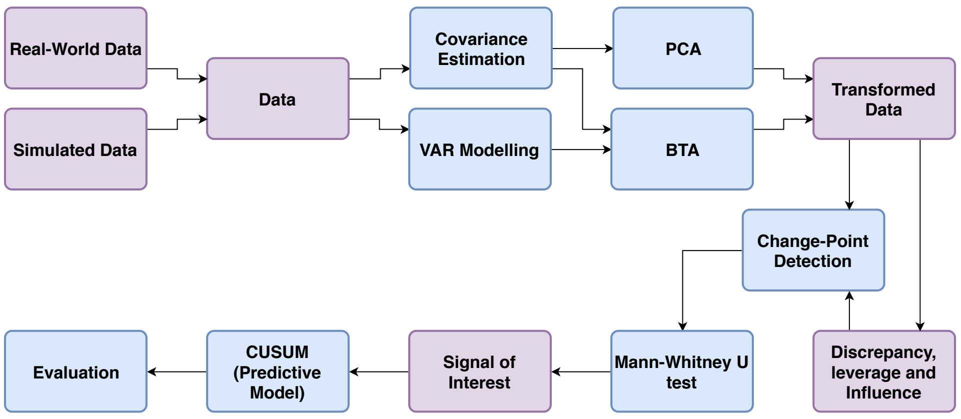


**Figure 3.1:** The analysis pipeline: An overview over the mathematical methods provided in this work and how they are used in combination with each other to make a failure prediction model using historical data. The boxes for mathematical methods are colored blue while the boxes for different types of data are colored purple. All the methods and types of data are presented in sufficient detail in this chapter.

dictive information. For the selection of the signal of interest, a Change-Point Detection framework is also needed. This is presented in Section 3.3. VAR modelling presented in Section 3.2 is a tool also used in our change-point detection framework. The use of this tool in change-point detection is described in Subsection 3.3.5. Using a Mann-Whitney U test as described in Subsection 3.4.5, one can choose the signal with the lowest $p$-value as the signal of interest among some alternatives. Our proposed alternatives are the principal components, the Box-Tiao components or one of the signals Discrepancy, Leverage or Influence computed after the PCA or BTA transformation. The construction of the discrepancy, leverage and influence is also described in Subsection 3.4.5.

In Section 3.1, some notation is introduced. This notation will be used throughout this chapter. However, some extra notation may be needed in the individual sections. This extra notation will be introduced in a subsection in the beginning of the corresponding section, and may be referred to also in later sections of this chapter.

## 3.1 Notation

- **Bold** capital letters are used for naming matrices. Non-bold capital letters are used for naming vector variables while the corresponding non-capital letter is used to refer to an observation of the same variable.

- For any set of indices $\{t_1, ..., t_N\}$, the dummy indices $\{1, ..., N\}$ can be used without loss of generality.

- For a set of indices, the corresponding observations of some multidimensional

time series
$x = \{x_t\}_{t=1}^{N}$ is called a signal.

- For a signal $x$, $x_{i...j}$ is used to refer to the set of observations of the variable $X_t$ at time point $i$ to $j$, namely $x_{i...j} = (x_i, x_{i+1}, ..., x_j)$.
- For a matrix $\mathbf{A}$, $\mathbf{A}_{i,j}$ is used to refer to the element in the $i$-th row and the $j$-th column of $\mathbf{A}$.
- For an $m \times n$ matrix, $\mathbf{A}$, the $n \times m$ matrix, $\mathbf{A}^T$, is the transposed of $\mathbf{A}$.
- For a random vector variable $X$, with a multivariate Gaussian distribution with mean $\mu_X$ and covariance matrix $\Sigma_X$), write $X \sim \mathcal{N}(\mu_X, \Sigma_X)$.

## 3.2 Vector Autoregressive (VAR) Models

In time series analysis the Vector Autoregressive (VAR) model is an extension of the well-known Autoregressive (AR) model from univariate to multivariate analysis. In a VAR model of order $p$, it is assumed that $X_t | X_{t-1}, X_{t-2}, ..., X_{t-p}$ has a multivariate Gaussian distribution with mean $\mu + \mathbf{A}_1 X_{t-1} + \mathbf{A}_2 X_{t-2} + ... + \mathbf{A}_p X_{t-p}$ and some constant covariance matrix $\mathbf{\Sigma} = \mathbf{\Sigma}_{X_t | X_{t-1}, X_{t-2}, ..., X_{t-p}}$. The $X_t$'s and $\mu$ are vectors of dimension $r$ while the $\mathbf{A}_j$'s and $\mathbf{\Sigma}$ are matrices of dimension $r \times r$. This means that for $t = p+1, ..., N$, $X_t$ can be described by the following model:

$$X_t = \mu + \sum_{j=1}^{p} \mathbf{A}_j X_{t-j} + U_t, \tag{3.1}$$

where $U_t$'s are independent and identically distributed *innovation terms* following a multivariate Gaussian distribution with mean 0 and covariance matrix $\mathbf{\Sigma}$. Tsay (2013, Chapter 2) discusses how one can derive Granger causality between the different dimensions of $X_t$ (sensors) based on $\mathbf{A}_j$'s. This analysis however is out of the scope of this work. Our main focus is on estimating $\mathbf{\Sigma}$, which is later used for dimensionality reduction in Subsection 3.4.2.

### 3.2.1 Notation

- For any positive integer, $r$, $\mathbf{I}_r$ is defined as the $r \times r$ identity matrix. In matrix equations, $\mathbf{I}$ might also be used to refer to the identity matrix. In this case the dimension $r$ is chosen appropriately with respect to the other variables in the equation.
- For an $m \times n$ matrix $\mathbf{A}$, the columns of $\mathbf{A}$ are referred to as $\mathbf{A}_{:,1}, \mathbf{A}_{:,2}, ..., \mathbf{A}_{:,n}$ and the rows of $\mathbf{A}$ are referred to as $\mathbf{A}_{1,:}, \mathbf{A}_{2,:}, ..., \mathbf{A}_{m,:}$.
- Let $\lfloor . \rfloor$ represent the floor function. Then for an $m \times n$ matrix, $\mathbf{A}$, and a $p \times q$ matrix, $\mathbf{B}$, the Kronecker product of the two is defined as the $mp \times nq$

matrix, $\mathbf{C} = \mathbf{A} \otimes \mathbf{B}$, where for $i = 1, 2, ..., mp$ and $j = 1, 2, ..., nq$, one has: $\mathbf{C}_{i,j} = \mathbf{A}_{\lfloor \frac{i-1}{p} \rfloor +1, \lfloor \frac{j-1}{q} \rfloor +1} \mathbf{B}_{i - \lfloor \frac{i-1}{p} \rfloor p, j - \lfloor \frac{j-1}{q} \rfloor q}$.

- For an $m \times n$ matrix $\mathbf{A}$, the *vec*-operator is defined as $vec(\mathbf{A}) = [\mathbf{A}_{:,1}^T, \mathbf{A}_{:,2}^T, ..., \mathbf{A}_{:,n}^T]^T$, and is used to stack the columns of $\mathbf{A}$ into one single $mn \times 1$ column.

### 3.2.2 Stability and Stationarity

Lütkepohl (2005, Chapter 2) defines stability in a time series following the model presented in (3.1) as follows:

#### Stability

$X_t$ is *stable* if for any complex number $z$, with absolute value $|z| \leq 1$:

$$\det(\mathbf{I} - \sum_{j=1}^{p} z^j \mathbf{A}_j) \neq 0. \tag{3.2}$$

For a VAR process of order 1, the condition above is the same as the matrix $\mathbf{A}_1$ having eigenvalues with their absolute value being all less than 1.

In the same chapter, Lütkepohl (2005) defines stationarity in the following way:

#### Stationarity

A stochastic process is *stationary* if its first and second moments are time-invariant. This means a time series $X_t$, is stationary if for all time points $t$, and time lags $\Delta t$, one has:

$$E(X_t) = \nu$$
$$\text{and}$$
$$E[(X_t - \nu)(X_{t-\Delta t} - \nu)^T] = \mathbf{\Gamma}(\Delta t) = (\mathbf{\Gamma}(-\Delta t))^T,$$

where $E(.)$ represents the expected value, $\nu$ is a constant and $\mathbf{\Gamma}(.)$ is a covariance matrix, only depending on the lag, $\Delta t$.

Lütkepohl (2005, p. 25) shows that a stable VAR process is also stationary. As a result, the condition in (3.2) is often referred to as the *stationarity condition* in the literature. In this work, we always assume stationarity for time series generated by a VAR process.

### 3.2.3 Estimation

Given a realization of a multidimensional time series, $x = \{x_t\}_{t=1}^{N}$, assumed to be generated by a VAR process of order $p$, a Generalized Least Squares (GLS) method can be used to estimate the model components including $\mu$, $\mathbf{A}_j$'s and $\mathbf{\Sigma}$. Lütkepohl (2005) introduces the following notation:

$$\begin{aligned}
\mathbf{Y} &:= [x_{p+1}, x_{p+2}, ..., x_N] \qquad r \times (N-p) \\
Z_t &:= [1, x_t^T, x_{t-1}^T, ..., x_{t-p+1}^T]^T \qquad (rp+1) \times 1 \\
\mathbf{Z} &:= [Z_p, Z_{p+1}, ..., Z_{N-1}] \qquad (rp+1) \times (N-p) \\
\mathbf{B} &:= [\mu, \mathbf{A}_1, ..., \mathbf{A}_p] \qquad r \times (rp+1) \\
\mathbf{U} &:= [u_{p+1}, u_{p+2}, ..., u_N] \qquad r \times (N-p) \\
Y &:= vec(\mathbf{Y}) \qquad r(N-p) \times 1 \\
\beta &:= vec(\mathbf{B}) \qquad r(rp+1) \times 1 \\
U &:= vec(\mathbf{U}) \qquad r(N-p) \times 1
\end{aligned} \tag{3.3}$$

For $t = p+1, p+2, ..., N$, the model in (3.1) can be written as follows:

$$\mathbf{Y} = \mathbf{BZ} + \mathbf{U}. \tag{3.4}$$

Following the calculations in Lütkepohl (2005, p. 70), one obtains

$$Y = (\mathbf{Z}^T \otimes \mathbf{I}_r)\beta + U, \tag{3.5}$$

where the covariance matrix of $U$, $\mathbf{\Sigma}_U$ is shown to be equal to the Kronecker product of an $(N-p) \times (N-p)$ identity matrix and the covariance matrix of $U_t$, $I_{N-p} \otimes \mathbf{\Sigma}$. If the covariance matrix of $U$, $\mathbf{\Sigma}_U$ is of the form $\mathbf{\Sigma}_U = \sigma_U^2 \mathbf{I}_{r(N-p)}$, where $\sigma_U$ is a constant, the Ordinary Least Squares (OLS) method can be used to estimate $\beta$ and $U$ in (3.5), where the elements of $\beta$ are the model coefficients. If the condition on $\mathbf{\Sigma}_U$ is not satisfied, a Generalized Least Squares (GLS) method can be used for this in a similar way. In this case one would expect $\mathbf{\Sigma}_U$ to be a part of the equation representing the GLS estimator for $\beta$. Lütkepohl (2005, p. 71-72) shows that the GLS estimator for $\beta$ is:

$$\hat{\beta} = ((\mathbf{ZZ}^T)^{-1}\mathbf{Z} \otimes \mathbf{I}_r)Y. \tag{3.6}$$

This is also the maximum likelihood estimator of $\beta$. One can observe that $\mathbf{\Sigma}_U$ is not a part of this equation. Interestingly one obtains the same estimator for $\beta$ using a GLS method as one would have obtained using an OLS method. Knowing that $\mathbf{B}$ is an $r \times (tp+1)$ matrix, $vec$ is an invertible transformation. The first $r$ elements of $vec(\mathbf{B})$ are the first column of $\mathbf{B}$, the second $r$ elements of $vec(\mathbf{B})$ are the second column of $\mathbf{B}$, and so on. In this way, we can obtain the GLS estimator for $\mathbf{B}$:

$$vec(\hat{\mathbf{B}}) = \hat{\beta} = vec(\mathbf{YZ}^T(\mathbf{ZZ}^T)^{-1}) \Rightarrow \hat{\mathbf{B}} = \mathbf{YZ}^T(\mathbf{ZZ}^T)^{-1}.$$

### Asymptotic Properties of $\hat{\beta}$

For a sequence of real-valued random variables $X_1, X_2, ...,$ and a random variable $X$, let $F_n$ and $F$ be the cumulative distribution functions (CDF) of random variables $X_n$ and $X$, respectively. According to Casella and Berger (2002, Section 5.5), the sequence $X_1, X_2, ...$ is said to *converge in distribution* to the random variable $X$ if

$$\lim_{n\to\infty} F_n(x) = F(x),$$

at every point $x$ at which F is continuous.

Casella and Berger (2002, Section 5.5) also define *convergence in probability.* The sequence $X_1, X_2, ...$ is said to converge in probability to the random variable $X$ if for all $\epsilon > 0$

$$\lim_{n\to\infty} Pr(|X_n - X| > \epsilon) = 0,$$

where $Pr(.)$ denotes the probability of an event.

Denote convergence in distribution by $\xrightarrow{d}$, and if $X_1, X_2, ... \xrightarrow{d} X$, where $X \sim \mathcal{N}(\mu_X, \Sigma_X)$, use the shorter notation $X_i \xrightarrow{d} \mathcal{N}(\mu_X, \Sigma_X)$. If the sequence $X_1, X_2, ...$ converges in probability to the random variable $X$, one can write $\operatorname{plim}_{n\to\infty} X_n = X$.

Consider the notation introduced in (3.3) for a stationary VAR process $X_t$ defined in (3.1), and let

$$\mathbf{\Pi} := \operatorname*{plim}_{N\to\infty} \frac{1}{N-p}\mathbf{ZZ}^T$$

exist and be non-singular. Lütkepohl (2005, Subsection 3.2.2) shows that under some regularity assumptions about the innovation term, $U_t$, one obtains:

$$\sqrt{N}(\hat{\beta} - \beta) \xrightarrow{d} \mathcal{N}(0, \mathbf{\Sigma} \otimes \mathbf{\Pi}^{-1}), \tag{3.7}$$

where $\hat{\beta}$ is the GLS estimator for $\beta$ given in (3.6) and $\mathbf{\Sigma}$ is the covariance matrix of $U_t$.

### Estimation of $\mathbf{\Sigma}$

Lütkepohl (2005, p. 75-77) introduces the following estimator for $\mathbf{\Sigma}$:

$$\hat{\mathbf{\Sigma}} = \frac{1}{N-(r+1)p-1}\mathbf{Y}(\mathbf{I}_{N-p} - \mathbf{Z}^T(\mathbf{ZZ}^T)^{-1}\mathbf{Z})\mathbf{Y}^T, \tag{3.8}$$

and shows that under some regularity conditions on $U_t$, this estimator converges in probability to the true covariance matrix of $U_t$:

$$\operatorname*{plim}_{N\to\infty} \hat{\mathbf{\Sigma}} = \mathbf{\Sigma}.$$

Note that $N-(r+1)p-1$ is the number of degrees of freedom in the regression problem presented in (3.4). The number of degrees of freedom is equal to the number of equations minus the number of estimated parameters. Equation (3.4) represents $N-p$ linear equations, one for each $t = p+1, p+2, ..., N$. On the other hand, there are $rp+1$ parameters in $\mathbf{B}$ that need to be estimated, namely $\mu, \mathbf{A}_1, \mathbf{A}_2, ..., \mathbf{A}_p$. As a result, the number of degrees of freedom is

$N - p - (rp+1) = N - (r+1)p - 1$. Lütkepohl (2005, p. 75) claims that in a regression problem with fixed, non-stochastic regressors, the adjustment for the number of degrees of freedom in the estimation of the covariance matrix yields an unbiased estimator. Therefore we divide by $N-(r+1)p-1$ when estimating $\boldsymbol{\Sigma}$ in (3.8).

## 3.3 Change-Point Detection

Given a realization of some multidimensional time series $x = \{x_t\}_{t=t_1}^{t_N} = \{x_{t_1}, ..., x_{t_k}, x_{t_{k+1}}, ..., x_{t_N}\}$, $t_k$ is defined as a change-point if there exists a change in the data generating process of the series between the observations $x_{t_k}$ and $x_{t_{k+1}}$. This change can be considered the cause of an anomaly as it will be discussed in Section 3.4.

We will present some methods for detection of such time points based on observational data. The methods presented in this section are offline and unsupervised.

A change-point detection problem can be formulated as a statistical hypothesis test where $H_0$: No changes occur in the data generating process, and $H_1$: a change occurs in the data generating process.

### 3.3.1 Notation

- For the first change-point in a signal, denoted by $\tau_1 \in \{t_1, ..., t_N\}$ such that $\tau_1 > t_1$, define the time interval $r_0 = [t_1, \tau_1]$ as *regime* 0. For the last change-point in a signal, denoted by $\tau_K < t_N$, define the time interval $r_K = [\tau_K + 1, t_N]$ as *regime $K$*. For any other two consecutive change-points, $\tau_k$ and $\tau_{k+1}$ define the time interval $r_k = [\tau_k + 1, \tau_{k+1}]$ as *regime $k$*. This is illustrated in Figure 3.2

- In each regime $k$, the corresponding sub-signal is denoted by ${}_kx = \{x_t | t \in r_k\}$. The sub-signal excluding observations in regime $k$ is denoted by ${}_{-k}x = \{x_t | t \notin r_k\}$.

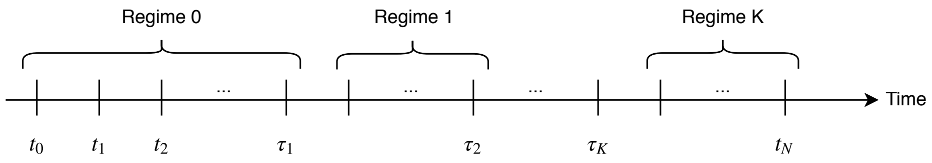


**Figure 3.2:** The time axis, where an observation of some multidimensional time series exists at each $t_i$. Each change-point $\tau_k$ is the end of its preceding regime $k-1$.

### 3.3.2 General Framework

In order to find the change-points in a signal, a contrast function $V(.)$ is defined. This function should be minimized over the whole set of possible change-points.

For any set of $K$ change-points $\tau = \{\tau_k\}_{k=1}^{K} \subset \{t_1, ..., t_N\}$ and a signal $x$, one has:

$$V(\tau, x) := \sum_{k=0}^{K} c({}_k x) \tag{3.9}$$

In each regime a chosen type of model is fit to the corresponding sub-signal ${}_k x$, and $c(.)$ is a cost function measuring the goodness-of-fit of this model to the data. The cost function is often related to the likelihood function given a model. Its values are supposed to be low on sub-signals that are well-approximated by the model and high on sub-signals that are not. Some search methods discussed in Section 3.3.4 and Truong et al. (2018) also make additional assumptions on the cost function. Several cost functions and models are discussed in Section 3.3.5. If some appropriate $c(.)$ is defined, then the change-point detection problem has been transformed to an optimization problem. Here $c(.)$ is defined based on what type of data generating process is assumed.

### 3.3.3 Penalty Function

In the above it is assumed that the number of change-points $K$ is known. This does not necessarily have to be the case. If one tries to minimize the contrast function $V(.)$ for any possible number of change-points $K$, one will quickly realize that $K$ has to be set equal to $N$, such that any possible change-point is in fact detected as a change-point because this will give the perfect fit. This is a typical over-fitting problem, which can be solved by introducing a penalty for the complexity of the set of change-points $\tau$ in the contrast function, such that

$$V(\tau, x) := \sum_{k=0}^{K} c({}_k x) + p(\tau), \tag{3.10}$$

where $p(\tau)$ is a penalty function for the complexity of $\tau$. The penalty function has to be such that the number of change-points can be estimated consistently using this method.

The linear penalty, $l_0$, was first introduced in Hinkley (1970) as:

$$p_{l_0}(\tau) := \omega|\tau| = \omega K, \tag{3.11}$$

where $\omega > 0$. Small values for $\omega$ will result in detection of many change-points and large values make the method sensitive only to the most drastic and significant change-points, which cause large reduction in the contrast function. The choice of $\omega$ should in theory depend on the choice of the cost function $c(.)$ for consistent estimation of the number of change-points $K$. This discussion is however out of the scope of this work. Truong et al. (2018) suggest that $\omega$ should at least depend on the number of observations, $N$, and the dimension of the signal, $r$. One could think that if this wasn't the case, then the algorithm would overestimate the number of change-points for long or high-dimensional signals. The Bayesian Information Criterion (BIC) and the Akaike Information Criterion

(AIC) are popular statistical procedures used in order to choose a value for $\omega$. Let $m$ be the number of parameters one needs to estimate. Then for the AIC, $\omega = \frac{m}{2}$, and for the BIC, $\omega = \frac{m}{2} \log N$, where $N$ is the length of $x$. For instance if one assumes that the 1-dimensional signal $x$ comes from a 1-dimensional Gaussian distribution with mean $\mu_k$ and standard deviation $\sigma_k$ within regime $k$, then $m = 2$, because two variables $\mu_k$ and $\sigma_k$ have to be estimated within each regime. Based on earlier argumentation the BIC is then a better choice when the method is applied to signals of different lengths. In Truong et al. (2018), one can find results about consistency in estimating the number of change-points using the AIC and the BIC under certain strong assumptions. In Bakka (2018) more variants of the AIC and the BIC are discussed. In fact, Haynes et al. (2017) argue that it is not appropriate to use the simple AIC or BIC in change-point detection problems where the true distribution of the data is unknown, because the consistency of the estimated number of change-points cannot be guaranteed. Instead they propose an algorithm called the Changepoints for a Range of PenaltieS (CROPS). Given data, CROPS finds the optimal number of change-points for every constant $\omega$ within an interval $[\omega_{min}, \omega_{max}]$ with a linear penalty function. Using this, one can choose a value for $\omega$ that is the most suitable for the problem at hand.

In the implementation we will take inspiration from the BIC and set $\omega = \frac{m}{2} \log N$. We will however tune the value of $m$ in an ad hoc setting to suit our specific change-point detection problem. This will be explained in more details in Chapter 4.

### 3.3.4 Search Methods

Several optimization algorithms are suggested in Truong et al. (2018) for finding the set of change-points $\tau$ which minimizes the contrast function $V(.)$ in (3.9). For a known number of change-points $K$, it can be shown that

$$\min_{|\tau|=K} V(\tau, x) = \min_{\tau_1 \leq N-K} [c({}_0x) + \min_{|\tau|=K-1} V(\tau, {}_{-0}x)],$$

where $\tau_1$ is the earliest change-point, and ${}_0x$ is the observed signal in the time period $[t_1, \tau_1]$. The proof is to be found in Truong et al. (2018). By following this equation in each step one can reduce the size of the problem, and solve it recursively using a dynamic programming approach, referred to as *optimal detection* or *Opt* in Truong et al. (2018), which sequentially detects change-points with optimal accuracy. For an unknown $K$, a naive approach is to apply Opt to the contrast function in (3.9) for all possible number of change-points $K = 1, ..., N$, and then choose the one that minimizes the contrast function in (3.10). However, this approach would prove to be very computationally expensive. A faster approach is provided in Killick et al. (2012) under the name of Pruned Exact Linear Time (PELT). PELT is a modification of Opt which uses a pruning rule to discard a number of possible prospective solutions, reducing the computational cost significantly. In the implementation of PELT an assumption is made on the minimum possible distance between consecutive change-points in addition to regularity assumptions on the cost function and linearity assumption of the penalty function. The detailed algorithms for both Opt and PELT in addition to several other search methods are presented in Truong et al. (2018).

### 3.3.5 Models

The cost function $c({}_k x)$ should be decided based on an assumption about the underlying data generating process for the signal $x$. One possible assumption is that the data comes from a multivariate Gaussian distribution, such that at any given time $t_i$, $x_i$ is the realization of the $r$-dimensional variable $X_t$ with a multivariate Gaussian distribution. The Gaussianity assumption might not hold in the case of an observed time series $x$. In order to address this issue a Vector Autoregressive (VAR) data generating process of some order $p$ is introduced. In VAR, at any given time point $t_i$, $x_i$ is assumed to be linearly dependent on the $p$ preceding observations of $x$, $x_{i-1}, ...x_{i-p}$ and some random innovation term. Multivariate Gaussian process and vector autoregressive process are both discussed below. The cost function $c(.)$ should be derived for both of them.

#### Multivariate Gaussian Process

A vector variable $X_t$ of dimension $r$ with expected value $\mu$ and positive definite covariance matrix $\mathbf{\Gamma}$, generated by a multivariate Gaussian process has the following distribution:

$$f(X_t) = (2\pi)^{-\frac{r}{2}} \det(\mathbf{\Gamma})^{-\frac{1}{2}} \exp(-\frac{1}{2}(X_t - \mu)^T \mathbf{\Gamma}^{-1}(X_t - \mu)).$$

Now divide the time interval $[t_1, t_N]$ into $K+1$ regimes, labeled from 0 to $K$. For $1 \leq k \leq K$, the change-point $\tau_k$ is the last time point in regime $k-1$. In each regime $k$, the vector variable $X_t$ is assumed to have a Gaussian distribution $f_k$ with mean $\mu_k$ and covariance matrix $\mathbf{\Gamma}_k$. This is illustrated in Figure 3.3.

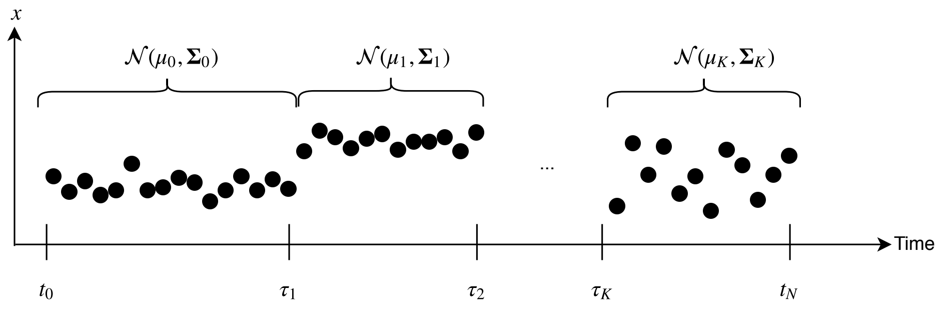

$x$
$\mathcal{N}(\mu_0, \mathbf{\Sigma}_0)$
$\mathcal{N}(\mu_1, \mathbf{\Sigma}_1)$
$\mathcal{N}(\mu_K, \mathbf{\Sigma}_K)$
...
Time
$t_0$
$\tau_1$
$\tau_2$
$\tau_K$
$t_N$

**Figure 3.3:** This plot illustrates different distributions of the vector variable $X_t$ in different regimes. The observations are shown in one dimension for convenience.

Let $\hat{\mathbf{\Gamma}}_k$ be the empirical covariance matrix of the sub-signal ${}_kx$, then:

$$\hat{\mathbf{\Gamma}}_k = \frac{1}{n_k - 1} \sum_{t \in r_k} (x_t - \overline{{}_kx})(x_t - \overline{{}_kx})^T,$$

where $n_k$ is the number of data points in regime $k$, and $\overline{{}_kx}$ is the empirical mean of the sub-signal ${}_kx$. According to Truong et al. (2018), the appropriate cost function is:

$$c({}_kx) := n_k \log \det \hat{\mathbf{\Gamma}}_k. \tag{3.12}$$

This is because for a $r$-dimensional multivariate Gaussian distribution with unknown mean and unknown covariance matrix the log-likelihood of $n$ independent observations, can be expressed as

$$l(\hat{\mu}, \hat{\mathbf{\Gamma}}) = -\frac{1}{2}(rn(\log(2\pi) + 1) + n \log \det \hat{\mathbf{\Gamma}}), \tag{3.13}$$

where $\hat{\mu}$ is the empirical mean and $\hat{\mathbf{\Gamma}}$ is the empirical variance. This means in order to maximize the log-likehood function one has to minimize $c(.) = n \log \det \hat{\mathbf{\Gamma}}$. A proof for (3.13) is found in Bakka (2018).

### Vector Autoregressive (VAR) Process

Assume a signal $x$ is generated by a VAR process of order $p$, as described in Section 3.2. Divide the time interval $[t_1, t_N]$ into $K+1$ regimes, as it was done for the multivariate Gaussian process. From one regime to the next, one or several of the model parameters $\mu, \mathbf{A}_1, \mathbf{A}_2, ..., \mathbf{A}_p$, and $\mathbf{\Sigma}$ change. Then Bai (2000) writes the data generating process as:

$$X_t = \begin{cases} \mu_0 + \mathbf{A}_{0,1}X_{t-1} + \mathbf{A}_{0,2}X_{t-2} + ... + \mathbf{A}_{0,p}X_{t-p} + (\mathbf{\Sigma}_0)^{1/2}\eta_t & t \in r_0 \\ \mu_1 + \mathbf{A}_{1,1}X_{t-1} + \mathbf{A}_{1,2}X_{t-2} + ... + \mathbf{A}_{1,p}X_{t-p} + (\mathbf{\Sigma}_1)^{1/2}\eta_t & t \in r_1 \\ ... \\ \mu_{K+1} + \mathbf{A}_{K+1,1}X_{t-1} + \mathbf{A}_{K+1,2}X_{t-2} + ... + \mathbf{A}_{K+1,p}X_{t-p} + (\mathbf{\Sigma}_{K+1})^{1/2}\eta_t & t \in r_K \end{cases}$$

where $\eta_t$ is a random variable with multivariate Gaussian distribution, mean 0 and covariance matrix $\mathbf{I}$. $\mathbf{\Sigma}_k^{1/2}$ is defined such that $\mathbf{\Sigma}_k = \mathbf{\Sigma}_k^{1/2}\mathbf{\Sigma}_k^{1/2}$. Then $(\mathbf{\Sigma}_k)^{1/2}\eta_t$ has a multivariate Gaussian distribution with mean 0 and covariance matrix $\mathbf{\Sigma}_k^{1/2}\mathbf{I}\mathbf{\Sigma}_k^{1/2} = \mathbf{\Sigma}_k$:

$$\mathbf{\Sigma}_k^{1/2}\eta_t \sim \mathcal{N}(0, \mathbf{\Sigma}).$$

The derivation of the cost function for this model is out of the scope of this work. This can be found in Bai (2000). However, Truong (2017, Personal Correspondence) has confirmed that the following cost function is appropriate in the case of a VAR process:

$$c({}_kx) := n_k \log \det \hat{\mathbf{\Sigma}}_k, \tag{3.14}$$

where $\hat{\mathbf{\Sigma}}_k$ can be computed with the method provided in Subsection 3.2.3. For a 1-dimensional time series assumed to be generated by a 1-dimensional autoregressive process, Truong (2017) shows that the cost function introduced in Bai (2000) is reduced to:

$$c(_kx) := \min_{\mu_k,\mathbf{A}_{k,1},\mathbf{A}_{k,2},\ldots,\mathbf{A}_{k,p}} \sum_{t\in r_k} ||x_t-(\mu_k+\mathbf{A}_{k,1}X_{t-1}+\mathbf{A}_{k,2}X_{t-2}+\ldots+\mathbf{A}_{k,p}X_{t-p})||_2^2, \tag{3.15}$$

where $||.||_2^2$ is the squared $l^2$ norm.

Truong (2017) provides a Python package for change-point detection following the framework presented in this section. To this date, Truong (2017) only has implemented the cost function in (3.15) for 1-dimensional autoregressive processes, leaving out the cost function in (3.14) for VAR processes. Truong (2017, Personal Correspondence) has however expressed interest in a collaboration with the author about implementing the cost function in (3.14).

## 3.4 Dimensionality Reduction

We refer to the number of dimensions used for analysis as the dimensionality. In this section we present methods for dimensionality reduction given a high-dimensional data set. The main reason for dimensionality reduction is that many dimensions in the original data might not contain any information that is useful for predicting the failures in the system, and including these dimensions in the predictive model might only add noise to the results. In this work it is desired to reduce the dimensionality to one using the methods presented in this section. We refer to the final 1-dimensional signal as the *signal of interest*, and we try to maximize the predictive information in this signal. The predictive model presented in Section 3.5 can be applied to this 1-dimensional signal. It is also possible to develop a multivariate predictive model based on a multidimensional signal of interest. This is however out of the scope of this work.

The simplest method for dimensionality reduction is picking one of the original dimensions in the data as the signal of interest. Assuming that a training data set that includes both normal and tripping groups is available, the Mann-Whitney U test presented in Subsection 3.4.5 can be performed to find the dimension which is expected to contain the most predictive information. In this case for every dimension in the original data, the offline change-point detection presented in Section 3.3 is applied to all the running groups in the training data set. For high-dimensional data, this might be a very computationally demanding task. In addition, the combination of several dimensions might contain more predictive information than each of the dimensions alone.

Here a semi-supervised method of dimensionality reduction is developed. The ultimate goal in this section is to develop a method for finding the directions in the data with maximum predictive information. First a linear transformation is created. This is done by taking advantage of Principal Component Analysis (PCA) or Box-Tiao Analysis (BTA) in a semi-supervised manner described in

Subsection 3.4.3. PCA and BTA are respectively described in Subsections 3.4.1 and 3.4.2. Finally in Subsection 3.4.5, we present some methods for reducing the dimensionality of the data to the desired number of dimensions using the linear transformations presented in this section.

In this section consider a $1 \times r$ vector variable $X_t$, with mean $\mu_X$ and covariance matrix $\mathbf{\Gamma}$. Let us assume with no loss of generality that $\mu_X = 0$. One can do so because if this is not the case, one can simply consider the $1 \times r$ vector variable $X_t - \mu_X$ instead. If $\mu_X$ is unknown, it can be estimated by the sample mean $\hat{\mu}_X$. Let $x = \{x_t\}_{t=1}^{N}$ be a series of observations of $X_t$ in the time interval $[t_1, t_N]$. Using $x$, create the $N \times r$ matrix, $\mathbf{X}$, where the $i$-th row is $x_t$ for $t = t_i$.

## 3.4.1 Principal Component Analysis (PCA)

The goal in PCA is to find directions in the data with maximal variance. Let us refer to the $i$-th principal component of $X_t$ as $PC_i$. For $i = 1, 2, ..., r$, $PC_i$ is defined as the linear combination of the columns of $X_t$ with the $i$-th most variance such that $PC_i$ and $PC_j$ are orthogonal to each other for $i \neq j$. Seber (2009, Section 5.2) proves that the principal components of $X_t$ are the projection of the data on the eigenvectors of the covariance matrix of $X_t$, $\mathbf{\Gamma}$. Additionally, $PC_i$ is the projection of the data on the eigenvector corresponding to the $i$-th largest eigenvalue of $\mathbf{\Gamma}$. Let $\mathbf{\Gamma}$ have the following diagonalization:

$$\mathbf{\Gamma} = \mathbf{\Psi}\mathbf{\Delta}^2\mathbf{\Psi}^T,$$

where $\mathbf{\Delta}^2$ is a diagonal matrix with the eigenvalues of $\mathbf{\Gamma}$ in descending order on its diagonal and $\mathbf{\Psi}$ is a $r \times r$ orthonormal matrix. Note that $\mathbf{\Gamma}$ is a positive definite matrix and its eigenvalues are therefore are greater than 0. The columns of $\mathbf{\Psi}$ are the eigenvectors of $\mathbf{\Gamma}$. The PCA-transformed data is then defined as:

$$\mathbf{P} = \mathbf{X}\mathbf{\Psi},$$

where the $i$-th column of $\mathbf{P}$ is the $i$-th principal component $PC_i$. Let $\lambda_i$ be the $i$-the largest eigenvalue of $\mathbf{\Gamma}$. In other words, $\lambda_i$ is the diagonal element in the $i$-the row and the $i$-th column of $\mathbf{\Delta}^2$. Then Seber (2009, Section 5.2) proves that:

$$\text{var}(PC_i) = \lambda_i,$$

where $\text{var}(PC_i)$ is the variance of the $i$-the principal component.

The true mean and covariance matrix of $X_t$ is usually unknown. The sample principal components of $X_t$ are the projection of the centered data on the eigenvectors of the estimated covariance matrix of $X_t$, when the centring has been performed using the sample mean of $X_t$, referred to as $\hat{\mu}_X$, and calculated as:

$$\hat{\mu}_X = \frac{1}{N}\sum_{t=1}^{N} x_t.$$

We assume that the observation matrix $\mathbf{X}$ has full rank. This means $N - 1 > r$ and the rank of $\mathbf{X}$ is $r$. Let $\mathbf{X}$ have the following singular value decomposition

(SVD):

$$\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^T, \tag{3.16}$$

where $\mathbf{U}$ is a $N \times r$ orthonormal matrix, $\mathbf{V}$ is a $r \times r$ orthonormal matrix, and $\mathbf{D}$ is a $r \times r$ diagonal matrix with elements larger than or equal to zero. This decomposition should be done such that the elements of $\mathbf{D}$ are sorted in descending order. Given that $\mathbf{X}$ is centered, Seber (2009, Section 1.4) shows that an unbiased estimator for the covariance matrix of $X_t$ is:

$$\hat{\mathbf{\Gamma}} = \frac{1}{N-1}\mathbf{X}^T\mathbf{X} = \frac{1}{N-1}\mathbf{V}\mathbf{D}^2\mathbf{V}^T. \tag{3.17}$$

In (3.16), columns of $U$ are eigenvectors of $\mathbf{X}\mathbf{X}^T$, columns of $V$ are eigenvectors of $\mathbf{X}^T\mathbf{X}$ and diagonal elements of $\mathbf{D}$ are the square root of the eigenvalues of $\mathbf{X}^T\mathbf{X}$. The PCA-transformed data, using *sample principal components* are then defined as:

$$\mathbf{P}_{\text{sample}} = \mathbf{X}\mathbf{V},$$

where the $i$-th column of $\mathbf{P}_{\text{sample}}$ is the $i$-th sample principal component. Seber (2009, Subsection 5.2.4) shows that by using $\hat{\mu}_X$ and $\hat{\mathbf{\Gamma}}$, there is essentially no difference between the properties of the true and the sample principal components. From here we refer to the sample principal components as principal components since we only have knowledge about the estimated mean and covariance matrix of $X_t$. We also will refer to $\mathbf{P}_{\text{sample}}$ as $\mathbf{P}$ and call its $i$-th column as $PC_i$.

The estimated correlation coefficient between the principal component $PC_i$ and the signal $\mathbf{X}_{:,k}$, which is the $k$-the column of the matrix $\mathbf{X}$, measures the importance of the $k$-th variable to $PC_i$. This estimated correlation is given by:

$$\hat{\rho}_{ik} = \frac{\hat{\sigma}_{\mathbf{P}_{:,i},\mathbf{X}_{:,k}}}{\hat{\sigma}_{\mathbf{X}_{:,k}}\hat{\sigma}_{\mathbf{P}_{:,i}}}, \tag{3.18}$$

where $\hat{\sigma}_{\mathbf{P}_{:,i},\mathbf{X}_{:,k}}$ is the estimated covariance between $\mathbf{X}_{:,k}$ and the $i$-th column of $\mathbf{P}$, $\mathbf{P}_{:,i}$, and $\hat{\sigma}_{\mathbf{X}_{:,k}}$ and $\hat{\sigma}_{\mathbf{P}_{:,i}}$ are the estimated variances of respectively $\mathbf{X}_{:,k}$ and $\mathbf{P}_{:,i}$. The expression in (3.18) can be simplified given the eigenvalues and the eigenvectors of the estimated covariance matrix $\hat{\mathbf{\Gamma}}$. However, this is not done here since the correlations are computed according to (3.18) in the implementation.

### 3.4.2 Box-Tiao Analysis (BTA)

Box and Tiao (1977) propose a slightly different approach from PCA for reducing the dimensionality of multidimensional time series data. They call it a canonical analysis of multiple time series. Here we will refer to it as BTA. The shortcoming of PCA when applied to time series is that the estimator used for the covariance matrix is not consistent for time series data. This is shown among others by Xiao et al. (2012). Box and Tiao (1977) try to take into account the time-dependency between observations. They propose to maximize predictability instead of variance.

For a 1-dimensional stationary time series, $y = \{y_t\}_{t=1}^{N}$, the predictability is defined as following. Let $y$ be generated by the following process:

$$y_t = f(y_{1\dots(t-1)}) + u_t,$$

where $f$ is a function of some or all of the preceding observations of $y$, and $u_t$ a random error term. One can also define $u_t$ as the difference between the observed value of the variable $y$ at time $t$ and the best possible prediction for $y_t$ given all the observations of $y$ prior to this time point. Let $u_t$ be called the innovation term, and assume that the $u_t$'s are independent and have an identical distribution with mean 0 and constant variance $\sigma_u^2$. Note that $f(.)$ is a deterministic function while $y_t$, $u_t$ are random variables. However, $f(y_{1\dots(t-1)})$ is a function of some random variables and therefore a random variable itself. Assume that for every time point $t$, the innovation term $u_t$ is independent from the prediction $f(y_{1\dots(t-1)})$. Let $\sigma_y$ be the standard deviation of $y$ and assume $\sigma_y > 0$. Then the variance of $f(y_{1\dots(t-1)})$ can be written as:

$$\mathrm{var}(f(y_{1\dots(t-1)})) = \sigma_y^2 - \sigma_u^2.$$

The predictability of $y$, $q_y$ is the fraction of the variation in $y$, which is explained by the effect of its previous observations on its current value. This is defined as:

$$q_y = \frac{\sigma_y^2 - \sigma_u^2}{\sigma_y^2} = 1 - \sigma_y^{-2}\sigma_u^2. \tag{3.19}$$

As an example, take the following time series: $y_{2k-1} = 0$ and $y_{2k} = 1$ for $k = 1, 2, \dots$. This time series is completely deterministic, meaning that given the observation for $y_{t-1}$, one can predict $y_t$ with no error. The variance of $y$ is however not 0, while its predictability is 1. For this time series, $\sigma_y = 0.5$ and $\sigma_u = 0$. If $y$ had been a part of a bivariate time series with another truly random signal with a standard deviation much smaller than 0.5, the first principal component would have been almost identical to $y$. However one can observe that $y$ would not have explained any of the random variation in the data. This is an example of the challenge Box and Tiao (1977) try to address.

Now let us move on from univariate to multivariate analysis. Consider the $r$-dimensional vector variable $X_t$. Assume that $X_t$ follows the process:

$$X_t = g(X_{1\dots(t-1)}) + \epsilon_t, \tag{3.20}$$

where $g$ is a function of some or all of the preceding observations of $X_t$, and $\epsilon_t$ is an $r$-dimensional error term with mean 0 and a time-invariant covariance matrix $\mathbf{\Sigma}$. We refer to $\epsilon_t$ as the *innovation* term. Assume that $\epsilon_t$'s are independent and identically distributed. Let $\mathbf{\Gamma}$ be the covariance matrix of $X_t$. The predictability matrix, $Q$, is defined as:

$$\mathbf{Q} = \mathbf{I} - \mathbf{\Gamma}^{-1}\mathbf{\Sigma}$$

Box and Tiao (1977) prove that the linear combinations of the data with maximum predictability are in the direction of the eigenvectors corresponding to the largest eigenvalues of the predictability matrix. Let us call the projection of the data on the eigenvector corresponding to the $i$-th largest eigenvalue of the predictability for $BT_i$. Comparing this to PCA, where $PC_i$ is the linear combination with the $i$-th largest variance, $BT_i$ is the linear combination with the $i$-th most predictability. Let $\mathbf{Q}$ have eigenvectors $m_i$ corresponding to the eigenvalues $\lambda_i$. Then we have:

$$\mathbf{Q}m_i = (\mathbf{I} - \mathbf{\Gamma}^{-1}\mathbf{\Sigma})m_i = \lambda_i m_i \Rightarrow \mathbf{\Gamma}^{-1}\mathbf{\Sigma}m_i = (1-\lambda_i)m_i.$$

By taking the inverse of the both sides of the last equation one obtains:

$$\mathbf{\Sigma}^{-1}\mathbf{\Gamma}m_i = \alpha_i m_i, \tag{3.21}$$

where $\alpha_i = (1-\lambda_i)^{-1}$. One can easily show that $\lambda_i$'s and $\alpha_i$'s are in the same order. This means the $i$-th largest $\lambda_i$ corresponds to the $i$-th largest $\alpha_i$'s. Let $\alpha_i$'s be in descending order. If the innovation term has covariance matrix equal to a positive constant times identity, BTA is equivalent to a regular PCA.

Let $\mathbf{\Sigma}^{-1}\mathbf{\Gamma} = \mathbf{\Omega}\mathbf{\Theta}^2\mathbf{\Omega}^T$ be the diagonalization of $\mathbf{\Sigma}^{-1}\mathbf{\Gamma}$, such that the diagonal elements of $\mathbf{\Theta}^2$ are in descending order. Consider the observation matrix $\mathbf{X}$. In this analysis we need to compute the projection of the rows of $\mathbf{X}$ on the eigenvectors of $\mathbf{\Sigma}^{-1}\mathbf{\Gamma}$. The BTA transformed data can therefore be defined as:

$$\mathbf{T} = \mathbf{X}\mathbf{\Omega},$$

where the $i$-the column of $\mathbf{T}$, also referred to as $BT_i$, is the $i$-th most predictable component of the observed time series $x$. Let $q_i$ and $\lambda_i$ be the predictability of $BT_i$ and the $i$-th largest eigenvalue of the predictability matrix $\mathbf{Q}$. Then one has:

$$q_i = \lambda_i = 1 - \alpha_i^{-1},$$

where $\alpha_i$ is the $i$-th largest eigenvalue of $\mathbf{\Sigma}^{-1}\mathbf{\Gamma}$ and the $i$-th diagonal element of $\mathbf{\Omega}^2$.

The true $\mathbf{\Sigma}$ and $\mathbf{\Gamma}$ are usually unknown. The sample Box-Tiao components of $X_t$ are the projection of the data on the eigenvectors of the estimated predictability matrix of $X_t$. We assume that the observation matrix $\mathbf{X}$ has full rank. This means $N-1 > r$ and the rank of $\mathbf{X}$ is $r$. Then we estimate $\mathbf{\Sigma}$ and $\mathbf{\Gamma}$ according to the equations methods provided in Subsections 3.2.3 and 3.4.1, respectively. Let the estimate $\hat{\mathbf{\Sigma}}^{-1}\hat{\mathbf{\Gamma}}$ have the following diagonalization:

$$\hat{\mathbf{\Sigma}}^{-1}\hat{\mathbf{\Gamma}} = \mathbf{L}\mathbf{S}^2\mathbf{L}^T \tag{3.22}$$

The BTA-transformed data, using sample Box-Tiao components is then defined as:

$$\mathbf{T}_{\text{sample}} = \mathbf{X}\mathbf{L},$$

where the $i$-th column of $\mathbf{T}_{\text{sample}}$ is the $i$-th sample Box-Tiao component. From here we refer to the sample Box-Tiao components as Box-Tiao components since we only have knowledge about the estimated predictability matrix of $X_t$. We also will refer to $\mathbf{T}_{\text{sample}}$ as $\mathbf{T}$ and call its $i$-th column as $BT_i$.

The estimated correlation coefficient between the Box-Tiao component $BT_i$ and the signal $\mathbf{X}_{:,k}$, which is the $k$-the column of the matrix $\mathbf{X}$, measures the importance of the $k$-th variable to $BT_i$. This estimated correlation is given by:

$$\hat{\pi}_{ik} = \frac{\hat{\sigma}_{\mathbf{T}_{:,i},\mathbf{X}_{:,k}}}{\hat{\sigma}_{\mathbf{X}_{:,k}}\hat{\sigma}_{\mathbf{T}_{:,i}}},$$

where $\hat{\sigma}_{\mathbf{T}_{:,i},\mathbf{X}_{:,k}}$ is the estimated covariance between $\mathbf{X}_{:,k}$ and $BT_i$, and $\hat{\sigma}_{\mathbf{X}_{:,k}}$ and $\hat{\sigma}_{\mathbf{P}_{:,i}}$ are the estimated variances of respectively $\mathbf{X}_{:,k}$ and $BT_i$.

### 3.4.3 Semi-Supervised Nature of the Transformations

Kuncheva and Faithfull (2014) explain that when change-point detection is applied for failure prediction, one could think that there might be changes is some directions in the data that do not cause a failure. These are referred to as normal change-points and in a failure prediction problem it is not desirable to detect them. This might be when a change in some other directions in the data in fact causes a failure. These are referred to as non-normal change-points, and are interesting to detect. To devise a semi-supervised approach to dimensionality reduction, one first needs to identify some time interval $I = \{t_a, t_{a+1}, ..., t_s\}$ where no interesting change-points occur in the corresponding sub-signal $_Ix = \{x_t | t \in I\}$, where $I$ is labeled as a normal time interval. A transformation can then be created based on $_Ix$. This means that the covariance matrices needed to build the transformation are estimated using only $_Ix$. An estimate for the covariance matrix of the data, $\mathbf{\Gamma}$, is needed for both PCA and BTA. For BTA an estimate for the covariance matrix of the innovation term, $\mathbf{\Sigma}$, is also needed. The estimates are then used in order to transform the whole data set.

Following this approach in a PCA, the normal variations in the data are intuitively captured by the principal components with large variances, while one would expect the principal components with small variances to be the most sensitive to non-normal changes and therefore the most valuable in a failure prediction problem. These are the $PC_i$'s for $i$'s close to $r$. On the other hand one could argue that the principal components with the smallest variances might not include any valuable information and only capture the noise in the data, and therefore it might be of interest to look at the components with larger variances. In addition an anomalously large change in the principal components with large variances means that the change is in a direction of the data in which there also exists a large degree of normal variation, but the change is of a non-normal amplitude. This might also be a good predictor for failures in the system. The principal components with large variances are $PC_i$'s for $i$'s close to 1.

In the case of a BTA, normal changes will intuitively be captured in the components with small predictability, while one would expect the components with large predictability not to include drastic changes under normal operation mode. A change in these components might therefore be a good predictor for failure. These are the $BT_i$'s for $i$'s close to 1. One could however argue that the components with small predictability are expected to show a *white-noise* behaviour under the normal operation mode. A deviation from this behaviour might therefore be a good predictor for failure. These are the $BT_i$'s for $i$'s close to $r$.

Intuitively, if the changes that cause failures increase the variance in some direction of the data, PCA might be a suitable approach to find the optimal signal of interest. However if the changes that cause failures are in the structure of the relationships between the dimensions, BTA might be a more suitable approach than PCA.

Following a semi-supervised approach, it is not clear exactly which linear combinations include the most predictive information in a failure prediction problem. The components most sensitive to interesting changes may differ from a problem to another and can only be determined by experimentation with the methods presented in Subsection 3.4.5. In Section 4.2, we present more details on how an experiment can be designed to determine this. In Section 4.2, we also provide a discussion about the kind of problems in which each principle or Box-Tiao component should be used for failure prediction. We propose the use of the semi-supervised approach to separate the normal variations in the data from the variations that cause a failure in the system. This is the strength of this approach compared to any unsupervised anomaly detection method.

### 3.4.4 Covariance Matrix Estimation

Applying methods of dimensionality reduction in our business case follows with its challenges. The covariance matrix $\mathbf{\Gamma}$ used in both PCA and BTA as well as the function $g$ and the covariance matrix $\mathbf{\Sigma}$ used in BTA are unknown and need to be estimated. However, there does not exist a continuous stream of data that can be used for this purpose. Instead, the data is divided in many running groups. A running group marks the time from when one compressor is turned on until the consecutive shut-down. Since there might be a long gap in time between running groups, it is not reasonable to concatenate the data and treat it as one continuous signal. This might be the case in many similar industrial settings. Therefore we present a pooling method here that can be used for estimation of the unknowns in situations similar to our business case.

First for every running group, $i$, the estimated covariance matrix $\hat{\mathbf{\Gamma}}_i$ is computed within the running group using the estimator in (3.17). Then one obtains the following equation:

$$\hat{\mathbf{\Gamma}}_i = \frac{1}{n_i - 1}\mathbf{X}_i^T\mathbf{X}_i,$$

where $\mathbf{X}_i$ is an $N_i \times r$ matrix with $n_i$ observations of the $r$-dimensional vector variable $X_t$ during the running group $i$. It is assumed that $\mathbf{X}_i$ has full rank. This means $N_i - 1 > r$ and the rank of $\mathbf{X}_i$ is $r$. The observations along each of the $r$ dimensions are centered around 0 within each running group. Here it is assumed that within one and the same running group, the $r$-dimensional vector variable $X_t$ has a constant expected value with respect to time. This assumption clearly holds if the process is stationary. However, if there exists a change-point in $X_t$ during a running group, the expected value of $X_t$ is not necessarily constant during that running group. Assuming constant expected value in this case might not yield a consistent estimator for the covariance matrix. This is the case for the data analyzed in this work, however this effect will be ignored in our analysis. One reason for this is that we have no knowledge of when change-points occur in the real-world data, and therefore we can not adjust the estimator of $\mathbf{\Gamma}$ to obtain consistent results. Another reason is that the inconsistency in the estimator of $\mathbf{\Gamma}$ might actually be desirable in our case. This is because the variance of the dimensions in the data where there exists normal change-points will be overestimated in this case. This overestimated variance will hopefully be captured by one of the first principal components, which makes the last principal components suitable as alternative signals of interest. With the same argumentation we assume that the true covariance matrix $\mathbf{\Gamma}$ is also constant with respect to time across all the running groups. The overall $\hat{\mathbf{\Gamma}}$ is then the weighted average of $\hat{\mathbf{\Gamma}}_i$'s:

$$\hat{\mathbf{\Gamma}} = \frac{\sum_i (n_i - 1)\hat{\mathbf{\Gamma}}_i}{\sum_i (n_i - 1)},$$

where $n_i$ is the length of the running group $i$ and the summing of matrices is done element-wise.

Calculating $\hat{\mathbf{\Sigma}}$ is a more demanding task, since one needs insight into the function $g$ introduced in (3.20). Let us assume that the time series is generated by a VAR process of order $p$, as explained in Section 3.2. We assume that the order $p$ and the true covariance matrix of the innovation term, $\mathbf{\Sigma}$, are constant with respect to time across all the running groups. These assumption will clearly not hold if there exist change-points in the data. This will be ignored in our analysis with a similar argumentation to the one provided for ignoring a possibly non-constant expected value for $X_t$ in the estimation of $\mathbf{\Gamma}_i$'s. The order $p$ can be selected using the methods developed in Lütkepohl (2005, Chapter 4). However, the methods developed in Lütkepohl (2005, Chapter 4) will not be used in our analysis. Instead. we pick $p$ in consultation with the field experts. Within a running group $i$, $\hat{\mathbf{\Sigma}}_i$ is calculated using the method presented in Subsection 3.2.3. The overall estimated covariance matrix $\hat{\mathbf{\Sigma}}$ is the weighted average of $\hat{\mathbf{\Sigma}}_i$'s, where $(n_i - (r+1)p)$ are the weights:

$$\hat{\mathbf{\Sigma}} = \frac{\sum_i (n_i - (r+1)p)\hat{\mathbf{\Sigma}}_i}{\sum_i (n_i - (r+1)p)}.$$

If $\hat{\mathbf{\Gamma}}_i$'s and $\mathbf{\Sigma}_i$'s are unbiased estimators of $\mathbf{\Gamma}$ and $\mathbf{\Sigma}$, respectively, then $\hat{\mathbf{\Gamma}}$ and $\hat{\mathbf{\Sigma}}$ are also unbiased estimators of respectively $\mathbf{\Gamma}$ and $\mathbf{\Sigma}$. Assume that $E(\hat{\mathbf{\Gamma}}_i) = \mathbf{\Gamma}$ and $E(\hat{\mathbf{\Sigma}}_i) = \mathbf{\Sigma}$. Then:

$$E(\hat{\mathbf{\Gamma}}) = \frac{\sum_i (n_i - 1)E(\hat{\mathbf{\Gamma}}_i)}{\sum_i (n_i - 1)} = E(\hat{\mathbf{\Gamma}}_i) = \mathbf{\Gamma},$$

$$E(\hat{\mathbf{\Sigma}}) = \frac{\sum_i (n_i - (r+1)p)E(\hat{\mathbf{\Sigma}}_i)}{\sum_i (n_i - (r+1)p)} = E(\hat{\mathbf{\Sigma}}_i) = \mathbf{\Sigma}.$$

### 3.4.5 Signal Selection

After transforming the data using PCA or BTA, we want to select one signal among many to reduce the dimensionality. We call this the *signal of interest*. Following the semi-supervised approach presented in Subsection 3.4.3, this signal should be selected by experimentation. In the following two alternative criteria are presented to select the signal of interest.

For the selection of the signal of interest training data is required. The training data set should include both periods of normal operation mode and periods prior to a failure in the system, and to avoid over-fitting, it should ideally be separate from the data used to create the (BTA or PCA) transformation and the data used to evaluate the model. In this data set each time point is labeled as belonging to either a *normal state*, a *transition state*, or a *failure state*. A normal state consists of time points not close enough to a failure in the system for possibly containing information about the cause of the failure. A failure state consists of the time points too close to a failure in the system, such that predicting the failure in this time window cannot give the engineers enough time to take preventing actions. A transition state consists of all the time points in between the other two states. We are interested in being able to predict the failure in this time window. Using a change-point detection method for prediction, we are interested in detecting change-points during the transition state and minimize the number of change-points detected in the normal state. Change-points detected in the failure state will not affect our model.

Time points prior to and closer than $f$ minutes to a failure in the system should be labeled as belonging to a failure state. Time points prior to a planned shut-down should be labeled as belonging to a normal state. Time points prior to and farther away than $\nu$ minutes from a failure in the system should also be labeled as belonging to a normal state based on the given definition. All other time points should be labeled as belonging to a transition state. The numerical values for $f$ and $\nu$ differ from a business case to another and should be selected in consultation with the field experts. This is illustrated in Figure 3.4.

Note that labeling the data points as illustrated in Figure 3.4 makes our signal selection approach semi-supervised. This is because we do not assume that all the data points in the transition states are non-normal, as one would have assumed in a fully supervised approach. However, we assume that there exists a change-point in each transition state. This only implies that the data points that appear after the change-point has occurred are non-normal.

In Figure 3.4, one can see that the different states are not necessarily of the same length. One could therefore think that the probability of detecting a change-point might be higher in longer states. Since normal states last usually much longer than transition states, this might introduce an undesirable bias into the selection of the signal of interest. Note that for the selection of the signal of interest, the offline change-point detection method presented in Section 3.3 is used. In this method the length of a signal should not affect the number of change-points detected in it as long as the penalty function discussed in Subsection 3.3.3 is a function of the length of the signal such that the number of

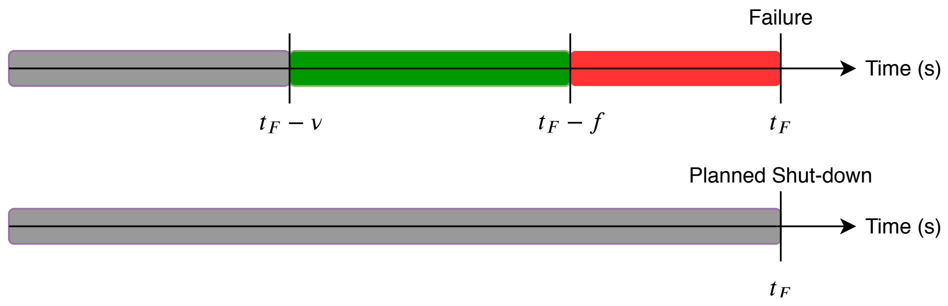


**Figure 3.4:** The first time line is prior to a shut-down due to a failure in the system, and the second time line is prior to a normal planned shut-down. The red, green and grey color mark respectively the time referred to as a failure state, a transition state and a normal state.

change-points is estimated consistently. In order to avoid a bias in the selection of the signal of interest, will use a penalty function inspired by the BIC. In our case the penalty function will be a function of $\log N$, where $N$ is the length of the signal. Note that we have not proven that in this case the number of change-points will be estimated consistently. This is however assumed in this work. Therefore we can use the division presented in Figure 3.4 for the selection of the signal of interest.

### The Mann-Whitney U test

In this method the number of change-points detected in different states are compared to each other for the selection of the signal of interest among some alternatives. The alternatives might be linear combinations of the data, like the principle or the Box-Tiao components. However, other functions of the data can also be among the alternatives. Later in this subsection, three non-linear functions of the data, called the *discrepancy*, the *leverage* and the *influence* are introduced. These three signals can also be alternatives for the signal of interest selected by the Mann-Whitney U test method. One can also use the dimensions in the data directly and pick a signal of interest among them. However, constructing a function, such as the PCA and the BTA transformation, which combine several dimensions in the data into one signal of interest might yield better results. This is because the combination of several dimensions might contain more predictive information than each of the dimensions alone. Here we will describe the Mann-Whitney U test as it is applied to pick one of the principal or the Box-Tiao components as the signal of interest. However, the method can easily be adjusted to pick the signal of interest among other alternatives.

Divide the data into normal, transition and failure states as illustrated in Figure 3.4, and consider a column in the transformed data set $C_q$ for $q = 1, 2, ..., r$. For each period of transition state, $i = 1, 2, ..., n$, let $\alpha_i = 1$ if a change-point is detected in $C_q$ during period $i$, and $\alpha_i = 0$ if no change-point is detected in $C_q$ during this period. Similarly for each period of normal state

$j = 1, 2, ..., m$, let $\beta_j = 1$ if a change-point is detected in $C_q$ during period $j$, and $\beta_j = 0$ if no change-point is detected in $C_q$ during this period. One might observe that for many $C_q$'s, $\bar{\alpha}_i > \bar{\beta}_j$, where $\bar{\alpha}_i$ and $\bar{\beta}_j$ are correspondingly the empirical average of $\alpha_i$'s and $\beta_j$'s. It can be hypothesized that the distribution of $\alpha_i$'s is *stochastically greater* than the distribution of $\beta_j$'s. The distribution of the random variable $\alpha \in \mathcal{R}$ is stochastically greater than the distribution of the random variable $\beta \in \mathcal{R}$ if:

$$\Pr(\beta > x) < \Pr(\alpha > x) \text{ for all } x \in \mathcal{R},$$

where Pr(.) is the probability of an event. Whether the distribution of $\alpha_i$'s is stochastically greater than the distribution of $\beta_j$'s or not, can be tested using a one-sided Mann-Whitney U test, introduced in Mann and Whitney (1947). This test is formulated as:

$$\begin{aligned} H_0 &: \Pr(\alpha > \beta) \leq \Pr(\alpha < \beta) \\ H_1 &: \Pr(\alpha > \beta) > \Pr(\alpha < \beta) \end{aligned}$$

This formulation is for the one-sided Mann-Whitney U test. There exists a two-sided version of this test, which is formulated as:

$$\begin{aligned} H_0 &: \Pr(\alpha > \beta) = \Pr(\alpha < \beta) \\ H_1 &: \Pr(\alpha > \beta) > \Pr(\alpha < \beta) \quad \text{or} \quad \Pr(\alpha > \beta) < \Pr(\alpha < \beta) \end{aligned}$$

In this work we only apply the one-sided Mann-Whitney U test and therefore we refer to this test as the Mann-Whitney U test.

The Mann-Whitney U test is a non-parametric test, where the assumptions are:

1. The samples being compared to one another have the same distribution except for a possible shift in mean.
2. The observations are independent of each other.
3. The variables are ordinal, i.e., from two observations one can in a meaningful way say which one is the greatest.

The Mann-Whitney U test can be applied to compare the distribution of any two random variables that fulfill the assumptions listed above. We will take advantage of the Mann-Whitney U test for this purpose many times in our analysis, for example for comparing the performance of models. However, here the test is described in the context of the selection of the signal of interest.

One must justify that the $\alpha_i$'s and the $\beta_j$'s, as described here fulfill the assumptions made by the Mann-Whitney U test. The first assumption means that under the null hypothesis, $H_0$, the distributions from which the $\alpha_i$'s and the $\beta_j$'s are drawn have to be equal, and under the alternative hypothesis, $H_1$, they are only non-equal in mean, and otherwise similar in form. One could think that this assumption might not be fulfilled in the case at hand, since the $\alpha_i$'s belong to transition state periods and the $\beta_j$'s belong to normal state periods. On the other hand, we argue that both the $\alpha_i$'s and the $\beta_j$'s are the number of change-points detected on the same signal using the same change-point detection method, and whether the system is in a transition state or a normal state is a random event. Therefore it is reasonable to assume the distributions from which the two samples come from are of the same form and the first assumption is fulfilled. The other two assumptions are clearly plausible to make.

For $\alpha_1, \alpha_2, ..., \alpha_n, \beta_1, \beta_2, ..., \beta_m$, put the two sets into one set, $\gamma$, of size $n+m$ and sort $\gamma$ in an ascending order. Assign each member of $\gamma$ its numeric rank $\zeta(\gamma_i)$, starting with 1 for the smallest member and adjusting for ties such that the rank is set equal to the mean of the unadjusted rankings. For instance for $\alpha_1 = 4$, $\alpha_2 = 7$, $\alpha_3 = 11$, $\beta_1 = 8$, $\beta_2 = 8$, one obtains $\gamma = (\alpha_1 = 4, \alpha_2 = 7, \beta_1 = 8, \beta_2 = 8, \alpha_3 = 11)$, and one obtains $\zeta(\alpha_1) = 1$, $\zeta(\alpha_2) = 2$, $\zeta(\beta_1) = \frac{3+4}{2} = 3.5$, $\zeta(\beta_2) = \frac{3+4}{2} = 3.5$, $\zeta(\alpha_3) = 5$. Now define the test-statistic $U$ as following:

$$U = \sum_{j=1}^{m} \zeta(\beta_j)$$

This formulation of $U$ differs slightly from the original formulation by Mann and Whitney (1947), and is taken from Zar (1998). The two formulations give the same value for $U$. The distribution of $U$ is derived in Mann and Whitney (1947). Let $U_{obs}$ be the observed value for $U$. If $\Pr(U \leq U_{obs}) \leq \alpha$ under the null hypothesis, the null hypothesis will be rejected at the significance level $\alpha$. Mann and Whitney (1947) also gives a table providing the largest value of $\alpha$ on which $H_0$ is rejected for any value of $U_{obs}$ and $m$. Larsen and Marx (1981) present the Mann-Whitney U test as a variant of Wilcoxon test and show that for large $nm$, $U$ is approximately normally distributed with mean $\frac{n(n+m+1)}{2}$ and variance $\frac{nm(n+m+1)}{12}$. Jones et al. (2001–) argue that it is reasonable to use normal approximation when $nm > 20$.

In order to choose which linear combination to use, one can compare the $p$-value of their corresponding Mann-Whitney U test, and pick the linear combination, $C_q$, with the lowest $p$-value when the Mann-Whitney U test is performed on its corresponding $\alpha_i$'s and $\beta_j$'s. If the null hypothesis in the Mann-Whitney U test is rejected for $C_q$ on the significance level $\alpha = 0.05$, one could claim that on average transition state periods include change-points more often than normal state periods in the $q$-th column of the transformed data, when the transformation is done according to the semi-supervised method introduced in Section 3.4.3 using a PCA or a BTA transformation. If one is interested in a multivariate predictive model, one can select $p$ signals of interest by running this test on every $p$-tuple of columns, selecting the $p$-tuple yielding the lowest $p$-value.

### Discrepancy, Leverage and Influence

In the previous two method, we tried to select one linear combination among many to reduce the dimensionality. However, here we want to combine all linear combinations to obtain a signal of interest. The idea presented here is inspired by Shyu et al. (2003), who called their method for Principal Component Classifier (PCC). They developed an anomaly detection method using the discrepancy and leverage (not under these names) as defined here with a semi-supervised PCA approach as explained in the previous subsections. We combine this idea with BTA and also introduce the influence as another possible signal of interest.

Let $x = \{x_t\}_{t=1}^{N}$ be a $r$-dimensional signal, and $\mathbf{X}$ be its corresponding $N \times r$ matrix. For each column of $\mathbf{X}$, subtract the mean value in the column from the observations in that column such that the data is centered around 0. Now create a PCA or a BTA transformation using $\mathbf{X}$. In the case of PCA, select a $s < r$, such that the first $s$ principal components explain a large proportion, $\kappa$, of the variance in the data. For instance one could set $\kappa = 0.90$. In the case of BTA, select a $s < r$, such that the first $s$ components explain a large proportion, $\kappa$, of the predictability in the data. For a new data point, use the same mean values calculated for the columns of $\mathbf{X}$ to centre the data point around 0, and call the new data point for $x_{new}$. Now use the transformation built based on $\mathbf{X}$ to transform $x_{new}$, and obtain $T(x_{new})$, which is an $r$-dimensional vector. Here it is assumed that $\mathbf{X}_i$ has full rank. This means $N_i - 1 > r$ and the rank of $\mathbf{X}_i$ is $r$. Refer to $T(x_{new})$'s $i$-th element with $T_i(x_{new})$.

The *leverage*, $l$, of $T(x_{new})$ is defined as:

$$l = \left( \sum_{i \leq s} \frac{1}{\lambda_i^{'}} \big(T_i(x_{new})\big)^2 \right)^{\frac{1}{2}},$$

where $\lambda_i^{'}$ is the variance of $PC_i$ or $BT_i$ in the case of PCA and BTA, respectively. As stated in Subsection 3.4.1, for PCA, $\lambda_i^{'}$ is the $i$-th largest eigenvalue of the covariance matrix $\mathbf{\Gamma}$. This is the same as $\lambda_i$ defined in Subsection 3.4.1. For BTA, one can show that $\lambda_i^{'}$ is the $i$-th diagonal element of the matrix $\mathbf{\Omega}^T \mathbf{\Gamma} \mathbf{\Omega}$, where $\mathbf{\Gamma}$ and $\mathbf{\Omega}$ are defined in Subsection 3.4.2. The true covariance matrix $\mathbf{\Gamma}$ as well as the true $\mathbf{\Omega}$ are often unknown in practice. In this case, the estimated matrices $\hat{\mathbf{\Gamma}}$ and $\mathbf{L}$ should be used instead of $\mathbf{\Gamma}$ and $\mathbf{\Omega}$, respectively. The estimated covariance matrix $\hat{\mathbf{\Gamma}}$ is calculated using the method presented in Subsection 3.4.4, while the matrix $\mathbf{L}$ is calculated according to the diagonalization in (3.22).

The *discrepancy*, $d$, of $T(x_{new})$ is defined as:

$$d = \left( \sum_{i > s} \frac{1}{\lambda_i^{'}} \big(T_i(x_{new})\big)^2 \right)^{\frac{1}{2}},$$

where $\lambda_i^{'}$'s are defined in the same way as for the leverage.

The *influence*, $c$, of $T(x_{new})$ is defined as:

$$c = l \times d.$$

Any of the signals $l$, $d$ or $c$ can be selected as the signal of interest. In the case of BTA, $l$ will have a large predictability under normal operation mode and therefore a deviation from its predictable trajectory is a sign of anomaly that can be a source of failure in the system. The discrepancy doesn't have a large predictability and will therefore include many normal changes. However an anomalously large change in this signal means that the change is in a normal direction of the data but of a non-normal amplitude, and this might also be a good predictor for failures in the system. The two signals are combined in the influence, $c$, which will show changes in both predictable and non-predictable directions of the data.

In the case of PCA, $d$ will have a small variance under the normal operation mode and therefore a deviation from its almost constant trajectory is a sign of anomaly that can be a source of failure in the system. The leverage has a large variance under the normal operation mode and will therefore include many normal changes. However an anomalously large change in this signal means that the change is in a normal direction of the data but of a non-normal amplitude, and this might also be a good predictor for failures in the system. The two signals are combined in the influence, $c$, which will show changes in both predictable and non-predictable directions of the data.

It is not clear which of the three signals leverage, discrepancy and influence should be picked as the signal of interest in any problem. The choice of the parameter $\kappa$ is not trivial either. In the case of PCA, $\kappa$ is the proportion of the variance in the data explained by the first $s$ principal components. In the case of BTA, $\kappa$ is the proportion of the predictability in the data explained by the first $s$ Box-Tiao components. In this analysis, we use $\kappa = 0.90$. One may use a change-point detection method with a Mann-Whitney U test to select the *best* signal of interest. This can be done in a similar way to what was described previously for the Mann-Whitney U test method, with the difference being that the alternative signals of interest are now the discrepancy, the leverage and the influence instead of linear combinations of the data. This means $C_q$ for $q = 1, 2, 3$ is now the discrepancy, the leverage and the influence. One should then select the signal of interest with the most significant Mann-Whitney U test. If desired, this could be repeated for different choice of $\kappa$ selecting the value which yields the most significant Mann-Whitney U test. This is however not done in this work.

**Discussion**

One of the methods presented in this subsection should be used for selection of the signal of interest. There are advantages and disadvantages with both methods. On one hand, there might exist only one PCA or BTA component which contains a large amount of predictive information. In this case combining several components to construct the discrepancy, the leverage and the influence only adds noise to the signal of interest. On the other hand, combining several PCA or BTA components into one signal of interest by calculating the discrepancy, the leverage and the influence might yield a signal of interest which contains more predictive information than each of the components alone. The proper approach for every business case should be determined by experimentation.

The offline change-point detection presented in Section 3.3 is a computationally demanding task. For every alternative signal of interest, this method is applied to all the running groups in the training data set for signal selection. Using the discrepancy, the leverage and the influence, there are only three alternative signals of interest for the Mann-Whitney U test. Using the PCA or the BTA components, there are as many alternatives as there are linear combinations. In our analysis this is equal to the number of dimensions in the data, $r$. If $r$ is large, this will be a computationally demanding task.

The Mann-Whitney U test method allows in theory for selection of as many signals of interest as desired. In this way a multidimensional predictive model can be developed. The computational complexity of this method grows however exponentially with the desired number of signals of interest. Using the discrepancy, the leverage and the influence, one can develop a predictive model of maximum three dimensions, using all three signals for prediction. This number cannot be surpassed using the PCA or the BTA components in our business case due to the computational complexity.

## 3.5 Online Predictive Models

As predictive model, we propose to use statistical control charts. The purpose of a control chart is to determine if a process is behaving as it is expected to. In the literature of control charts, the term *in-control* is used to refer to the normal state of a process and the term *out-of-control* is used to refer to the anomaly states. The fundamental idea behind control charts is similar to that of hypothesis testing. One wants to control the probability of concluding that the process is out-of-control when in fact it is not, while maximizing the detection power of the out-of-control state.

First the in-control state should be defined on a variable, $s_t$. In our case $s_t$ is the signal of interest, selected in Section 3.4. For this, one needs some training data on $s_t$ from its in-control state. Using this data, one should establish an in-control state. In practice, this means estimating the expected value and the variance of $s_t$.

Walpole et al. (1993, Chapter 17) introduce the $\bar{X}$-chart as the simplest control chart. Here we will present this method to give the reader insight into the nature of statistical control charts. In this method, one assumes that in the in-control state, the observations of $s_t$ are independent and identically distributed in time. In addition, one assumes that the standard deviation and the expected value of $s_t$ are known. Let us refer to these as $\sigma_s$ and $\mu_s$, respectively. If $\sigma_s$ and $\mu_s$ are not known, one can use in-control training data to estimate these variables. Under these condition, one takes advantage of the Central Limit Theorem (CLT) to compute a Lower Control Limit (LCL) and an Upper Control Limit (UCL) in every time interval. Assume that in a time interval $k$, $n_k$ values of $s_t$ have been recorded. Let us refer to these as $s_1, ..., s_{n_k}$. Then the *sample* *mean* of $s_t$ in this time interval is defined as:

$$_k\bar{s}_t = \frac{1}{n_k}\sum_{t=1}^{n_k} s_t.$$

According to the CLT, we have under the condition that the process is in-control:

$$_k\bar{s}_t \sim \mathcal{N}(\mu_s, \frac{\sigma_s^2}{n_k}). \tag{3.23}$$

For some $\alpha < 1$, consider the following limits:

$$\begin{aligned}\text{LCL} &= \mu - z_{\alpha/2}\frac{\sigma_s}{\sqrt{n_k}}\\ \text{UCL} &= \mu + z_{\alpha/2}\frac{\sigma_s}{\sqrt{n_k}},\end{aligned}$$

where $z_{\alpha/2}$ is the $Z$-score for $\frac{\alpha}{2}$, being computed from the *standard normal distribution*. Declare the process out-of-control only when $_k\bar{s}_t >$ UCL or $_k\bar{s}_t <$ LCL. Equation (3.23) tells us that for $100(1-\alpha)\%$ of the in-control time intervals, the $_k\bar{s}_t$-values will be classified as in-control. In practice, $z_{\alpha/2}$ is often set to 3, as a rule of thumb.

In the industry, more sophisticated control charts are often used for the purpose of quality control of industrial processes. The Exponentially Weighted Moving Average (EWMA) chart, originally introduced by Roberts (1959), and the Cumulative Sum (CUSUM) chart, originally proposed by Page (1954), are two of the most popular ones. According to Arnold et al. (2018), the main advantage of the CUSUM chart compared to the EWMA and the $\bar{X}$ charts is that each point on a CUSUM chart is based on information from all the observations up to and including the current one. In addition, a CUSUM chart is more efficient for detecting small but consistent shifts in the mean of the signal of interest. In the rest of this section we will present the CUSUM method, which will be applied as a predictive model in our experiments.

### 3.5.1 Cumulative Sum (CUSUM)

The Cumulative Sum (CUSUM) control chart is often used as an online change-point detection tool. Let us assume $s_t$ has a constant expected value $\mu_s$ and standard deviation $\sigma_s$ when the process $s_t$ is in-control. Now at a time point $t = i$ define $C_i$ in the following way:

$$\begin{aligned}C_0 &= 0,\\ C_i &= \sum_{t=1}^{i}(s_t - \mu_s) \quad \text{for } i > 0.\end{aligned} \tag{3.24}$$

This can also be written as:

$$\begin{aligned}C_0 &= 0,\\ C_i &= C_{i-1} + (s_t - \mu_s) \quad \text{for } i > 0.\end{aligned}$$

If the process is in-control for $t < i$:

$$E[s_t - \mu_s] = 0 \Rightarrow E[C_i] = \sum_{t=1}^{i} E[s_t - \mu_s] = 0,$$

where $E[.]$ is the expected value function. In CUSUM, one declares the process to have become out-of-control if $C_i$ is sufficiently different from 0. In other words, at any time point $i$, one performs the following hypothesis test:

$$\begin{aligned} &H_0 : E(s_1) = E(s_2) = ... = E(s_i) = \mu_s, \\ &H_1 : E(s_t) \neq \mu_s \quad \text{for some } t \leq i. \end{aligned}$$

The following test statistics are used:

$$\begin{aligned} C_i^+ &= \max(0, s_i - (T + K) + C_{i-1}^+), \\ C_i^- &= \max(0, -s_i + (T + K) + C_{i-1}^-), \end{aligned} \tag{3.25}$$

where max(., .) is the maximum function, $T$ is called the *target value* for $s_i$ and $K$ is called the *acceptable margin*. The target value is the expected value of $s_i$ given that the process is in-control, while $K$ is the acceptable deviation of $s_i$ from the target value under the in-control state. For some constant threshold $h$, the process is declared to have become out-of-control if either $C_i^+ > h$ or $C_i^- > h$. One can create a one-sided test by only looking at one of $C_i^+$ or $C_i^-$. One should only look at $C_i^+$ if one is only interested in detecting when the observed values are higher than the target, while one should only look at $C_i^-$ if one is only interested in detecting when the observed values are lower than the target.

Determining $T$, $K$ and $h$ is a non-trivial problem. We call these the *CUSUM features*. If $s_t$'s are independent and identically distributed and follow a Gaussian distribution with different means under the in-control and out-of-control states, then one can use a training data set from the in-control state to set reasonable values for CUSUM features. One should estimate the mean and the standard deviation of $s_t$ when the process is in-control. Let us call these estimators $\hat{\mu}_s$ and $\hat{\sigma}_s$, respectively. It is assumed that the training data set used to calculate $\hat{\mu}_s$ and $\hat{\sigma}_s$ is large enough, such that the estimation error is kept under control. Then it is reasonable to set $T = \hat{\mu}_s$. Some widely used rules of thumb for the other two variables are: $K = \frac{1}{2}\hat{\sigma}_s$ and $h = 5\hat{\sigma}_s$. These work well under the independence and normality assumptions and for detection of changes in the mean of $s_t$ of an amplitude greater than or equal to $\hat{\sigma}_s$.

If one or several of the assumptions mentioned above are violated, determining the CUSUM features will be a substantially more difficult task. In our case, serial dependence in time series data is a challenge that should be addressed.

In this work, we present an ad hoc method to determine the CUSUM features. A competing approach is provided by Aue and Horváth (2013). They present a review on the work done to modify the original CUSUM method to also work for data exhibiting serial dependence. Using a different notation from ours, Aue and Horváth (2013) show that the CUSUM features for serially dependent data should be of the same form as for independent data but differ in a scaling parameter $\omega$ on $K$ and $h$. Aue and Horváth (2013) refer to $\omega^2$ as the long-run variance, and provides estimators for $\omega^2$ under different assumptions about the distribution of the data. Using $\omega^2$, Aue and Horváth (2013) formulate the CUSUM method as a hypothesis test under different assuptions about the data. A test statistic $M_i$ is also provided for the hypothesis test. The asymptotic distribution of $M_i$ is tabulated in Shorack and Wellner (1986).

Lu and Reynolds Jr (2001) provide a performance analysis of the CUSUM method when applied to autoregressive data. The idea is to test if CUSUM performs better when applied to the residuals after fitting an autoregressive model to the data than it does when applied to the original data. Lu and Reynolds Jr (2001) conclude that CUSUM based on original data performs as well as CUSUM based on the residuals except in the case in which both the level of autocorrelation and the shift in the mean are high.

Here we will use CUSUM as a predictive model on our signal of interest selected in Section 3.4. We will perform the following tuning method to determine the CUSUM features.

Assume that we have access to a training data set. This training data should be from our signal of interest $s_t$, in both periods of normal operation mode and periods prior to a failure in the system. One can use the same data set used for signal selection in Subsection 3.4.5, and transform it to arrive to a 1-dimensional signal of interest. Assume that each time point in this data set is labeled as belonging to either a normal state, a transition state or a failure state, as described in Subsection 3.4.5. In Figure 3.4, one can see that the different states are not necessarily of the same length. One could therefore think that the probability of declaring the process out-of-control by the model might increase with the length of the states. Since the normal states last usually much longer than the transition states, this might introduce an undesirable bias both into the selection of the CUSUM features. To account for this bias, we will divide all normal states in intervals of the same length as the transition states. This is illustrated in Figure 3.5.

In this part of the analysis we will only use data from the normal groups for simplicity. Use the data in the normal states to compute the sample mean and the sample variance of $s_t$. Let the normal states be labeled as $j = 1, 2, ..., J$, and each be of length $\nu - f$ as shown in Figure 3.5. Refer to the sub-signal of interest corresponding to the normal state $j$ as ${}_j s_t$. Then the sample mean and the sample variance of $s_t$ under the normal states is computed in the following

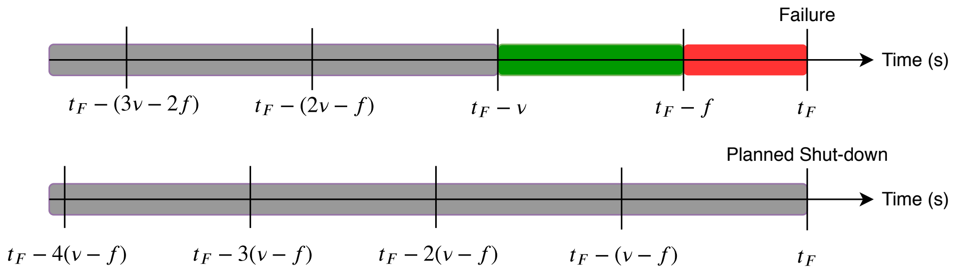


**Figure 3.5:** The first time line is prior to a shut-down due to a failure in the system, and the second time line is prior to a normal planned shut-down. The red, green and grey color mark respectively the time referred to as a failure state, a transition state and a normal state. The normal state is divided into intervals of lengths equal to that of transition states.

way:

$$\begin{aligned}
{}_j\hat{\mu}_s &= \frac{1}{\nu-f}\sum_t {}_js_t,\\
{}_j\hat{\sigma}_s^2 &= \frac{1}{\nu-f-1}\sum_t ({}_js_t-{}_j\hat{\mu}_s)^2,\\
\hat{\mu}_s &= \frac{1}{J(\nu-f)}\sum_j(\nu-f)_j\hat{\mu}_s\\
\hat{\sigma}_s^2 &= \frac{1}{J(\nu-f-1)}\sum_j(\nu-f-1)_j\hat{\sigma}_s^2.
\end{aligned} \tag{3.26}$$

The target value $T$ for CUSUM is set to $\hat{\mu}_s$. It remains to determine $K$ and $h$. One should now determine a *searching space* for $K$, where the optimal $K$ is picked among some alternatives. We will use $K^{\text{set}} = \{\frac{\hat{\sigma}_s}{2}, \hat{\sigma}_s, 2\hat{\sigma}_s, 4\hat{\sigma}_s, 8\hat{\sigma}_s\}$. Now we need a searching space for $h$. For $K \in K^{\text{set}}$ execute the following algorithm:

1. Let $T = \hat{\mu}_s$ and compute ${}_jC_i$'s for all normal states following (3.24).
2. If any ${}_jC_i < 0$, set ${}_jC_i = -{}_jC_i$.
3. For each normal state $j$, compute $\max({}_jC_i)$. Save all the values in a set and call it $h^{\text{set}}$.
4. Sort $h^{\text{set}}$ in descending order.

The set $h^{\text{set}}$ is our searching space for $h$. Note that only the normal states that are a part of a normal group are used in determining the searching spaces for $h$ and $K$.

For every pair $(K,h) \in K^{\text{set}} \times h^{\text{set}}$, apply the CUSUM method to the training data. For this part of the analysis, we will use all the normal and the transition states in the training data. The performance of the CUSUM method is measured for each pair $(K, h)$ using one of the statistical metrics provided in Subsection 3.5.2. The pair $(K, h)$ yielding the best performance on the training data is selected.

Note that labeling the data points as illustrated in Figure 3.5 to determine the CUSUM features with an ad hoc approach makes our predictive model semi-supervised. In this approach, the normal data is labeled and the change from the normal state to the non-normal state is assumed to occur in a predetermined time-window. We do not assume that all the data points in the transition states are non-normal, as one would have assumed in a fully supervised approach. However, we assume that there exists a change-point in each transition state. This only implies that the data points that appear after the change-point has occurred are non-normal.

The estimation error when estimating $\mu_s$ and $\sigma_s$ is ignored in the method presented in this subsection. This can possibly affect the performance of the final predictive model. Gandy and Kvaløy (2013) suggest a method based on bootstrapping the data used to estimate these parameters to overcome this issue. Taking advantage of this method to estimate the distribution of the data in the in-control state is out of the scope of this work. The estimation error should however be kept in mind as a source of the prediction error when evaluating the model, specially since the unbiasedness of our estimators is not guaranteed for time series data with serial dependence.

### 3.5.2 Evaluation

The predictive model constructed with the CUSUM method needs to be tested. For this we present several statistical metrics that can measure the performance of the predictive model. Given a test data set including both normal and tripping groups, we divide the data into normal, transition and failure states as illustrated in Figure 3.5, and define the following concepts:

- True Positive (TP): A transition states in the test data during which the predictive CUSUM model has declared the process out-of-control at least once.

- False Positive (FP): A normal states in the test data during which the predictive CUSUM model has declared the process out-of-control at least once.

- True Negative (TN): A normal states in the test data during which the predictive CUSUM model has not declared the process out-of-control at all.

- False Negative (FN): A transition states in the test data during which the predictive CUSUM model has not declared the process out-of-control at all.

Denote the number of TP's, FP's, TN's and FN's with #TP, #FP, #TN and #FN, respectively. In a predictive model, it is desired to maximize #TP and #TN and minimize #FP and #FN. These values can be summarized in a *confusion matrix.* For a binary classification problem, such as our case, the confusion matrix is defined as:

$$\begin{bmatrix} \#\text{TN} & \#\text{FP} \\ \#\text{FN} & \#\text{TP} \end{bmatrix}$$

In Koehrsen (2018) the following metrics are presented to measure the performance of a model:

$$\begin{aligned} \text{Accuracy} &= \frac{\#\text{TP} + \#\text{TN}}{\#\text{TP} + \#\text{FP} + \#\text{TN} + \#\text{FN}} \\ \text{Precision} &= \frac{\#\text{TP}}{\#\text{TP} + \#\text{FP}} \\ \text{Recall} &= \frac{\#\text{TP}}{\#\text{TP} + \#\text{FN}} \\ \text{F1 score} &= 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \end{aligned}$$

The accuracy of a model is the proportion of the states identified correctly by the model. The precision is the proportion of the failure predictions made by the model that actually were true, the recall is the proportion of failures that the model could correctly predict, and the F1 score is the harmonic mean of the precision and the recall. The recall is also referred to as the *sensitivity* or the *true positive rate* in some literature. The precision is also referred to as the *positive predictive value* in some literature.

The simplest solution to the model selection problem is to select the model with the highest accuracy. This approach might however not work well if the data set is imbalanced. An imbalanced data set is a data set in which the number of observations belonging to one class is substantially larger than the number of observations belonging to the other classes. Given an imbalanced enough data set the model with the highest accuracy is the model that always predicts the majority class. This is the case in our business case, since the number of normal states is much larger than the number of transition states. Therefore the model with the highest accuracy is the model that never declares the process to be out-of-control and therefore maximizes #TN. This model will have #TP= 0, but this will not decrease the accuracy substantially since the maximum possible value of #TP is much smaller than the maximum value of #TN.

For an imbalanced data set one of the metrics recall, precision or F1 score should be used. To be able to predict more failures, one should increase #TP. An approach that selects the model with the highest precision maximizes #TP while minimizing #FP. In this case the final model might have a large #FN. Note that for very imbalanced data sets, the model is expected to have a low precision, since even with a small or moderate false positive rate ((#FP)/(#FP+#TN)), #FP might still be much larger than the maximum possible value of #TP. An approach that selects the model with the highest recall maximizes #TP, while minimizing #FN. In this case the final model might have a large #FP.

Every time the model declares the process to be out-of-control, the business owner has to take some actions to prevent a failure in the system. These actions involve a cost for the business owner. In case of a FP, the actions taken were actually not needed. In this case, we call the actions' cost for *the cost of a FP*. On the other hand, a failure in the system also involves a cost for the business owner. If the model doesn't declare the process to be out-of-control in the time prior to a failure, no actions can be taken by the business owner to prevent the failure. In this case, we call the failure's cost for *the cost of a FN*. If the cost of a FN is much higher than the cost of a FP, one should prioritize to maximize the recall. Otherwise the precision should also be kept as high as possible. In the cases where it is not clear which metric should be maximized, one could select the model with the highest F1 score, which is the harmonic mean of the precision and the recall.

# Chapter 4

# Experiments

In this chapter, we will design experiments to test the different approaches presented in Chapter 3. Looking at Figure 3.1, there are four possible ways to arrive at a 1-dimensional signal of interest given a multidimensional data set. One can use either the PCA or the BTA to transform the data. For each choice of transformation, one can either pick a linear combination as the signal of interest directly, or compute the discrepancy, the leverage and the influence and pick one of these as the signal of interest. We will test out all the four different approaches to some degree.

First we will simulate a data set based on the real-world data in our business case. The description of the real-world data and the details about how the data simulation is performed are presented in Section 4.1. The four different approaches are tested on the simulated data. The results can be compared at two levels. One can compare the performance of two approaches either by comparing the $p$-value of the Mann-Whitney U test on their corresponding selected signals of interest, or by using one of the statistical metrics presented in Subsection 3.5.2 on their corresponding CUSUM models using some test data set. The result of the different methods of evaluation might be different. If one uses the Mann-Whitney U test, a predictive model does not need to be developed for the simulation study. However, testing a predictive model based on the simulated data is beneficial not only for comparing the different approaches but also for proving the value of using the methods presented in Chapter 3 in failure prediction problems.

The main goal of the simulation study is to illustrate the power of the methods in predicting failures given the *the right type of data*. In addition, we want to determine if one gains a significant improvement in the model by using a BTA transformation instead of a PCA transformation. Another goal is to compare the different methods of selecting the signal of interest. Based on the results from the simulation study, one or several approaches will be applied to the real-world data.

## 4.1 Data Description

In this section the data used for the experiments is described. In Subsection 4.1.1, we describe the sensor-based data, which has been the source of inspiration for this work. This data is provided to us by the Norwegian company DNV GL. In Subsection 4.1.2, we present our method of data simulation. The data simulation is inspired by the real-world data.

### 4.1.1 Real-World Data

We will use real-world sensor-based data from 10 pairs of gas compressor and gas turbine units to test some of the methods presented in Chapter 3. Each gas compressor is powered by its corresponding gas turbine. The data is provided by a European Gas Transmission System (GTS) owner and operator, through the Norwegian DNV GL. As discussed in Chapter 1, the ultimate goal is to predict failures on these machines. One or several of the four approaches presented in Section 4.2 can be used for creating a signal of interest. One can decide on which approaches are applied based on the results from the simulation study. A CUSUM method can be developed as a predictive model and the performance of the model is assessed using the measures presented in Section 4.4.

There is data available on 10 systems from 2013 til 2017. The possible reasons for a failure can be bearing problems, seals, lube oil system issues, process conditions or combustion related. Failures are further divided into two groups of *starting trips* and *running trips* depending on when the failure happens. A starting trip is a failure to start the machine and a running trip is a failure that happens while the machine is running. Here the scope of the analysis is reduced by only looking at the running trips. The streams of data available for this analysis are sensor-based values related to measurements in physical quantities such as temperature, speed, efficiency, pressure, etc.

These gas compressors are used to compress gas in some pipelines, and they do not run continuously in time. They are turned on when needed and switched off when the pressure of the gas in the pipeline reaches the desired value. Therefore the data is divided into different running groups, as described in Section 1.2. The running groups are categorized based on their type of shut-down. The possible categories are normal stop, running trip and starting trip. As discussed in Section 1.2, we ignore the running groups ending in a starting trip and refer to the running groups ending in a normal stop and a running trip as a normal group and a tripping group, respectively.

The length of each running group varies from about one hour to about 43 days. All the running groups that are shorter than 180 minutes are removed from the data. Then the first 45 minutes of each running group is removed in order to avoid detection of change-points in the starting phase, as it is normal to make adjustments in the starting phase until the compressor reaches the desired operational mode. For normal groups, the last 15 minutes are also removed. This is because the process of shutting down the compressor with a normal stop takes around 15 minutes, and change-points in this time interval are of no interest. An unplanned shut-down sequence usually takes between 5 to 10 minutes. Therefore the last 10 minutes of the tripping groups are also removed from the data.

The raw data provided by the GTS operator company is not suitable for analysis. Different sensors register values with different frequencies and the time of measurements are not synced. A data preparation process is performed and the data is re-sampled with a frequency of one minute. This is done by DNV GL. DNV GL has provided a detailed description of the data preparation process in a report presented in Appendix A. The data provided to us by DNV GL still includes some Not Available (NA) values. Therefore we perform a data cleaning process.

If the value recorded by a sensor is NA in more than 10% of the time, the sensor is removed from the analysis. Then within a running group, the NA values are replaced with the Last Observation Carried Forward (LOCF). If the data from a sensor is NA during a whole tripping group, the sensor is removed from the analysis. This is because we want to avoid removing tripping groups from the analysis due to their scarcity. If the data from a sensor is NA during a whole normal group, the normal group is removed.

After the data manipulation described in the last two paragraphs is performed, the number of streams, $r$, left for this analysis is 62. The 62 dimensions of the real-world data set, after the data manipulation is performed, are listed in a table in Appendix B. The number of remaining tripping groups is 76 and the number of remaining normal groups is 497. The distributions of the lengths of the remaining normal groups and the remaining tripping groups are illustrated with histograms in Figures 4.1 and 4.2, respectively. The length of the longest remaining normal group is 61806 minutes while that of the longest remaining tripping group is 18751 minutes. In Figures 4.1 and 4.2, one can observe that most of the remaining running groups are shorter than 1000 minutes.

The real-world data might need to be re-scaled prior to the analysis to obtain the best possible performance by some of the methods like PCA. This is because if there is a large difference between the variance of the different dimensions, the first principal components will be dominated by the dimensions with the largest variances. Note that if the unit in which a physical quantity is measured changes from a large to a small unit, its variance will increase. As a result it may go from having little impact to dominating the first principal component. The PCA transformation should not be dependent on such re-scaling. Therefore if there is a large difference between the variance of the different dimensions, all the dimensions should be re-scaled to have variance 1. In Figure 4.3, the logarithm of the estimated variance of each of the 62 dimensions is shown separately in the normal groups and in the tripping groups. The dimensions are sorted in the descending order of their variance in the normal groups. The rank of each dimension in Figure 4.3 can be found in a table in Appendix B. The estimated variances are calculated in the following way: Let $G_j$ refer to the time interval corresponding to running group $j$, and let $|G_j|$ be the length of $G_j$. Let $d_i$ refer to the $i$-th dimension in the data and let $d_{i,t}$ be the value of $d_i$ at time point

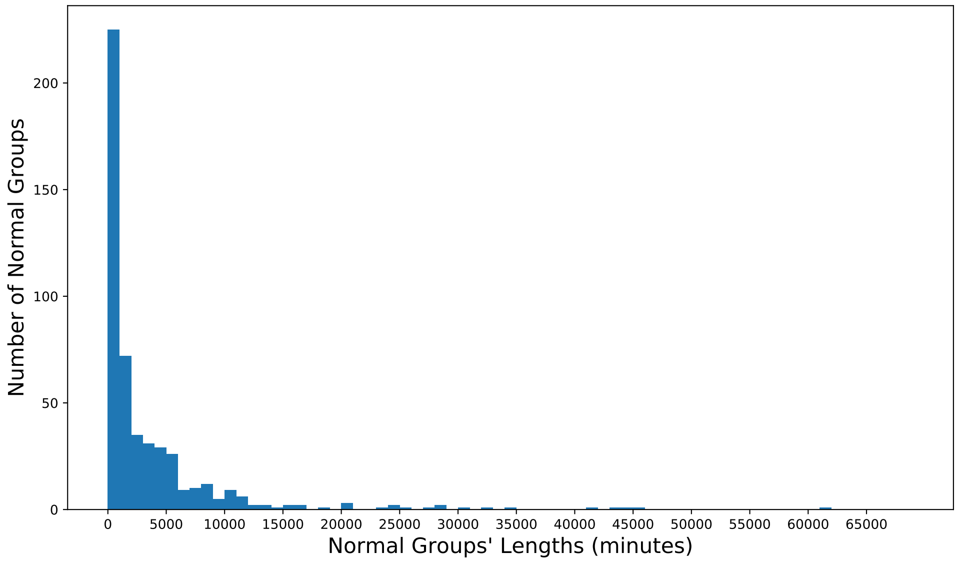


**Figure 4.1:** Histogram of the lengths of the normal groups in the real-world data. Each bar covers 1000 minutes. The height of a bar shows the number of normal groups the in the respective interval.

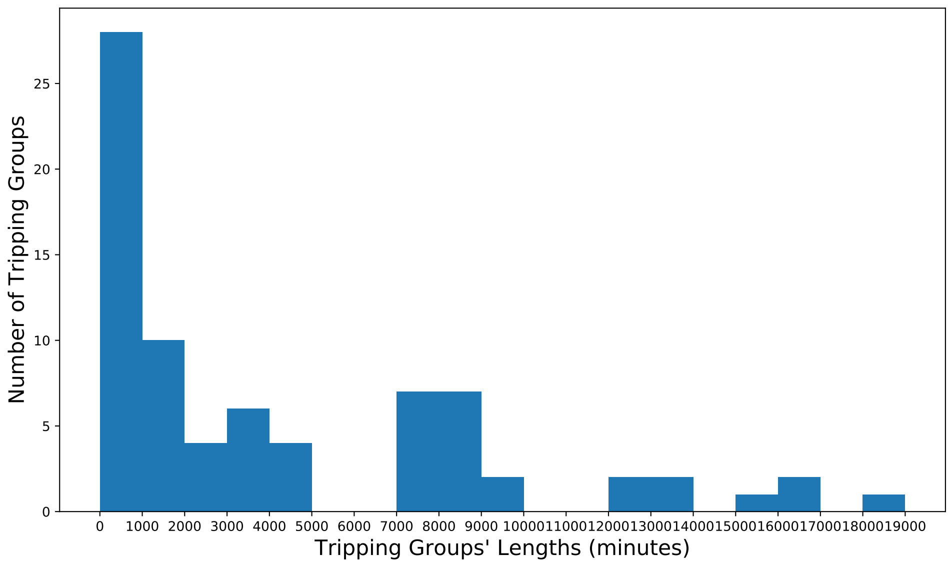


**Figure 4.2:** Histogram of the lengths of the tripping groups in the real-world data. Each bar covers 1000 minutes. The height of a bar shows the number of tripping groups in the respective interval.

$t$. Finally, let $NG$ and $TG$ be the set of normal groups and tripping groups, respectively. Then:

$$\begin{aligned}
{}_j\hat{\mu}_{d_i} &= \frac{1}{|G_j|}\sum_{t\in G_j} d_{i,t}, \\
{}_j\hat{\sigma}^2_{d_i} &= \frac{1}{|G_j|-1}\sum_{t\in G_j} (d_{i,t} - {}_j\hat{\mu}_{d_i})^2, \\
{}_{\text{Normal}}\hat{\sigma}^2_{d_i} &= \frac{1}{\sum_{j\in NG}(|G_j|-1)}\sum_{j\in NG} (|G_j|-1)_j\hat{\sigma}^2_{d_i}, \\
{}_{\text{Tripping}}\hat{\sigma}^2_{d_i} &= \frac{1}{\sum_{j\in TG}(|G_j|-1)}\sum_{j\in TG} (|G_j|-1)_j\hat{\sigma}^2_{d_i},
\end{aligned} \tag{4.1}$$

where ${}_j\hat{\mu}_{d_i}$ and ${}_j\hat{\sigma}^2_{d_i}$ are respectively the sample mean and the sample variance of dimension $d_i$ during running group $j$ and ${}_{\text{Normal}}\hat{\sigma}^2_{d_i}$ and ${}_{\text{Tripping}}\hat{\sigma}^2_{d_i}$ are the sample variance of dimension $d_j$ in the normal groups and the tripping groups, respectively. One can see in Figure 4.3 that there is substantial differences between variances of the 62 dimensions. Therefore, the data needs to be rescaled.

One can also see in Figure 4.3 that the estimated variances of the different dimensions in the tripping groups do not differ much from that in the normal groups. This might be because the process is mostly in a normal state even during the tripping groups. This means that most of the data used for the estimation of the variances in the tripping groups comes actually from a normal state. Therefore it might be interesting to calculate the estimated variances using only the data from the transition states. This will be presented in Chapter 5.

From the 497 normal groups and the 76 tripping groups, 421 normal groups are randomly selected for constructing the PCA and the BTA transformations. This is called the *training set*. Another 38 normal groups together with 38 tripping groups are randomly selected and used for the selection of the signal of interest. This is called the *signal selection set*. Note that for the selection of the signal of interest, the data is divided into normal, transition and failure states following the method presented in Subsection 3.4.5. In Figure 3.4, one can see that for every normal group, there exists a normal state and for every tripping group, there exists a normal, a transition and a failure state. This means that there are twice as many normal states as there are transition states in the signal selection set. Finally the 38 remaining normal groups together with the 38 remaining tripping groups are used for evaluating the predictive model. This is called the *test set*. Note that the signal selection set is also used for the training of the predictive model. For this purpose, the data in the signal selection set is divided into normal, transition and failure states following the method presented in Subsection 3.5.1. In Figure 3.5, one can see that following this method, the number of normal states will be substantially larger than the number of transition states in the signal selection set. The same is the case for

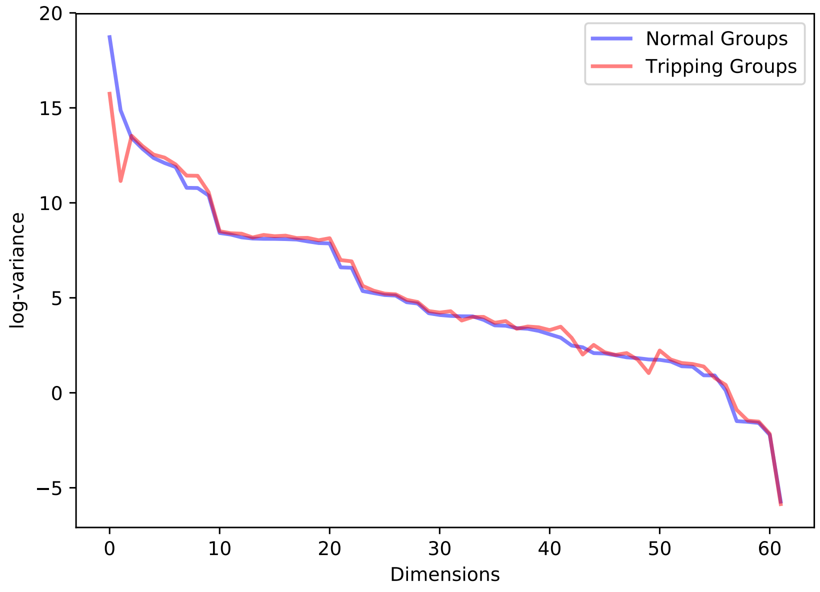


**Figure 4.3:** The natural logarithm of the estimated variance of each of the 62 dimensions in the real-world data is plotted. The estimations for the normal groups and the tripping groups are performed separately. The estimation is performed using the method presented in (4.1). The dimensions are sorted in the descending order of their estimated variance in the normal groups. The rank of each dimension can be found in a table in Appendix B.

the test set.

For each dimension $d_i$, the mean and the variance of $d_i$ in the normal states are estimated in the same way as in (4.1) using the 421 normal groups in the training set. Then all the values in $d_i$ are centered around its estimated mean and divided by the square root of its estimated variance in the normal states.

### 4.1.2 Simulated Data

We perform 100 simulations, each containing a 10-dimensional data set with 600 normal groups and 100 tripping groups. A running group is a 10-dimensional time series with a frequency of one minute, where the data point $i$ represents *"measurements"* recorded by 10 *"sensors"* in minute $i$. The length of each running group is drawn from a discrete uniform distribution over the interval $[300, 700]$. The normal groups and the tripping groups are initially generated by the same VAR process within each simulation. Each data set is generated by a 10-dimensional random stationary VAR process of order 1. There exists a change-point in every running group. The change-points in the tripping groups are in a different dimension from the change-points in the normal groups.

In order to differentiate between normal and tripping groups, two different kinds of change-points can occur in the data, namely normal and non-normal change-points. Normal change-points appear only in normal groups while non-normal change-points appear only in tripping groups. For each simulation, let $d_n$ and $d_t$ be two distinct random positive integers less than or equal to 10, drawn from the discrete uniform distribution over the interval $[1, 10]$. Normal change-points occur in the mean of the $d_n$-th dimension of the data, while non-normal change-points occur in the mean of the $d_t$-th dimension. The amplitudes of the normal and non-normal changes are respectively equal to the estimated standard deviations of the $d_n$-th and the $d_t$-th dimension. In order to estimate the standard deviations of the $d_n$-th and the $d_t$-th dimension, first a signal of length 1000 minutes is simulated by the VAR process. The data in this signal is used to estimate the standard deviations of the $d_n$-th and the $d_t$-th dimension. Each normal group contains exactly one normal change-point while each tripping group contains exactly one non-normal one. Normal change-points can occur at any time during a normal group with equal probability for each minute. Non-normal change-points can only occur between 30 and 90 minutes prior to the end of a tripping group, with equal probability for every minute in this interval.

For each simulation, the VAR process is created in the following way:

1. Sample 10 distinct values from the continuous uniform distribution over the open interval $(-1, 1)$, and use them as the diagonal elements of the diagonal $10 \times 10$ matrix $\mathbf{D_A}$.

2. Generate a $10 \times 10$ random orthogonal matrix $\mathbf{Q_A}$, drawn from the $O(N)$ Haas distribution, as described in Mezzadri (2006).

3. Let $\mathbf{A} = \mathbf{Q_A D_A Q_A^T}$.

4. Generate a random symmetric positive-definite matrix $\mathbf{\Sigma}$ with the method provided in Pedregosa et al. (2011).

5. Let the VAR process be defined as $X_t = \mathbf{A} X_{t-1} + U_t$ for $t > 1$, where $U_t$'s are independent and identically distributed with $U_t \sim \mathcal{N}(0, \mathbf{\Sigma})$.

6. Let $x_1$ be a vector with 10 elements and set $x_1 = 0$.

7. For $t = 2, 3, ..., 1000$, set $x_t = \mathbf{A} x_{t-1} + u_t$ where $u_t$ is a random vector drawn from the multivariate Gaussian distribution $\mathcal{N}(0, \mathbf{\Sigma})$.

8. Using $x_t$'s, create the $1000 \times 10$ matrix $\mathbf{X}$, where for $t = 1, 2, ..., 1000$, the $t$-th row of $\mathbf{X}$ is $x_t$. Estimate the standard deviation of the $d_n$-th and the $d_t$-th column of $\mathbf{X}$, and call them $\hat{\sigma}_n$ and $\hat{\sigma}_t$, respectively. These will be the amplitudes of the changes in the mean of the $d_n$-th and $d_t$-th dimension, respectively.

Following this construction, $\mathbf{A}$ is guaranteed to have 10 real distinct eigenvalues with their absolute value less than 1. According to the theory presented in Subsection 3.2, this guarantees stationarity for the time series.

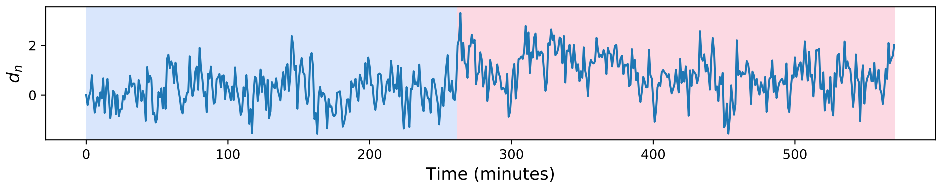


**Figure 4.4:** An illustration of the $d_n$-th dimension of a simulated normal group. There exists a normal change-point in this dimension in minute 262. The data before and after the change-point are marked with the blue and the red background color, respectively.

Given the VAR process in a simulation, the running groups are generated in the following way:

1. Draw a random integer $l$, from the discrete uniform distribution over the interval $[300, 700]$. This will be the length of the running group in minutes.
2. If the running group is a normal group, draw a random integer $\tau$ from the discrete uniform distribution over the half-open interval $[0, l)$. This is the time point at which a change will occur in the data.
3. If the running group is a tripping group, draw $\tau$ from the discrete uniform distribution over the half-open interval $[l - 90, l - 30)$.
4. Let $x_1$ be a vector with 10 elements and set $x_1 = 0$.
5. For $t = 2, 3, ..., l$, set $x_t = \mathbf{A}x_{t-1} + u_t$ where $u_t$ is a random vector drawn from the multivariate Gaussian distribution $\mathcal{N}(0, \mathbf{\Sigma})$.
6. If the running group is a normal group, add $\hat{\sigma}_n$ to the $d_n$-th element of $x_t$ for all $t \geq \tau$.
7. If the running group is a tripping group, add $\hat{\sigma}_t$ to the $d_t$-th element of $x_t$ for all $t \geq \tau$.

The simulated data is illustrated in Figures 4.4, 4.5, 4.6 and 4.7. Figures 4.4 and 4.5 are from one and the same simulated normal group while Figures 4.6 and 4.7 are from one and the same simulated tripping group. In the normal group, there is a change of amplitude 0.83 in the mean of the $d_n$-th dimension in minute 262, while there is no change in the $d_t$-th dimension in this running group. In the tripping group, there is a change of amplitude 3.48 in the mean of the $d_t$-th dimension in minute 528, while there is no change in the $d_n$-th dimension in this running group.

The methods presented in Chapter 3 can be tested on the simulated data. From the 600 normal groups and the 100 tripping groups generated in a simulation, 500 normal groups are used for constructing the PCA and the BTA transformations. This is called the training set. Another 50 normal groups together with 50 tripping groups are used for the selection of the signal of interest.

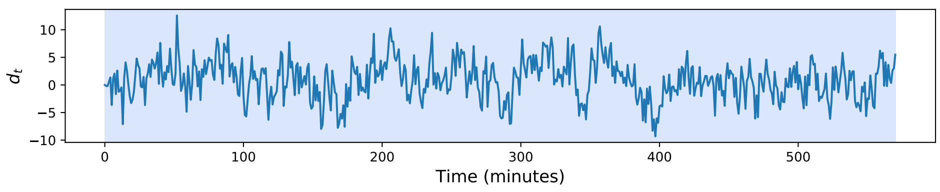


**Figure 4.5:** An illustration of the $d_t$-th dimension of a simulated normal group. There exists no change-points in this dimension during this running group.

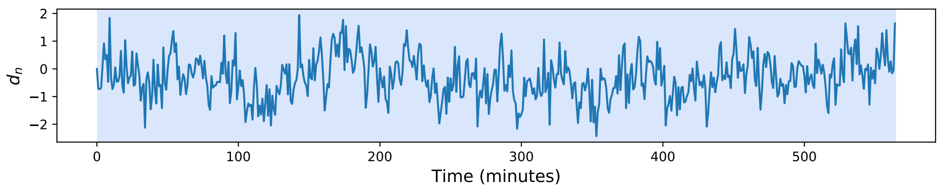


**Figure 4.6:** An illustration of the $d_n$-th dimension of a simulated tripping group. There exists no change-points in this dimension during this running group.

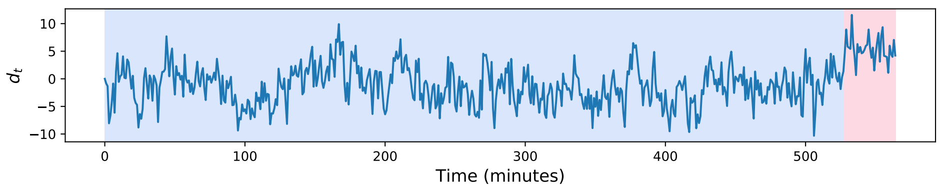


**Figure 4.7:** An illustration of the $d_t$-th dimension of a simulated tripping group. There exists an non-normal change-point in this dimension in minute $528$. The data before and after the change-point are marked with the blue and the red background color, respectively.

| Alternative Signals of Interest | PCA | BTA |
|---|---|---|
| BTA and PCA Components | Experiment 1 | Experiment 3 |
| Discrepancy, Leverage and Influence | Experiment 2 | Experiment 4 |

**Table 4.1:** The design of the experiments is illustrated in this table. The four experiments are used as dimensionality reduction approaches in this work. They differ in the type of transformation used in them and their alternative signals of interest among which one signal of interest is to be selected.

This is called the signal selection set. Finally 50 normal groups together with 50 tripping groups are used for evaluating the method. This is called the test set.

## 4.2 Experiment Design for Signal Selection

Here four experiments are designed to test the four possible approaches presented in Chapter 3. In each subsection the corresponding approach is marked in the analysis pipeline. All experiments are possible to perform both on the simulated and on the real-world data. We will however determine which experiments to perform on the real-world data based on the results from performing all four on the simulated data.

First, one should divide the data in three sets as explained in Section 4.1. The three sets are the training set, the signal selection set and the test set. The training set must include normal groups only. The majority of the normal groups should be in this data set since it is used to estimate the distribution of the data in the normal states and experimentation has shown that a large amount of data is required for convergent results in this part. The signal selection set and the test set must include both normal and tripping groups.

The four different experiments differ only in their method of constructing the signal of interest. In Experiments 1 and 2, a PCA transformation is used, while in Experiments 3 and 4, a BTA transformation is used. In Experiments 1 and 3, one of the PCA or BTA components is selected as the signal of interest while in Experiments 2 and 4, one of the signals discrepancy, leverage or influence is computed based on the PCA or BTA transformed data and used as the signal of interest. This is illustrated in Table 4.1. The method applied to construct a predictive model based of the signal of interest is common for all four experiments. Therefore only the approaches applied to obtain the signal of interest are described in every subsection and the description of the construction of the predictive model is presented for all experiments at the end of this section.

For the training of the predictive model, the running groups in the data are divided into different states as explained in Figure 3.5. The goal is to make a predictive model that predicts tripping at least once in the transition state of most tripping groups while it predicts no tripping in most normal states. The statistical metrics presented in Subsection 3.5.2 can be used for measuring and comparing the performance of the predictive model on the simulated test data for the four different approaches. The four experiments are performed on all of the 100 simulations. The four different approaches can therefore be compared by using their yielded mean value for a predetermined metric over the 100 simulations.

### 4.2.1 Experiment 1

In this experiment, one of the principal components is directly used as the signal of interest. The approach applied in this experiment is marked with red in the analysis pipeline in Figure 4.8. The PCA is performed as explained in Subsection 3.4.1. The PCA transformation has a semi-supervised nature as explained in Subsection 3.4.3. The training set is used to calculate the estimated covariance matrix $\hat{\mathbf{\Gamma}}$ of the data in the normal state. The estimation is performed according to the method presented in Subsection 3.4.4. Thereafter, $\hat{\mathbf{\Gamma}}$ is used to PCA-transform the rest of the running groups. This is done following the method presented in Subsection 3.4.1.

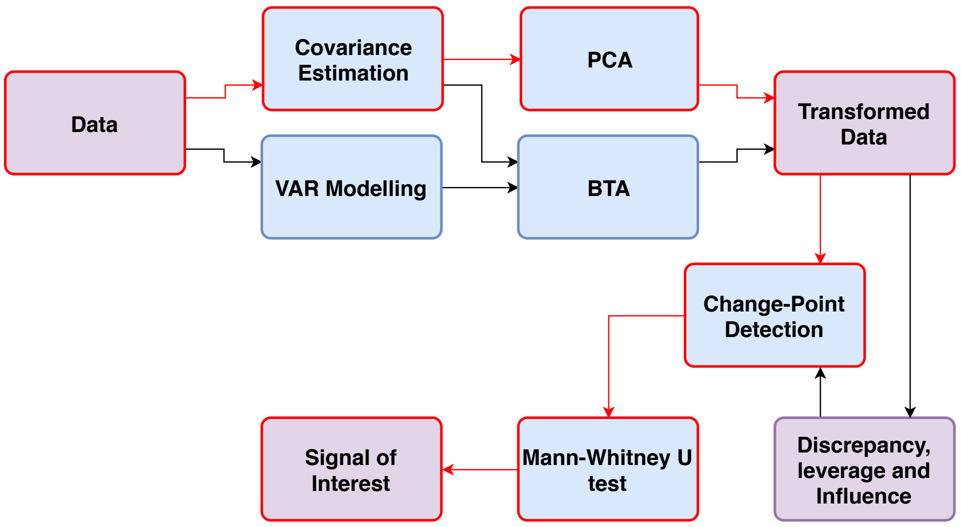


**Figure 4.8:** The analysis pipeline for Experiment 1. The performed analysis in this experiment is marked with the red color.

Within each running group in the signal selection set, an unsupervised offline change-point detection algorithm, based on the theory presented in Section 3.3, is applied to every principal component. With an unknown number of change-points, a penalty function is needed as explained in Subsection 3.3.3. The following penalty function, inspired from the BIC, is used in the algorithm:

$$p_{l_0}(\tau) = \Big(\frac{m}{2}\log(N)\Big)K, \tag{4.2}$$

where $N$ is the length of the running group, $K$ is the number of change-points, log is the natural logarithm function and $m$ is a constant. It is reasonable that the penalty function is a function of the length of the running group since the running groups can be of different lengths. The reason for this is explained both in Subsection 3.3.3 and Subsection 3.4.5. The constant $m$ in this equation is to be selected by the user. An ad hoc method can be applied to select this constant. For the simulated data, we set $m = 4$. The details about how $m$ should be selected and the assumptions under which the number of change-points is estimated consistently using this penalty function is out of the scope of this work. A brief overview on the literature is provided in Subsection 3.3.3. If the signals to which the change-point detection method is applied are of equal length, one can use the following penalty function, inspired by the AIC:

$$p_{l_0}(\tau) = \frac{m}{2}K,$$

where $K$ is the number of change-points and $m$ is a constant.

PELT, presented in Subsection 3.3.4, is used as the search method. For the cost function, (3.15) is used, assuming a 1-dimensional autoregressive process of order 1. This is because one could argue that a linear combination of the dimensions of some data that is generated by a VAR process is best modelled with an autoregressive model. For some linear combinations, this can be guaranteed. Note the following theorem proven by the author:

**Theorem** Let $X_t$ be an $r \times 1$ time series generated by a VAR process of order 1, such that for $t > 1$: $X_t = \mathbf{A}_1 X_{t-1} + U_t$, where $U_t \sim \mathcal{N}(0, \mathbf{\Sigma})$. Let $B$ be a $1 \times r$ vector. Then $Y_t = BX_t$ is a 1-dimensional time series that is a linear combination of the dimensions of $X_t$. If $B$ is a left eigenvector of $\mathbf{A}_1$, then $Y_t$ is generated by an autoregressive process of order 1.

**Proof** Assume that $B\mathbf{A}_1 = \lambda B$ and $V_t = BU_t$. One has:

$$\begin{aligned} X_t = \mathbf{A}_1 X_{t-1} + U_t &\Rightarrow BX_t = B\mathbf{A}_1 X_{t-1} + BU_t \Rightarrow \\ BX_t &= \lambda BX_{t-1} + B\mathbf{A}_1 X_{t-1} - \lambda BX_{t-1} + BU_t \Rightarrow \\ Y_t &= \lambda Y_{t-1} + B(\mathbf{A}_1 - \lambda \mathbf{I}_r)X_{t-1} + V_t. \end{aligned}$$

Now we will show that $B(\mathbf{A}_1 - \lambda \mathbf{I}_r)X_{t-1} = 0$:

$$B\mathbf{A}_1 = \lambda B \Rightarrow B(\mathbf{A}_1 - \lambda \mathbf{I}_r) = 0 \Rightarrow B(\mathbf{A}_1 - \lambda \mathbf{I}_r)X_{t-1} = 0.$$

Therefore one now has:

$$Y_t = \lambda Y_{t-1} + V_t,$$

where $V_t \sim \mathcal{N}(0, B\mathbf{\Sigma}B^T)$. It is proven that $Y_t$ is generated by an autoregressive process of order 1.

In the case of the simulated data, it is known that the data comes from a VAR process, while in the case of the real-world data, this is an assumption we make. One cannot guarantee that the principal components are best modelled by an autoregressive process, since they are not necessarily in the direction of one of the left eigenvectors of the matrix $\mathbf{A}_1$. However, for the offline change-point detection in this analysis, the cost function presented in (3.15) is used, assuming a 1-dimensional autoregressive process of order 1 for the principal components.

Following the method in Subsection 3.4.5 and Figure 3.4, every running group is divided into different states. The $f$ minutes before the end of a tripping group are labeled as a failure state, while the period between $\nu$ minutes before the end and $f$ minutes before the end of a tripping groups is labeled as the transition state. The rest of the time points in both normal and tripping groups are labeled as normal states as illustrated in Figure 3.4. In this experiment, $f = 15$ minutes for the simulated data and $f = 5$ minutes the real-world data. Note that the last 10 minutes of the tripping groups in the real-world data are previously removed from the data set. Therefore setting $f = 5$ minutes means we want to be able to predict the unplanned shut-downs at least 15 minutes before the speed of the compressor goes below 500 RPM. For the simulated data $\nu = 105$ minutes, and for the real-world data $\nu = 120$ minutes. For the simulated data, the length of the transition states is increased by 15 minutes in each direction compared to the true length of the transition states given in Subsection 4.1.2. This is done to allow for the early and the late detection of the transition state since it is possible that the algorithm detects a change-point before the transition state starts or after the transition state has ended. The acceptable margin is 15 minutes. The values for the real-world data are selected in consultation with the field experts. The change-points detected in each principal component during every running group in the signal selection set are used to calculate the $\alpha_i$'s and $\beta_j$'s needed to perform the Mann-Whitney U test, as described in Subsection 3.4.5. Using the values of the $\alpha_i$'s and the $\beta_j$'s, the Mann-Whitney U test is performed on every principal component, and the principal component with the lowest $p$-value is selected as the signal of interest.

Instead of the principal components, one could apply this experiment to select one of the original dimensions in the data set as the signal of interest. There is reason to believe that transforming the data using the PCA or the BTA would improve the performance of the final predictive model. However, this experiment is also applied to the original dimensions of the real-world data set in order to provide a comparison ground to which the performance of the transformation approaches can be compared. The results are presented in Chapter 5.

The semi-supervised CUSUM method, presented in Subsection 3.5.1, is a competing method that could replace the unsupervised offline change-point detection method applied for the selection of the signal of interest in this experiment. Applying the CUSUM method, one would need to use historical data to determine the CUSUM features. In this analysis, the signal selection data set is used both for the selection of the signal of interest and for determining the CUSUM features. Using the CUSUM method for the selection of the signal of interest, the CUSUM features should be determined for each alternative signal.

Then the CUSUM method should be applied to the same data set and the signal yielding the best performance should be selected as the signal of interest. One can argue that using the signal selection data set for both purposes is a type of overfitting. Therefore in this analysis the unsupervised offline change-point detection algorithm is applied for the selection of the signal of interest. Given a larger data set, one could use the semi-supervised CUSUM method for this purpose, in which case one should use different data sets for determining the CUSUM features and the selection of the signal of interest.

## 4.2.2 Experiment 2

In this experiment one of the signals discrepancy, leverage or influence based on a PCA transformation is used as the signal of interest. The approach applied in this experiment is marked with red in the analysis pipeline in Figure 4.9. The PCA transformation is constructed and applied in the same way as in Experiment 1.

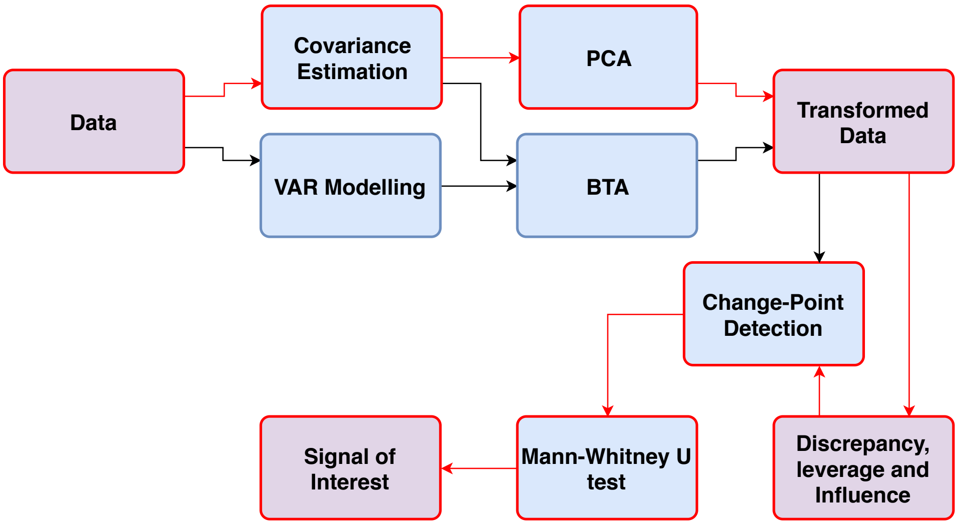


**Figure 4.9:** The analysis pipeline for Experiment 2. The performed analysis in this experiment is marked with the red color.

After transforming each running group in the signal selection set, the discrepancy, the leverage and the influence are calculated as alternative signals of interest. This is performed following the method presented in Subsection 3.4.5. The first $s$ principal components are used to calculate the leverage while the rest of the principal components are used to calculate the discrepancy. The number $s$ is selected such that the first $s$ principal components explain a large proportion $\kappa$ of the variance in the training data set. We set $\kappa = 0.90$ both for the simulated and for the real-world data.

Within each running group in the signal selection set, a change-point detection algorithm, based on the theory presented in Section 3.3, is applied to all three possible signals of interest. The same penalty function used in Experiment 1 is also used here, except here we set $m = 10$ for the simulated data and $m = 60$ for the real-world data. These values are selected by trial and error. PELT, presented in Subsection 3.3.4, is used as the search method. The cost function presented in (3.12) is used here. Since the signal is one dimensional, the determinant of the estimated covariance matrix in this equation is replaced with the estimated variance.

After the change-point detection is performed on the three signals discrepancy, leverage and influence in all of the running groups in the signal selection set, the three signals are compared to each other using the Mann-Whitney U test. This is done in a similar way as in Experiment 1. Comparing the $p$-values of the performed test, one of the three signals is selected as the signal of interest.

### 4.2.3 Experiment 3

In this experiment, one of the Box-Tiao components is selected as the signal of interest. The approach applied in this experiment is marked with red in the analysis pipeline in Figure 4.10. The BTA is performed as explained in Subsection 3.4.2. The BTA transformation has a semi-supervised nature as explained in Subsection 3.4.3. The training set is used to estimate a VAR model of order 1 for the data. Thereafter the estimated covariance matrix of the data $\hat{\mathbf{\Gamma}}$ and the estimated covariance matrix of the residuals $\hat{\mathbf{\Sigma}}$ are calculated for the data in the normal state. The estimation is performed according to the method presented in Subsection 3.4.4. Then $\hat{\mathbf{\Sigma}}^{-1}\hat{\mathbf{\Gamma}}$ is used to BTA-transform the rest of the running groups. This is done following the method presented in Subsection 3.4.2.

Within each running group in the signal selection set, a change-point detection algorithm is applied to every Box-Tiao component. The penalty function, the search method and the cost function used here are all the same as the ones used in Experiment 1. The Box-Tiao components are linear combinations of the dimensions of the data. Therefore with a similar argument to the one provided for Experiment 1 the cost function presented in (3.15) is used for the change-point detection, assuming a 1-dimensional autoregressive process of order 1 for the Box-Tiao components.

After the change-point detection is performed on the Box-Tiao components in all of the running groups in the signal selection set, the Box-Tiao components are compared to each other using the Mann-Whitney U test. This is done in a similar way as in Experiment 1. The Box-Tiao component yielding the lowest $p$-value is selected as the signal of interest.

### 4.2.4 Experiment 4

In this experiment one of the signals discrepancy, leverage or influence based on a BTA transformation is used as the signal of interest. The approach applied

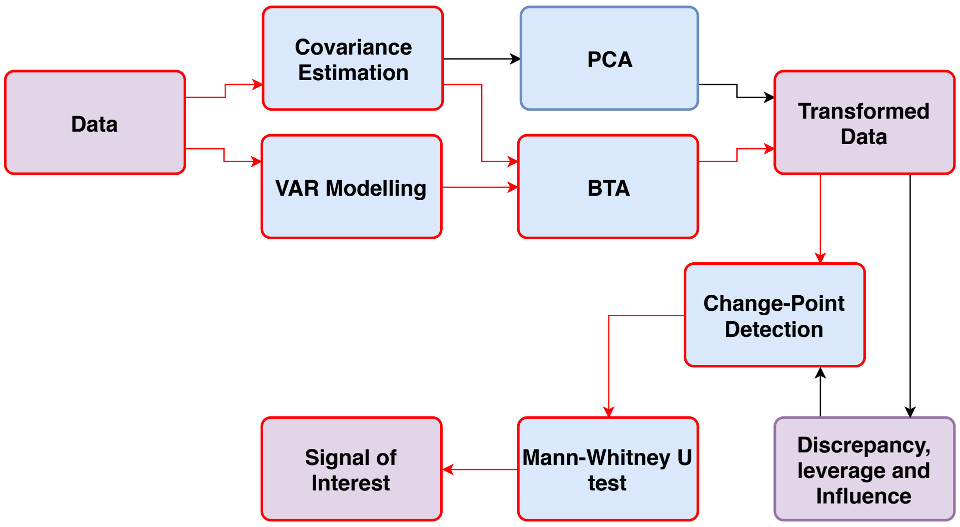


**Figure 4.10:** The analysis pipeline for Experiment 3. The performed analysis in this experiment is marked with the red color.

in this experiment is marked with red in the analysis pipeline in Figure 4.11. The BTA transformation is constructed and applied in the same way as in Experiment 3.

After transforming each running group in the signal selection set, the discrepancy, the leverage and the influence are calculated as alternative signals of interest in the same way as for Experiment 2, with the same value for $\kappa$.

Within each running group in the signal selection set, a change-point detection algorithm is applied to all three possible signals of interest. The penalty function, the search method and the cost function used here are all the same as the ones used in Experiment 2.

After the change-point detection is performed on the three signals discrepancy, leverage and influence in all of the running groups in the signal selection set, the three signals are compared to each other using the Mann-Whitney U test. This is done in a similar way as in Experiment 1. The signal yielding the lowest $p$-value is selected as the signal of interest.

### 4.2.5 Discussion

The different experiments presented in this section cover a variety of possible assumptions on the nature of the problem. BTA is used in Experiments 3 and 4, while PCA is used in Experiments 1 and 2.

BTA makes the assumption that the data comes from a VAR process of order 1. If the distribution of the data is very different from a VAR process, the approaches presented in Experiments 3 and 4 might not be suitable for the problem. If both the normal and the tripping change-points are in the mean

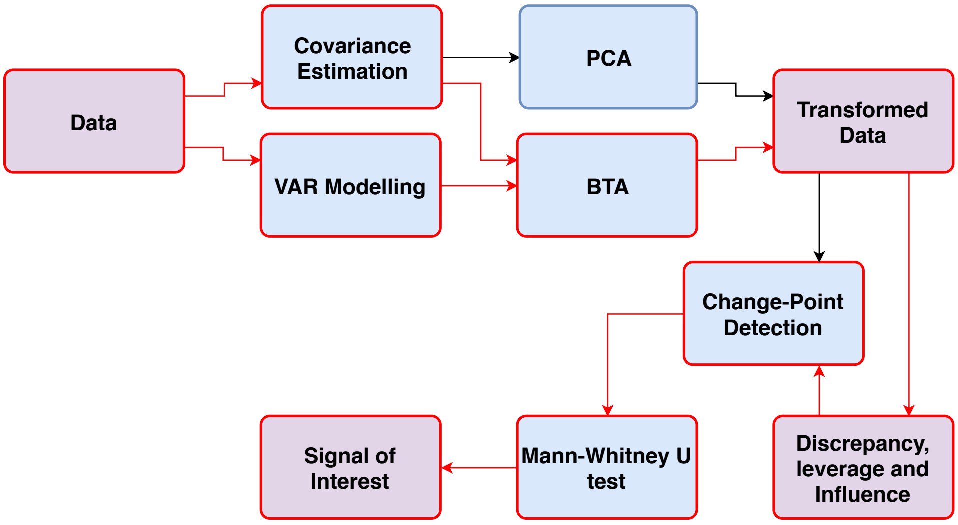


**Figure 4.11:** The analysis pipeline for Experiment 4. The performed analysis in this experiment is marked with the red color.

or the variance of the $d_n$-th and the $d_t$-th dimension of the data, respectively, the PCA approach should work well. This is because a change in the mean or the variance of the $d_n$-th dimension in the normal groups will increase its estimated variance, while the estimated variance of $d_t$ in the normal states remains *low*. This means that $d_n$ will be a larger factor than $d_t$ in the first principal components while $d_t$ will be a larger factor than $d_n$ in the last ones. In this case the principal components in which $d_t$ is a large factor will have small variances in the normal states and larger variances prior to a tripping. Therefore, intuitively one of the last principal components or the discrepancy should be selected as the signal of interest. However, this will only be the case if the variance of $d_t$ in the normal states is not in itself substantially larger than the variance of $d_n$ in the normal states. If this is the case, $d_t$ will be a larger factor than $d_n$ in the first principal components, which will have large variances in the normal states and even larger variances prior to a tripping. In this case, one of the first principal components or the leverage should be selected as the signal of interest. If the estimated variance of $d_t$ and $d_n$ in the normal states are almost equal, it is not clear which principal component will have the most predictive information.

If one or both of the normal and the tripping changes occur, not in the mean or the variance of some dimensions, but in the structure of the relations between them, the BTA approach might be more suitable than the PCA approach. This is for instance the case if the changes occur in the matrix $\mathbf{A}_1$ in a VAR process of order 1 such that $X_t = \mathbf{A}_1 X_{t-1} + U_t$ for $t > 1$, where $U_t \sim \mathcal{N}(0, \mathbf{\Sigma})$ for some covariance matrix $\mathbf{\Sigma}$. This is because such changes affect the predictability of the different dimensions but not necessarily their estimated variance. Changes in the relationships between the different dimensions of the data are not included in the simulated data. Therefore although the argumentation in this paragraph seems plausible, it is not tested in this work.

In Experiments 2 and 4, one of the signals discrepancy, leverage or influence is used as the signal of interest while in Experiments 1 and 3, a linear combination of the data is selected as the signal of interest. Choosing between the discrepancy, the leverage and the influence needs less computation power as the change-point detection needs to be performed only 3 times on each running group. In the approaches presented in Experiments 1 and 3, the change-point detection needs to be performed as many times as the number of dimensions in the data. The difference in the computation time will be more significant for high-dimensional data. On the other hand, using one of the signals discrepancy, leverage or influence will not result in any useful information about the cause of the failures in the system. In other words, this approach has little interpretability. This is while using a linear combination of the data, one knows how important a sensor measurement is for the signal of interest. This might provide the field experts with useful information about the cause of the failures in the system.

## 4.3 Predictive Model Development

For all the experiments presented in Section 4.2, a predictive model is developed using the CUSUM method presented in Subsection 3.5.1. We will use the signal selection set to determine the CUSUM features $T$, $K$ and $h$ with an ad hoc method.

The running groups in the signal selection set are divided into different states as illustrated in Figure 3.5. For the simulated data $\nu = 90$ minutes, as given in Subsection 4.1.2. We allow for the late detection of the transition state by the predictive model but not for the early detection since this is not possible given the online nature of the model. The late detection margin is 15 minutes. Therefore $f = 30 - 15 = 15$ minutes. The length of each state is then 75 minutes. For the real-world data $\nu$ and $f$ are set to 120 and 5 minutes, respectively in consultation with the field experts. Note that the last 10 minutes of the tripping groups in the real-world data are previously removed from the data set. Therefore setting $f = 5$ minutes means we want to be able to predict the unplanned shut-downs at least 15 minutes before the speed of the compressor goes below 500 RPM.

We set $T$ to the sample mean of the signal of interest in the normal states. The value of the sample mean $\hat{\mu}_s$ is calculated using the normal groups in the signal selection data set and following the method provided in (3.26). Using the signal selection data set and following the ad hoc method presented in Subsection 3.5.1, the optimal $K$ and $h$ can be selected such that some statistical metric is maximized. One of the metrics presented in Subsection 3.5.2 can be selected for this purpose. The suitable metric is specific to the business case, as explained in Subsection 3.5.2. In our experiments, $K$ and $h$ are selected such that the F1 score of the model is maximized on the signal selection data set.

Looking at (3.25), in Experiments 2 and 4, the process is declared out-of-control only if $C_i^+ > h$. This is because selecting one of the signals discrepancy, leverage or influence, we are not interested in declaring the process out-of-control when the observed value for the signal of interest is *"too low"*. In Experiments 1 and 3, the process is declared out-of-control if either $C_i^+ > h$ or $C_i^- > h$.

## 4.4 Evaluation

One can compare the performance of two approaches either by comparing the $p$-value of the Mann-Whitney U test on their corresponding selected signals of interest, or by using one of the statistical metrics presented in Subsection 3.5.2 on their corresponding CUSUM models using some test data set. In the simulation study, for each of the 100 simulations, a $p$-value is yielded for each of the four experiments. The Mann-Whitney U test is used to compare the estimated distribution of the $p$-values for every combination of two experiments. The Mann-Whitney U test is performed on the $p$-value's to test if the $p$-values yielded by one experiment are significantly lower than that yielded by another. The four different approaches for the selection of the signal of interest are compared in this way.

A test set, which includes both normal and tripping groups, can be used to evaluate the predictive models developed based on the signal of interest selected by the four different approaches presented in Section 4.2. The statistical metrics presented in Subsection 3.5.2 are used to evaluate the predictive models. All four metrics presented in Subsection 3.5.2 are calculated for each experiment, and the approaches are evaluated based on the value of all four metrics. The value of the four statistical metrics calculated on some test data set is the final evaluation of an approach.

## 4.5 Implementation

The implementation of the experiments is done in Python, and the code is provided in Appendix C. The code for the estimation of the covariance matrices $\mathbf{\Gamma}$ and $\mathbf{\Sigma}$ needed for PCA and BTA presented in Section 3.4 is developed by the author. The code for the PCA and the BTA transformations and the calculations of the discrepancy, the leverage and the influence with respect to these transformations is also developed by the author. The methods are implemented in a Python object called `VARBT`, presented in Appendix C. The estimation of the covariance matrix of the innovation term in a VAR process $\mathbf{\Sigma}$ used for BTA is performed with the help of the Statsmodel package developed by Seabold and Perktold (2010).

The VAR model is implemented in
`statsmodel.tsa.vector_ar.var_model`. For the data simulation a `VARProcess` object should be created. This object takes in the $\mathbf{A_i}$ matrices and

the covariance matrix of the innovation term $\mathbf{\Sigma}$ used for the construction of the VAR process as described in Section 3.2. A `simulate_var` function is linked to each `VARProcess` object. Using this function, one can simulate data following the pre-determined VAR process. This function only takes in the desired length of the simulated data. The random matrices used for the data simulation described in 4.1.2 are generated with the help of functions from the Scipy and the Sklearn packages of Python, developed by Jones et al. (2001–) and Pedregosa et al. (2011), respectively. A random covariance matrix function can be generated with the help of the `sklearn.datasets.make_spd_matrix` function of Sklearn, which takes in the number of dimensions and generates a random symmetric positive-definite matrix. A random orthogonal matrix given the number of dimensions can be generated using the `scipy.stats.ortho_group.rvs` function of Scipy.

In order to estimate a VAR model given data, one should create a `VAR` object. This object takes in the data used to fit the model. A `fit` function is linked to each `VAR` object. This function only takes in the desired order of the VAR model, and returns a result object. Calling the `resid` attribute of the result object, one can access the residuals of the fitted VAR model. Using the residuals, the covariance matrix of the innovation term, $\mathbf{\Sigma}$ can be estimated.

The change-point detection method presented in Section 3.3 is implemented in a Python package developed by Truong (2017) called Ruptures. This package is used here for the offline change-point detection. Several search methods including Opt and PELT, introduced in Subsection 3.3.4 are implemented in Ruptures. It is possible to use any cost function under some mild regularity conditions given in Truong (2017). Ruptures has implemented the cost function for several models including multivariate Gaussian model and univariate autoregressive process. The implementation of PELT in Ruptures requires the variables `model`, `min_size` and `jump`. Here `model` is set to `"normal"` for Experiments 2 and 4. For Experiments 1 and 3, `model` is set to `"ar"`. In these cases, the cost functions presented in (3.12) and (3.15) are used, respectively. The variable `min_size` is the minimum possible length of a regime and the variable `jump` is used to create a sub-sample of possible change-points, such that every

Chapter 5

# Statistical Analyses and Results

In this chapter we will present the result of performing the experiments designed in Section 4.2 on the simulated and the real-world data sets, described in Section 4.1.

First, we will perform all four experiments presented in Section 4.2 on the 100 simulated data sets. The data simulation process is described in Section 4.1.2. The results are presented in Section 5.1. In Subsection 5.1.3, the results yielded by the four different approaches are compared and discussed.

In Section 5.2, we will perform two of the four experiments designed in 4.2 on the real-world data. The experiments performed on the real-world data are selected based on the discussion of the results of the simulation analysis, presented in Subsection 5.1.3. In Subsection 5.2.4, we will present some results that can inspire for further work.

## 5.1 Simulation Analysis

Here we present the results of performing Experiments 1-4 designed in 4.2 on the 100 simulated data sets. In Subsection 5.1.1, the results for Experiments 1 and 3 are presented. In these experiments, one of the principal or Box-Tiao components is selected as the signal of interest. The predictive model is developed such that it detects deviations from the target value in both directions, as discussed in Section 4.3. In Subsection 5.1.2, the results for Experiments 2 and 4 are presented. In these experiments, one of the signals discrepancy, leverage or influence of the PCA or BTA transformed data is selected as the signal of interest. The predictive model is developed such that it detects deviations from the target value only in the positive directions, as discussed in Section 4.3. The evaluation is done according to the method discussed in Section 4.4. In Subsection 5.1.3, the results of all 4 experiments are compared and discussed. Based on the discussion presented in Subsection 5.1.3, two of the experiments are selected to be performed on the real-world data set.

### 5.1.1 Experiments 1 and 3

Experiments 1 and 3, presented in Section 4.2, are performed on the 100 simulated data sets. These experiments are marked in the analysis pipeline in Figures 4.8 and 4.10, respectively. In Experiments 1 and 3, first a PCA or BTA transformation is created in a semi-supervised manner, as discussed in Section 3.4.3. Then the offline change-point detection method presented in 3.3 is performed on all principal and Box-Tiao components. The Mann-Whitney U test is performed on the results of the offline change-point detection, as discussed in Subsection 3.4.5. The principal or the Box-Tiao component yielding the lowest $p$-value for the Mann-Whitney U test are respectively selected as the signal of interest in Experiment 1 and 3 in each of the 100 simulations.

In Figure 5.1, one can see the number of the simulations in which a principal or Box-Tiao component is selected as the signal of interest. One can see that neither the first principal component nor the first Box-Tiao component are ever selected as the signal of interest. Note that for the normal states the following holds:

$$\begin{aligned} \mathrm{var}(PC_1) > \mathrm{var}(PC_2) > ... > \mathrm{var}(PC_{10}), \\ q(BT_1) > q(BT_2) > ... > q(BT_{10}), \end{aligned}$$

where $\mathrm{var}(PC_i)$ and $q(BT_i)$ are respectively the variance of the $i$-th principal component and the predictability of the $i$-th Box-Tiao component. In Figure 5.1, one can see that if $i > j$, $PC_i$ is selected more often as the signal of interest than $PC_j$. For the BTA, the ninth Box-Tiao component is the most often selected signal of interest, while the frequencies of the selection of $BT_i$ for $i \in \{5, 6, 7, 8, 10\}$ are close to each other.

Looking at Figure 5.1, one could argue that in Experiment 3, the predictive information seems to be concentrated in one Box-Tiao component, which is not necessarily the last one, while in Experiment 1, the predictive information of $PC_i$ seems to gradually increase with increasing $i$. Therefore it is plausible to predict that if a predictive model is developed based on one single principal or Box-Tiao component, the BTA approach might outperform the PCA approach. On the other hand, one can see that $PC_{10}$, which is the most frequently selected principal component as the signal of interest, is more frequently selected as such than $BT_9$, which is the most frequently selected Box-Tiao component as the signal of interest. However, for $i \in \{2, 3, 5, 6, 7\}$, $BT_i$ is selected more frequently than $PC_i$. Therefore one could argue that in Experiment 3, the predictive information might be spread out among the Box-Tiao components while in Experiment 1, the last few principal components might contain most of the predictive information. Therefore it is plausible to predict that if a predictive model is developed using one of the signals discrepancy, leverage or influence, the discrepancy will outperform the other two signal with a PCA approach, while with a BTA approach, the competition between the discrepancy and the leverage or the influence will be closer than for the PCA. One should note that the discussion here might only hold in our simulation study and is not necessarily generalizable to other data sets.

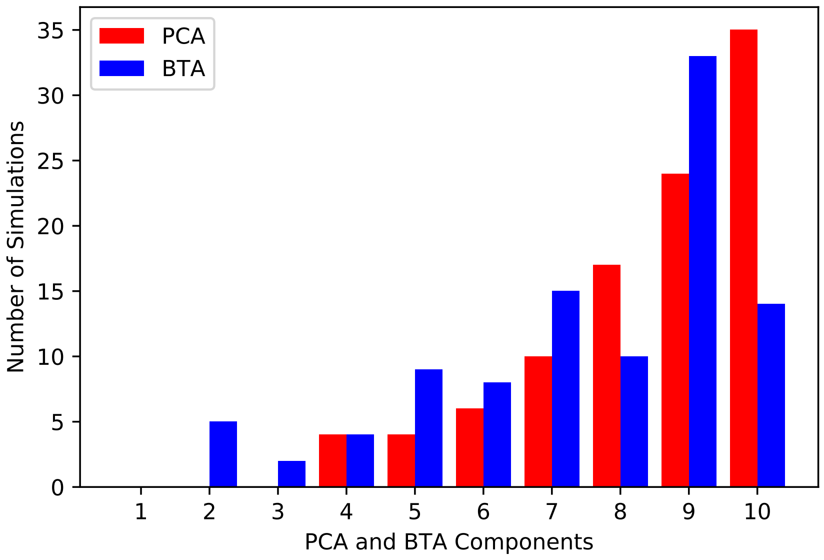


**Figure 5.1:** A histogram showing how frequently a principal or Box-Tiao component is selected as the signal of interest in Experiments 1 and 3 in the simulation study. The results for Experiment 1 are in red while the results for Experiment 3 are in blue. The total number of simulations is 100 for each experiment.

In each simulation, the Box-Tiao or the principal component with the lowest $p$-value of the Mann-Whitney U test is selected as the signal of interest. Inspired by the Fisher's method, first introduced in Fisher (1925, p. 103), we compute the natural logarithm of the $p$-value yielded by the selected signal of interest in each simulation. Then the sample mean and the sample standard deviation of the log($p$-values) are calculated. This is done separately for Experiments 1 and 3. In Experiment 1, the sample mean and the sample standard deviation of the log($p$-values) are respectively $-57.51$ and 22.25. In Experiment 3, the sample mean and the sample standard deviation of the log($p$-values) are respectively $-57.55$ and 20.64. Based on these values, one can see that for most of the selected signals of interest, the Mann-Whitney U test, under a significance level of 0.05, concludes that the distribution of the transition states in which a change-point is detected is stochastically greater than the distribution of the normal states in which a change-point is detected.

The Mann-Whitney U test is used to test if the distribution of the $p$-values yielded by the selected signal of interest in Experiment 1 is stochastically less than those Experiment 3. The normalization of the distributions is not necessary here since the Mann-Whitney U test is a non-parametric test. The $p$-value of the test is 0.34. One can see that the $H_0$ is not rejected at a significance level of 0.05. It is therefore concluded that none of the approaches, PCA or BTA, are necessarily superior to the other in our simulation study, when one compares the $p$-values of the Mann-Whitney U test yielded by the selected signal of interest in Experiment 1 and 3.

For each simulation, a predictive model is constructed based on the signal of interest selected by each of the two approaches, following the method presented in Subsections 4.3 and 3.5.1. As described in Subsection 4.3, the CUSUM features are selected such that the F1 score of the model is maximized on the signal selection data set. For each simulation and experiment, the predictive model is applied to the test data set. The recall, the precision, the accuracy and the F1 score of the models are calculated following the method presented in Subsection 3.5.2. The results of the calculation of these metrics are illustrated as histograms in Figures 5.2, 5.3, 5.4 and 5.7, respectively.

In Figure 5.2, the distribution of the recall of the predictive models applied to the simulated test data sets is presented. This is done separately for Experiments 1 and 3. In this figure, one can see that the predictive models in a large number of simulations have a recall that is larger than 0.95 both in Experiment 1 and 3. In these simulations, the predictive model can predict more than 95% of the trippings in the test data set between 15 and 90 minutes before they happen. One can see that in the interval $(0.95, 1]$, the number of simulations with Experiment 3 is larger than that with Experiment 1. This might mean that Experiment 3 yields a better recall for the predictive model than Experiment 1. This hypothesis can be tested using the Mann-Whitney U test. The Mann-Whitney U test is used to test if the distribution of the recall yielded by the predictive models on the simulated test data sets in Experiment 3 is stochastically greater than that of Experiment 1. The $p$-value of the test is

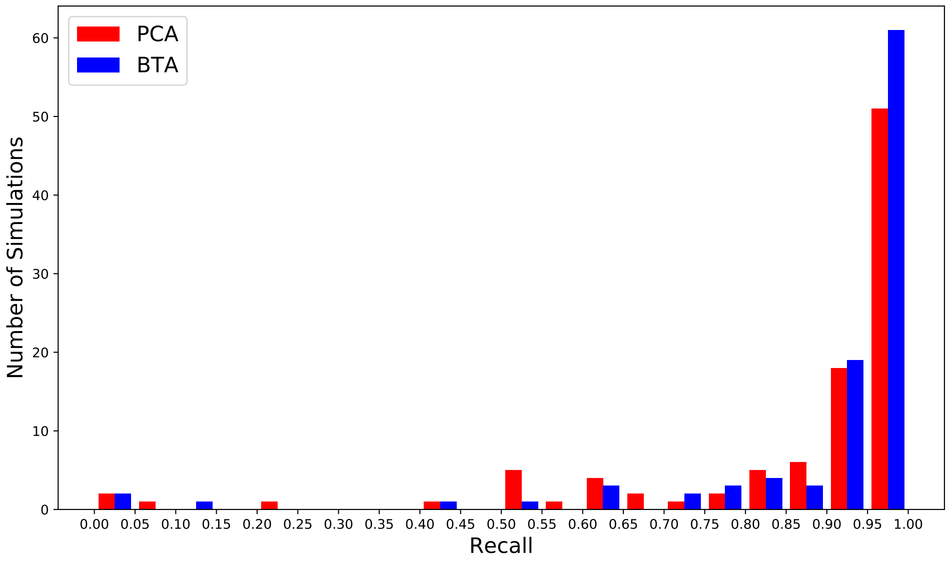


**Figure 5.2:** A histogram showing the distribution of the recall of the predictive models in Experiments 1 and 3 applied to the simulated test data sets. The results for Experiment 1 are in red while the results for Experiment 3 are in blue. The total number of simulations is 100 for each experiment.

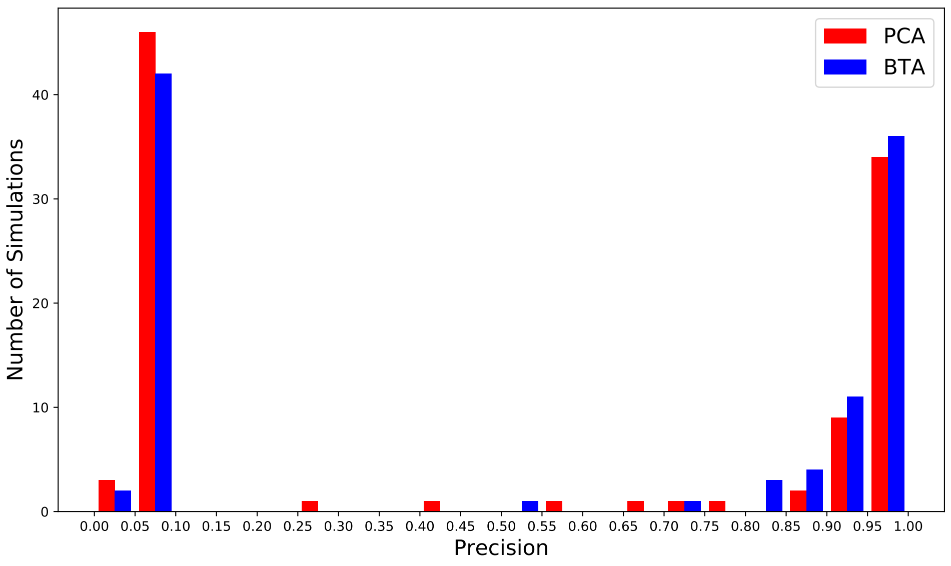


**Figure 5.3:** A histogram showing the distribution of the precision of the predictive models in Experiments 1 and 3 applied to the simulated test data sets. The results for Experiment 1 are in red while the results for Experiment 3 are in blue. The total number of simulations is 100 for each experiment.

0.046. One can see that the $H_0$ is rejected at a significance level of 0.05. It is therefore concluded that Experiment 3 yields a better recall for the predictive model than Experiment 1 in our simulation study.

The distribution of the precision and the accuracy of the predictive models on the test data sets are presented in Figures 5.3 and 5.4, respectively. This is done separately for Experiments 1 and 3. In these figures, it can be observed that in both experiments, the precision and the accuracy of the predictive model for a large number of simulations are either in the interval $(0.05, 0.1]$ or in the interval $(0.95, 1]$. This means when one of the Experiments 1 or 3 is applied as the method for the selection of the signal of interest, the resulting predictive model can have either very low and very high precision and accuracy. Therefore it is plausible to think that there might be a latent factor affecting the precision and the accuracy of the predictive model.

One could wonder if the precision and the accuracy are affected by a latent factor in the same way. In Figure 5.5, the precision and the accuracy of the predictive model are plotted against each other separately for Experiment 1 and 3. One can see in this figure that for most of the the simulations the precision and the accuracy of the predictive model are either both high or both low. This

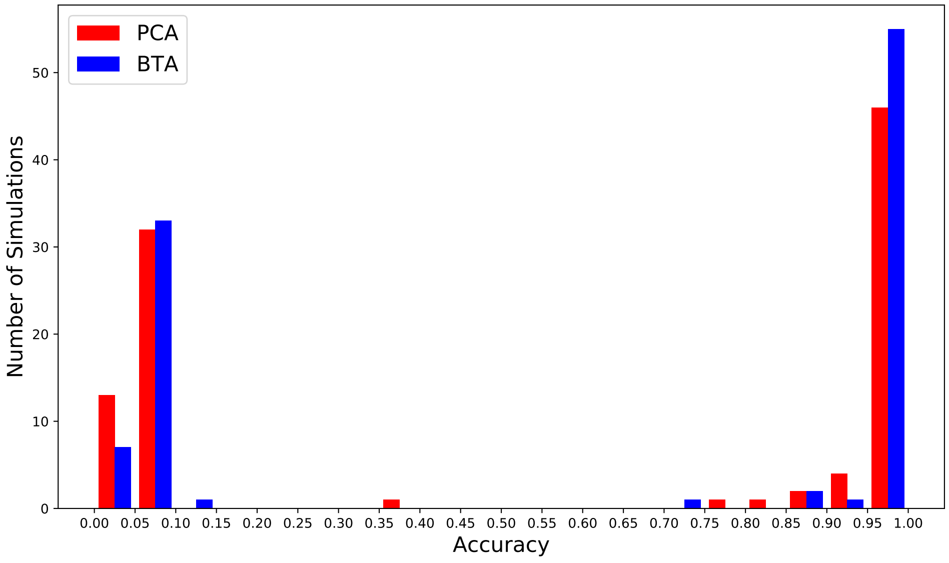


**Figure 5.4:** A histogram showing the distribution of the accuracy of the predictive models in Experiments 1 and 3 applied to the simulated test data sets. The results for Experiment 1 are in red while the results for Experiment 3 are in blue. The total number of simulations is 100 for each experiment.

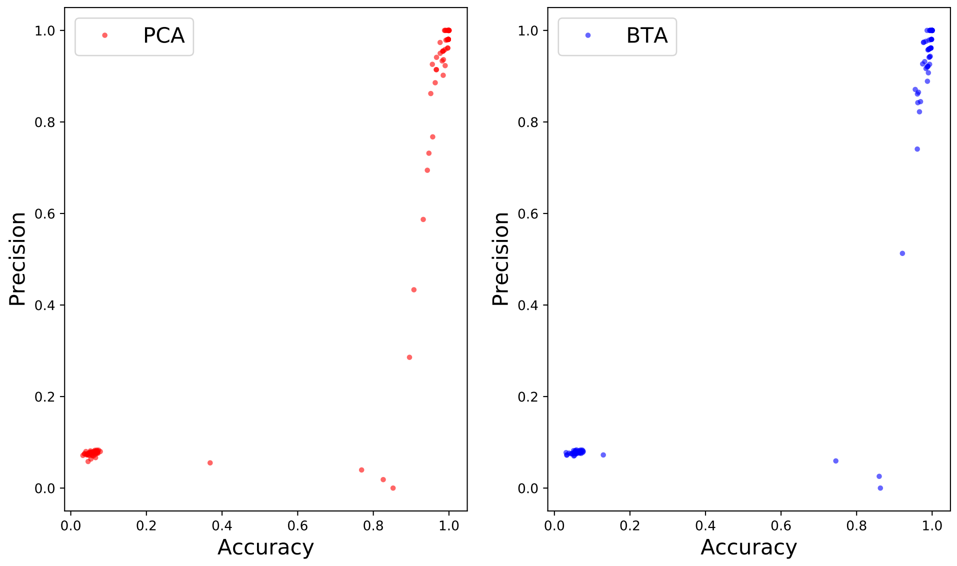


**Figure 5.5:** The precision and the accuracy of the predictive models in Experiments 1 and 3, applied to the simulated test data sets are plotted against each other. The plot on the left side is for Experiment 1 and the plot on the right side is for Experiment 3. The total number of points in each plot is 100.

supports the hypothesis that the accuracy and the precision are affected in the same way by a latent factor.

The above-mentioned division in the distribution of the precision and the accuracy is not observed in the distribution of the recall, presented in Figure 5.2. One could therefore think that the recall might not be affected in the same way as the precision and the accuracy are by a latent factor. Looking at the definition of the precision and the recall in Subsection 3.5.2, one can see that the only difference is that the #FP in the formula for the precision is replaced with a #FN in the formula for the recall. Therefore it is fair to conclude the the precision and the accuracy might be affected by a latent factor through #FP in the model.

It might be hypothesized that the latent factor affecting the precision and the accuracy is the selected signal of interest. In Figure 5.6, the precision and the accuracy of the predictive model applied to the test data set in a simulation are plotted against the selected signal of interest in that simulation. This is done separately for Experiments 1 and 3. One can observe that for almost all choices of the signal of interest, there exist simulations in which the precision and the accuracy are both high and low. For each experiment, the linear regression lines are plotted for both the precision and the accuracy. One can see that the lines are almost flat. Therefore it is fair to conclude that the latent factor affecting the precision and the accuracy is not the selected signal of interest.

The latent variable affecting the precision and the accuracy of the predictive model on the simulated test data sets might be a property of the underlying VAR process used to generate the data set. However, investigating this further is out of the scope of this work.

Looking at Figures 5.3 and 5.4, it seems that the distribution of the precision and the accuracy of the predictive model in Experiment 3 might be stochastically greater than that in Experiment 1. This is tested using a Mann-Whitney U test. The $p$-values of the Mann-Whitney U test comparing the precision and the accuracy of the two methods are 0.13 and 0.11, respectively. Neither for the precision nor for the accuracy, the null hypothesis in the Mann-Whitney U test is rejected under a significance level of 0.05. It is therefore concluded that Experiment 3 doesn't yield significantly better precision or accuracy for the predictive model than Experiment 1 in our simulation study.

In Figure 5.7, the distribution of the F1 score of the predictive model is presented in two histograms separately for Experiment 1 and 3. One can see that a similar division to the one observed in the distribution of the precision and the accuracy is observed in the distribution of the F1 score. In most of the simulations, the predictive model has either a F1 score in the interval $(0.1, 0.15]$ or in the interval $(0.95, 1]$. This is not surprising since the F1 score is the harmonic mean of the recall and the precision. It is known that in most of the simulations, the predictive model has a recall in the interval $(0.95, 1]$ on the simulated test data sets. Therefore the division in the distribution of the precision is carried to the distribution of the F1 score.

Looking at Figure 5.7, one can hypothesize that the distribution of the F1

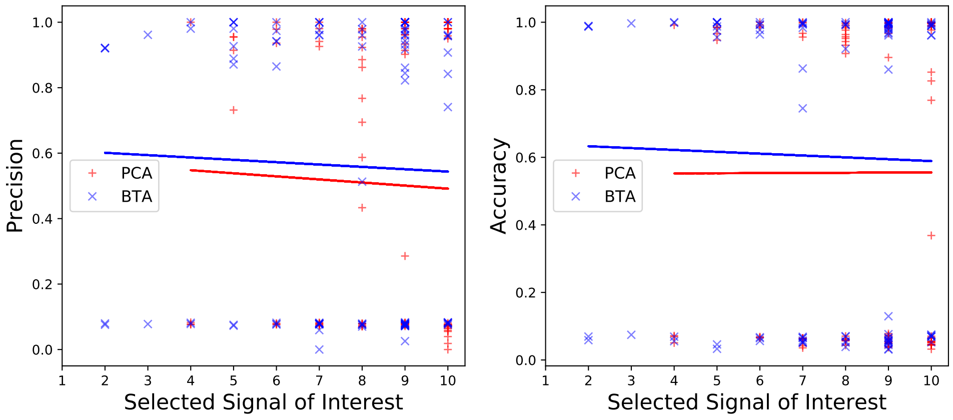


**Figure 5.6:** The precision and the accuracy of the predictive model applied to the test data set in a simulation are plotted against the selected signal of interest in that simulation. This is done separately for Experiments 1 and 3. The points belonging to Experiment 1 are colored red and have the shape of a ”+”, while the points belonging to Experiment 3 are colored blue and have the shape of a ”×”. On the $x$-axis, the numbers $1, \ldots, 10$ represent the 10 principal components in Experiment 1 and the 10 Box-Tiao components in Experiment 3. The red lines are linear regression lines for the points belonging to Experiment 1, and the blue lines are linear regression lines for the points belonging to Experiment 3. The total number of points for each experiment is 100.

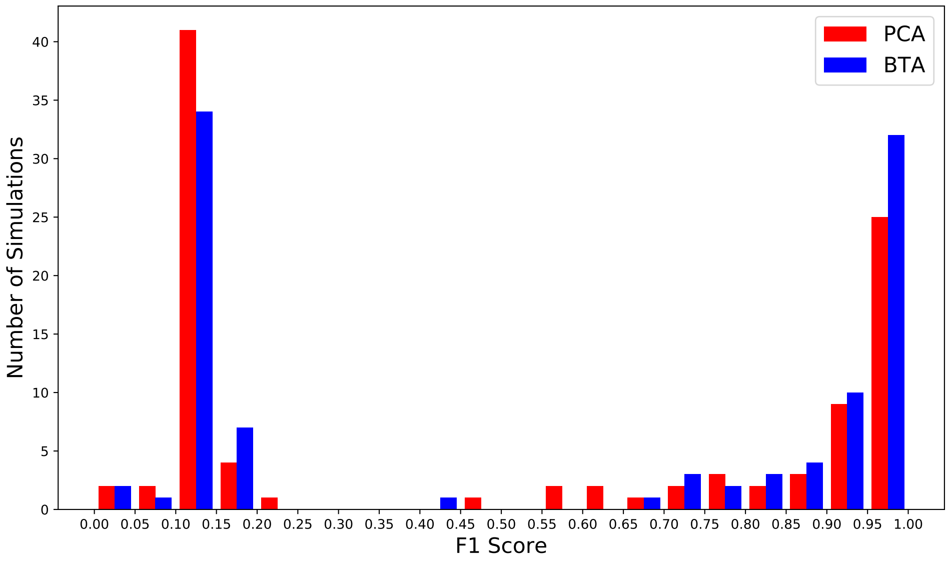


**Figure 5.7:** A histogram showing the distribution of the F1 score of the predictive models in Experiments 1 and 3 applied to the simulated test data sets. The results for Experiment 1 are in red while the results for Experiment 3 are in blue. The total number of simulations is 100 for each experiment.

score in Experiment 3 is stochastically greater than that in Experiment 1. This is tested using the Mann-Whitney U test. The $p$-value of the test is 0.08 and the null hypothesis is not rejected using a significance level of 0.05. It is therefore concluded that Experiment 3 doesn't a yield significantly better F1 score for the predictive model than Experiment 1 in our simulation study.

### 5.1.2 Experiments 2 and 4

Experiments 2 and 4, presented in Section 4.2, are performed on the 100 simulated data sets. These experiments are marked in the analysis pipeline in Figures 4.9 and 4.11, respectively. In Experiments 2 and 4, first a PCA or BTA transformation is created in a semi-supervised manner, as discussed in Section 3.4.3. The PCA transformation is used in Experiment 2 while the BTA transformation is used in Experiment 4. Then the discrepancy, the leverage and the influence of the PCA or the BTA transformed data are calculated. These signals are defined in Subsection 3.4.5. The offline change-point detection method presented in 3.3 is performed on these three signals in each experiment. Then the Mann-Whitney U test is performed on the results of the offline change-point detection with the leverage, the discrepancy and the influence as the alternative signals of interest, as discussed in Subsection 3.4.5. The signal yielding the lowest $p$-value by the Mann-Whitney U test is selected as the signal of interest in each of the 100 simulations.

In Figure 5.8, one can see the number of the simulations in which each signal is selected as the signal of interest. This is illustrated separately for Experiment 2 and 4. One can see that in both experiments, the discrepancy is the most frequent choice of the signal of interest. The leverage is more frequently chosen in Experiment 4 than in Experiment 2. This is in agreement with the prediction made in Subsection 5.1.1 based on Figure 5.1. It can be observed that the influence is chosen as the signal of interest in a very few number of simulations.

In a similar way to the analysis performed for Experiment 1 and 3, we compute the natural logarithm of the $p$-value yielded by the selected signal of interest in each simulation to normalize their distribution. The sample mean and the sample standard deviation of the log($p$-values) are calculated, separately for Experiments 2 and 4. In Experiment 2, the sample mean and the sample standard deviation of the log($p$-values) are respectively $-41.95$ and 25.55. In Experiment 4, the sample mean and the sample standard deviation of the log($p$-values) are respectively $-34.51$ and 26.84. Based on these values, one can see that for most of the selected signals of interest, the Mann-Whitney U test under a significance level of 0.05 concludes that the distribution of the transition states in which a change-point is detected is stochastically greater than the distribution of the normal states in which a change-point is detected.

The Mann-Whitney U test is used to test if the distribution of the $p$-values yielded by the selected signal of interest in Experiment 2 is stochastically less than that in Experiment 4. The $p$-value of the test is 0.04. The $H_0$ is rejected at a significance level of 0.05. It is therefore concluded that the PCA approach is *superior* to the BTA approach in our simulation study, when one compares the

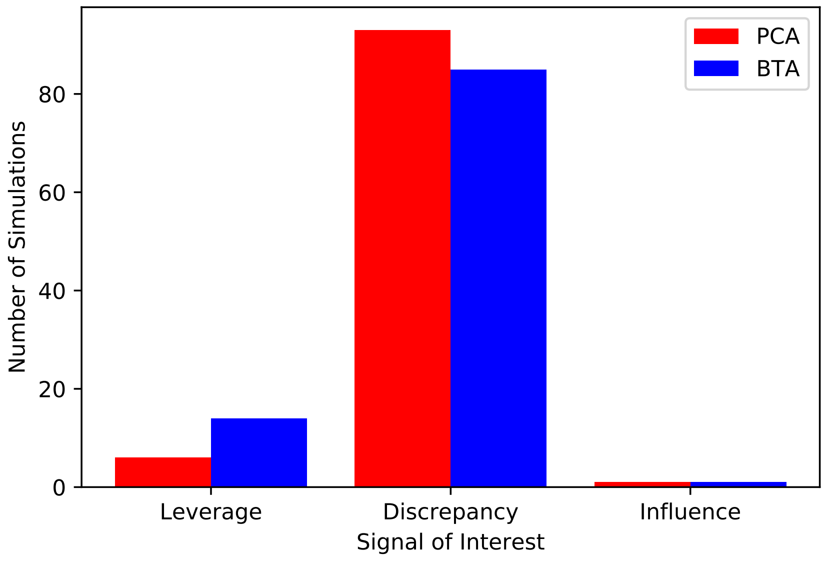


**Figure 5.8:** A histogram showing how frequently a signal is selected as the signal of interest in Experiments 2 and 4 in the simulation study. The results for Experiment 2 are in red while the results for Experiment 4 are in blue. The total number of simulations is 100 for each experiment.

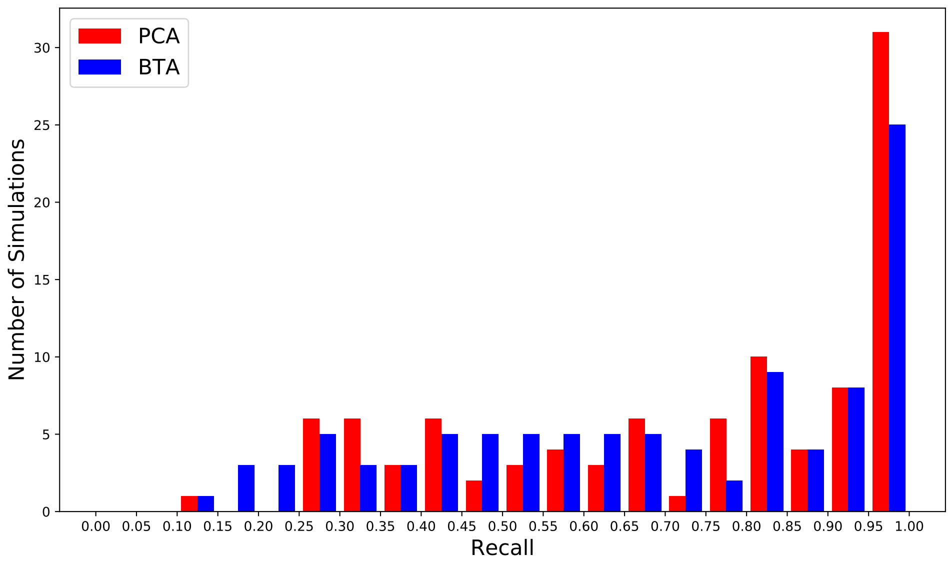


**Figure 5.9:** A histogram showing the distribution of the recall of the predictive models in Experiments 2 and 4 applied to the simulated test data sets. The results for Experiment 2 are in red while the results for Experiment 4 are in blue. The total number of simulations is 100 for each experiment.

$p$-values of the Mann-Whitney U test yielded by the selected signal of interest in Experiments 2 and 4.

For each simulation, a predictive model is constructed based on the signal of interest selected in each of the Experiments. This is done following the method presented in Subsection 4.3 and 3.5.1. As described in Subsection 4.3, the CUSUM features are selected such that the F1 score of the model is maximized on the simulated signal selection data sets. For each experiment, the predictive model is applied to the simulated test data sets. The recall, the precision, the accuracy and the F1 score of the models are calculated following the method presented in Subsection 3.5.2. The results of the calculation of these metrics are illustrated as histograms in Figures 5.9, 5.10, 5.11 and 5.12, respectively.

In Figures 5.9, 5.10, 5.11 and 5.12, one can see that in both experiments, all the statistical metrics calculated to measure the performance of the predictive model are in the interval $(0.95, 1]$ for a large number of simulations. For all of the four metrics, the number of the simulations in which the metric is larger than 0.95 is larger for Experiment 2 than it is for Experiment 4. It can therefore be hypothesized that using the discrepancy, the leverage or the influence of the PCA transformed data as the signal of interest might yield better results

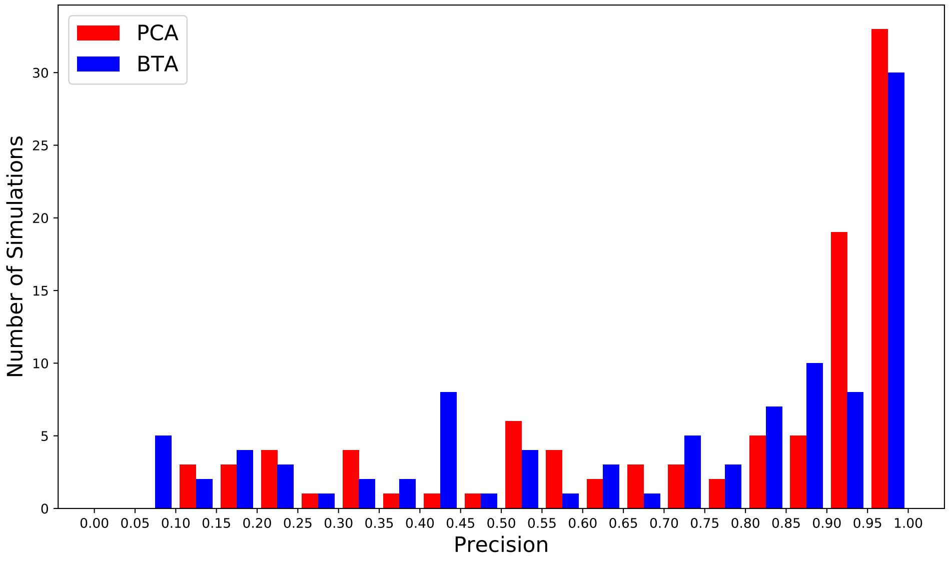


**Figure 5.10:** A histogram showing the distribution of the precision of the predictive models in Experiments 2 and 4 applied to the simulated test data sets. The results for Experiment 2 are in red while the results for Experiment 4 are in blue. The total number of simulations is 100 for each experiment.

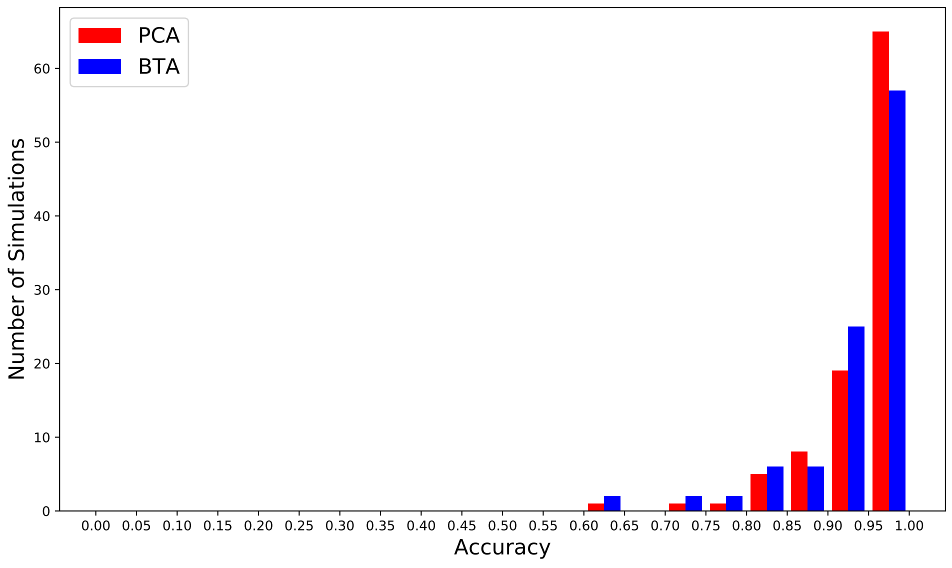


**Figure 5.11:** A histogram showing the distribution of the accuracy of the predictive models in Experiments 2 and 4 applied to the simulated test data sets. The results for Experiment 2 are in red while the results for Experiment 4 are in blue. The total number of simulations is 100 for each experiment.

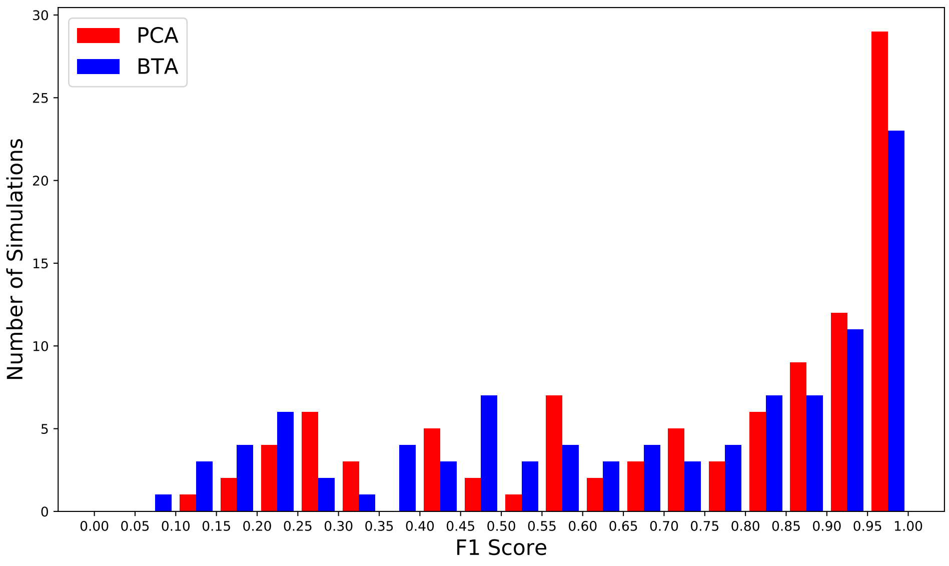


**Figure 5.12:** A histogram showing the distribution of the F1 score of the predictive models in Experiments 2 and 4 applied to the simulated test data sets. The results for Experiment 2 are in red while the results for Experiment 4 are in blue. The total number of simulations is 100 for each experiment.

for the predictive model than using that of the BTA transformed data in our simulation study. This hypothesis can be tested using the Mann-Whitney U test. The Mann-Whitney U test is used to test if the distribution of each of the four metrics in Experiment 2 is stochastically greater than that in Experiment 4. The $p$-values of the test on the recall, the precision, the accuracy and the F1 score are 0.10, 0.11, 0.12 and 0.13 respectively. One can see that the $H_0$ is not rejected at a significance level of 0.05 in any of the tests. It is therefore concluded that Experiment 2 does not yield significantly better results for the predictive model than Experiment 4 in our simulation study.

### 5.1.3 Comparison and Discussion

Looking at Figures 5.10, 5.11 and 5.12 for Experiments 2 and 4, one can not observe the same division in the distribution of the precision, the accuracy and the F1 score observed in the distribution of these metrics in Experiments 1 and 3 in Figures 5.3, 5.4 and 5.7. One may therefore infer that the latent factor affecting the performance of the predictive model in Experiments 1 and 3 is not present in Experiments 2 and 4. Another possible conclusion is that selecting only one principal or Box-Tiao component as the signal of interest makes the approaches in Experiments 1 and 3 non-robust. The difference in the Mann-Whitney U test's $p$-values yielded by the different principal or Box-Tiao components might be small. In this case, the principal or Box-Tiao component yielding the least $p$-value might not yield the best performance by the predictive model.

In Experiments 1 and 3, the last principal and Box-Tiao components are selected more frequently as the signal of interest than the first ones while in Experiments 2 and 4 the discrepancy is the most frequently selected signal of interest. Note that the discrepancy is calculated based on the last principal or Box-Tiao components. It is therefore fair to conclude that the principal components with the smallest variances and the Box-Tiao components with the smallest predictabilities in the normal states contain the most predictive information in our simulation study.

Comparing the recall of the predictive models in Experiments 1 and 3, it was concluded that the distribution of the recall is stochastically greater in Experiment 3 than in Experiment 1. On the other hand comparing the $p$-values of the Mann-Whitney U test yielded by the selected signal of interest in Experiments 2 and 4, it was concluded that the distribution of the $p$-values in Experiment 2 is stochastically less than in Experiment 4. Note that the PCA transformation is used in Experiments 1 and 2, while the BTA transformation is used in Experiments 3 and 4. Other than the above-mentioned differences, the performance yielded by the PCA and the BTA are not significantly different.

In our simulation study, we calculate the average recall, precision, F1 score and accuracy of the predictive model in each of the four experiments designed in Subsection 4.2. The results are presented in Figure 5.13. Experiments 1 and 3 yield lower average precision, recall and accuracy than Experiment 2 and 4. This is expected given the division in the distribution of these metrics in

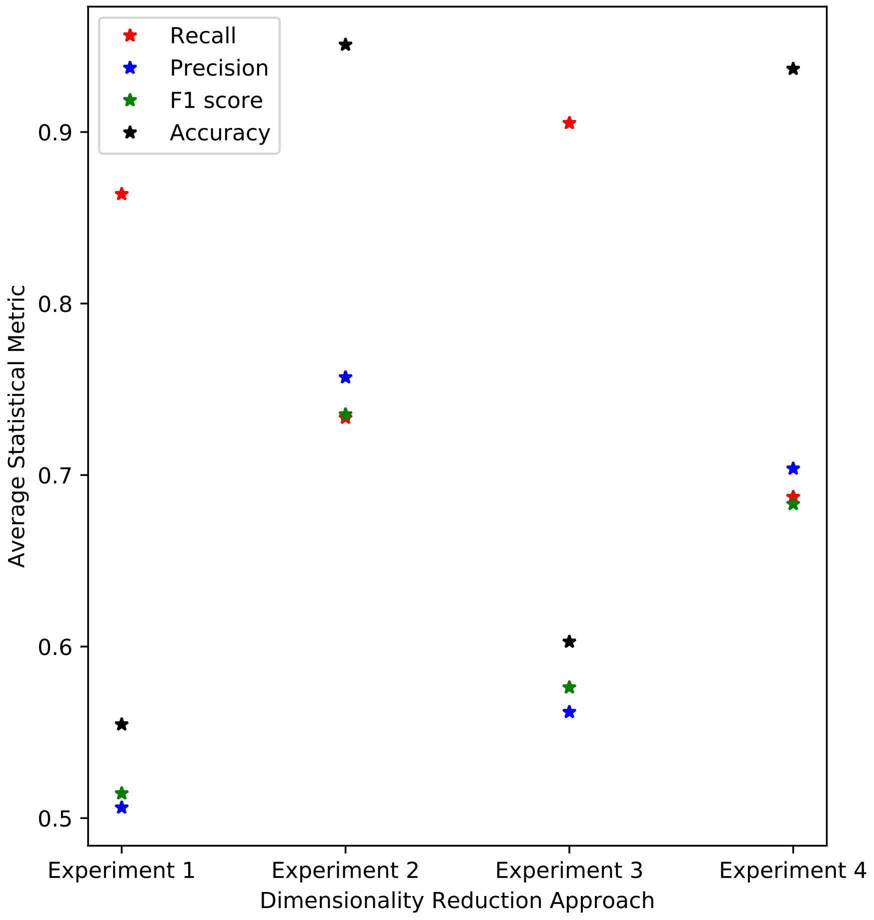


**Figure 5.13:** The average recall, precision, F1 score and accuracy of the predictive model in each of the four experiments designed in Subsection 4.2 applied to the simulated test data sets are illustrated here. The total number of simulations used to calculate the average metrics is 100. The average values of the recall for Experiments $1-4$ are respectively 0.86, 0.73, 0.91 and 0.69. The average values of the precision for Experiments $1-4$ are respectively 0.51, 0.76, 0.56 and 0.70. The average values of the F1 score for Experiments $1-4$ are respectively 0.51, 0.74, 0.58 and 0.68. The average values of the accuracy for Experiments $1-4$ are respectively 0.55, 0.95, 0.60 and 0.94.

these experiments. Experiments 1 and 3 yield however larger average recall than Experiments 2 and 4.

## 5.2 Real-World Analysis

Based on the results of the simulation study, presented in Section 5.1, we choose to perform Experiments 2 and 4 on the real-world data set. This is also partly to save computation time, in addition to avoiding a very low precision and accuracy as one obtained for many simulations with Experiments 1 and 3.

In Subsection 5.2.1, we present an analysis of the real-world data. This subsection is an extension of the description of the real-world data presented in Subsection 4.1.1. The results of the dimensionality reduction performed with the methods in Experiments 2 and 4 are presented in Subsection 5.2.2. Following the method presented in Subsection 4.3, a predictive model is developed based on the selected signal of interest in each of the experiments. The predictive models are evaluated on a test data set, using the statistical metrics presented in Subsection 3.5.2. The results of this are presented in Subsection 5.2.3.

Finally, in Subsection 5.2.4, we present the results of a simple multivariate model, demonstrating the potential in these models. The theory for this experiment is not developed or presented in this work. The results in Subsection 5.2.4 only serve the purpose of inspiring for further work in the field.

### 5.2.1 Data Analysis

In Figure 4.3, it is observed that the estimated variances for most of the dimensions in the tripping groups are not very different from those in the normal groups. It is argued that this might be because the system is in a normal state for most of the time even in the tripping groups. In Figure 5.14, the estimated predictability and the natural logarithm of the estimated variance of the 62 dimensions of the data are shown separately in the normal and in the transition states. We refer to the variance and the predictability of a dimension in the normal states as its *normal variance* and *normal predictability*, respectively. The variance and the predictability of a dimension in the transition states are referred to as its *transition variance* and *transition predictability*, respectively. The normal and the transition states here are defined in Subsection 3.5.1 and illustrated in Figure 3.5. Here $\nu = 120$ minutes and $f = 5$ minutes. Only the normal states in the normal groups are used to estimate the normal variances and the normal predictabilities. For estimation in the transition states, the data from 120 minutes to 5 minutes before the end of the tripping groups is used. For the estimation in the normal states, the normal groups are therefore divided into intervals of length 115 minutes. Note that the last 10 minutes of each tripping group is previously removed from the data set. As a result a transition state is from 130 to 15 minutes before a failure in the system. The final estimation of the normal and the transition variance and predictability of a dimension are the average of the estimated variances and predictabilities of that dimension in

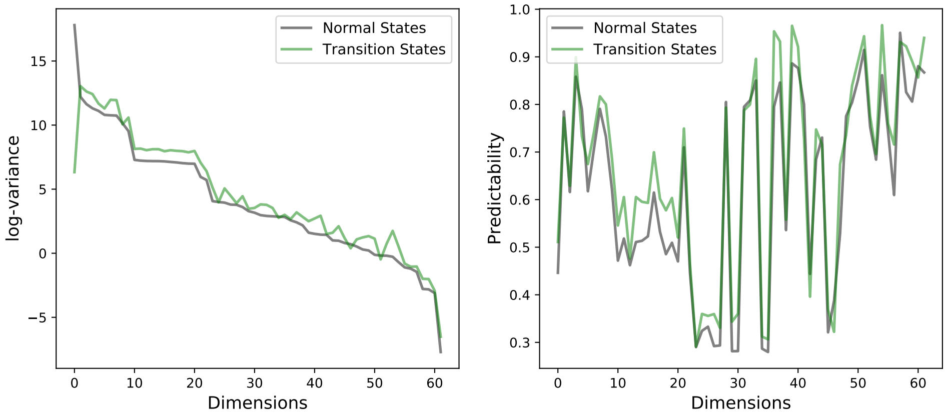


**Figure 5.14:** In the left plot, the natural logarithm of the estimated normal and transition variance of the 62 dimensions of the real-world data are shown separately. In the right plot, the estimated normal and transition predictability of the 62 dimensions of the data are shown separately. The predictability is estimated assuming an autoregressive process of order 1 for each dimension. The estimates in the normal states are shown in dark grey while the estimates in the transition states are shown in green. The dimensions are sorted in the descending order of their estimated normal variance in both plots. The rank of each dimension can be found in a table in Appendix B.

the normal and the transition states, respectively. The dimensions are sorted in the descending order of their estimated normal variance in both plots. This is not necessarily the same as the indexing used in Figure 4.3, since the estimation is done in a different way here. The rank of each dimension in Figure 5.14 can be found in a table in Appendix B. The predictability is estimated assuming an autoregressive process of order 1 for each dimension. The predictability $q$ of a 1-dimensional signal is defined in (3.19).

In Figure 5.14, one can see that the estimated transition variances are slightly larger than the estimated normal variances for most of the dimensions. Note that the difference is larger in the values of the estimated variances than it is in the values of the logarithm of the estimated variances, which are plotted in Figure 5.14. This difference in the variance in the different states can be exploited in developing a predictive model.

The plots in Figure 5.14 support the hypothesis that the variance of a signal and its predictability are independent in the real-world data set. This is because while the dimensions are sorted in the descending order of their estimated normal variance in both plots, the plot for the estimated predictabilities shows no clear trend. Note that the predictabilities are estimated assuming an autoregressive process of order 1 for each dimension. This assumption is clearly not guaranteed to hold.

The mean estimated normal and transition predictability of the dimensions

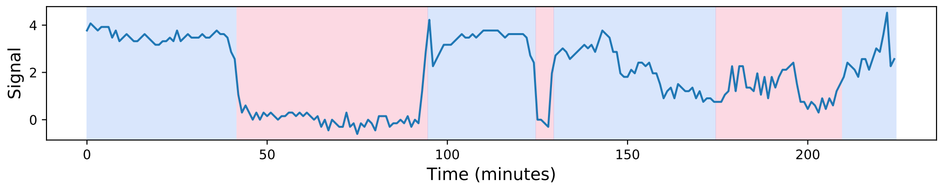


**Figure 5.15:** An illustration of the normal changes in a dimension of the real-world data set in a normal group. A change-point is marked with a shift in the background color.

in the data are respectively 0.63 and 0.67. Note that the relations between the different dimensions are disregarded in the estimation of the predictabilities. Looking at the definition of the predictability in (3.19), one can interpret that the assumption of a 1-dimensional autoregressive process of order 1 explains in average about 65% of the variance in each dimension. This supports the hypothesis that the real-world data can be modelled with a VAR model of order 1.

Note that prior to the analysis, the whole data set is scaled by dividing each observation in each dimension by the square root of its normal variance, shown in Figure 4.3. However, the calculations of the variances and the predictabilities illustrated in Figure 5.14 are done prior to the scaling.

In Subsection 5.2.2, the offline change-point detection method together with the Mann-Whitney U test method are performed on the original dimensions of the real-world data to provide a comparison ground for the dimensionality reduction approaches in Experiments 2 and 4. The dimension yielding the lowest $p$-value is then selected as the signal of interest. In Subsection 5.2.3, a predictive model is developed based on this dimension. In Figure 5.15, this dimension is plotted in a normal group. In Section 3.4.3, we propose not to develop change-point detection models based on the sensor measurements directly, since there might be many normal changes in the operation mode of the system. This is illustrated in Figure 5.15. In this figure, one can see that there are several changes in the distribution of this dimension during a normal group. Note that according to the results from applying the offline change-point detection and the Mann-Whitney U test method to the original dimensions, the difference between the distribution of the transition and the normal states in which a change-point is detected is largest for this dimension. Looking at Figure 5.15, it is reasonable to predict that a model that is supposed to predict failures in the system based on changes in the distribution of this signal might not have a good performance.

## 5.2.2 Signal Selection Results

Experiments 2 and 4, presented in Section 4.2 are performed on the real-world data set. Here the training data set is used to construct the BTA and the PCA transformations and the signal selection data set is used to select the signal of interest among the signals leverage, discrepancy and influence in each case. The different data sets are defined in Subsection 4.1.1.

The Mann-Whitney U test is performed on the results of the offline change-point detection applied to the signal selection data set for Experiments 2 and 4. The log($p$-values) of the tests are displayed in Figure 5.16. The $p$-values for all the tests are larger than 0.99999. This might mean that none of the alternative signals of interest contain any predictive information, since the distribution of the transition states in which a change-point is detected is not stochastically greater than the distribution of the normal states in which a change-point is detected. This might however be caused by the difference in the lengths of the different states.

The transition states are all of length $\nu - f = 115$ minutes. This is while the length of a normal state can be as large as several thousands minutes, looking at Figures 4.1 and 4.2. By dividing the data set into normal, transition and failure states following the method presented in Subsection 3.4.5 and illustrated in Figure 3.4, we assume that the probability of detecting a change-point in an interval is independent from the length of the interval since the penalty function in the offline change-point detection method is a function of the length of the interval, as discussed in Subsection 3.4.5. This assumption is however in no way guaranteed. This problem was not encountered in the simulation study, probably since the difference in the length of the normal states and the length of the transition states in the simulated data sets is not as large as it is in the real-world data set. To solve this problem, one should divide the normal states into intervals of equal length to the transition states, as it is illustrated in Figure 3.5. Then the normal and the transition states will have equal lengths and the penalty function in the offline change-point detection method will not need to depend on the length of the running group to estimate the number of change-points consistently. In this case a linear penalty function inspired by the AIC can be used. One should note that using a penalty function inspired by the AIC in this case still does not guarantee the consistency in estimating the number of change-points. However, non-consistent estimation in this case has a smaller effect on our results since the intervals that are being compared to each other are of equal lengths. Even though the division illustrated in Figure 3.5 is used to calculate the $\alpha_i$'s and $\beta_j$'s used to perform the Mann-Whitney U test, the change-point detection method is still applied to whole running groups. To avoid the above-mentioned issue completely, one should divide the data into normal and transition states of equal lengths and then apply the change-point detection method with the same penalty function to each individual state separately instead of applying it to whole running groups. This is however not done in this work. We have divided the normal states into intervals of equal length to the transition states, as it is illustrated in Figure 3.5, and applied the change-point detection method with a linear penalty to the running groups. The results are presented in Figure 5.17. Inspired by the AIC, a constant $\omega = 200$ is used in the penalty function $p_{l_0}$, presented in (3.11). One can see in Figure 5.17 that labeling the data points as it is illustrated in Figure 3.5 results in $p$-values that are in the expected range. Figure 5.17 shows that the

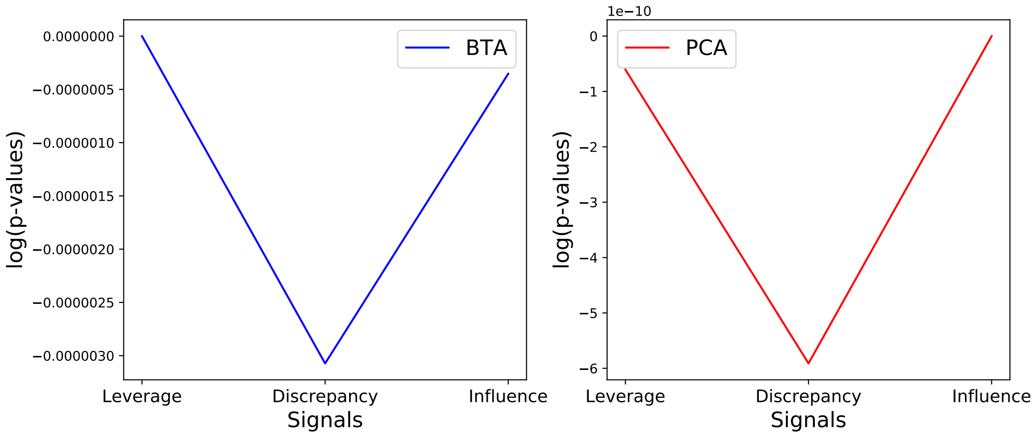


**Figure 5.16:** The $\log(p\text{-values})$ of the Mann-Whitney U test performed on the results of the offline change-point detection applied to the signals discrepancy, leverage and influence of the transformed real-world signal selection data set. In Experiment 2, the result of which is presented in the plot to the right, the PCA transformation is used. In Experiment 4, the result of which is presented in the plot to the left, the BTA transformation is used. Here the labeling of the data points is performed as described in Subsection 3.4.5 and illustrated in Figure 3.4. The penalty function used here for the change-point detection is provided in (4.2), where $m = 60$.

discrepancy yields the lowest $p$-value for both experiments. The $p$-values yielded by the discrepancy for Experiments 2 and 4 are respectively 0.3612 and 0.0002. This signal is therefore selected as the signal of interest in both experiments.

The offline change-point detection method presented in Section 3.3 together with the Mann-Whitney U test method presented in Subsection 3.4.5 are also performed on the original dimensions to provide a comparison ground for the dimensionality reduction approaches in Experiments 2 and 4. Here the labeling of the data points is performed as illustrated in Figure 3.4. By applying the change-point detection to the original dimensions, we assume that one of them is selected as the signal of interest on which a predictive model is developed. This provides a basis with which the performance of our methods in Experiments 2 and 4 can be compared. The offline change-point detection is performed in a similar way to Experiment 1, assuming an autoregressive model of order 1, and setting $m = 0.02$ in the BIC-inspired penalty function. The dimension yielding the lowest $p$-value is then selected as the signal of interest. The $p$-values yielded by the 62 dimensions are illustrated in Figure 5.18. The lowest $p$-value is 0.36, and one can see that most of the $p$-values are close to 1. The observed effect in the $p$-values yielded by the signals discrepancy, leverage and influence when the labeling of the data points is performed as illustrated in Figure 3.4 is also observed here. However, this issue is ignored here and the dimension yielding the lowest $p$-value is selected as the signal of interest, since the difference between the $p$-value yielded by this dimension and the $p$-values yielded by the other

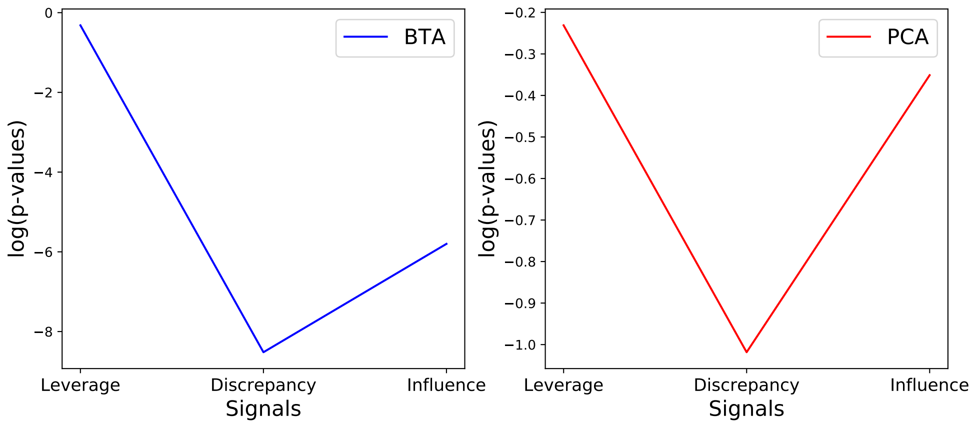


**Figure 5.17:** The $\log(p\text{-values})$ of the Mann-Whitney U test performed on the results of the offline change-point detection applied to the signals discrepancy, leverage and influence of the transformed real-world signal selection data set. In Experiment 2, the result of which is presented in the plot to the right, the PCA transformation is used. In Experiment 4, the result of which is presented in the plot to the left, the BTA transformation is used. Here the labeling of the data points is performed as described in Subsection 3.5.1 and illustrated in Figure 3.5. The penalty function used here for the change-point detection is $p_{l_0}(\tau) = 200K$, where $K$ is the number of change-points.

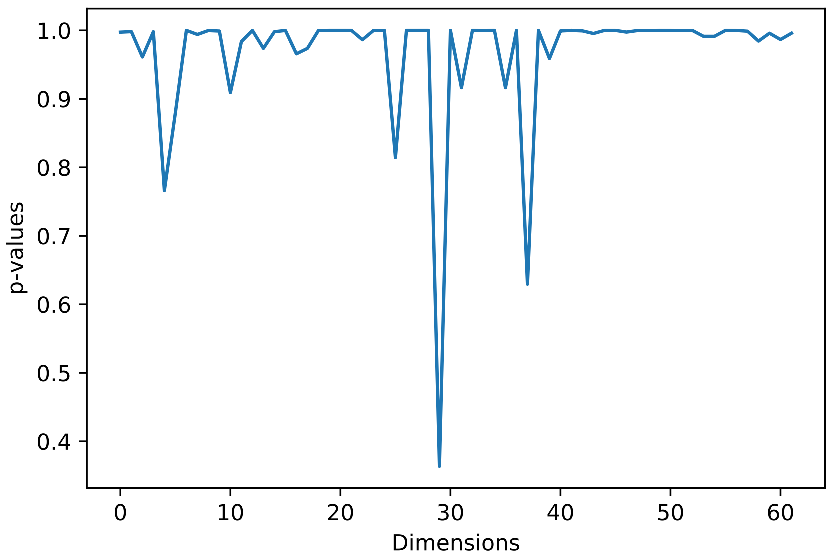


**Figure 5.18:** The $p$-values of the Mann-Whitney U test performed on the results of the offline change-point detection applied to the original dimensions of the real-world data. The signal selection data set is used for this analysis.

dimensions is *large enough*.

### 5.2.3 Predictive Model Results

Two CUSUM models are developed based on the discrepancy of the BTA and the PCA transformed data, as described in Subsection 4.3. Another CUSUM model based one of the original dimensions in the data is also developed as a comparison ground. Looking at (3.25), in Experiments 2 and 4, the process is declared out-of-control only if $C_i^+ > h$, as discussed in Subsection 4.3. Selecting one of the original dimensions as the signal of interest, the process is declared out-of-control if either $C_i^+ > h$ or $C_i^- > h$. The dimension yielding the lowest $p$-value of the Mann-Whitney U test is selected for this purpose. The signal selection data set consisting of 38 normal groups and 38 tripping groups is used to determine the CUSUM features in each case such that the F1 score of the model is optimized, following the method in Subsection 3.5.1. The CUSUM models are tested on the test data set, also consisting of 38 normal groups and 38 tripping groups. The different data sets are defined in Subsection 4.1.1. The data is divided into normal, transition and failure states following the method presented in Subsection 3.5.1 and Figure 3.5. Then there are in total 2627 normal states and 38 transition states in the test data set.

The predictive models are evaluated using the statistical metrics presented in Subsection 3.5.2. The results are illustrated in Figure 5.19. From the 38 transition states in the test data set, only 3 are detected by the model based on one of the original dimensions. Using the discrepancy of the PCA and the BTA transformed data as the signal of interest, the CUSUM model detects respectively 9 and 10 transition states out of 38. With the PCA and the BTA transformations, the number of false positives is respectively 336 and 495 out of 2627 normal states in the test data set. Using one of the original dimensions in the data, the number of false positives is 52. A lower false positive rate is the reason why the accuracy of this model is higher than that of the models using the PCA and the BTA transformations. One can see that while the number of true positives using the PCA is triple that using no transformations, the number of false positives using the PCA is more than 6 times that using no transformations. Therefore one might argue that applying the PCA might not have a real positive effect on the performance of the model. However, in further experimentation, we determined the CUSUM features based on the same individual dimension in the data maximizing the recall of the model on the signal selection data set. Note that earlier this was done maximizing the F1 score of the model on the signal selection data set. After applying the new model to the test data set, we obtained 22 true positives out of 38, and 1868 false positives out of 2627. One can see that while the number of true positives in this model using no transformations is slightly more than twice that using the PCA, the number of false positives in this model is more than 5 times that using the PCA. Therefore we conclude that our dimensionality reduction approaches should have a positive effect on the performance of the final predictive model.

### 5.2.4 A Simple Multivariate Predictive Model

Here we present the result of a model that can show the potential of further work on multivariate CUSUM models. The same PCA and BTA transformations constructed for Experiments 2 and 4 are used here. We use the last 10 principal and Box-Tiao components in this model. For each of them, using the signal selection data set and the method presented in 3.5.1, the CUSUM features are determined independently such that the F1 score is optimized. A CUSUM model is developed on each of the signals. Using the BTA transformation, the process is declared out-of-control if at least one of the last 10 Box-Tiao components is out-of-control and in-control if all of the last 10 Box-Tiao components are in-control. Using the PCA transformation, the process is declared out-of-control if at least one of the last 10 principal components is out-of-control and in-control if all of the last 10 principal components are in-control. A principal or Box-Tiao component is out-of-control if either $C_i^+ > h$ or $C_i^- > h$, where $C_i^+$ and $C_i$ are defined in (3.25). The models are tested on the test data set and the results are illustrated in Figure 5.20.

One can see that the recall is improved substantially compared to the univariate models while the loss of accuracy is limited. From 38 transition states 17 and 21 are detected using the last 10 Box-Tiao and principal components,

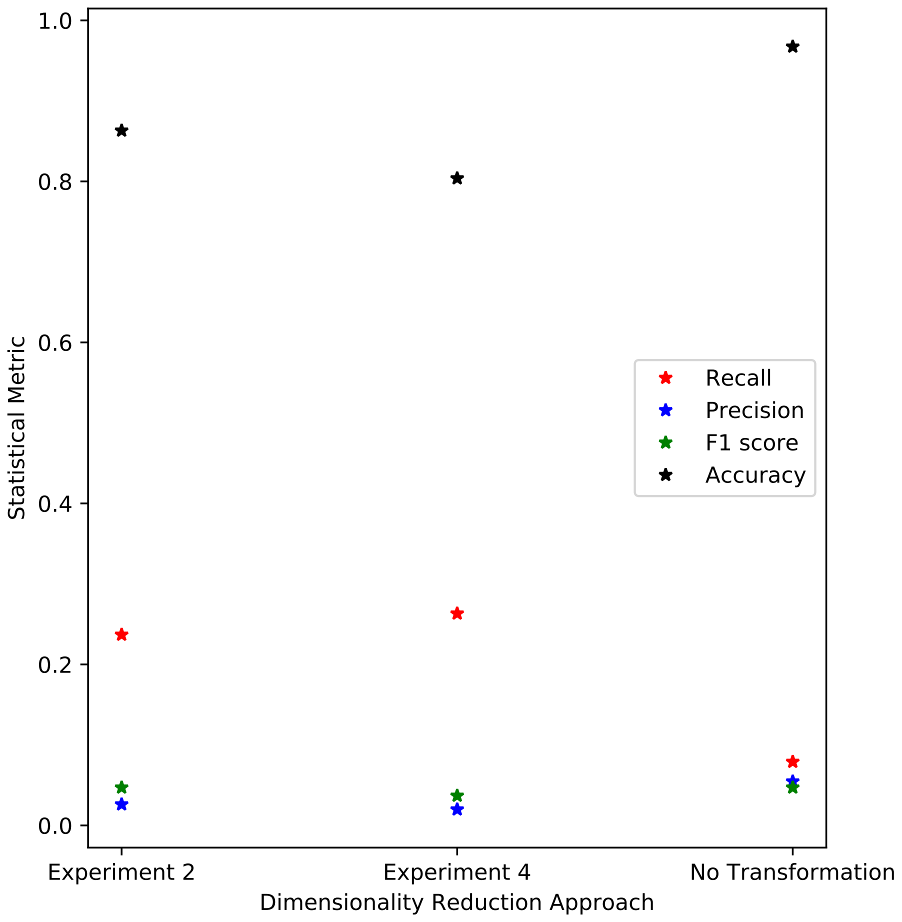


**Figure 5.19:** The recall, precision, F1 score and accuracy of the predictive model in Experiments 2 and 4 applied to the real-world test data set are illustrated here. These metrics are also calculated and illustrated for the predictive model based on one of the original dimensions of the data. The values of the recall for Experiments 2 and 4 are respectively 0.24 and 0.26. The values of the precision for Experiments 2 and 4 are respectively 0.03 and 0.02. The values of the F1 score for Experiments 2 and 4 are respectively 0.05 and 0.04. The values of the accuracy for Experiments 2 and 4 are respectively 0.86 and 0.80. The values of the recall, the precision, the F1 score and the accuracy of the predictive model based on one of the original dimensions of the data are respectively 0.08, 0.05, 0.06 and 0.97.

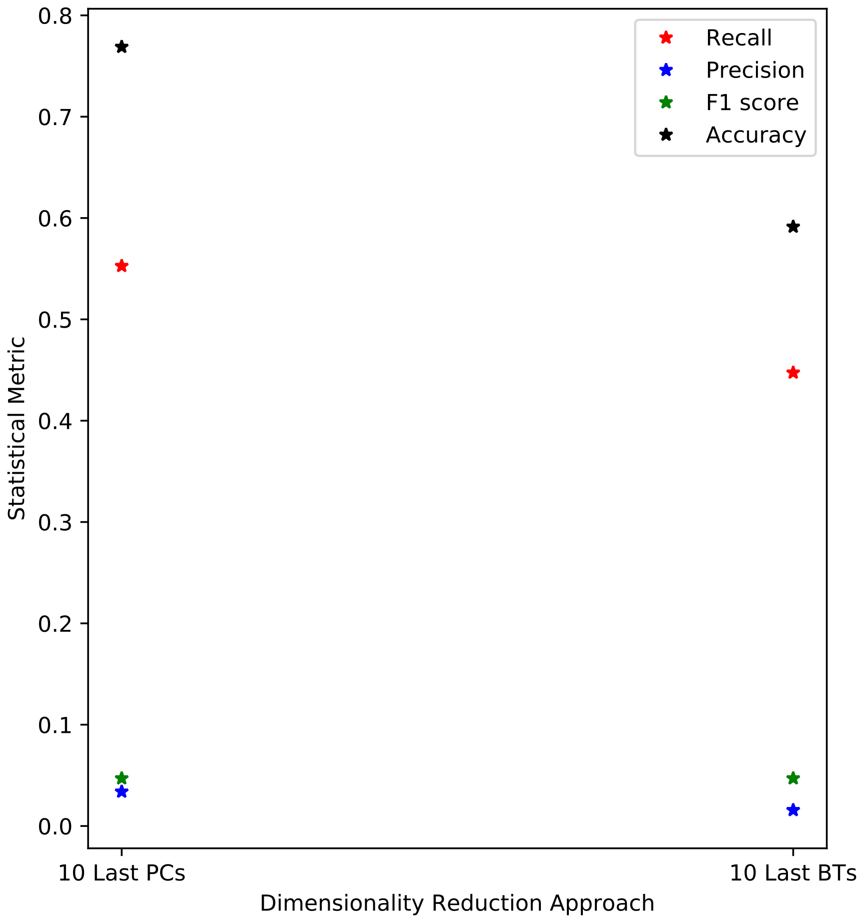


**Figure 5.20:** The recall, precision, F1 score and accuracy of two simple multivariate predictive models applied to the real-world test data set are illustrated here. One of the models is based on the last $10$ principal components and the other on the last $10$ Box-Tiao components. The values of the recall, the precision, the F1 score and the accuracy of the predictive model based on the last $10$ principal components are respectively $0.55$, $0.03$, $0.05$ and $0.77$. The values of the recall, the precision, the F1 score and the accuracy of the predictive model based on the last $10$ Box-Tiao components are respectively $0.45$, $0.02$, $0.05$ and $0.59$.

respectively. The number of false positives using the Box-Tiao and PCA transformation is respectively 1068 and 599 out of 2627 normal states.

The improvement in the performance of the multivariate models compared to the univariate ones is as expected, since there exist many types of failures in the system. A principal or Box-Tiao component might contain predictive information about one type of failure while another one might contain information about another type of failure. The results presented in this subsection only show the potential in multivariate models, and can be improved upon by selecting the optimal combination of the principal and Box-Tiao components and developing a more advanced multivariate CUSUM model.

Chapter 6

# Discussion and Conclusion

Based on the results of the experiments performed in this work, it is concluded that both in the simulation and in the real-world study, the semi-supervised nature of the transformations, described in Subsection 3.4.3 is advantageous in finding the signals in the data with the most predictive information. The results of all the experiments support the hypothesis that in most cases, the principal components with the least variances under the normal state contain the most predictive information. It can be argued that a similar conclusion can be drawn for the Box-Tiao components with the least predictabilities. However, the simulation study shows that this is not always the case and careful experimentation should be done in order to select the signal of interest. The semi-supervised labeling of the data points, described in Subsection 3.4.5 and illustrated in Figures 3.4 and 3.5, enables the construction of a predictive model without prior knowledge about exactly when the anomalies occur with respect to the failures in the system.

One can see in Figure 5.13 that in our simulation study, the average performance of the predictive models based on the Box-Tiao components is slightly better than that based on the principal components. On the other hand, the average performance of the predictive models based on the signals discrepancy, leverage or influence of the PCA transformed data is slightly better than that of the BTA transformed data. The Mann-Whitney U test is used to test if the differences in the four statistical performance metrics of the models in different experiments are significant. Using a significance level of 0.05, the only significant difference between PCA and BTA in our simulation study is in the recall of the predictive models based on the Box-Tiao components and that based on the principal components. In all the other cases, the difference is not significant. Overall, our study doesn't confirm the advantage of using the BTA transformation instead the PCA transformation in our model. However this can be investigated further with a more comprehensive simulation study which looks at several types of changes in the distribution of the data.

In the simulation study, we compare the results of Experiments 2 and 4, presented in Subsection 5.1.2, to the results of Experiments 1 and 3, presented in Subsection 5.1.1. In Experiments 1 and 3, one obtains very low precision and accuracy metrics for the predictive model in a substantial number of simulations. This effect is not observed for Experiments 2 and 4. It is therefore possible to conclude that selecting only one principal or Box-Tiao component as the signal of interest makes the approaches in Experiments 1 and 3 non-robust. The results of Experiments 2 and 4 show that combining the principal and Box-Tiao components into the signals discrepancy, leverage and influence is a more robust approach to the selection of the signal of interest than choosing only one principal or Box-Tiao component.

In Subsection 3.5.1, we present the CUSUM method as an online change-point detection model applied for failure prediction. We refer to the parameters of this model as the CUSUM features and develop an ad hoc method to determine the CUSUM features, with a semi-supervised approach, described in Subsection 3.5.1. This method is tested in our simulation and real-world analyses, and the results are presented in Chapter 5. Based on the performances of the predictive models in our analyses, it is fair to conclude that our proposed semi-supervised ad hoc approach for determining the CUSUM features yields acceptable performances by the predictive model. In a more extensive study, one can apply and compare the results of several methods for determining the CUSUM features. This is however out of the scope of this work.

The results in Chapter 5 show the potential in developing change-point detection models to predict failures in industrial instruments using sensor-based data. However, there are several modifications that can be made to improve the performance of the final predictive model. In Section 6.1, we present recommendations for future work on the subject.

## 6.1 Further Work

In order to solve our business case, a competing approach for this work is to use the unsupervised offline change-point detection method, presented in Section 3.3, to detect change-points in the normal groups, and refer to the time between each pair of consecutive change-points as a normal state. Then one can construct several models such that the distribution of the data in each normal state is explained to a sufficient degree by at least one of the models. Then a predictive model can be developed based on the residuals of a new data point with respect to all of the models constructed based on the normal states. If the residuals are large with respect to all of the models, one can conclude that the new data point is an anomaly that might cause an unplanned shut-down. This approach can be tested in future work on the subject.

In this work, only Experiments 2 and 4 were performed on the real-world data set. These experiments are chosen among the four experiments designed in Section 4.2, based on the results of the simulation study. However, one could argue that the conclusions drawn from performing the experiments on the simulated data sets are not necessarily applicable to the real-world case. In this work, limited computation time prevented us from performing all of the experiments on the real-world data set. We recommend performing Experiments 1 and 3 on the real-world data set in further work on the subject.

Further in this section we will present achievable improvements on the analysis in this work that should be kept in mind in further work on the subject. In Subsection 6.1.1, possible improvements on the process of data preparation are presented. In Subsection 6.1.2, we propose improvements on the method of offline change-point detection presented in Section 3.3. In Subsection 6.1.3, improvements on the simulation study designed in Chapter 4 are proposed. In Subsection 6.1.4, we propose improvements on the methods of dimensionality reduction, presented in Section 3.4. Finally in Subsection 6.1.5, we propose improvements on the online predictive model, presented in Subsection 3.5.1.

### 6.1.1 The Real-World Data Preparation

The failures in the gas compressors studied in this work are categorized into different groups. The possible reasons for a failure can be bearing problems, seals, lube oil system issues, process conditions or combustion related. In this analysis we ignore the categorization of the failures and refer to all types of failures as a starting or a running trip depending on the time at which the failure occurs. The analysis can be improved upon by only looking at one type of failure at a time. This is because the signals containing the most useful information for predicting different types of failures might be different. In addition the anomalies that lead to different types of failures might happen in different time-windows with respect to their corresponding unplanned shut-downs. For example in our case, one could hypothesize that while it might take several minutes for the temperature in a component of the system to rise to a dangerous level, the pressure of the gas might change drastically in a matter of seconds. Therefore one needs different models to detect anomalies in the temperature measurements and in the pressure measurements. This was not done in this work due to the limited number of tripping groups in the data. Dividing the data set into smaller data sets would have resulted in too few tripping groups in each data set for a meaningful analysis to be performed. This issue can be overcome by combining the data from more compressors of the same kind.

The data preparation is performed by DNV GL. A description of this process is provided in Appendix A. A data cleaning process is performed by the author after the data preparation is performed by DNV GL. A description of the data cleaning process is presented in Subsection 4.1.1. The methods applied for the data preparation and the data cleaning can affect the results of the analysis presented in this work. The data preparation and cleaning should be performed according to the assumptions and the requirements of the methods applied in the analysis. For example, one might need a higher frequency than one observation per minute to be able to detect some certain anomalies. The methods applied to remove or replace the NA values, described in Subsection 4.1.1, might also affect the results of the analysis. The NA values recorded by some sensors might themselves be considered as anomalies that can be used to predict some types of the failures in the system. This is however ignored in this work. An analysis of the data preparation and the data cleaning processes and their implications for the results of our methods can be performed. This is however out of the scope of this work. We recommend a better integration of the data preparation and the data cleaning processes in the statistical analysis of the data in future work.

### 6.1.2 The Offline Change-Point Detection Method

The change-point detection method, presented in Section 3.3, is used as a tool in the selection of the signal of interest. Although the theory presented in Section 3.3 allows for multivariate change-point detection, we only use this method on 1-dimensional signals. One could apply multivariate change-point detection to select a multidimensional signal of interest. This signal could for example consist of several principal components. The Mann-Whitney U test, as described in Subsection 3.4.5, can be used to select the optimal $n$-tuple of the principal components for any $n \leq r$.

The change-point detection method, presented in Section 3.3, is implemented in the Ruptures package of Python, developed by Truong (2017). The cost function used in this method, discussed in Subsection 3.3.5, should be derived from some assumptions about the distribution of the data and the types of changes that are desirable to detect. Truong (2017) has implemented a few popular cost functions in the Ruptures package. However, Ruptures allows the user to implement any cost function under some mild regularity conditions. The cost function for a VAR process is not implemented in this package. This function is presented in (3.14). Truong (2017, Personal Correspondence) has expressed interest in a collaboration with the author about implementing the cost function in (3.14). This cost function can be used for multivariate change-point detection in time series data.

When we apply the change-point detection method to the signal discrepancy, leverage and influence, we use the cost function for a Gaussian distribution. However, one might be able to derive the distributions of these signals based on some assumptions about the distribution of the original data. Then a cost function based on the distributions of these signals can be derived and implemented for the purpose of change-point detection. One might obtain more accurate results from the change-point detection if the cost function is derived and implemented based on reasonable assumptions about the distribution of the data.

In our experiments, the penalty function in the offline change-point detection method, discussed in 3.3.3, is inspired by the BIC. The constant $m$ in this function is chosen by trial and error in each experiment. A more extensive analysis of what this function should be in the problem at hand can be performed using CROPS developed by Haynes et al. (2017). If the normal and the transition states used in the selection of the signal of interest are of different lengths, it is important that the number of change-points is estimated consistently using the penalty function. Without the use of a normalizing penalty function, there is statistically a higher chance of detecting change-points in longer time-intervals. In our case, we first assume that the dependence of the penalty function on the length of the running group balances out the above-mentioned statistical effect. This is however not guaranteed. In Figure 5.16, the log($p$-value)s of the Mann-Whitney U test performed on the results of the offline change-point detection applied to the signals discrepancy, leverage and influence of the transformed real-world data set are shown. One can see that the $p$-values are very large. We argue that the reason for this might be the above-mentioned statistical effect. To avoid this issue, we divide the normal states into intervals of equal length to the transition states, as it is illustrated in Figure 3.5. One can see in Figure 5.17 that doing so, one obtains more meaningful $p$-values. Note that even though the division illustrated in Figure 3.5 is used to calculate the $\alpha_i$s and $\beta_j$s used to perform the Mann-Whitney U test, the change-point detection method is still applied to whole running groups. To avoid the above-mentioned issue completely, one should divide the data into normal and transition states of equal lengths and then apply the change-point detection method with the same penalty function to each individual state separately instead of applying it to whole running groups. This is however not done in this work. For the selection of the signal of interest in the simulation study, we divide the running groups into normal, transition and failure states as it is illustrated in Figure 3.4. This is because even though the transition and the normal states are not of the same lengths in this case, the differences in the lengths in the simulated data sets are not as large as they are in the real-world data set. However, this issue should be kept in mind in future simulation studies. Further work is needed for finding penalty functions that yield consistent estimation of the number of change-points under different assumptions.

### 6.1.3 The Simulation Study

The only possible change-points in the simulated data are in the mean of a dimension. However this does not need to be the case. One can test the methods on simulated data sets with different types of change-points. The change-points can for example be in the variance of a dimensions, the predictability of a dimension, or in the structure of the relations between the dimensions. By conducting a comprehensive simulation study including several possible types of change-points, one can determine which one of the four approaches presented in Subsection 4.2 are suitable for detecting each type of change-points.

In Subsection 5.1.1, we present the results of the application of Experiments 1 and 3, designed in Subsection 4.2, to the simulated data sets. In Figures 5.4, 5.3 and 5.7 one can see a division in the distribution of the precision, the accuracy and the F1 score of the predictive models applied to the simulated test data sets. In this work we argue that this division might be caused by a latent factor affecting the performance of the model. In Figure 5.5, it is illustrated that the precision and the accuracy of the predictive models are affected in the same way. Looking at Figure 5.6, we argue that the selected signal of interest can not be causing the division in the distribution of the precision, the accuracy and the F1 score of the predictive models. The latent factor affecting these statistical performance metrics is not identified in this work despite our attempts to do so. One could argue that the reason for the division in the distribution of these metrics is that selecting only one principal or Box-Tiao component as the signal of interest might be a non-robust approach. The difference in the Mann-Whitney U test's $p$-values yielded by the different principal or Box-Tiao components might be small. In this case, the principal or Box-Tiao component yielding the least $p$-value might not yield the best performance by the predictive model. This issue should be addressed and analyzed in future work.

### 6.1.4 The Method of Dimensionality Reduction

The test statistic used in the Mann-Whitney U test is the binary statistic which shows whether or not a change-point is detected in a state. However, this is not the only option, and one could think of several other appropriate test statistics, for example the number of change-points detected in a state. The choice of the test statistic should to some degree be determined by the nature of the predictive model. If the predictive model is developed such that it raises an alarm whenever a change occurs, the test statistic used in this work is appropriate. However, if the predictive model operates on the concentration of the change-points and only raises an alarm when there are too many change-points close to each other in time, then the number of change-points detected in a state is an appropriate test statistic.

The data points are labeled as belonging to a normal, transition or failure state both for the selection of the signal of interest and for the training of the predictive model, as described in Subsections 3.4.5 and 3.5.1 and illustrated in Figures 3.4 and 3.5. For the real-world data set, the variables $\nu$ and $f$ are set in consultation with the field experts. In future work on business cases like ours, one can tune the values of these variables in an ad hoc setting. This might improve the results yielded by the predictive model.

The offline change-point detection method, presented in Section 3.3, combined with the Mann-Whitney U test method, presented in Subsection 3.4.5, are applied for the selection of the signal of interest. The point of applying these two methods is estimating the amount of predictive information in a signal. This can also be done by using the CUSUM method, presented in Subsection 3.5.1, combined with one of the statistical metrics, presented in Subsection 3.5.2. In the analysis in this work, the signal selection data set is used both for the methods applied to determine the signal of interest and for determining the CUSUM features, as described in Subsection 3.5.1. In this analysis, the signal of interest is selected before the CUSUM features are determined. However using the CUSUM method for the selection of the signal of interest, the CUSUM features should be determined for each alternative signal. Then the CUSUM method should be applied to the same signals, and the signal yielding the best performance should be selected as the signal of interest. One can argue that using the signal selection data set for both purposes in this case is a type of overfitting.

On the other hand, since the CUSUM method is used as the predictive model, it is reasonable to use the same method for the selection of the signal of interest. If more data is available, different data sets should be used for the selection of the signal of interest and determining the CUSUM features.

The selection of the signal of interest can also be done in consultation with the field experts. The principal and Box-Tiao components are linear combinations of the dimensions of the scaled data set. One can therefore interpret a principal or Box-Tiao component by comparing the coefficients of the dimensions in its equation. The results of the interpretation can be communicated to the field experts, and one can use their knowledge in the selection of the signal of interest.

### 6.1.5 The Predictive Model

In this work, we propose the CUSUM method as an online change-point detection model for failure prediction. Here this model is only presented for 1-dimensional signals. However there exist variants of the CUSUM method that are applicable to multidimensional data. In Subsection 5.2.4, we present the result of a simple multivariate model that illustrates the potential of further work on multivariate CUSUM models. Golosnoy et al. (2009) present the multivariate CUSUM method and investigate its properties. In addition, they suggest several refinements of the method. Following the mathematical theory in Golosnoy et al. (2009), one can see that applying multivariate CUSUM to principal or Box-Tiao components yields interesting mathematical properties since these signals are orthogonal to each other. We recommend the use of the multivariate CUSUM method as the predictive model in future work.

As described in Subsection 3.5.1, we use an ad hoc method to determine the CUSUM features in this work. One could instead take advantage of the approach provided by Aue and Horváth (2013). They present a review on the work done to modify the original CUSUM method to also work for data exhibiting serial dependence. Aue and Horváth (2013) also prove some asymptotic properties for the test statistics they present.

The estimation error when estimating the distribution of the data in the in-control state is ignored in this work. Inspired by the work done by Gandy and Kvaløy (2013), one could take advantage of methods based on bootstrapping the data used to estimate the distribution in the in-control state to overcome this issue.

The results of applying our proposed predictive model to the real-world test data set are presented in Subsection 5.2.3. One can see that an imbalanced data set results in a large number of false positive predictions by the model. In further work, one could analyze the false positive predictions and try to modify the method to filter them out. This is only possible if one is able to separate the false positive predictions from the true positive ones to a sufficient degree. In this case, a filter can be constructed that allows the majority of the true positive predictions to pass, filtering out the majority of the false positive ones. This can potentially improve the performance of the predictive model substantially.

In this work the CUSUM method is used as the predictive model. Aminikhanghahi and Cook (2017), in their survey article, enumerate, categorize, and compare many of the methods for change-point detection in time series. They also introduce several criteria to compare the algorithms. Other online change-point models, presented in Aminikhanghahi and Cook (2017), can be used instead of the CUSUM method in future works. The choice of the model should depend on the assumptions and the specifications of the problem setting at hand.

## 6.2 Conclusion

In this work we propose a semi-supervised change-point detection method applied for failure prediction in industrial instruments. Four possible approaches are tested with experiments designed in Chapter 4. A test case is provided by a European GTS owner and operator through Norwegian DNV GL. The case is analyzing the possibility of developing a failure prediction model for the gas compressors and the gas turbines used by the GTS operator, using historical sensor-based data from these machines.

Since many aspects of the problem at hand were unknown prior to this analysis, few assumptions could be made when trying to select an optimal approach. It is not known which parts of the data could be labeled as normal or non-normal. This challenge is overcome by introducing semi-supervised change-point detection methods. Applying change-point detection methods for failure prediction, it is challenging to distinguish between normal changes in the operation mode of the system and non-normal changes that cause failures. To overcome this issue, principal component analysis and Box-Tiao analysis are used in a semi-supervised manner for dimensionality reduction. For the Box-Tiao analysis a model should be assumed for the data. We use the vector autoregressive model for this purpose. Finally a predictive model is developed and evaluated on a test data set. The CUSUM method is used for this purpose. All the methods presented in this work are also tested and evaluated in a simulation study.

The analyses in this work could be improved upon given a longer time frame. Achievable improvements are proposed in Section 6.1. Given the analyses performed in this work and the current results, we conclude that applying the methods in this work for failure prediction in our business case might be economically profitable if the economic value of a true positive prediction is much larger than the economic loss of a false positive prediction. A profitability analysis of the predictive model can be performed based on the results presented in Subsection 5.2.3. This is however out of the scope of this work.

In this work, as an original work by the author, the theories of the vector autoregressive model, the offline change-point detection method, the principal and the Box-Tiao analysis, the Mann-Whitney U test, the signals discrepancy, leverage and influence, the CUSUM method and statistical performance metrics have been combined to create end-to-end solutions to the problem of sensor-based failure prediction in industrial instruments. The main contribution of the author, as illustrated in Figure 3.1 is combining all the methods presented in this work into four possible end-to-end solutions and testing and evaluating the solutions on the simulated and the real-world data sets. The semi-supervised signal selection methods, discussed in Subsection 3.4.5, the ad hoc method of determining the CUSUM features, presented in Subsection 3.5.1, and the pooling method of estimating the covariance matrices, presented in Subsection 3.4.4, are original ideas of the author. The author has developed large parts of the implementation of the methods. The Python script developed by the author and used in this work is available in Appendix C.

In Chapter 5, the performance of our proposed end-to-end solutions and the predictive information in the last principal and Box-Tiao components are illustrated. Given the results presented in Chapter 5, we conclude that there is a potential in applying methods of change-point detection for predicting failure in the gas compressors analyzed in this work. This might also be the case for other industrial instruments with a sensor-based monitoring system. We recommend further work on the subject by conducting a similar multivariate analysis.

# Bibliography

Aminikhanghahi, S., Cook, D. J., 2017. A survey of methods for time series change point detection. Knowledge and information systems 51 (2), 339–367.

Arnold, T., et al., 2018. Sas/qc 15.1 users guide. [Online; accessed 2019-06-27]. URL https://documentation.sas.com/?docsetId=qcug&docsetTarget=qcug_cusum_sect039.htm&docsetVersion=15.1&locale=en

Aue, A., Horváth, L., 2013. Structural breaks in time series. Journal of Time Series Analysis 34 (1), 1–16.

Bai, J., 2000. Vector autoregressive models with structural changes in regression coefficients and in variance-covariance matrices. ANNALS OF ECONOMICS AND FINANCE 1 (1), 303–339.

Bakka, K. B., 2018. Changepoint model selection in Gaussian data by maximization of approximate Bayes factors with the pruned exact linear time algorithm. Master's thesis at NTNU, 108–114.
URL https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2558597

Box, G. E. P., Tiao, G. C., 08 1977. A canonical analysis of multiple time series. Biometrika 64 (2), 355–365.
URL https://doi.org/10.1093/biomet/64.2.355

Casella, G., Berger, R. L., 2002. Statistical inference. Vol. 2. Duxbury Pacific Grove, CA.

Chandola, V., Banerjee, A., Kumar, V., Jul. 2009. Anomaly detection: A survey. ACM Comput. Surv. 41 (3), 15:1–15:58.
URL http://doi.acm.org/10.1145/1541880.1541882

Choudhary, P., 2017. Introduction to anomaly detection. [Online; accessed 2018-11-19].

URL https://www.datascience.com/blog/python-anomaly-detection

Fisher, R. A., 1925. Statistical methods for research workers. Genesis Publishing Pvt Ltd.

Gandy, A., Kvaløy, J. T., 2013. Guaranteed conditional performance of control charts via bootstrap methods. Scandinavian Journal of Statistics 40 (4), 647–668.

GE Power, 2018. Lm2500 aeroderivative gas turbine. [Online; accessed 2018-12-03].
URL https://www.ge.com/power/gas/gas-turbines/lm2500

Golosnoy, V., Ragulin, S., Schmid, W., 2009. Multivariate CUSUM chart: properties and enhancements. AStA Advances in Statistical Analysis 93 (3), 263–279.

Haynes, K., Eckley, I. A., Fearnhead, P., 2017. Computationally efficient changepoint detection for a range of penalties. Journal of Computational and Graphical Statistics 26 (1), 134143.
URL https://doi.org/10.1080/10618600.2015.1116445

Hinkley, D., 1970. Inference about the change-point in a sequence of random variables. Biometrika 57.

Jones, E., Oliphant, T., Peterson, P., et al., 2001–. SciPy: Open source scientific tools for Python. [Online; accessed 2018-12-14].
URL http://www.scipy.org/

Kadous, W., 2011. What are some time-series classification methods? [Online; accessed 2018-11-19].
URL https://www.quora.com/What-are-some-time-series-\classification-methods

Killick, R., Fearnhead, P., Eckley, I., 12 2012. Optimal detection of changepoints with a linear computational cost. Journal of the American Statistical Association 107, 1590–1598.

Koehrsen, W., 2018. Beyond accuracy: Precision and recall. [Online; accessed 2019-06-04].
URL https://towardsdatascience.com/beyond-accuracy-precision-and-recall-3da06bea9f6c

Kuncheva, L. I., Faithfull, W. J., Jan 2014. PCA feature extraction for change detection in multidimensional unlabeled data. IEEE Transactions on Neural Networks and Learning Systems 25 (1), 69–80.

Larsen, R., Marx, M. L., 1981. An introduction to mathematical statistics and its applications. Prentice-Hall.

Lu, C.-W., Reynolds Jr, M. R., 2001. CUSUM charts for monitoring an autocorrelated process. Journal of Quality Technology 33 (3), 316–334.

Lütkepohl, H., 2005. New introduction to multiple time series analysis. Springer Science & Business Media.

Mann, H. B., Whitney, D. R., 1947. On a test of whether one of two random variables is stochastically larger than the other. The Annals of Mathematical Statistics 18 (1), 50–60.
URL `http://www.jstor.org/stable/2236101`

Mezzadri, F., 2006. How to generate random matrices from the classical compact groups. arXiv preprint math-ph/0609050.

Page, E. S., 1954. Continuous inspection schemes. Biometrika 41 (1/2), 100–115.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E., 2011. Scikit-learn: Machine learning in Python. Journal of Machine Learning Research 12, 2825–2830.

Perera, S., 2015. Introduction to anomaly detection: Concepts and techniques. [Online; accessed 2018-11-19].
URL `https://iwringer.wordpress.com/2015/11/17/anomaly-detection-concepts-and-techniques/`

Roberts, S., 1959. Control chart tests based on geometric moving averages. Technometrics 1 (3), 239–250.

Seabold, S., Perktold, J., 2010. Statsmodels: Econometric and statistical modeling with python. In: 9th Python in Science Conference.

Seber, G. A., 2009. Multivariate observations. Vol. 252. John Wiley & Sons.

Shorack, G., Wellner, J., 1986. Empirical processes with statistical application.

Shyu, M.-L., Chen, S.-C., Sarinnapakorn, K., Chang, L., 2003. A novel anomaly detection scheme based on principal component classifier. Tech. rep., MIAMI UNIV CORAL GABLES FL DEPT OF ELECTRICAL AND COMPUTER ENGINEERING.

Steinwart, I., Hush, D., Scovel, C., Dec. 2005. A classification framework for anomaly detection. J. Mach. Learn. Res. 6, 211–232.
URL `http://dl.acm.org/citation.cfm?id=1046920.1058109`

Truong, C., 2017. Ruptures. [Online; accessed 2018-11-29].
URL `http://ctruong.perso.math.cnrs.fr/ruptures-docs/build/html/index.html`

Truong, C., Oudre, L., Vayatis, N., 2018. A review of change point detection methods. CoRR abs/1801.00718.
URL `http://arxiv.org/abs/1801.00718`

Tsay, R. S., 2013. Multivariate time series analysis: with R and financial applications. John Wiley & Sons.

Walpole, R. E., Myers, R. H., Myers, S. L., Ye, K., 1993. Probability and statistics for engineers and scientists. Vol. 5. Macmillan New York.

Xiao, H., Wu, W. B., et al., 2012. Covariance matrix estimation for stationary time series. The Annals of Statistics 40 (1), 466–493.

Zar, J. H., 1998. Biostatistical Analysis. New Jersey: Prentice Hall International, INC.

# Appendices

Appendix A

# The Description of the Data Preparation Process

This document is provided to us by DNV GL and is not written by the author.

The raw data available in this project comprises timestamped sensor and trip log data from six stations (locations) covering a total of 10 compressors spanning a period of 14 years. This data is distributed over 28000 files in `.hlg` historian format.

## A.1 Data Standardization

### A.1.1 Format converting

Process of data:

- Download of all alert historical data for 6 sites ($> 28000$ files)
- Consolidation of alert baseline configuration to ensure as far as possible a common configuration of baselines with the maximum value for machinery monitoring
- Collection of alarm and trip configurations for each machine
- Software tools developed to allow initial data processing
- Operating log data consolidated back to 2004 using alert and HPMIS logs

Processed data is:

- One file per measurement per year
- Number of files: $> 172,000$ files

- File Format: `.csv` format
- Size of data: Over 5TB of data

### A.1.2 Sensor name harmonisation

In order to compare the same sensors across different compressors, the sensor names have to be harmonized. Each compressor typically has its own local sensor-naming convention, so a large amount of manual work has been performed to harmonize sensor names for the 10 compressors studied here in order to perform analysis across multiple compressors.

## A.2 Data Preparation and Cleaning

The data we focus is the data when the compressor is on as the case we are trying to solve is predicting the tripping when compressor is working. Furthermore, as there are many factors which might cause the tripping, we would like to look for information within many relevant sensors at the same time.

The sensor *GasGenerator-Speed-Speed* is used as an indication of the status of a compressor. If the speed is above 500 rpm, it is assumed that the compressor state is labelled ON. Otherwise, the compressor state is labelled OFF. In this way, the sensor data has been divided into distinct running groups. Each running group corresponds to a session of use of the compressor: each running group has a start timestamp, an end timestamp and comprises all the sensor data between those two timestamps.

Aside from sensor data, historical log data is available which contains manually-logged information about the timestamps when the compressor was turned on, when the compressor was off, what is the reason for the stopping of the compressor and the failure causes if the stop was unnormal. It was decided to restrict the data set to trips concerned with combustion problems, since these are common. Based on expert advice concerning which sensors are relevant to combustion trips, an active sensor set was defined as a subset of the available sensors.

Different sampling frequency across sensors: The raw sensors data are collected at different sampling frequency varying from several seconds to days. In order to consider all the focused sensors simultaneously, the sensors have been resampled at one-minute sampling frequency such that all the timestamp across sensors have been aligned for multivariate analysis.

The sensors related to combustion-related trips are resampled at 1-minute sample frequency. As a starting point, at each resampled timestamp, each sensors value is equal to the value in the latest timestamp in the raw data while the compressor is working.

The sensor data and log data are combined to create a data table containing both the input (observable) variables and the target feature (trip/no trip). The following processes for cleaning data have been performed:

1. If the maximum speed in a running group is below 3000rpm, it is assumed that this running group belongs to a specific shutdown sequence. Hence, it has been removed for further analysis.

2. It is assumed that the overlapping period between sensor data running group and the period in log data should be at least 50% of time coverage in period at log data. Otherwise, we will remove the data from that running group because of the uncertainty in mismatching start time and end time between a running group and the log data.

3. If there are multiple running groups from sensor data corresponding to one period in log data, then only the last of these running groups is retained.

4. The end of a running group should align with the end time in the log data. It is often Observed that there is a 1 hour difference between the two. This is assumed to be due to the difference between UTC and local time used for the timestamps in sensor data and log data. [sensor *StatusRunningTrip* has been checked against log data to confirm the rough one hours difference between end time in the period at log data and the running group ending time.]

5. During the compressor shutdown process, the data points with speed below idle speed are removed.

6. Data points with missing sensors values are removed. [Treatment for missing data: Deletion, Imputation]

7. For each running group, the first 50 minutes of data has been removed because the system takes some time to warm up and enter a stable operational state.

For running trip group, the last 20 datapoints are flagged as *Class* 1 *Running Trip* and others are flagged as *Class* 0 *Normal Stop.*

### A.2.1 Aggregation of temperature sensors

At each time stamp, eight exhaust gas temperature sensors (`GasGenerator-ExhaustTemperature-TempExhTC1-8ISO`) have been aggregated to min, max, median and standard deviation to capture the varying range and variance within those similar sensors. Through this aggregation, the problem among those eight sensors has been generalized to some extent. The same aggregation method has been applied to similar temperature sensors as well.

### A.2.2 Data quality of temperature sensors

There may be some data quality issues for similar exhaust gas temperature sensors which need to be addressed. For example, Sensor

`GasGenerator-ExhaustTemperature-TempExhTC2ISO` at Unit `Aberdeen-UnitA`, the maximum value of this sensor is above 1000 while other similar sensors are around 400 to 800. However, the value from this sensor was totally deviating from other similar sensors. This could be due to configuration errors in the sensor logging instrument. Another example is sensor value `GasGenerator-ExhaustTemperature-TempExhTC1ISO` is low however, the remaining 7 similar sensors are about 800. This may be due to the sensor is not working properly. For simplicity, as agreed with domain experts we have removed those periods with sensor temperature too high (value above 1000) or too low when compressor is on (below 250). More thorough data quality check should be performed for all focused sensors. Sensors of bad quality should be removed for further analysis.

### A.2.3 Imbalanced Dataset

After pre-processing and cleaning of data, the dataset is ready for further analysis. The proportion of normal stop runs against number of running trips is about 26.65 : 1. Because the running trip is rather rare among the runs for compressors, the data is by nature an unbalanced class problem which makes the prediction more challenging. The model will always try to predict the majority class by default. This shall be considered when building up model. Therefore, in order to make algorithm easier to find pattern within the data, we perform modification for the number of data points used for analysis. For a normal run, we randomly pick up 20 data points within the run. For a running trip, we pick up the last 20 data points.

Appendix B

# Dimensions List

Here we present the name of the sensors analyzed in this work. These are the dimensions in the real-world data set after the data cleaning process is performed. The number in column *Rank* 1 refers to the rank of the dimension in Figure 4.3, and the number in column *Rank* 2 refers to the rank of the dimension in Figure 5.14.

| Dimensions | Rank 1 | Rank 2 |
|---|---|---|
| GasCompressor_BearingTemperature_TempBrgDriveEnd1 | 4 | 4 |
| GasCompressor_BearingTemperature_TempBrgNonDriveEnd1 | 5 | 44 |
| GasCompressor_BearingTemperature_TempBrgThrustActive1 | 44 | 40 |
| GasCompressor_BearingTemperature_TempBrgThrustInactive1 | 40 | 59 |
| GasCompressor_Efficiency_EffIsen | 42 | 42 |
| GasCompressor_FlowProcessND_FlowProcessND | 59 | 5 |
| GasCompressor_FlowProcessStandardVolume_FlowProcessStdVol | 41 | 55 |
| GasCompressor_HeadPolyAct_HeadPolyAct | 55 | 56 |
| GasCompressor_PowerAbsorbedGas_PowerAbsorbedGas | 56 | 41 |
| GasCompressor_PressRatio_PressRatio | 43 | 43 |
| GasCompressor_PressureDischarge_PressDisch | 34 | 34 |
| GasCompressor_PressureSuction_PressSuct | 24 | 38 |
| GasCompressor_SpeedND_SpeedND | 21 | 21 |
| GasCompressor_Vibration_VibDriveEndY | 30 | 28 |
| GasCompressor_Vibration_VibNonDriveEndX | 38 | 32 |
| GasCompressor_Vibration_VibNonDriveEndY | 19 | 24 |
| GasGenerator_AirInlet_TempAirInlet | 32 | 19 |

| Dimensions | Rank 1 | Rank 2 |
|---|---|---|
| GasGenerator_CompressorDelivery_PressCompDelISO | 36 | 26 |
| GasGenerator_CompressorDelivery_TempCompDelISO | 26 | 36 |
| GasGenerator_ExhaustTemperature_TempExhISO | 20 | 30 |
| GasGenerator_ExhaustTemperature_TempExhMax | 28 | 20 |
| GasGenerator_ExhaustTemperature_TempExhMin | 39 | 18 |
| GasGenerator_ExhaustTemperature_TempExhSpread | 18 | 39 |
| GasGenerator_ExhaustTemperature_TempExhTC1DevMean | 22 | 22 |
| GasGenerator_ExhaustTemperature_TempExhTC1ISO | 23 | 27 |
| GasGenerator_ExhaustTemperature_TempExhTC2DevMean | 33 | 23 |
| GasGenerator_ExhaustTemperature_TempExhTC2ISO | 27 | 35 |
| GasGenerator_ExhaustTemperature_TempExhTC3DevMean | 51 | 33 |
| GasGenerator_ExhaustTemperature_TempExhTC3ISO | 35 | 51 |
| GasGenerator_ExhaustTemperature_TempExhTC4DevMean | 50 | 37 |
| GasGenerator_ExhaustTemperature_TempExhTC4ISO | 52 | 29 |
| GasGenerator_ExhaustTemperature_TempExhTC5DevMean | 37 | 50 |
| GasGenerator_ExhaustTemperature_TempExhTC5ISO | 31 | 52 |
| GasGenerator_ExhaustTemperature_TempExhTC6DevMean | 25 | 6 |
| GasGenerator_ExhaustTemperature_TempExhTC6ISO | 29 | 25 |
| GasGenerator_ExhaustTemperature_TempExhTC7DevMean | 58 | 31 |
| GasGenerator_ExhaustTemperature_TempExhTC7ISO | 45 | 57 |
| GasGenerator_ExhaustTemperature_TempExhTC8DevMean | 57 | 45 |
| GasGenerator_ExhaustTemperature_TempExhTC8ISO | 6 | 49 |
| GasGenerator_FlameTemperature_TempFlame | 2 | 58 |
| GasGenerator_FlowFuel_FlowFuelMassISO | 7 | 7 |
| GasGenerator_FlowFuel_FlowFuelMassInner | 49 | 2 |
| GasGenerator_FlowFuel_FlowFuelMassOuter | 8 | 13 |
| GasGenerator_FlowFuel_FlowFuelMassPilot | 13 | 8 |
| GasGenerator_FlowFuel_FlowFuelVolISO | 3 | 3 |
| GasGenerator_FuelBurnerConfiguration_FuelBurnerConfigDmd | 60 | 15 |
| GasGenerator_FuelGas_PressFuelGasInner2 | 16 | 14 |
| GasGenerator_FuelGas_TempFuelGas1 | 10 | 47 |
| GasGenerator_GasProperties_GasPropFuelRho | 0 | 17 |
| GasGenerator_GasTurbineEfficiency_EffGT | 14 | 46 |
| GasGenerator_PowerNTIISO_PowerNTIISO | 15 | 10 |
| GasGenerator_PowerNTIPct_PowerNTIPct | 47 | 60 |
| GasGenerator_PowerNTI_PowerNTI | 17 | 0 |
| GasGenerator_PressExhISO_PressExhISO | 46 | 1 |
| GasGenerator_PressExh_PressExh | 1 | 12 |
| GasGenerator_SpeedISO_SpeedISO | 11 | 11 |
| GasGenerator_Speed_Speed | 12 | 16 |
| PowerTurbine_BearingTemperature_TempBrgDriveEnd1 | 61 | 61 |
| PowerTurbine_BearingTemperature_TempBrgNonDriveEnd1 | 53 | 53 |
| PowerTurbine_Speed_Speed | 54 | 54 |
| Station_TemperatureDischarge_TempDisch | 48 | 48 |
| Station_TemperatureSuction_TempSuct | 9 | 9 |

Appendix C

# Python Script

## C.1 Functions and Packages

Here one can find the Python script for the functions and packages needed to perform the analysis in this work.

### C.1.1 External Packages

First several packages need to be imported.

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from statsmodels.tsa.vector_ar.var_model
    import VAR, VARProcess
import ruptures as rpt
import matplotlib.pyplot as plt
from random import uniform
from sklearn.datasets import make_spd_matrix
from scipy.stats import ortho_group
from scipy.stats import mannwhitneyu
from tqdm import tqdm
from sklearn.model_selection import train_test_split
from statsmodels.tsa.ar_model import AR
```

### C.1.2 BTA and PCA

The following class is used for BTA and PCA transformations. If a BTA transformation is desired, one should set `BoxTiao` to `True`, and otherwise `False`. One can also calculate the discrepancy, the leverage and the influence of a transformed point. A VAR process of order `order` is assumed in the estimation of the predictability matrix. This has to be determined when the object is created. The other functions only take in data. If one or both of the covariance matrices $\mathbf{\Sigma}$ and $\mathbf{\Gamma}$ are known, they can be taken in as inputs in the `fit` function. Then one should set `SigmaEstimation` and `GammaEstimation` to `False` when the object is created. If the data is divided into running groups, the data should be in a Pandas' data frame, and the column `gr_nr` should indicate which running group each data point belongs to. Then the input `group_c_name` should be set to the string `gr_nr`.

```
class VARBT:
    def __init__(self, order = 1, SigmaEstimation = True,
        GammaEstimation = True,
            groug_c_name = 'There is only one group',
                BoxTiao = True):
        self.order = order
        self.SigmaEstimation = SigmaEstimation
        self.GammaEstimation = GammaEstimation
        self.groug_c_name = groug_c_name
        self.BoxTiao = BoxTiao
        if(BoxTiao == False):
            self.SigmaEstimation = False
    def fit(self,data, Sigma = 0, Gamma = 0):
        data = pd.DataFrame(data)
        if(self.groug_c_name in data.columns):
            self.TrueColumns =
                data.columns.drop(self.groug_c_name)
            nrOFgr =
                len(np.unique(data[self.groug_c_name]))
            grSums = data.groupby(self.groug_c_name).
                apply(lambda x: np.sum(x))
                    [self.TrueColumns]
            self.meanToScale = np.sum(grSums)/len(data)
        else:
            self.meanToScale = np.mean(data)

        if(self.GammaEstimation):
            def estimate_ScaledGamma(GammaData):
                GammaData = GammaData.
                    reset_index(drop=True)
                GammaDataCentered =
                    StandardScaler(with_std=False).
                        fit_transform(GammaData)
                return(np.matmul(GammaDataCentered.
                    transpose(),GammaDataCentered))
            if(self.groug_c_name in data.columns):
                ScaledGammas = data.groupby(self.
                    groug_c_name).apply(lambda x:
                        estimate_ScaledGamma(x[self.
                            TrueColumns]))
                Gamma = (sum(ScaledGammas))/
                    (len(data)-nrOFgr)
            else:
                Gamma = (estimate_ScaledGamma(data))/
                    (len(data)-1)

        if(self.SigmaEstimation):
            def estimate_ScaledSigma(SigmaData):
                SigmaData = SigmaData.
                    reset_index(drop=True)
                TSmodel = VAR(SigmaData)
                try:
                    TSresult = TSmodel.fit(self.order)
                except:
                    TSresult = TSmodel.fit(self.order,
                        trend='nc')
                return(np.matmul(TSresult.resid.
                    transpose(),TSresult.resid))
            if(self.groug_c_name in data.columns):
                ScaledSigmas = data.groupby(self.
                    groug_c_name).apply(lambda x:
                        estimate_ScaledSigma(x
                            [self.TrueColumns]))
                Sigma = (sum(ScaledSigmas))/
                    (len(data)-(self.order*
                        (len(data.columns))*
                            nrOFgr))
            else:
                Sigma= (estimate_ScaledSigma(data))/
                    (len(data)-(self.order*
                        (1+len(data.columns))))
        if(self.groug_c_name in data.columns):
            data = data[self.TrueColumns]
        self.Sigma = Sigma
        self.Gamma = Gamma
        if(self.BoxTiao):
            AdjQ = np.matmul(np.linalg.inv(Sigma),Gamma)
        else:
            AdjQ = Gamma
        D_unsorted, eigVec_unsorted = np.linalg.eig(AdjQ)
        idx = np.argsort(-D_unsorted)
        self.eigVec = (eigVec_unsorted.transpose()[idx]).
```

```
            transpose()
        if(self.BoxTiao):
            self.predictabilities = 1-np.power
                (D_unsorted[idx],-1)
            self.variances = (np.matmul
                (self.eigVec.transpose(),
                    np.matmul(self.Gamma,self.eigVec))).
                        diagonal()
        else:
            self.variances = D_unsorted[idx]
    def transform(self,trans_data):
        trans_data = pd.DataFrame(trans_data)
        if(self.groug_c_name in trans_data.columns):
            trans_data = trans_data[self.TrueColumns]
        trans_data = trans_data-self.meanToScale
        return(np.matmul(trans_data,self.eigVec))
    def fit_transform(self,ft_data, ft_Sigma = 0,
        ft_Gamma = 0,
            ft_groug_vr_name = 'There is only one group'):
        ft_data = pd.DataFrame(ft_data)
        self.fit(data=ft_data, Sigma = ft_Sigma,
            Gamma = ft_Gamma,
                groug_vr_name = ft_groug_vr_name)
        return(self.transform(trans_data = ft_data))
    def inverse_transform(self, it_data):
        return(pd.DataFrame(np.matmul
            (it_data,
                np.linalg.inv(self.eigVec)))+
                    self.meanToScale)
    def Leverage(self, lev_data,s_l=1):
        lev_trans_data = self.transform(lev_data)
            [:,0:s_l]
        lev_trans_data_squared =
            lev_trans_data*lev_trans_data
        lev_final_data = lev_trans_data_squared
        for counter in range(0,s_l):
            lev_final_data[:,counter] /=
                self.variances[counter]

        return(sum(lev_final_data.transpose()))
    def Discrepancy(self,dis_data,s_di=1):
        dis_trans_data = self.transform(dis_data)
        n_c = dis_trans_data.shape[1]
        dis_trans_data = dis_trans_data[:,s_di:n_c]
        dis_trans_data_squared =
            dis_trans_data*dis_trans_data
        dis_final_data = dis_trans_data_squared
        for counter in range(s_di, n_c):
            dis_final_data[:, counter-s_di] /=
                self.variances[counter]
        return(sum(dis_final_data.transpose()))
    def Influence(self, inf_data, s_i=1):
        return(self.Discrepancy(inf_data, s_di = s_i)*
            self.Leverage(inf_data, s_l=s_i))
```

## C.1.3 CUSUM

Here we present the functions used for the creation and the evaluation of the predictive model.

The following function takes in a signal and the CUSUM features $T$ and $K$. If `side='hi'`, the function returns the sequence $C_i^+$ and if `side='lo'`, the function returns the sequence $C_i^-$. These sequences are defined in (3.25).

```
def Compute_CUSUM(signal, target, k, side = 'hi'):
    c = 0
    result = []
    if(side=='hi'):
        for xi in signal:
            c= np.max([0,(xi-(target+k)+c)])
            result.append(c)
    else:
        for xi in signal:
            c= np.max([0,(target-k+c-xi)])
            result.append(c)
    return(result)
```

Given a list of signals that are observations of the same variable, the following function estimates the mean and the standard deviation of that variable.

```
def Compute_target_and_std(gr_signals):
    means = [np.mean(x) for x in gr_signals]
    variances = [np.var(x, ddof=1) for x in gr_signals]
    lens = [len(x) for x in gr_signals]
    target = sum(x*y for x,y in zip(means, lens))/
        sum(lens)
    std = np.sqrt(sum(x*(y-1) for x,y in
        zip(variances, lens))/(sum(lens)-len(gr_signals)))
    return([target, std])
```

The following function divides the `max_signal` into intervals of the length `per_len`, and for each interval, it calculates the maximum of `max_signal` in that interval. Then it returns a list of the maximums.

```
def Period_Maxes(max_signal, per_len):
    result = [np.max(max_signal
        [(x*per_len):((x+1)*per_len)]) for x in
            range(0,int(len(max_signal)/per_len))]
    if((int(len(max_signal)/per_len)*per_len) <
        len(max_signal)):
        result.append(np.max(max_signal
            [(int(len(max_signal)/per_len)*per_len):]))
    return(result)
```

Given data, the following function determines the CUSUM features with an ad hoc approach as described in Subsection 3.5.1. The inputs `NormalSOFs` and `TripSOFs` are lists of signals in the normal groups and in the tripping groups, respectively. Looking at Figure 3.5, the inputs `NormalLen` and `TransitionEnd` should respectively be set to $\nu - f$ and $\nu$. If we allow for the late detection of the transition state, `DetectMarg` should be set to the allowed delay for the detection after the transition state has ended. The CUSUM feature $K$ is selected among the values of the list `k_space` times the estimated standard deviation of the signal. The variable `include_low` is set to `True` if it is desired to also declare the process out of control when $C_i^- > h$. Otherwise only $C_i^+$ will be taken into consideration. The input `metric` should be selected among `'accuracy'`, `'recall'`, `'precision'` or `'f1'`. This input indicates which statistical metric should be maximized in the ad hoc approach. The function returns the CUSUM features in addition to the estimated standard deviation of the signal and the performance of the model on the signal selection data set.

```
def Make_CUSUM(NormalSOFs, TripSOFs, NormalLen,
    TransitionEnd, DetectMarg=0, k_space=[0.5,1,2,4,8],
        include_low = True,metric = 'f1',
            priority = 'largest'):
    if(priority == 'largest'):
        priority_ind = -1
    else:
        priority_ind = 0
    NormalSOFs=[x[(len(x)%(NormalLen+DetectMarg)):]
        for x in NormalSOFs]
    TripSOFs=[x[((len(x)-NormalLen-TransitionEnd)%
        (NormalLen+DetectMarg)):] for x in TripSOFs]
    NormalTarget, NormalSTD =
        Compute_target_and_std(NormalSOFs)
    k_space = np.asarray(k_space)
    k_set = np.sort(k_space * NormalSTD)
    TripTimes = np.asarray
        ([len(x) for x in TripSOFs])
    NormalTimes = np.asarray
        ([len(x) for x in NormalSOFs])
    TotalNrOfN = 0
```

```
for counter in range(0,len(NormalSOFs)):
    TotalNrOfN += np.ceil(NormalTimes[counter]/
        (NormalLen+DetectMarg))
for counter in range(0,len(TripSOFs)):
    TotalNrOfN += np.ceil((TripTimes[counter]-
        NormalLen-TransitionEnd) /
            (NormalLen+DetectMarg))
TotalNrOfP = len(TripSOFs)
best_kh_metric=[]
best_hs=[]
for temp_k in k_set:
    NormalCUSUMhis = [Compute_CUSUM
        (x,NormalTarget,temp_k)
            for x in NormalSOFs]
    TripCUSUMhis = [Compute_CUSUM
        (x,NormalTarget,temp_k)
            for x in TripSOFs]
    h_set = [Period_Maxes(x,NormalLen)
        for x in NormalCUSUMhis]
    if(include_low):
        NormalCUSUMlos = [Compute_CUSUM
            (x,NormalTarget,temp_k,side='lo')
                for x in NormalSOFs]
        TripCUSUMlos = [Compute_CUSUM
            (x,NormalTarget,temp_k,side='lo')
                for x in TripSOFs]
        h_set.extend([Period_Maxes
            (x,NormalLen)
                for x in NormalCUSUMlos])
    h_set=np.sum(h_set)
    h_set = np.sort(np.unique(h_set))
    best_h_metric=[]
    for temp_h in h_set:
        TripOutWheres = [np.where(x)[0]
            for x in [x>temp_h
                for x in TripCUSUMhis]]
        NormalOutWheres = [np.where(x)[0]
            for x in [x>temp_h
                for x in NormalCUSUMhis]]
        if(include_low):
            TripOutWheres.extend([np.where(x)[0]
                for x in [x>temp_h
                    for x in TripCUSUMlos]])
            NormalOutWheres.extend(
                [np.where(x)[0] for x in
```

```
                    [x>temp_h
                        for x in NormalCUSUMlos]])
            TripOutWheres = [np.asarray(x)
                for x in TripOutWheres]
            NormalOutWheres = [np.asarray(x)
                for x in NormalOutWheres]
            TrueP = sum([(len(np.intersect1d(
                range((TransitionEnd-DetectMarg),
                    (TransitionEnd+NormalLen)),
                        (TripTimes[x]-
                            TripOutWheres[x]))) > 0)
                                for x in range
                                    (0,len(TripSOFs))])
            FalseP = 0
            for x in NormalOutWheres:
                FalseP += len(np.unique(np.floor
                    (x/(NormalLen+DetectMarg))))
            for x in range(0,len(TripSOFs)):
                FalseP += len(np.unique(np.floor
                    ((TripOutWheres[x])
                        [(TripOutWheres[x] <=
                            (TripTimes[x]-NormalLen
                                -TransitionEnd))]/
                                    (NormalLen+
                                        DetectMarg))))
            TrueN = TotalNrOfN - FalseP
            FalseN = TotalNrOfP - TrueP
            accuracy = (TrueP+TrueN)/
                (TotalNrOfP+TotalNrOfN)
            recall=TrueP/TotalNrOfP
            precision=0
            if(TrueP>0):
                precision=TrueP/(TrueP+FalseP)
            f1 = 0
            if((precision+recall)>0):
                f1=(2*precision*recall)/
                    (precision+recall)
            specificity = TrueN/(TotalNrOfN)
            best_h_metric.append(eval(metric))
        best_h_metric = np.asarray(best_h_metric)
        best_hs.append(h_set[np.where(best_h_metric
            == np.max(best_h_metric))[0][priority_ind]])
        best_kh_metric.append(np.max(best_h_metric))
    best_kh_metric = np.asarray(best_kh_metric)
    best_metric = np.max(best_kh_metric)
    best_k=k_set[np.where(best_kh_metric == best_metric)
        [0][priority_ind]]
    best_h = best_hs[np.where(best_kh_metric ==
        best_metric)[0][priority_ind]]
    return([NormalTarget, NormalSTD,best_k,best_h,
        best_metric])
```

Given test data and the CUSUM features, the following function evaluates a CUSUM model, and returns a confusion matrix. The inputs `NormalSOFs` and `TripSOFs` are lists of signals in the normal groups and in the tripping groups, respectively. The variables `NormalTarget`, `k` and `h` are the CUSUM features. Looking at Figure 3.5, the inputs `NormalLen` and `TransitionEnd` should respectively be set to $\nu - f$ and $\nu$. If we allow for the late detection of the transition state, `DetectMarg` should be set to the allowed delay for the detection after the transition state has ended. The variable `include_low` is set to `True` if it is desired to also declare the process out of control when $C_i^- > h$. Otherwise only $C_i^+$ will be taken into consideration.

```
def Evaluate_CUSUM(NormalSOFs, TripSOFs,NormalTarget, k,h,
    NormalLen,TransitionEnd, DetectMarg,
        include_low = True):
    NormalSOFs=[x[(len(x)%(NormalLen+DetectMarg)):]
        for x in NormalSOFs]
    TripSOFs=[x[((len(x)-NormalLen-TransitionEnd)%
        (NormalLen+DetectMarg)):] for x in TripSOFs]
    TripTimes = np.asarray([len(x) for x in TripSOFs])
    NormalTimes = np.asarray([len(x)
        for x in NormalSOFs])
    NormalCUSUMhis = [Compute_CUSUM(x,NormalTarget,k)
        for x in NormalSOFs]
    TripCUSUMhis = [Compute_CUSUM(x,NormalTarget,k)
        for x in TripSOFs]
    NormalOutWheres = [np.where(x)[0] for x in [x>h
        for x in NormalCUSUMhis]]
    TripOutWheres = [np.where(x)[0] for x in [x>h
        for x in TripCUSUMhis]]
    if(include_low):
        NormalCUSUMlos = [Compute_CUSUM
            (x,NormalTarget,k,side='lo')
                for x in NormalSOFs]
        TripCUSUMlos = [Compute_CUSUM
            (x,NormalTarget,k,side='lo')
                for x in TripSOFs]
        TripOutWheres.extend([np.where(x)[0]
            for x in [x>h for x in TripCUSUMlos]])
        NormalOutWheres.extend([np.where(x)[0]
            for x in [x>h for x in NormalCUSUMlos]])
    TripOutWheres = [np.asarray(x)
        for x in TripOutWheres]
    NormalOutWheres = [np.asarray(x)
        for x in NormalOutWheres]
    TripOutWheres = np.asarray(TripOutWheres)
    NormalOutWheres = np.asarray(NormalOutWheres)
    TrueP = sum([(len(np.intersect1d(range(
        (TransitionEnd-DetectMarg),
            (TransitionEnd+NormalLen)),
                (TripTimes[x]-TripOutWheres[x]))) > 0)
                    for x in range(0,len(TripSOFs))])
    TotalNrOfN = 0
    FalseP = 0
    for counter in range(0,len(NormalSOFs)):
        TotalNrOfN += np.ceil(NormalTimes[counter] /
            (NormalLen+DetectMarg))
        FalseP += len(np.unique(np.floor(
            NormalOutWheres[counter] /
                (NormalLen+DetectMarg))))
    for counter in range(0,len(TripSOFs)):
        TotalNrOfN += np.ceil(
            (TripTimes[counter]-NormalLen-TransitionEnd)/
                (NormalLen+DetectMarg))
        FalseP += len(np.unique(np.floor(
            (TripOutWheres[counter])
                [(TripOutWheres[counter] <=
                    (TripTimes[counter]-NormalLen-
                        TransitionEnd))]/
                            (NormalLen+DetectMarg))))
    TotalNrOfP = len(TripSOFs)
    FalseN = TotalNrOfP-TrueP
    TrueN = TotalNrOfN-FalseP
    return(pd.DataFrame([[TrueN,FalseP],[FalseN,TrueP]],
        columns=['Predicted Normal','Predicted Trip'],
            index=['Actual Normal', 'Actual Trip']))
```

## C.2 Simulation Study

Here we will present the Python script for the simulation study.

### C.2.1 Data Simulation

First some variables should be set.

```
NumberOfSimulations = 100
```

```
NumberOfDimensions = 10
NumberOfNormalGr = 500
NumberOfNormalSSGr = 50
NumberOfNormalTestGr = 50
NumberOfTripSSGr = 50
NumberOfTripTestGr = 50
lengthSD = 200
minLength = 300
TmodeStart = 90
TmodeEnd = 30
TmodeLength = TmodeStart - TmodeEnd
TDetectionMargin = 15
SearchLen = TmodeStart - TmodeEnd + 2*TDetectionMargin
r_s_vec=range(0,NumberOfSimulations)
```

In the following code, the simulated data sets are generated. The simulated normal training data sets are saved in the list `VecSimulatedNormal`. The simulated signal selection data sets are saved in the lists `VecSimulatedNormalSS` and `VecSimulatedTripSS`, and the simulated test data sets are saved in the lists `VecSimulatedNormalTest` and `VecSimulatedTripTest`.

```
VecNormalChangePoints=[]
VecNormalSSChangePoints=[]
VecNormalTestChangePoints=[]
VecTripSSChangePoints=[]
VecTripTestChangePoints=[]
VecSimulatedNormal = []
VecSimulatedNormalSS = []
VecSimulatedNormalTest = []
VecSimulatedTripSS = []
VecSimulatedTripTest = []
VecTrainingData = []
for simulationCounter in tqdm(r_s_vec):
    r_s=simulationCounter
    np.random.seed(r_s)
    #Making Sigma
    covMat = make_spd_matrix(NumberOfDimensions,
        random_state=r_s)
    #Making A
    A_eigs = (2*np.random.random_sample
        (size=NumberOfDimensions)) -1
    while((-1) in A_eigs):
        A_eigs = (2*np.random.random_sample
            (size=NumberOfDimensions)) -1
    Q = ortho_group.rvs
        (dim=NumberOfDimensions, random_state=r_s)
    A = np.matmul(Q,np.matmul
        (np.diag(A_eigs),Q.transpose()))
    #Make the process
    y_N = VARProcess(coefs=np.asanyarray([A]),
        sigma_u=covMat,coefs_exog=(np.asanyarray([0])))
    #Selecting dimensions of change
    NormalChangeDim, TripChangeDim = [0,0]
    while(NormalChangeDim == TripChangeDim):
        NormalChangeDim, TripChangeDim =
            np.random.randint(NumberOfDimensions,size=2)
    #Initialize values for simulation
    grMade = 0
    NormalChangeAmp = np.std
        (y_N.simulate_var(steps=1000)[:,NormalChangeDim])
    TripChangeAmp = np.std
        (y_N.simulate_var(steps=1000)[:,TripChangeDim])
    NormalChangePoints=[]
    NormalSSChangePoints=[]
    NormalTestChangePoints=[]
    TripSSChangePoints=[]
    TripTestChangePoints=[]
    SimulatedNormal = []
    SimulatedNormalSS = []
    SimulatedNormalTest = []
    SimulatedTripSS = []
    SimulatedTripTest = []
    # Make normal groups for training
    for counter in range(0,NumberOfNormalGr):
        length = int(uniform(minLength,
            (minLength+2*lengthSD+1)))
        changeWhere = int(uniform(0,length))
        NormalChangePoints.append(changeWhere)
        temp = y_N.simulate_var(steps=length)
        temp[changeWhere:,NormalChangeDim] =
            temp[changeWhere:,NormalChangeDim]
                + NormalChangeAmp
        temp = pd.DataFrame(temp)
        temp['gr_nr'] = [grMade+counter]*length
        SimulatedNormal.append(temp)
    grMade += NumberOfNormalGr
    TrainingData = pd.concat(SimulatedNormal)
    # Make normal groups for signal selection
    for counter in range(0,NumberOfNormalSSGr):
        length = int(uniform(minLength,
            (minLength+2*lengthSD+1)))
        changeWhere = int(uniform(0,length))
```

```
        NormalSSChangePoints.append(changeWhere)
        temp = y_N.simulate_var(steps=length)
        temp[changeWhere:,NormalChangeDim] =
            temp[changeWhere:,NormalChangeDim]
                + NormalChangeAmp
        temp = pd.DataFrame(temp)
        temp['gr_nr'] = [grMade+counter]*length
        SimulatedNormalSS.append(temp)
    grMade+= NumberOfNormalSSGr
    # Make trip groups for signal selection
    for counter in range(0,NumberOfTripSSGr):
        length = int(uniform(minLength,
            (minLength+2*lengthSD+1)))
        changeWhere = int(uniform(
            length-TmodeStart,length-TmodeEnd))
        TripSSChangePoints.append(changeWhere)
        temp = y_N.simulate_var(steps=length)
        temp[changeWhere:,TripChangeDim] =
            temp[changeWhere:,TripChangeDim]
                + TripChangeAmp
        temp = pd.DataFrame(temp)
        temp['gr_nr'] = [grMade+counter]*length
        SimulatedTripSS.append(temp)
    grMade+= NumberOfNormalSSGr
    # Make normal groups for testing
    for counter in range(0,NumberOfNormalTestGr):
        length = int(uniform(minLength,
            (minLength+2*lengthSD+1)))
        changeWhere = int(uniform(0,length))
        NormalTestChangePoints.append(changeWhere)
        temp = y_N.simulate_var(steps=length)
        temp[changeWhere:,NormalChangeDim] =
            temp[changeWhere:,NormalChangeDim]
                + NormalChangeAmp
        temp = pd.DataFrame(temp)
        temp['gr_nr'] = [counter]*length
        SimulatedNormalTest.append(temp)
    grMade+= NumberOfNormalSSGr
    # Make trip groups for testing
    for counter in range(0,NumberOfTripTestGr):
        length = int(uniform(minLength,
            (minLength+2*lengthSD+1)))
        changeWhere = int(uniform
            (length-TmodeStart,length-TmodeEnd))
        TripTestChangePoints.append(changeWhere)
```

```
            temp = y_N.simulate_var(steps=length)
            temp[changeWhere:,TripChangeDim] =
                temp[changeWhere:,TripChangeDim]
                    + TripChangeAmp
            temp = pd.DataFrame(temp)
            temp['gr_nr'] = [grMade+counter]*length
            SimulatedTripTest.append(temp)
        grMade+= NumberOfNormalSSGr
        VecNormalChangePoints.
            append(NormalChangePoints)
        VecNormalSSChangePoints.
            append(NormalSSChangePoints)
        VecNormalTestChangePoints.
            append(NormalTestChangePoints)
        VecTripSSChangePoints.
            append(TripSSChangePoints)
        VecTripTestChangePoints.
            append(TripTestChangePoints)
        VecSimulatedNormal.
            append(SimulatedNormal)
        VecSimulatedNormalSS.
            append(SimulatedNormalSS)
        VecSimulatedNormalTest.
            append(SimulatedNormalTest)
        VecSimulatedTripSS.
            append(SimulatedTripSS)
        VecSimulatedTripTest.
            append(SimulatedTripTest)
        VecTrainingData.
            append(TrainingData)
```

### C.2.2 Signal Selection

Here we present the implementation of the four experiments, designed in 4.2, for the purpose of selecting the signal of interest in the simulation study.

In each experiment, one can select the constant $m$ in the BIC inspired penalty function with an ad hoc approach among some alternatives in the list `penVec`. However, we only use one alternative. Note that the element in `penVec` is $\frac{m}{2}$ for each experiment.

#### Experiment 1

The following is the implementation of Experiment 1 for the simulated data. For each simulation, the natural logarithm of the Mann-Whitney U test's $p$-value yielded by the selected signal of interest is saved in `PCALinCombBestLogPvalues`, and the selected signal of interest is saved in `PCALinCombBestDim`.

```
algo=rpt.Pelt(model="ar",jump=int(TmodeLength/5),params
    ={"order": 1})
penVec=[2]
PCALinCombBestLogPvalues = []
PCALinCombBestDim=[]
PCALinCombBestPens=[]
for simulationCounter in tqdm(r_s_vec):
    NormalChangePoints = VecNormalChangePoints
        [simulationCounter]
    NormalSSChangePoints= VecNormalSSChangePoints
        [simulationCounter]
    NormalTestChangePoints = VecNormalTestChangePoints
        [simulationCounter]
    TripSSChangePoints = VecTripSSChangePoints
        [simulationCounter]
    TripTestChangePoints = VecTripTestChangePoints
        [simulationCounter]
    SimulatedNormal = VecSimulatedNormal
        [simulationCounter]
    SimulatedNormalSS = VecSimulatedNormalSS
        [simulationCounter]
    SimulatedNormalTest = VecSimulatedNormalTest
        [simulationCounter]
    SimulatedTripSS = VecSimulatedTripSS
        [simulationCounter]
    SimulatedTripTest = VecSimulatedTripTest
        [simulationCounter]
    TrainingData = VecTrainingData
        [simulationCounter]

    #Making the PCA transformation:
    myPC = VARBT(groug_c_name='gr_nr',BoxTiao=False)
    myPC.fit(TrainingData)
    ExpVar = myPC.variances/sum(myPC.variances)

    #Selecting pen function:
    PCApvalMins=[]
    PCAPSbestDims=[]
    for penConst in penVec:
        PCAMWUtests = []
        for linComb in range(0,len(myPC.TrueColumns)):
            alphas = []
            betas = []
            for groupDF in SimulatedTripSS:
                SOF = myPC.transform(groupDF)
```

```
                    [:,linComb]
                SOF = SOF[(((len(SOF)-TmodeStart-
                    TDetectionMargin)%SearchLen):]
                algo = algo.fit(SOF)
                DFresult = algo.predict(pen =
                    penConst*np.log(len(SOF)))
                DFresult = np.asarray(DFresult)
                if(len(np.intersect1d(range(
                    TmodeEnd-TDetectionMargin,
                        TmodeStart+TDetectionMargin)
                            ,(len(SOF)-DFresult)))>0):
                    alphas.append(1)
                else:
                    alphas.append(0)
                DFresult=DFresult[(DFresult <=
                    (len(SOF)-TmodeStart-
                        TDetectionMargin))]
                if(len(DFresult)>0):
                    betas.append(1)
                else:
                    betas.append(0)

            for groupDF in SimulatedNormalSS:
                SOF = myPC.transform(groupDF)
                    [:,linComb]
                SOF = SOF[(len(SOF) % SearchLen):]
                algo = algo.fit(SOF)
                DFresult = algo.predict(pen =
                    penConst*np.log(len(SOF)))
                DFresult = np.asarray(DFresult)
                if(len(DFresult)>1):
                    betas.append(1)
                else:
                    betas.append(0)

            try:
                PCAMWUtests.append(mannwhitneyu(
                    x=betas,y=alphas,
                        alternative='less'))
            except:
                PCAMWUtests.append(mannwhitneyu(
                    x=[1,-1]*1000,y=[-1,1]*1000,
                        alternative='less'))
        PCApvalMins.append(np.min(
            [np.log(x.pvalue)
```

```
                for x in PCAMWUtests]))
        PCAPSbestDims.append(np.where(
            PCAMWUtests==np.min(
                PCAMWUtests))[0][0])
    PCALinCombBestPens.append(penVec
        [np.where(PCApvalMins ==
            min(PCApvalMins))[0][0]])
    PCALinCombBestLogPvalues.
        append(min(PCApvalMins))
    PCALinCombBestDim.append(
        PCAPSbestDims[np.where
            (PCApvalMins ==
                min(PCApvalMins))[0][0]])
```

**Experiment 2**

The following is the implementation of Experiment 2 for the simulated data. For each simulation, the natural logarithm of the Mann-Whitney U test's $p$-value yielded by the selected signal of interest is saved in `PCADLIBestLogPvalues`, and the selected signal of interest is saved in `PCADLIBestDim`.

```
VarLimit = 0.9
DLISignals = ['myPC.Leverage(groupDF, s_l=S)',
    'myPC.Discrepancy(groupDF, s_di=S)',
        'myPC.Influence(groupDF, s_i=S)']
algo=rpt.Pelt(model="normal",jump=int(TmodeLength/5))
penVec=[5]
PCADLIBestLogPvalues = []
PCADLIBestDim=[]
PCADLIBestPens=[]
for simulationCounter in tqdm(r_s_vec):
    NormalChangePoints = VecNormalChangePoints
        [simulationCounter]
    NormalSSChangePoints= VecNormalSSChangePoints
        [simulationCounter]
    NormalTestChangePoints = VecNormalTestChangePoints
        [simulationCounter]
    TripSSChangePoints = VecTripSSChangePoints
        [simulationCounter]
    TripTestChangePoints = VecTripTestChangePoints
        [simulationCounter]
    SimulatedNormal = VecSimulatedNormal
        [simulationCounter]
    SimulatedNormalSS = VecSimulatedNormalSS
        [simulationCounter]
    SimulatedNormalTest = VecSimulatedNormalTest
```

```
        [simulationCounter]
    SimulatedTripSS = VecSimulatedTripSS
        [simulationCounter]
    SimulatedTripTest = VecSimulatedTripTest
        [simulationCounter]
    TrainingData = VecTrainingData
        [simulationCounter]


    #Making the PCA transformation:
    myPC = VARBT(groug_c_name='gr_nr',BoxTiao=False)
    myPC.fit(TrainingData)
    ExpVar = myPC.variances/sum(myPC.variances)
    S = (np.where(np.asanyarray([sum(ExpVar[:x+1])
        for x in range(0,len(ExpVar))]) >
            VarLimit)[0][0])+1
    #Selecting pen function:
    PCApvalMins=[]
    PCAPSbestDims=[]
    for penConst in penVec:
        PCAMWUtests = []
        for DLI in range(0,len(DLISignals)):
            alphas = []
            betas = []
            for groupDF in SimulatedTripSS:
                SOF = eval(DLISignals[DLI])
                SOF = SOF[((len(SOF)-TmodeStart-
                    TDetectionMargin)%SearchLen):]
                algo = algo.fit(SOF)
                DFresult = algo.predict(pen = penConst
                    *np.log(len(SOF)))
                DFresult = np.asarray(DFresult)
                if(len(np.intersect1d(range(TmodeEnd-
                    TDetectionMargin,TmodeStart+
                        TDetectionMargin),
                            (len(SOF)-DFresult)))>0):
                    alphas.append(1)
                else:
                    alphas.append(0)
                DFresult=DFresult[(DFresult <=
                    (len(SOF)-TmodeStart-
                        TDetectionMargin))]
                if(len(DFresult)>0):
                    betas.append(1)
                else:
                    betas.append(0)

            for groupDF in SimulatedNormalSS:
                SOF = eval(DLISignals[DLI])
                SOF = SOF[(len(SOF) % SearchLen):]
                algo = algo.fit(SOF)
                DFresult = algo.predict(
                    pen = penConst*np.log(len(SOF)))
                DFresult = np.asarray(DFresult)
                if(len(DFresult)>1):
                    betas.append(1)
                else:
                    betas.append(0)
            try:
                PCAMWUtests.append(mannwhitneyu(
                    x=betas,y=alphas,
                        alternative='less'))
            except:
                PCAMWUtests.append(mannwhitneyu(
                    x=[1,-1]*1000,y=[-1,1]*1000,
                        alternative='less'))
        PCApvalMins.append(np.min([np.log(x.pvalue)
            for x in PCAMWUtests]))
        PCAPSbestDims.append(np.where(PCAMWUtests==
            np.min(PCAMWUtests))[0][0])

    PCADLIBestPens.append(penVec[np.where(PCApvalMins ==
        min(PCApvalMins))[0][0]])
    PCADLIBestLogPvalues.append(min(PCApvalMins))
    PCADLIBestDim.append(PCAPSbestDims[np.where(
        PCApvalMins == min(PCApvalMins))[0][0]])
```

**Experiment 3**

The following is the implementation of Experiment 3 for the simulated data. For each simulation, the natural logarithm of the Mann-Whitney U test's $p$-value yielded by the selected signal of interest is saved in `BTALinCombBestLogPvalues`, and the selected signal of interest is saved in `BTALinCombBestDim`.

```
algo=rpt.Pelt(model="ar",jump=int(TmodeLength/5),params=
    {"order": 1})
penVec=[2]
BTALinCombBestLogPvalues = []
BTALinCombBestDim=[]
BTALinCombBestPens = []
for simulationCounter in tqdm(r_s_vec):
    NormalChangePoints = VecNormalChangePoints
        [simulationCounter]
    NormalSSChangePoints= VecNormalSSChangePoints
        [simulationCounter]
    NormalTestChangePoints = VecNormalTestChangePoints
        [simulationCounter]
    TripSSChangePoints = VecTripSSChangePoints
        [simulationCounter]
    TripTestChangePoints = VecTripTestChangePoints
        [simulationCounter]
    SimulatedNormal = VecSimulatedNormal
        [simulationCounter]
    SimulatedNormalSS = VecSimulatedNormalSS
        [simulationCounter]
    SimulatedNormalTest = VecSimulatedNormalTest
        [simulationCounter]
    SimulatedTripSS = VecSimulatedTripSS
        [simulationCounter]
    SimulatedTripTest = VecSimulatedTripTest
        [simulationCounter]
    TrainingData = VecTrainingData
        [simulationCounter]

    #Making the PCA transformation:
    myBT = VARBT(groug_c_name='gr_nr')
    myBT.fit(TrainingData)
    ExpVar = myBT.variances/sum(myBT.variances)

    #Selecting pen function:
    BTApvalMins=[]
    BTAPSbestDims=[]
    for penConst in penVec:
        BTAMWUtests = []
        for linComb in range(0,len(myBT.TrueColumns)):
            alphas = []
            betas = []
            for groupDF in SimulatedTripSS:
                SOF = myBT.transform(groupDF)[:,linComb]
                SOF = SOF[((len(SOF)-TmodeStart-
                    TDetectionMargin)%SearchLen):]
                algo = algo.fit(SOF)
                DFresult = algo.predict(pen =
                    penConst*np.log(len(SOF)))
                DFresult = np.asarray(DFresult)
                if(len(np.intersect1d(range(TmodeEnd-
                    TDetectionMargin , TmodeStart+
                        TDetectionMargin ) ,( len (SOF)
                            −DFresult ) ) ) >0):
                    alphas . append (1)
                else :
                    alphas . append (0)
                DFresult=DFresult [ ( DFresult <= ( len (SOF)
                    −TmodeStart−TDetectionMargin ) ) ]
                if ( len ( DFresult ) >0):
                    betas . append (1)
                else :
                    betas . append (0)

            for groupDF in SimulatedNormalSS :
                SOF = myBT. transform ( groupDF ) [ : , linComb ]
                SOF = SOF[ ( len (SOF) % SearchLen ) : ]
                algo = algo . fit (SOF)
                DFresult = algo . predict ( pen =
                    penConst∗np . log ( len (SOF) ) )
                DFresult = np . asarray ( DFresult )
                if ( len ( DFresult ) >1):
                    betas . append (1)
                else :
                    betas . append (0)

            try :
                BTAMWUtests . append ( mannwhitneyu (
                    x=betas , y=alphas , alternative='less ' ) )
            except :
                BTAMWUtests . append ( mannwhitneyu (
                    x=[1 , −1]∗1000 ,y=[−1 ,1]∗1000 ,
                        alternative='less ' ) )
        BTApvalMins . append (np . min ( [ np . log ( x . pvalue )
            for x in BTAMWUtests ] ) )
        BTAPSbestDims . append (np . where (BTAMWUtests==
            np . min (BTAMWUtests ) ) [ 0 ] [ 0 ] )
    BTALinCombBestPens . append ( penVec [ np . where ( BTApvalMins
        == min ( BTApvalMins ) ) [ 0 ] [ 0 ] ] )
    BTALinCombBestLogPvalues . append ( min ( BTApvalMins ) )
    BTALinCombBestDim . append ( BTAPSbestDims [ np . where (
        BTApvalMins == min ( BTApvalMins ) ) [ 0 ] [ 0 ] ] )
```

**Experiment 4**

The following is the implementation of Experiment 4 for the simulated data. For each simulation, the natural logarithm of the Mann-Whitney U test's $p$-value yielded by the selected signal of interest is saved in `BTADLIBestLogPvalues`, and the selected signal of interest is saved in `BTADLIBestDim`.

```
PredLimit = 0.9
DLISignals = ['myBT.Leverage(groupDF, s_l=S)',
    'myBT.Discrepancy(groupDF, s_di=S)',
        'myBT.Influence(groupDF, s_i=S)']
algo=rpt.Pelt(model="normal",jump=int(TmodeLength/5))
penVec=[5]
BTADLIBestLogPvalues = []
BTADLIBestDim=[]
BTADLIBestPens=[]
for simulationCounter in tqdm(r_s_vec):
    NormalChangePoints = VecNormalChangePoints
        [simulationCounter]
    NormalSSChangePoints= VecNormalSSChangePoints
        [simulationCounter]
    NormalTestChangePoints = VecNormalTestChangePoints
        [simulationCounter]
    TripSSChangePoints = VecTripSSChangePoints
        [simulationCounter]
    TripTestChangePoints = VecTripTestChangePoints
        [simulationCounter]
    SimulatedNormal = VecSimulatedNormal
        [simulationCounter]
    SimulatedNormalSS = VecSimulatedNormalSS
        [simulationCounter]
    SimulatedNormalTest = VecSimulatedNormalTest
        [simulationCounter]
    SimulatedTripSS = VecSimulatedTripSS
        [simulationCounter]
    SimulatedTripTest = VecSimulatedTripTest
        [simulationCounter]
    TrainingData = VecTrainingData
        [simulationCounter]

    #Making the PCA transformation:
    myBT = VARBT(groug_c_name='gr_nr')
    myBT.fit(TrainingData)
    ExpPred = myBT.predictabilities/
        sum(myBT.predictabilities)
    S = (np.where(np.asanyarray([sum(ExpPred[:x+1])
        for x in range(0,len(ExpPred))]) > PredLimit)
```

```
                [0][0])+1
    #Selecting pen function:
    BTApvalMins=[]
    BTAPSbestDims=[]
    for penConst in penVec:
        BTAMWUtests = []
        for DLI in range(0,len(DLISignals)):
            alphas = []
            betas = []
            for groupDF in SimulatedTripSS:
                SOF = eval(DLISignals[DLI])
                SOF = SOF[((len(SOF)-TmodeStart-
                    TDetectionMargin)%SearchLen):]
                algo = algo.fit(SOF)
                DFresult = algo.predict(pen =
                    penConst*np.log(len(SOF)))
                DFresult = np.asarray(DFresult)
                if(len(np.intersect1d(range(
                    TmodeEnd-TDetectionMargin,
                        TmodeStart+TDetectionMargin),
                            (len(SOF)-DFresult)))>0):
                    alphas.append(1)
                else:
                    alphas.append(0)
                DFresult=DFresult[(DFresult <=
                    (len(SOF)-TmodeStart-
                        TDetectionMargin))]
                if(len(DFresult)>0):
                    betas.append(1)
                else:
                    betas.append(0)

            for groupDF in SimulatedNormalSS:
                SOF = eval(DLISignals[DLI])
                SOF = SOF[(len(SOF) % SearchLen):]
                algo = algo.fit(SOF)
                DFresult = algo.predict(pen =
                    penConst*np.log(len(SOF)))
                DFresult = np.asarray(DFresult)
                if(len(DFresult)>1):
                    betas.append(1)
                else:
                    betas.append(0)
            try:
                BTAMWUtests.append(mannwhitneyu(
                    x=betas,y=alphas,alternative='less'))
            except:
                BTAMWUtests.append(mannwhitneyu(
                    x=[1,-1]*1000,y=[-1,1]*1000,
                        alternative='less'))
        BTApvalMins.append(np.min([np.log(x.pvalue)
            for x in BTAMWUtests]))
        BTAPSbestDims.append(np.where(BTAMWUtests==
            np.min(BTAMWUtests))[0][0])
    BTADLIBestLogPvalues.append(min(BTApvalMins))
    BTADLIBestDim.append(BTAPSbestDims[np.where(
        BTApvalMins == min(BTApvalMins))[0][0]])
    BTADLIBestPens.append(penVec[np.where(
        PCApvalMins == min(PCApvalMins))[0][0]])
```

### C.2.3 Predictive Model

Here, we present the Python script used for the training and the evaluation of the predictive model in the simulation study.

#### Experiment 1

In the following code, we use the signals of interest selected in Experiment 1 to construct and evaluate a predictive model using the CUSUM method. The confusion matrices of the models are saved in the list `PCALinCombConfMats`.

```
PCALinCombConfMats=[]
PCALinCombBestCusumH = []
PCALinCombBestCusumK = []
PCALinComPCArgets=[]
PCALinCombSTDs=[]
PCALinCombTrainingMetric=[]

for simulationCounter in tqdm(r_s_vec):
    SimulatedNormalSS = VecSimulatedNormalSS
        [simulationCounter]
    SimulatedNormalTest = VecSimulatedNormalTest
        [simulationCounter]
    SimulatedTripSS = VecSimulatedTripSS
        [simulationCounter]
    SimulatedTripTest = VecSimulatedTripTest
        [simulationCounter]
    TrainingData = VecTrainingData
        [simulationCounter]

    #Making the PCA transformation:
```

```
    myPC = VARBT(groug_c_name='gr_nr ',BoxTiao=False)
    myPC.fit(TrainingData)
    PCABestLinComb = PCALinCombBestDim
        [simulationCounter]

    #Transforming Data
    PCALinCombNormalssSOFs = [myPC.transform(x)
        [:,PCABestLinComb] for x in SimulatedNormalSS]
    PCALinCombTripssSOFs = [myPC.transform(x)
        [:,PCABestLinComb] for x in SimulatedTripSS]
    PCALinCombNormalTestSOFs = [myPC.transform(x)
        [:,PCABestLinComb] for x in SimulatedNormalTest]
    PCALinCombTripTestSOFs = [myPC.transform(x)
        [:,PCABestLinComb] for x in SimulatedTripTest]

    #Making CUSUM
    PCALinCombNormalTarget, PCALinCombNormalSTD,
        PCALinCombBest_k ,PCALinCombBest_h,
            PCALinCombBestTrainingMetric = Make_CUSUM(
                PCALinCombNormalssSOFs,
                    PCALinCombTripssSOFs, TmodeLength,
                        TmodeEnd, TDetectionMargin,
                            include_low = True)
    PCALinCombConfMats.append(Evaluate_CUSUM(
        PCALinCombNormalTestSOFs, PCALinCombTripTestSOFs,
            PCALinCombNormalTarget,PCALinCombBest_k,
                PCALinCombBest_h,TmodeLength,TmodeEnd,
                    TDetectionMargin,include_low = True))
    PCALinCombBestCusumH.append(PCALinCombBest_h)
    PCALinCombBestCusumK.append(PCALinCombBest_k)
    PCALinComPCArgets.append(PCALinCombNormalTarget)
    PCALinCombSTDs.append(PCALinCombNormalSTD)
    PCALinCombTrainingMetric.
        append(PCALinCombBestTrainingMetric)
```

### Experiment 2

In the following code, we use the signals of interest selected in Experiment 2 to construct and evaluate a predictive model using the CUSUM method. The confusion matrices of the models are saved in the list `PCADLIConfMats`.

```
DLISignals = ['myPC.Leverage(x,s_l=S)',
    'myPC.Discrepancy(x,s_di=S)',
        'myPC.Influence(x,s_i=S)']
VarLimit = 0.90
PCADLIConfMats=[]
```

```
PCADLIBestCusumH = []
PCADLIBestCusumK = []
PCADLITargets=[]
PCADLISTDs=[]
PCADLITrainingMetric=[]

for simulationCounter in tqdm(r_s_vec):
    SimulatedNormalSS = VecSimulatedNormalSS
        [simulationCounter]
    SimulatedNormalTest = VecSimulatedNormalTest
        [simulationCounter]
    SimulatedTripSS = VecSimulatedTripSS
        [simulationCounter]
    SimulatedTripTest = VecSimulatedTripTest
        [simulationCounter]
    TrainingData = VecTrainingData
        [simulationCounter]

    #Making the PCA transformation:
    myPC = VARBT(groug_c_name='gr_nr', BoxTiao=False)
    myPC.fit(TrainingData)
    ExpVar = myPC.variances/sum(myPC.variances)
    S = (np.where(np.asanyarray([sum(ExpVar[:x+1])
        for x in range(0,len(ExpVar))]) > VarLimit)
            [0][0])+1
    PCABestDLI = PCADLIBestDim[simulationCounter]

    #Transforming Data
    PCADLINormalssSOFs = [eval(DLISignals[PCABestDLI])
        for x in SimulatedNormalSS]
    PCADLITripssSOFs = [eval(DLISignals[PCABestDLI])
        for x in SimulatedTripSS]
    PCADLINormalTestSOFs = [eval(DLISignals[PCABestDLI])
        for x in SimulatedNormalTest]
    PCADLITripTestSOFs = [eval(DLISignals[PCABestDLI])
        for x in SimulatedTripTest]

    #Making CUSUM
    PCADLINormalTarget, PCADLINormalSTD,PCADLIBest_k ,
        PCADLIBest_h,PCADLIBestTrainingMetric =
            Make_CUSUM(PCADLINormalssSOFs, PCADLITripssSOFs,
                TmodeLength, TmodeEnd, TDetectionMargin,
                    include_low = False)
    PCADLIConfMats.append(Evaluate_CUSUM(
        PCADLINormalTestSOFs, PCADLITripTestSOFs,
            PCADLINormalTarget,PCADLIBest_k,PCADLIBest_h,
                TmodeLength,TmodeEnd, TDetectionMargin,
                    include_low = False))
    PCADLIBestCusumH.append(PCADLIBest_h)
    PCADLIBestCusumK.append(PCADLIBest_k)
    PCADLITargets.append(PCADLINormalTarget)
    PCADLISTDs.append(PCADLINormalSTD)
    PCADLITrainingMetric.
        append(PCADLIBestTrainingMetric)
```

**Experiment 3**

In the following code, we use the signals of interest selected in Experiment 3 to construct and evaluate a predictive model using the CUSUM method. The confusion matrices of the models are saved in the list `BTALinCombConfMats`.

```
BTALinCombConfMats=[]
BTALinCombBestCusumH = []
BTALinCombBestCusumK = []
BTALinCombTargets=[]
BTALinCombSTDs=[]
BTALinCombTrainingMetric=[]

for simulationCounter in tqdm(r_s_vec):
    SimulatedNormalSS = VecSimulatedNormalSS
        [simulationCounter]
    SimulatedNormalTest = VecSimulatedNormalTest
        [simulationCounter]
    SimulatedTripSS = VecSimulatedTripSS
        [simulationCounter]
    SimulatedTripTest = VecSimulatedTripTest
        [simulationCounter]
    TrainingData = VecTrainingData
        [simulationCounter]

    #Making the PCA transformation:
    myBT = VARBT(groug_c_name='gr_nr')
    myBT.fit(TrainingData)
    BTABestLinComb = BTALinCombBestDim[simulationCounter]

    #Transforming Data
    BTALinCombNormalssSOFs =
        [myBT.transform(x)[:,BTABestLinComb]
            for x in SimulatedNormalSS]
    BTALinCombTripssSOFs =
        [myBT.transform(x)[:,BTABestLinComb]
```

```
            for x in SimulatedTripSS]
    BTALinCombNormalTestSOFs =
        [myBT.transform(x)[:,BTABestLinComb]
            for x in SimulatedNormalTest]
    BTALinCombTripTestSOFs =
        [myBT.transform(x)[:,BTABestLinComb]
            for x in SimulatedTripTest]

    #Making CUSUM
    BTALinCombNormalTarget, BTALinCombNormalSTD,
        BTALinCombBest_k ,BTALinCombBest_h,
            BTALinCombBestTrainingMetric = Make_CUSUM(
                BTALinCombNormalssSOFs,
                    BTALinCombTripssSOFs, TmodeLength,
                        TmodeEnd, TDetectionMargin,
                            include_low = True)
    BTALinCombConfMats.append(Evaluate_CUSUM(
        BTALinCombNormalTestSOFs, BTALinCombTripTestSOFs,
            BTALinCombNormalTarget,BTALinCombBest_k,
                BTALinCombBest_h,TmodeLength,TmodeEnd,
                    TDetectionMargin,include_low = True))
    BTALinCombBestCusumH.append(BTALinCombBest_h)
    BTALinCombBestCusumK.append(BTALinCombBest_k)
    BTALinCombTargets.append(BTALinCombNormalTarget)
    BTALinCombSTDs.append(BTALinCombNormalSTD)
    BTALinCombTrainingMetric.
        append(BTALinCombBestTrainingMetric)
```

### Experiment 4

In the following code, we use the signals of interest selected in Experiment 1 to construct and evaluate a predictive model using the CUSUM method. The confusion matrices of the models are saved in the list `BTADLIConfMats`.

```
DLISignals = ['myBT.Leverage(x,s_l=S)',
    'myBT.Discrepancy(x,s_di=S)',
        'myBT.Influence(x,s_i=S)']
PredLimit = 0.90
BTADLIConfMats=[]
BTADLIBestCusumH = []
BTADLIBestCusumK = []
BTADLITargets=[]
BTADLISTDs=[]
BTADLITrainingMetric=[]

for simulationCounter in tqdm(r_s_vec):
```

```
SimulatedNormalSS = VecSimulatedNormalSS
    [simulationCounter]
SimulatedNormalTest = VecSimulatedNormalTest
    [simulationCounter]
SimulatedTripSS = VecSimulatedTripSS
    [simulationCounter]
SimulatedTripTest = VecSimulatedTripTest
    [simulationCounter]
TrainingData = VecTrainingData
    [simulationCounter]

#Making the BTA transformation:
myBT = VARBT(groug_c_name='gr_nr')
myBT.fit(TrainingData)
ExpPred = myBT.predictabilities/
    sum(myBT.predictabilities)
S = (np.where(np.asanyarray([sum(ExpPred[:x+1])
    for x in range(0,len(ExpPred))]) > PredLimit)
        [0][0])+1
BTABestDLI = BTADLIBestDim[simulationCounter]

#Transforming Data
BTADLINormalssSOFs = [eval(DLISignals[BTABestDLI])
    for x in SimulatedNormalSS]
BTADLITripssSOFs = [eval(DLISignals[BTABestDLI])
    for x in SimulatedTripSS]
BTADLINormalTestSOFs = [eval(DLISignals[BTABestDLI])
    for x in SimulatedNormalTest]
BTADLITripTestSOFs = [eval(DLISignals[BTABestDLI])
    for x in SimulatedTripTest]

#Making CUSUM
BTADLINormalTarget, BTADLINormalSTD,BTADLIBest_k
    ,BTADLIBest_h,BTADLIBestTrainingMetric =
        Make_CUSUM(BTADLINormalssSOFs, BTADLITripssSOFs,
            TmodeLength, TmodeEnd, TDetectionMargin,
                include_low = False)
BTADLIConfMats.append(Evaluate_CUSUM(
    BTADLINormalTestSOFs, BTADLITripTestSOFs,
        BTADLINormalTarget,BTADLIBest_k,BTADLIBest_h,
            TmodeLength,TmodeEnd, TDetectionMargin,
                include_low = False))
BTADLIBestCusumH.append(BTADLIBest_h)
BTADLIBestCusumK.append(BTADLIBest_k)
BTADLITargets.append(BTADLINormalTarget)
```

```
    BTADLISTDs.append(BTADLINormalSTD)
    BTADLITrainingMetric.append(BTADLIBestTrainingMetric)
```

### C.2.4 Results and Figures

The code used to calculate the mean and the standard deviation of the log($p$-values), presented in Subsection 5.1.1:

```
print(np.mean(PCALinCombBestLogPvalues))
print(np.mean(BTALinCombBestLogPvalues))
print(np.std(PCALinCombBestLogPvalues))
print(np.std(BTALinCombBestLogPvalues))
```

The code used to calculate the mean and the standard deviation of the log($p$-values), presented in Subsection 5.1.2:

```
print(np.mean(PCADLIBestLogPvalues))
print(np.mean(BTADLIBestLogPvalues))
print(np.std(PCADLIBestLogPvalues))
print(np.std(BTADLIBestLogPvalues))
```

The code used to create Figure 5.1:

```
PCALinCombBestDimP1=(np.asarray(PCALinCombBestDim)+1)
BTALinCombBestDimP1=(np.asarray(BTALinCombBestDim)+1)
colors = ['red', 'blue']
n, bins, patches = plt.hist([PCALinCombBestDimP1,BTALinCombBestDimP1],
    bins=np.arange(1,12)-0.5 , histtype='bar',
        color=colors, label=['PCA', 'BTA'])
plt.legend(prop={'size': 10})
plt.xticks(range(1,11))
plt.xlabel('PCA and BTA Components')
plt.ylabel('Number of Simulations')
plt.savefig('SimLinCombhist.pdf')
```

The code used to create Figure 5.8:

```
colors = ['red', 'blue']
n, bins, patches = plt.hist([PCADLIBestDim,BTADLIBestDim],
    bins=np.arange(4)-0.5 , histtype='bar', color=colors,
        label=['PCA', 'BTA'])
plt.legend(prop={'size': 10})
plt.xticks(range(0,3),['Leverage', 'Discrepancy','Influence'])
plt.xlabel('Signal of Interest')
plt.ylabel('Number of Simulations')
plt.savefig('DLISignalhist.pdf')
```

The code used to compare the Mann-Whitney U tests' $p$-values yielded by the different approaches:

```
print('PCALinComb < BTALinComb')
print(mannwhitneyu(x=PCALinCombBestLogPvalues,
    y=BTALinCombBestLogPvalues, alternative='less'))
print('PCALinComb < BTADLI')
print(mannwhitneyu(x=PCALinCombBestLogPvalues,
    y=BTADLIBestLogPvalues, alternative='less'))
print('PCALinComb < PCADLI')
print(mannwhitneyu(x=PCALinCombBestLogPvalues,
    y=PCADLIBestLogPvalues, alternative='less'))
print('BTALinComb < BTADLI')
print(mannwhitneyu(x=BTALinCombBestLogPvalues,
    y=BTADLIBestLogPvalues, alternative='less'))
print('BTALinComb < PCADLI')
print(mannwhitneyu(x=BTALinCombBestLogPvalues,
    y=PCADLIBestLogPvalues, alternative='less'))
print('PCADLI < BTADLI')
print(mannwhitneyu(x=PCADLIBestLogPvalues,
    y=BTADLIBestLogPvalues, alternative='less'))
```

In the following code we calculate the four statistical metrics accuracy, recall, precision and F1 score for each simulation experiment:

```
BTALinCombRecalls=np.asarray([(x.iloc[1,1])/
    (x.iloc[1,1]+x.iloc[1,0]) for x in BTALinCombConfMats])
PCALinCombRecalls=np.asarray([(x.iloc[1,1])/
    (x.iloc[1,1]+x.iloc[1,0]) for x in PCALinCombConfMats])
BTALinCombPrecisions=np.asarray([(x.iloc[1,1])/
    (x.iloc[1,1]+x.iloc[0,1]) for x in BTALinCombConfMats])
PCALinCombPrecisions = np.asarray([(x.iloc[1,1])/
    (x.iloc[1,1]+x.iloc[0,1]) for x in PCALinCombConfMats])
BTALinCombF1s = 2*(BTALinCombRecalls*BTALinCombPrecisions)/
    (BTALinCombRecalls+BTALinCombPrecisions)
PCALinCombF1s = 2*(PCALinCombRecalls*PCALinCombPrecisions)/
    (PCALinCombRecalls+PCALinCombPrecisions)
BTALinCombAccuracys=np.asarray([(x.iloc[0,0]+x.iloc[1,1])/
    (x.sum().sum()) for x in BTALinCombConfMats])
PCALinCombAccuracys = np.asarray([(x.iloc[0,0]+x.iloc[1,1])/
    (x.sum().sum()) for x in PCALinCombConfMats])
BTALinCombSpecificitys=np.asarray([(x.iloc[0,0])/
    (x.iloc[0,0]+x.iloc[0,1]) for x in BTALinCombConfMats])
PCALinCombSpecificitys = np.asarray([(x.iloc[0,0])/
    (x.iloc[0,0]+x.iloc[0,1]) for x in PCALinCombConfMats])
BTADLIRecalls=np.asarray([(x.iloc[1,1])/
    (x.iloc[1,1]+x.iloc[1,0]) for x in BTADLIConfMats])
PCADLIRecalls=np.asarray([(x.iloc[1,1])/
    (x.iloc[1,1]+x.iloc[1,0]) for x in PCADLIConfMats])
BTADLIPrecisions=np.asarray([(x.iloc[1,1])/
    (x.iloc[1,1]+x.iloc[0,1]) for x in BTADLIConfMats])
PCADLIPrecisions = np.asarray([(x.iloc[1,1])/
    (x.iloc[1,1]+x.iloc[0,1]) for x in PCADLIConfMats])
BTADLIF1s = 2*(BTADLIRecalls*BTADLIPrecisions)/
    (BTADLIRecalls+BTADLIPrecisions)
PCADLIF1s = 2*(PCADLIRecalls*PCADLIPrecisions)/
    (PCADLIRecalls+PCADLIPrecisions)
BTADLIAccuracys=np.asarray([(x.iloc[0,0]+x.iloc[1,1])/
    (x.sum().sum()) for x in BTADLIConfMats])
PCADLIAccuracys = np.asarray([(x.iloc[0,0]+x.iloc[1,1])/
    (x.sum().sum()) for x in PCADLIConfMats])
BTADLISpecificitys=np.asarray([(x.iloc[0,0])/
    (x.iloc[0,0]+x.iloc[0,1]) for x in BTADLIConfMats])
PCADLISpecificitys = np.asarray([(x.iloc[0,0])/
    (x.iloc[0,0]+x.iloc[0,1]) for x in PCADLIConfMats])
BTALinCombF1s=np.nan_to_num(BTALinCombF1s)
PCALinCombF1s=np.nan_to_num(PCALinCombF1s)
BTADLIF1s=np.nan_to_num(BTADLIF1s)
PCADLIF1s=np.nan_to_num(PCADLIF1s)
```

Now we set the font size in our figures: `bigFsize=16`.

The code for producing Figures 5.2, 5.9, 5.3, 5.10, 5.4, 5.11,5.7 and 5.12:

```
plt.figure(figsize=(12, 7))
colors = ['red', 'blue']
plt.hist([PCALinCombRecalls,BTALinCombRecalls],
    bins=np.arange(0,1.05,0.05),color=colors,
        label=['PCA', 'BTA'])
plt.xticks(np.arange(0,1.05,0.05))
plt.legend(prop={'size': bigFsize})
plt.xlabel('Recall', fontsize=bigFsize)
plt.ylabel('Number of Simulations', fontsize=bigFsize)
plt.savefig('SimLinCombRecallhist.pdf')
plt.show()

plt.figure(figsize=(12, 7))
colors = ['red', 'blue']
plt.hist([PCADLIRecalls,BTADLIRecalls],
    bins=np.arange(0,1.05,0.05),color=colors,
        label=['PCA', 'BTA'])
plt.xticks(np.arange(0,1.05,0.05))
plt.legend(prop={'size': bigFsize})
plt.xlabel('Recall', fontsize=bigFsize)
plt.ylabel('Number of Simulations', fontsize=bigFsize)
plt.savefig('SimDLIRecallhist.pdf')
plt.show()
plt.figure(figsize=(12, 7))
colors = ['red', 'blue']
plt.hist([PCALinCombPrecisions,BTALinCombPrecisions],
    bins=np.arange(0,1.05,0.05),color=colors,
        label=['PCA', 'BTA'])
plt.xticks(np.arange(0,1.05,0.05))
plt.legend(prop={'size': bigFsize})
plt.xlabel('Precision', fontsize=bigFsize)
plt.ylabel('Number of Simulations', fontsize=bigFsize)
plt.savefig('SimLinCombPrecisionhist.pdf')
plt.show()

plt.figure(figsize=(12, 7))
colors = ['red', 'blue']
plt.hist([PCADLIPrecisions,BTADLIPrecisions],
    bins=np.arange(0,1.05,0.05),color=colors,
        label=['PCA', 'BTA'])
plt.xticks(np.arange(0,1.05,0.05))
plt.legend(prop={'size': bigFsize})
plt.xlabel('Precision', fontsize=bigFsize)
plt.ylabel('Number of Simulations', fontsize=bigFsize)
plt.savefig('SimDLIPrecisionhist.pdf')
plt.show()

plt.figure(figsize=(12, 7))
colors = ['red', 'blue']
plt.hist([PCALinCombAccuracys,BTALinCombAccuracys],
    bins=np.arange(0,1.05,0.05),color=colors,
        label=['PCA', 'BTA'])
plt.xticks(np.arange(0,1.05,0.05))
plt.legend(prop={'size':bigFsize})
plt.xlabel('Accuracy', fontsize=bigFsize)
plt.ylabel('Number of Simulations', fontsize=bigFsize)
plt.savefig('SimLinCombAccuracyhist.pdf')
plt.show()

plt.figure(figsize=(12, 7))
colors = ['red', 'blue']
plt.hist([PCADLIAccuracys,BTADLIAccuracys],
    bins=np.arange(0,1.05,0.05),color=colors,
        label=['PCA', 'BTA'])
plt.xticks(np.arange(0,1.05,0.05))
plt.legend(prop={'size':bigFsize})
plt.xlabel('Accuracy', fontsize=bigFsize)
plt.ylabel('Number of Simulations', fontsize=bigFsize)
plt.savefig('SimDLIAccuracyhist.pdf')
plt.show()

plt.figure(figsize=(12, 7))
colors = ['red', 'blue']
plt.hist([PCALinCombF1s,BTALinCombF1s],
    bins=np.arange(0,1.05,0.05),color=colors,
        label=['PCA', 'BTA'])
plt.xticks(np.arange(0,1.05,0.05))
plt.legend(prop={'size': bigFsize})
plt.xlabel('F1 Score', fontsize=bigFsize)
plt.ylabel('Number of Simulations', fontsize=bigFsize)
plt.savefig('SimLinCombF1hist.pdf')
plt.show()

plt.figure(figsize=(12, 7))
colors = ['red', 'blue']
plt.hist([PCADLIF1s,BTADLIF1s], bins=np.arange(0,1.05,0.05),
    color=colors, label=['PCA', 'BTA'])
plt.xticks(np.arange(0,1.05,0.05))
plt.legend(prop={'size': bigFsize})
plt.xlabel('F1 Score', fontsize=bigFsize)
plt.ylabel('Number of Simulations', fontsize=bigFsize)
plt.savefig('SimDLIF1hist.pdf')
plt.show()
```

The code for comparing the performance metrics of the four different approaches:

```
print(mannwhitneyu(x=PCALinCombRecalls,
    y=BTALinCombRecalls, alternative='less'))
print(mannwhitneyu(x=BTADLIRecalls,
    y=PCADLIRecalls, alternative='less'))
print(mannwhitneyu(x=BTADLIPrecisions,
    y=PCADLIPrecisions, alternative='less'))
print(mannwhitneyu(x=PCALinCombPrecisions,
    y=BTALinCombPrecisions, alternative='less'))
print(mannwhitneyu(x=PCALinCombAccuracys,
    y=BTALinCombAccuracys, alternative='less'))
print(mannwhitneyu(x=BTADLIAccuracys,
    y=PCADLIAccuracys, alternative='less'))
print(mannwhitneyu(x=PCALinCombF1s,
    y=BTALinCombF1s, alternative='less'))
print(mannwhitneyu(x=BTADLIF1s,
    y=PCADLIF1s, alternative='less'))
```

The code for producing Figure 5.6:

```
plt.figure(figsize=(12, 5))
plt.subplot(1,2,1)
plt.plot((np.asarray(PCALinCombBestDim)+1),
    PCALinCombPrecisions,'+',color='red',
        alpha = 0.6,label='PCA')
m,b=np.polyfit((np.asarray(PCALinCombBestDim)+1),
    PCALinCombPrecisions,1)
plt.plot((np.asarray(PCALinCombBestDim)+1),
    m*(np.asarray(PCALinCombBestDim)+1)+b,'--r')
plt.plot((np.asarray(BTALinCombBestDim)+1),
    BTALinCombPrecisions,'x',color='blue',
        alpha = 0.5,label='BTA')
m,b=np.polyfit((np.asarray(BTALinCombBestDim)+1),
    BTALinCombPrecisions,1)
plt.plot((np.asarray(BTALinCombBestDim)+1),
    m*(np.asarray(BTALinCombBestDim)+1)+b,'--b')
plt.legend(prop={'size': 12})
plt.xticks(range(1,11))
plt.xlabel('Selected Signal of Interest',
    fontsize=bigFsize)
plt.ylabel('Precision', fontsize=bigFsize)
plt.subplot(1,2,2)
plt.plot((np.asarray(PCALinCombBestDim)+1),
    PCALinCombAccuracys,'+',color='red',
        alpha = 0.6,label='PCA')
m,b=np.polyfit((np.asarray(PCALinCombBestDim)+1),
    PCALinCombAccuracys,1)
plt.plot((np.asarray(PCALinCombBestDim)+1),
    m*(np.asarray(PCALinCombBestDim)+1)+b,'--r')
plt.plot((np.asarray(BTALinCombBestDim)+1),
    BTALinCombAccuracys,'x',color='blue',
        alpha = 0.5,label='BTA')
m,b=np.polyfit((np.asarray(BTALinCombBestDim)+1),
    BTALinCombAccuracys,1)
plt.plot((np.asarray(BTALinCombBestDim)+1),
    m*(np.asarray(BTALinCombBestDim)+1)+b,'--b')
plt.legend(prop={'size': 12})
plt.xticks(range(1,11))
plt.xlabel('Selected Signal of Interest', fontsize=bigFsize)
plt.ylabel('Accuracy', fontsize=bigFsize)
plt.savefig('SimLinCombPrecisionSOFs.pdf')
plt.show()
```

The code for producing Figure 5.5:

```
plt.figure(figsize=(12,7))
plt.subplot(1,2,1)
plt.plot(PCALinCombAccuracys,PCALinCombPrecisions,
      '.',color='red',alpha = 0.6,label='PCA')
plt.xlabel('Accuracy', fontsize=bigFsize)
plt.ylabel('Precision', fontsize=bigFsize)
plt.legend(prop={'size': bigFsize})
plt.subplot(1,2,2)
plt.plot(BTALinCombAccuracys,BTALinCombPrecisions,
      '.',color='blue',alpha = 0.6,label='BTA')
plt.legend(prop={'size': bigFsize})
plt.xlabel('Accuracy', fontsize=bigFsize)
plt.ylabel('Precision', fontsize=bigFsize)
plt.savefig('SimLinCombPrecisionAccuracy.pdf')
plt.show()
```

The code for taking the average of the performance metrics and producing Figure 5.13:

```
PCADLIRecall=np.mean(PCADLIRecalls)
BTADLIRecall=np.mean(BTADLIRecalls)
BTALinCombRecall=np.mean(BTALinCombRecalls)
PCALinCombRecall=np.mean(PCALinCombRecalls)
PCADLIPrecision = np.mean(PCADLIPrecisions)
BTADLIPrecision=np.mean(BTADLIPrecisions)
BTALinCombPrecision=np.mean(BTALinCombPrecisions)
PCALinCombPrecision = np.mean(PCALinCombPrecisions)
PCADLIF1 = np.mean(PCADLIF1s)
BTADLIF1 = np.mean(BTADLIF1s)
BTALinCombF1 = np.mean(BTALinCombF1s)
PCALinCombF1 = np.mean(PCALinCombF1s)
PCADLIAccuracy = np.mean(PCADLIAccuracys)
BTADLIAccuracy=np.mean(BTADLIAccuracys)
BTALinCombAccuracy=np.mean(BTALinCombAccuracys)
PCALinCombAccuracy = np.mean(PCALinCombAccuracys)
plt.figure(figsize=(6, 7))
plt.plot(['Experiment 1','Experiment 2','Experiment 3',
      'Experiment 4'],[PCALinCombRecall,PCADLIRecall,
            BTALinCombRecall,BTADLIRecall],'*',color='red',
                label='Recall')
plt.plot(['Experiment 1','Experiment 2','Experiment 3',
      'Experiment 4'],[PCALinCombPrecision,PCADLIPrecision,
            BTALinCombPrecision,BTADLIPrecision],'*',
                color='blue',label='Precision')
plt.plot(['Experiment 1','Experiment 2','Experiment 3',
      'Experiment 4'],[PCALinCombF1,PCADLIF1,BTALinCombF1,
            BTADLIF1],'*',color='green',label='F1 score')
plt.plot(['Experiment 1','Experiment 2','Experiment 3',
      'Experiment 4'],[PCALinCombAccuracy,PCADLIAccuracy,
        BTALinCombAccuracy,BTADLIAccuracy],'*',
            color='black',label='Accuracy')
plt.legend(prop={'size': 10})
plt.xlabel('Dimensionality Reduction Approach')
plt.ylabel('Average Statistical Metric')
plt.savefig('SimAllMetric.pdf')
```

## C.3 Real-World Study

Here we present the implementation of the study conducted on the real-world data set.

First we import the data in a Pandas data frame, and call it `df`. The first column is used to indicate the running group. The name of this column is `groupno`. The three last columns are meta data about the time of the measurement and type of the shut-down. The last column indicates the type of the shut-down. The name of this column is `StopReason`. The names of the other two columns are `logRowNumber` and `TimeStamp`. The `StopReason` can be `'Normal Stop'`, `'Running Trip'` or `'Starting Trip'`.

### C.3.1 Data Manipulation

The following code removes the edges of each running trip, as explained in Subsection 4.1.1.

```
def remove_edges(df_group,startCut = 45, endCutNormal=15,
    endCutAnomal = 10):
    notTripped = df_group['StopReason'][0] == 'Normal Stop'
    if (notTripped):
        if(endCutNormal != 0):
            return df_group.iloc[
                startCut:((-1)*endCutNormal),:]
        else:
            return df_group.iloc[startCut:,:]
    elif (endCutAnomal != 0):
        return df_group.iloc[startCut:((-1)*endCutAnomal),:]
    else:
        return df_group.iloc[startCut:,:]
df = df.groupby('groupno').apply(lambda x: remove_edges(x))
df.index = df.timeStamp
```

Then the following code handles the `NA` values in the data set. This process is described in Subsection 4.1.1. In the end, we divide the running groups into normal and tripping groups.

```
df=df.replace([np.inf, -np.inf], np.nan)
#removing columns with more than 10% missing
```

```
df = df[df.columns[df.isnull().mean() < 0.1]]
#Filling values in each group first with forward fill
#and then with backward fill
df = df.groupby('groupno').apply(lambda x:
    x.fillna(method='ffill'))
df = df.groupby('groupno').apply(lambda x:
    x.fillna(method='bfill'))
NormalGroups = np.unique(df['groupno']
    [df['StopReason']=='Normal Stop'])
TripGroups = np.unique(df['groupno']
    [df['StopReason'] == 'Running Trip'])
ColsToRemove = np.asarray([])
for gr_numbr in TripGroups:
    df_group = df[df['groupno']==gr_numbr]
    ColsToRemove = np.concatenate((ColsToRemove,
        np.asarray(df_group.isnull().mean().
            index[df_group.isnull().mean() > 0])),axis=0)
ColsToRemove = np.unique(ColsToRemove)
#Removing columns where NA exists in a trip group
for c in ColsToRemove:
    df = df.drop(c,axis=1)
#removing normal groups with columns that only have na
df= df.dropna()
NormalGroups = np.unique(df['groupno']
    [df['StopReason']=='Normal Stop'])
TripGroups = np.unique(df['groupno']
    [df['StopReason'] == 'Running Trip'])
```

In the following code we divide the data set into training, signal selection and test sets. In the training data set, there are only normal groups, while in the signal selection and the test data sets, there are both normal and tripping groups.

```
TrainingGroups, TestGroups=train_test_split(NormalGroups,
    test_size=int(len(TripGroups)),random_state=0)
NormalSSGroups, NormalTestGroups =
    train_test_split(TestGroups,test_size=0.5,
        random_state=0)
TripSSGroups, TripTestGroups = train_test_split(TripGroups,
    test_size=0.5,random_state=0)
TrainingData = df[df['groupno'].isin(TrainingGroups)].
    drop(['timeStamp','logRowNumber','StopReason'],axis=1)
```

Now we use the running groups in the training data set to estimate the variance of each dimension, and use the variances to scale the whole data set. Then we save the training, signal selection and test set in different variables.

```
NormalScaledVariances = TrainingData.groupby('groupno').
```

```
    apply(lambda x: ((len(x)-1)*np.var(x.
        drop(['groupno'], axis=1),ddof=1)))
NormalVariances=np.sum(NormalScaledVariances)/
    (len(TrainingData)-len(TrainingGroups))

df[df.columns.drop(['groupno','timeStamp','logRowNumber',
    'StopReason'])]=df[df.columns.
        drop(['groupno','timeStamp','logRowNumber',
            'StopReason'])]/np.sqrt(NormalVariances)

NormalSS = [df[df['groupno']==x].
    drop(['timeStamp','logRowNumber','StopReason'],axis=1)
        for x in NormalSSGroups]
NormalTest = [df[df['groupno']==x].
    drop(['timeStamp','logRowNumber','StopReason'],axis=1)
        for x in NormalTestGroups]
TripSS = [df[df['groupno']==x].
    drop(['timeStamp','logRowNumber','StopReason'],axis=1)
        for x in TripSSGroups]
TripTest = [df[df['groupno']==x].
    drop(['timeStamp','logRowNumber','StopReason'],axis=1)
        for x in TripTestGroups]
TrainingData = df[df['groupno'].isin(TrainingGroups)].
    drop(['timeStamp','logRowNumber','StopReason'],axis=1)
```

The following code prints the number of running groups in each data set.

```
print('Number of Normal Groups in
    the Signal Selection Set:')
print(len(NormalSSGroups))
print('Number of Normal Groups in the Test Set:')
print(len(NormalTestGroups))
print('Number of Trip Groups in the Signal Selection Set:')
print(len(TripSSGroups))
print('Number of Trip Groups in the Test Set:')
print(len(TripTestGroups))
print('Number of Normal Groups in the Training Set:')
print(len(TrainingGroups))
```

Finally we set some initial values for the analysis.

```
TmodeStart = 120
TmodeEnd = 5
TmodeLength = TmodeStart - TmodeEnd
TDetectionMargin = 0
SearchLen = TmodeStart - TmodeEnd + 2*TDetectionMargin
```

### C.3.2 Data Description

Here we present the code used for calculating numbers and making figures used to describe the real-world data set.

The following code calculates the length of the normal and the tripping groups:

```
TripLengths = df[df['groupno'].isin(TripGroups)].
    groupby('groupno').apply(lambda x: len(x))
NormalLengths = df[df['groupno'].isin(NormalGroups)].
    groupby('groupno').apply(lambda x: len(x))
```

The following code produces Figure 4.3:

```
idx = np.argsort(-NormalVariances)
plt.figure(figsize=(7, 5))
plt.plot(range(0,(len(NormalVariances))),
    np.log(NormalVariances[idx]),color='blue',
        label='Normal Groups',alpha=0.5,linewidth=2.0)
plt.plot(range(0,(len(TripVariances))),
    np.log(TripVariances[idx]),color='red',
        label='Tripping Groups',alpha=0.5,linewidth=2.0)
plt.legend(prop={'size': 10})
plt.ylabel('log-variance')
plt.xlabel('Dimensions')
plt.savefig('logvarplot.pdf')
plt.show()
```

The following functions are used to estimate the variances and the predictabilities of the dimensions in the real-world data set, separately in the normal and in the transition states:

```
def dividedVar(xdf):
    counter=-116
    resvec=[]
    while((-counter)<len(xdf)):
        resvec.append(np.var(xdf.iloc
            [counter:(counter+115),:],ddof=1))
        counter -= 115
    return(sum(resvec)/len(resvec))

def predictability(inputarray):
    inputarray=np.asarray(inputarray)
    return(1-(np.var(AR(inputarray).fit(1).resid)/
        np.var(inputarray)))

def DFpredictability(inputDF):
    result=[]
    for c in inputDF.columns:
        result.append(predictability(inputDF[c]))
    result=pd.core.series.Series(result)
    result.index = inputDF.columns
    return(result)

def dividedPred(xdf):
    counter=-116
    resvec=[]
    while((-counter)<len(xdf)):
        resvec.append(DFpredictability(xdf.iloc
            [counter:(counter+115),:]))
        counter -= 115
    return(sum(resvec)/len(resvec))
```

The following code uses the functions above to estimate the variances and the predictabilities of the dimensions in the real-world data set, separately in the normal and in the transition states:

```
NormalStatesVariances = NormalDF.groupby('groupno').
    apply(lambda x: dividedVar(x))
TransitionScaledVariances = TripDF.groupby('groupno').
    apply(lambda x: (np.var(x.iloc[-120:-5,:],ddof=1)))
TransitionVariances=np.mean(TransitionScaledVariances)
TransitionVariances=TransitionVariances.
    drop(['groupno','logRowNumber'])
NormalStateVariances=np.mean(NormalStatesVariances)
NormalStateVariances=NormalStateVariances.
    drop(['groupno','logRowNumber'])
stateIDX=np.argsort(-NormalStateVariances)
NormalStatesPreds = NormalDF.groupby('groupno').
    apply(lambda x: dividedPred(x.
        drop(['timeStamp', 'logRowNumber',
            'StopReason','groupno'],axis=1)))
NormalScaledPreds = NormalDF.groupby('groupno').
    apply(lambda x: len(x)*DFpredictability(x.
        drop(['timeStamp', 'logRowNumber',
            'StopReason','groupno'],axis=1)))
TripScaledPreds = TripDF.groupby('groupno').
    apply(lambda x: len(x)*DFpredictability(x.
        drop(['timeStamp', 'logRowNumber',
            'StopReason','groupno'],axis=1)))
TransitionScaledPreds = TripDF.groupby('groupno').
    apply(lambda x: DFpredictability((x.
        drop(['timeStamp', 'logRowNumber',
            'StopReason','groupno'],axis=1)).
                iloc[-120:-5,:]))
NormalPreds=np.sum(NormalScaledPreds)/len(NormalDF)
```

```
TripPreds=np.sum(TripScaledPreds)/len(TripDF)
TransitionPreds = np.mean(TransitionScaledPreds)
NormalStatePreds = np.mean(NormalStatesPreds)
```

The following code produces Figure 5.14:

```
plt.figure(figsize=(12, 5))
plt.subplot(1,2,2)
plt.plot(range(0,(len(NormalPreds))),
    NormalStatePreds[stateIDX],color='black',
        label='Normal States',alpha=0.5,linewidth=2.0)
plt.plot(range(0,(len(TripPreds))),
    TransitionPreds[stateIDX],color='green',
        label='Transition States',alpha=0.5,linewidth=2.0)
plt.legend(prop={'size': 11})
plt.ylabel('Predictability',fontsize=13)
plt.xlabel('Dimensions',fontsize=13)
plt.subplot(1,2,1)
plt.plot(range(0,(len(NormalStateVariances))),
    np.log(NormalStateVariances[stateIDX]),color='black',
        label='Normal States',alpha=0.5,linewidth=2.0)
plt.plot(range(0,(len(TransitionVariances))),
    np.log(TransitionVariances[stateIDX]),color='green',
        label='Transition States',alpha=0.5,linewidth=2.0)
plt.legend(prop={'size': 11})
plt.ylabel('log-variance',fontsize=13)
plt.xlabel('Dimensions',fontsize=13)
plt.savefig('Predplot.pdf')
plt.show()
```

### C.3.3 Signal Selection

Here we apply different experiments to select the signal of interest.

#### No Transformation

In the following code, we select the signal of interest among the dimensions of the data:

```
algo=rpt.Pelt(model="ar",jump=int(TmodeLength/5))
penConst= 0.01
Signals = df.columns.drop(['groupno','timeStamp',
    'logRowNumber','StopReason'])
#Selecting pen function:
NTMWUtests = []
for DLI in tqdm(Signals):
    alphas = []
    betas = []
    for groupDF in TripSS:
        SOF = np.asarray(groupDF[DLI])
        SOF = SOF[((len(SOF)-TmodeStart-
            TDetectionMargin)%SearchLen):]
        algo = algo.fit(SOF)
        DFresult = algo.predict(pen =
            penConst*np.log(len(SOF)))
        DFresult = np.asarray(DFresult)
        if(len(np.intersect1d(range(
            TmodeEnd-TDetectionMargin,
                TmodeStart+TDetectionMargin),
                    (len(SOF)-DFresult)))>0):
            alphas.append(1)
        else:
            alphas.append(0)
        DFresult=DFresult[(DFresult <=
            (len(SOF)-TmodeStart-TDetectionMargin))]
        if(len(DFresult)>0):
            betas.append(1)
        else:
            betas.append(0)

    for groupDF in NormalSS:
        SOF = np.asarray(groupDF[DLI])
        SOF = SOF[(len(SOF) % SearchLen):]
        algo = algo.fit(SOF)
        DFresult = algo.predict(pen =
            penConst*np.log(len(SOF)))
        DFresult = np.asarray(DFresult)
        if(len(DFresult)>1):
            betas.append(1)
        else:
            betas.append(0)
    try:
        NTMWUtests.append(mannwhitneyu(
            x=betas,y=alphas,alternative='less'))
    except:
        NTMWUtests.append(mannwhitneyu(x=[1,-1]*1000,
            y=[-1,1]*1000,alternative='less'))
NTpvalMins=np.min([np.log(x.pvalue) for x in NTMWUtests])
NTPSbestDims=np.where(NTMWUtests==np.min(NTMWUtests))[0][0]
```

**Experiment 2**

In the following code we select the signal of interest among the signals leverage, discrepancy and influence of the PCA transformed data. Here the $\alpha_i$s and $\beta_j$s are set according to the division in Figure 3.4.

```
VarLimit = 0.9
DLISignals = ['myPC.Leverage(groupDF, s_l=S)',
     'myPC.Discrepancy(groupDF, s_di=S)',
          'myPC.Influence(groupDF, s_i=S)']
algo=rpt.Pelt(model="normal",jump=int(TmodeLength/5))
penConst= 30
myPC = VARBT(groug_c_name='groupno', BoxTiao=False)
myPC.fit(TrainingData)
ExpVar = myPC.variances/sum(myPC.variances)
S = (np.where(np.asanyarray([sum(ExpVar[:x+1])
     for x in range(0,len(ExpVar))]) > VarLimit)[0][0])+1
PCAMWUtests = []
for DLI in tqdm(range(0,len(DLISignals))):
     alphas = []
     betas = []
     for groupDF in TripSS:
          SOF = eval(DLISignals[DLI])
          SOF = SOF[((len(SOF)-TmodeStart-
               TDetectionMargin)%SearchLen):]
          algo = algo.fit(SOF)
          DFresult = algo.predict(pen =
               penConst*np.log(len(SOF)))
          DFresult = np.asarray(DFresult)
          if(len(np.intersect1d(range(
               TmodeEnd-TDetectionMargin,
                    TmodeStart+TDetectionMargin),
                         (len(SOF)-DFresult)))>0):
               alphas.append(1)
          else:
               alphas.append(0)
          DFresult=DFresult[(DFresult <=
               (len(SOF)-TmodeStart-TDetectionMargin))]
          if(len(DFresult)>0):
               betas.append(1)
          else:
               betas.append(0)

     for groupDF in NormalSS:
          SOF = eval(DLISignals[DLI])
          SOF = SOF[(len(SOF) % SearchLen):]
          algo = algo.fit(SOF)
```

```
            DFresult = algo.predict(pen =
                  penConst*np.log(len(SOF)))
            DFresult = np.asarray(DFresult)
            if(len(DFresult)>1):
                  betas.append(1)
            else:
                  betas.append(0)
      try:
            PCAMWUtests.append(mannwhitneyu(
                  x=betas,y=alphas,alternative='less'))
      except:
            PCAMWUtests.append(mannwhitneyu(
                  x=[1,-1]*1000,y=[-1,1]*1000,
                        alternative='less'))
PCApvalMins=np.min([np.log(x.pvalue) for x in PCAMWUtests])
PCAPSbestDims=np.
      where(PCAMWUtests==np.min(PCAMWUtests))[0][0]
```

In the following code we select the signal of interest among the signals leverage, discrepancy and influence of the PCA transformed data. Here the $\alpha_i$s and $\beta_j$s are set according to the division in Figure 3.5.

```
VarLimit = 0.9
DLISignals = ['myPC.Leverage(groupDF, s_l=S)',
      'myPC.Discrepancy(groupDF, s_di=S)',
            'myPC.Influence(groupDF, s_i=S)']
algo=rpt.Pelt(model="normal",jump=int(TmodeLength/5))
penConst= 30
myPC = VARBT(groug_c_name='groupno',BoxTiao=False)
myPC.fit(TrainingData)
ExpVar = myPC.variances/sum(myPC.variances)
S = (np.where(np.asanyarray([sum(ExpVar[:x+1])
      for x in range(0,len(ExpVar))]) > VarLimit)[0][0])+1
PCAMWUtests = []
for DLI in tqdm(range(0,len(DLISignals))):
      alphas = []
      betas = []
      for groupDF in TripSS:
            SOF = eval(DLISignals[DLI])
            SOF = SOF[((len(SOF)-TmodeStart-
                  TDetectionMargin)%SearchLen):]
            algo = algo.fit(SOF)
            DFresult = algo.predict(pen = 200)
            DFresult = np.asarray(DFresult)
            if(len(np.intersect1d(range(
                  TmodeEnd-TDetectionMargin,TmodeStart+
                        TDetectionMargin),(len(SOF)-DFresult)))>0):
```

```
            alphas.append(1)
        else:
            alphas.append(0)
        DFresult=DFresult[(DFresult <=
            (len(SOF)-TmodeStart-TDetectionMargin))]
        TotalNrOfN =
            np.ceil((len(SOF)-TmodeStart-TDetectionMargin)/
                SearchLen)
        FalseN = len(np.unique(np.floor(DFresult /
            SearchLen)))
        betas.extend([1]*int(FalseN))
        betas.extend([0]*int(TotalNrOfN-FalseN))

    for groupDF in NormalSS:
        SOF = eval(DLISignals[DLI])
        SOF = SOF[(len(SOF) % SearchLen):]
        algo = algo.fit(SOF)
        DFresult = algo.predict(pen =
            penConst*np.log(len(SOF)))
        DFresult = np.asarray(DFresult)
        TotalNrOfN = np.ceil(DFresult[-1] / SearchLen)
        FalseN = len(np.unique(np.floor(DFresult[:-1] /
            SearchLen)))
        betas.extend([1]*int(FalseN))
        betas.extend([0]*int(TotalNrOfN-FalseN))
    try:
        PCAMWUtests.append(mannwhitneyu(
            x=betas,y=alphas,alternative='less'))
    except:
        PCAMWUtests.append(mannwhitneyu(
            x=[1,-1]*1000,y=[-1,1]*1000,
                alternative='less'))
PCApvalMins=np.min([np.log(x.pvalue) for x in PCAMWUtests])
PCAPSbestDims=np.
    where(PCAMWUtests==np.min(PCAMWUtests))[0][0]
```

**Experiment 4**

In the following code we select the signal of interest among the signals leverage, discrepancy and influence of the BTA transformed data. Here the $\alpha_i$s and $\beta_j$s are set according to the division in Figure 3.4.

```
PredLimit = 0.9
DLISignals = ['myBT.Leverage(groupDF, s_l=S)',
    'myBT.Discrepancy(groupDF, s_di=S)',
        'myBT.Influence(groupDF, s_i=S)']
```

```
algo=rpt.Pelt(model="normal",jump=int(TmodeLength/5))
penConst= 30
myBT = VARBT(groug_c_name='groupno')
myBT.fit(TrainingData)
ExpPred = myBT.predictabilities/sum(myBT.predictabilities)
S = (np.where(np.asanyarray([sum(ExpPred[:x+1])
     for x in range(0,len(ExpPred))]) > PredLimit)[0][0])+1
BTAMWUtests = []
for DLI in tqdm(range(0,len(DLISignals))):
     alphas = []
     betas = []
     for groupDF in TripSS:
          SOF = eval(DLISignals[DLI])
          SOF = SOF[((len(SOF)-TmodeStart-
               TDetectionMargin)%SearchLen):]
          algo = algo.fit(SOF)
          DFresult = algo.predict(pen =
               penConst*np.log(len(SOF)))
          DFresult = np.asarray(DFresult)
          if(len(np.intersect1d(range(TmodeEnd-
               TDetectionMargin,TmodeStart+
                    TDetectionMargin),(len(SOF)-
                         DFresult)))>0):
               alphas.append(1)
          else:
               alphas.append(0)
          DFresult=DFresult[(DFresult <=
               (len(SOF)-TmodeStart-TDetectionMargin))]
          if(len(DFresult)>0):
               betas.append(1)
          else:
               betas.append(0)

     for groupDF in NormalSS:
          SOF = eval(DLISignals[DLI])
          SOF = SOF[(len(SOF) % SearchLen):]
          algo = algo.fit(SOF)
          DFresult = algo.predict(pen =
               penConst*np.log(len(SOF)))
          DFresult = np.asarray(DFresult)
          if(len(DFresult)>1):
               betas.append(1)
          else:
               betas.append(0)
     try:
          BTAMWUtests.append(mannwhitneyu(
               x=betas,y=alphas,alternative='less'))
     except:
          BTAMWUtests.append(mannwhitneyu(
               x=[1,-1]*1000,y=[-1,1]*1000,
                    alternative='less'))
BTApvalMins=np.min([np.log(x.pvalue) for x in BTAMWUtests])
BTAPSbestDims=np.where(BTAMWUtests==
     np.min(BTAMWUtests))[0][0]
```

In the following code we select the signal of interest among the signals leverage, discrepancy and influence of the BTA transformed data. Here the $\alpha_i$s and $\beta_j$s are set according to the division in Figure 3.5.

```
PredLimit = 0.9
DLISignals = ['myBT.Leverage(groupDF,s_l=S)',
     'myBT.Discrepancy(groupDF,s_di=S)',
          'myBT.Influence(groupDF,s_i=S)']
algo=rpt.Pelt(model="normal",jump=int(TmodeLength/5))
penConst= 30
myBT = VARBT(groug_c_name='groupno')
myBT.fit(TrainingData)
ExpPred = myBT.predictabilities/sum(myBT.predictabilities)
S = (np.where(np.asanyarray([sum(ExpPred[:x+1])
     for x in range(0,len(ExpPred))]) > PredLimit)[0][0])+1
BTAMWUtests = []
for DLI in tqdm(range(0,len(DLISignals))):
     alphas = []
     betas = []
     for groupDF in TripSS:
          SOF = eval(DLISignals[DLI])
          SOF = SOF[((len(SOF)-TmodeStart-
               TDetectionMargin)%SearchLen):]
          algo = algo.fit(SOF)
          DFresult = algo.predict(pen = 200)
          DFresult = np.asarray(DFresult)
          if(len(np.intersect1d(range(TmodeEnd-
               TDetectionMargin,TmodeStart+
                    TDetectionMargin),
                         (len(SOF)-DFresult)))>0):
               alphas.append(1)
          else:
               alphas.append(0)
          DFresult=DFresult[(DFresult <=
               (len(SOF)-TmodeStart-TDetectionMargin))]
          TotalNrOfN =
               np.ceil((len(SOF)-TmodeStart-TDetectionMargin)/
                    SearchLen)
          FalseN = len(np.unique(np.floor(DFresult /
               SearchLen)))
          betas.extend([1]*int(FalseN))
          betas.extend([0]*int(TotalNrOfN-FalseN))

     for groupDF in NormalSS:
          SOF = eval(DLISignals[DLI])
          SOF = SOF[(len(SOF) % SearchLen):]
          algo = algo.fit(SOF)
          DFresult = algo.predict(pen =
               penConst*np.log(len(SOF)))
          DFresult = np.asarray(DFresult)
          TotalNrOfN = np.ceil(DFresult[-1] / SearchLen)
          FalseN = len(np.unique(np.floor(DFresult[:-1] /
               SearchLen)))
          betas.extend([1]*int(FalseN))
          betas.extend([0]*int(TotalNrOfN-FalseN))
     try:
          BTAMWUtests.append(mannwhitneyu(
               x=betas,y=alphas,alternative='less'))
     except:
          BTAMWUtests.append(mannwhitneyu(
               x=[1,-1]*1000,y=[-1,1]*1000,
                    alternative='less'))
BTApvalMins=np.min([np.log(x.pvalue) for x in BTAMWUtests])
BTAPSbestDims=np.where(BTAMWUtests==
     np.min(BTAMWUtests))[0][0]
```

### C.3.4 Predictive Models

Here, we present the code used for the training and the evaluation of the predictive models on the real-world data set.

#### No Transformation

The code for the training and the evaluation of a CUSUM model based on one of the original dimensions of the data, selected in the signal selection part:

```
Signals = df.columns.drop(['groupno','timeStamp',
    'logRowNumber','StopReason'])
DLI = Signals[NTPSbestDims]

NTNormalssSOFs = [np.asarray(x[DLI]) for x in NormalSS]
NTTripssSOFs = [np.asarray(x[DLI]) for x in TripSS]
NTNormalTestSOFs = [np.asarray(x[DLI]) for x in NormalTest]
```

```
NTTripTestSOFs = [np.asarray(x[DLI]) for x in TripTest]

#Making CUSUM
NTNormalTarget, NTNormalSTD,NTBest_k ,NTBest_h,
    NTBestTrainingMetric = Make_CUSUM(NTNormalssSOFs,
        NTTripssSOFs, TmodeLength, TmodeEnd,
            TDetectionMargin,
                include_low = True,metric="f1")
NTConfMat = Evaluate_CUSUM(NTNormalTestSOFs,
    NTTripTestSOFs,NTNormalTarget,NTBest_k,NTBest_h,
        TmodeLength,TmodeEnd, TDetectionMargin,
            include_low = True)
```

**Experiment 2**

The code for the training and the evaluation of a CUSUM model based on the discrepancy of the PCA transformed data:

```
DLISignals = ['myPC.Leverage(x,s_l=S)',
    'myPC.Discrepancy(x,s_di=S)',
        'myPC.Influence(x,s_i=S)']
DLI = DLISignals[PCAPSbestDims]
VarLimit = 0.90
S = (np.where(np.asanyarray([sum(ExpVar[:x+1])
    for x in range(0,len(ExpVar))]) > VarLimit)[0][0])+1
PCADLINormalssSOFs = [eval(DLI) for x in NormalSS]
PCADLITripssSOFs = [eval(DLI) for x in TripSS]
PCADLINormalTestSOFs = [eval(DLI) for x in NormalTest]
PCADLITripTestSOFs = [eval(DLI) for x in TripTest]

#Making CUSUM
PCADLINormalTarget, PCADLINormalSTD,PCADLIBest_k ,
    PCADLIBest_h,PCADLIBestTrainingMetric =
        Make_CUSUM(PCADLINormalssSOFs, PCADLITripssSOFs,
            TmodeLength, TmodeEnd, TDetectionMargin,
                include_low = False)
PCADLIConfMat = Evaluate_CUSUM(PCADLINormalTestSOFs,
    PCADLITripTestSOFs,PCADLINormalTarget,PCADLIBest_k,
        PCADLIBest_h,TmodeLength,TmodeEnd,
            TDetectionMargin,include_low = False)
```

**Experiment 4**

The code for the training and the evaluation of a CUSUM model based on the discrepancy of the BTA transformed data:

```
DLISignals = ['myBT.Leverage(x, s_l=S)',
     'myBT.Discrepancy(x, s_di=S)',
          'myBT.Influence(x, s_i=S)']
DLI = DLISignals[BTAPSbestDims]
PredLimit = 0.90
S = (np.where(np.asanyarray([sum(ExpPred[:x+1])
     for x in range(0,len(ExpPred))]) > PredLimit)[0][0])+1
BTADLINormalssSOFs = [eval(DLI) for x in NormalSS]
BTADLITripssSOFs = [eval(DLI) for x in TripSS]
BTADLINormalTestSOFs = [eval(DLI) for x in NormalTest]
BTADLITripTestSOFs = [eval(DLI) for x in TripTest]

#Making CUSUM
BTADLINormalTarget, BTADLINormalSTD,BTADLIBest_k ,
     BTADLIBest_h,BTADLIBestTrainingMetric =
          Make_CUSUM(BTADLINormalssSOFs, BTADLITripssSOFs,
               TmodeLength, TmodeEnd, TDetectionMargin,
                    include_low = False)
BTADLIConfMat = Evaluate_CUSUM(BTADLINormalTestSOFs,
     BTADLITripTestSOFs,BTADLINormalTarget,BTADLIBest_k,
          BTADLIBest_h,TmodeLength,TmodeEnd,
               TDetectionMargin,include_low = False)
```

**Multivariate Models**

First we define a new function for the evaluation of our multivariate CUSUM model:

```
def Evaluate_CUSUM_multi(NormalData, TripData,
     NormalTargets, ks,hs,NormalLen,TransitionEnd,
          DetectMarg,include_low = True,trsh=0):
     NormalCUSUMhisvec = np.asarray([])
     dimvec = NormalData[0].columns
     NormalOutWheresVec = []
     TripOutWheresVec = []
     for dimC in tqdm(range(0,len(dimvec))):
          dim=dimvec[dimC]
          NormalTarget=NormalTargets[dimC]
          k=ks[dimC]
          h=hs[dimC]
          NormalSOFs=np.asarray([np.asarray(x[dim])
               for x in NormalData])
          TripSOFs=np.asarray([np.asarray(x[dim])
               for x in TripData])
          NormalSOFs=[x[(len(x)%(NormalLen+DetectMarg)):]
               for x in NormalSOFs]
```

```
        TripSOFs=[x[(((len(x)-NormalLen-TransitionEnd)%
            (NormalLen+DetectMarg)):] for x in TripSOFs]
        TripTimes = np.asarray([len(x) for x in TripSOFs])
        NormalTimes = np.asarray([len(x)
            for x in NormalSOFs])
        NormalCUSUMhis = [Compute_CUSUM(x,NormalTarget,k)
            for x in NormalSOFs]
        TripCUSUMhis = [Compute_CUSUM(x,NormalTarget,k)
            for x in TripSOFs]
        NormalOutWheres = [np.where(x)[0] for x in
            [x>h for x in NormalCUSUMhis]]
        TripOutWheres = [np.where(x)[0] for x in
            [x>h for x in TripCUSUMhis]]
        if(include_low):
            NormalCUSUMlos = [Compute_CUSUM(
                x,NormalTarget,k,side='lo')
                    for x in NormalSOFs]
            TripCUSUMlos = [Compute_CUSUM(
                x,NormalTarget,k,side='lo')
                    for x in TripSOFs]
            TripOutWheres.extend([np.where(x)[0]
                for x in [x>h for x in TripCUSUMlos]])
            NormalOutWheres.extend([np.where(x)[0]
                for x in [x>h for x in NormalCUSUMlos]])
        TripOutWheres = [np.asarray(x)
            for x in TripOutWheres]
        NormalOutWheres = [np.asarray(x)
            for x in NormalOutWheres]
        TripOutWheres = np.asarray(TripOutWheres)
        NormalOutWheres = np.asarray(NormalOutWheres)
        NormalOutWheresVec.append(NormalOutWheres)
        TripOutWheresVec.append(TripOutWheres)
    TruePvec=[np.asarray([(len(np.intersect1d(
        range((TransitionEnd-DetectMarg),
            (TransitionEnd+NormalLen)),
                (TripTimes[x]-TripOutWheres[x]))) > 0)
                    for x in range(0,len(TripSOFs))])
                        for TripOutWheres in
                            TripOutWheresVec]

    TrueP = sum(sum(TruePvec)>trsh)
    TotalNrOfN = 0
    FalseP = 0
    for counter in range(0,len(NormalSOFs)):
        TotalNrOfN += np.ceil(NormalTimes[counter] /
            (NormalLen+DetectMarg))
    for counter in range(0,len(TripSOFs)):
        TotalNrOfN += np.ceil(
            (TripTimes[counter]−NormalLen−TransitionEnd)/
                (NormalLen+DetectMarg))
    TotalNrOfP = len(TripSOFs)

    FalsePvec=[]
    whereC=0
    for counter in range(0,len(NormalSOFs)):
        for NormalOutWheres in NormalOutWheresVec:
            FalsePvec += (whereC+np.unique(
                np.floor(NormalOutWheres[counter] /
                    (NormalLen+DetectMarg)))).tolist()
        whereC+=np.max(FalsePvec)+1
    for counter in range(0,len(TripSOFs)):
        for TripOutWheres in TripOutWheresVec:
            FalsePvec += (whereC+np.unique(np.floor((
                TripOutWheres[counter])[(TripOutWheres
                    [counter] <= (TripTimes[counter]−
                        NormalLen−TransitionEnd))]/
                            (NormalLen+DetectMarg)))).
                                tolist()
        whereC+=np.max(FalsePvec)+1
    uniqs,counts=np.unique(FalsePvec,return_counts=True)
    FalseP = len(uniqs[np.asarray(counts)>trsh])
    FalseN = TotalNrOfP−TrueP
    TrueN = TotalNrOfN−FalseP
    return(pd.DataFrame([[TrueN,FalseP],
        [FalseN,TrueP]],columns=
            ['Predicted Normal','Predicted Trip'],
                index=['Actual Normal', 'Actual Trip']))
```

The code for a multivariate model with the last 10 principal components:

```
Signals = range(52,62)
DLI = range(52,62)
PC10NormalTestSOFs = [pd.DataFrame(myPC.transform(x)
    [:,DLI]) for x in NormalTest]
PC10TripTestSOFs = [pd.DataFrame(myPC.transform(x)
    [:,DLI]) for x in TripTest]
PC10NormalTargets=[]
PC10Best_ks=[]
PC10Best_hs=[]
#Making CUSUM
for DLI in tqdm(Signals):
    PC10NormalssSOFs = [np.asarray(myPC.transform(x)
        [:,DLI]) for x in NormalSS]
    PC10TripssSOFs = [np.asarray(myPC.transform(x)
        [:,DLI]) for x in TripSS]
    aPC10NormalTarget, aPC10NormalSTD,aPC10Best_k ,
        aPC10Best_h,aPC10BestTrainingMetric =
            Make_CUSUM(PC10NormalssSOFs, PC10TripssSOFs,
                TmodeLength, TmodeEnd, TDetectionMargin,
                    include_low = True)
    PC10NormalTargets.append(aPC10NormalTarget)
    PC10Best_ks.append(aPC10Best_k)
    PC10Best_hs.append(aPC10Best_h)
aPC10ConfMat = Evaluate_CUSUM_multi([x.iloc[:,range(20,30)]
    for x in PC10NormalTestSOFs], [x.iloc[:,range(20,30)]
        for x in PC10TripTestSOFs],PC10NormalTargets[20:30],
            PC10Best_ks[20:30],PC10Best_hs[20:30],
                TmodeLength,TmodeEnd, TDetectionMargin,
                    include_low = True)
```

The code for a multivariate model with the last 10 Box-Tiao components:

```
Signals = range(52,62)
DLI = range(52,62)
BT10NormalTestSOFs = [pd.DataFrame(myBT.transform(x)
    [:,DLI]) for x in NormalTest]
BT10TripTestSOFs = [pd.DataFrame(myBT.transform(x)[:,DLI])
    for x in TripTest]
BT10NormalTargets=[]
BT10Best_ks=[]
BT10Best_hs=[]
#Making CUSUM
for DLI in tqdm(Signals):
    BT10NormalssSOFs = [np.asarray(myBT.transform(x)[:,DLI])
        for x in NormalSS]
    BT10TripssSOFs = [np.asarray(myBT.transform(x)[:,DLI])
        for x in TripSS]
    aBT10NormalTarget, aBT10NormalSTD,aBT10Best_k ,
        aBT10Best_h,aBT10BestTrainingMetric =
            Make_CUSUM(BT10NormalssSOFs, BT10TripssSOFs,
                TmodeLength, TmodeEnd, TDetectionMargin,
                    include_low = True)
    BT10NormalTargets.append(aBT10NormalTarget)
    BT10Best_ks.append(aBT10Best_k)
    BT10Best_hs.append(aBT10Best_h)
aBT10ConfMat = Evaluate_CUSUM_multi(BT10NormalTestSOFs,
    BT10TripTestSOFs,BT10NormalTargets,BT10Best_ks,
        BT10Best_hs,TmodeLength,TmodeEnd,
            TDetectionMargin,include_low = True)
```

## C.3.5 Results and Figures

Here, we present the code used for calculating the statistical metrics and producing the figures.

The code for producing Figures 5.18, 5.16 and 5.17:

```
plt.plot([x.pvalue for x in NTMWUtests])
plt.xlabel('Dimensions')
plt.ylabel('p-values')
plt.savefig('RWNTpvalues.pdf')

plt.figure(figsize=(12, 5))
plt.subplot(1,2,2)
plt.plot(['Leverage','Discrepancy','Influence'],
    [np.log(x.pvalue) for x in PCAMWUtests], color = 'red',
        label = 'PCA')
plt.xlabel('Signals', fontsize = 16)
plt.xticks(['Leverage','Discrepancy','Influence'],
    fontsize = 13)
plt.legend(prop={'size': 16})
plt.ylabel('log(p-values)',fontsize=16)
plt.subplot(1,2,1)
plt.plot(['Leverage','Discrepancy','Influence'],
    [np.log(x.pvalue) for x in BTAMWUtests],
        color = 'blue', label = 'BTA')
plt.xlabel('Signals', fontsize = 16)
plt.xticks(['Leverage','Discrepancy','Influence'],
    fontsize = 13)
plt.ylabel('log(p-values)',fontsize=16)
plt.legend(prop={'size': 16})
plt.savefig('RWpvalues.pdf')
```

The code used to produce Figure 5.15:

```
Signals = df.columns.drop(['groupno','timeStamp',
    'logRowNumber','StopReason'])
DLI = Signals[NTPSbestDims]
rpt.display(NormalSS[4][DLI],
    [42,95,125,130,175,210,len(NormalSS[4][DLI])])
plt.ylabel('Signal',fontsize=13)
plt.xlabel('Time (minutes)',fontsize=13)
plt.savefig('RWBestDimNormal.pdf', bbox_inches = "tight")
plt.show()
```

Now we calculate the four statistical metrics for our models:

```
x=BTADLIConfMat
BTADLIRecall=(x.iloc[1,1])/(x.iloc[1,1]+x.iloc[1,0])
BTADLIPrecision=(x.iloc[1,1])/(x.iloc[1,1]+x.iloc[0,1])
```

```
BTADLIF1 = 2*(BTADLIRecall*BTADLIPrecision)/
     (BTADLIRecall+BTADLIPrecision)
BTADLIAccuracy=(x.iloc[0,0]+x.iloc[1,1])/
     (x.sum().sum())

x=PCADLIConfMat
PCADLIRecall=(x.iloc[1,1])/(x.iloc[1,1]+x.iloc[1,0])
PCADLIPrecision = (x.iloc[1,1])/(x.iloc[1,1]+x.iloc[0,1])
PCADLIF1 = 2*(PCADLIRecall*PCADLIPrecision)/
     (PCADLIRecall+PCADLIPrecision)
PCADLIAccuracy = (x.iloc[0,0]+x.iloc[1,1])/(x.sum().sum())

x=NTConfMat
NTRecall=(x.iloc[1,1])/(x.iloc[1,1]+x.iloc[1,0])
NTPrecision = (x.iloc[1,1])/(x.iloc[1,1]+x.iloc[0,1])
NTF1 = 2*(PCADLIRecall*PCADLIPrecision)/
     (PCADLIRecall+PCADLIPrecision)
NTAccuracy = (x.iloc[0,0]+x.iloc[1,1])/(x.sum().sum())

x=aPC10ConfMat
NT10PCRecall=(x.iloc[1,1])/(x.iloc[1,1]+x.iloc[1,0])
NT10PCPrecision = (x.iloc[1,1])/(x.iloc[1,1]+x.iloc[0,1])
NT10PCF1 = 2*(PCADLIRecall*PCADLIPrecision)/
     (PCADLIRecall+PCADLIPrecision)
NT10PCAccuracy = (x.iloc[0,0]+x.iloc[1,1])/(x.sum().sum())

x=aBT10ConfMat
NT10BTRecall=(x.iloc[1,1])/(x.iloc[1,1]+x.iloc[1,0])
NT10BTPrecision = (x.iloc[1,1])/(x.iloc[1,1]+x.iloc[0,1])
NT10BTF1 = 2*(PCADLIRecall*PCADLIPrecision)/
     (PCADLIRecall+PCADLIPrecision)
NT10BTAccuracy = (x.iloc[0,0]+x.iloc[1,1])/(x.sum().sum())
```

The following code is used for the production of Figure 5.19:

```
plt.figure(figsize=(6, 7))
plt.plot(['Experiment 2','Experiment 4','No Transformation']
     ,[PCADLIRecall,BTADLIRecall,NTRecall],'*',
          color='red',label='Recall')
plt.plot(['Experiment 2','Experiment 4','No Transformation']
     ,[PCADLIPrecision,BTADLIPrecision,NTPrecision],'*',
          color='blue',label='Precision')
plt.plot(['Experiment 2','Experiment 4','No Transformation']
     ,[PCADLIF1,BTADLIF1,NTF1],'*',color='green',
          label='F1 score')
plt.plot(['Experiment 2','Experiment 4','No Transformation']
     ,[PCADLIAccuracy,BTADLIAccuracy,NTAccuracy],'*',
        color='black', label='Accuracy')
plt.legend(prop={'size': 10})
plt.xlabel('Dimensionality Reduction Approach')
plt.ylabel('Statistical Metric')
plt.savefig('RWAllMetric.pdf')
```

The following code is used for the production of Figure 5.20:

```
plt.figure(figsize=(6, 7))
plt.plot(['10 Last PCs','10 Last BTs'],
    [NT10PCRecall,NT10BTRecall],'*',color='red',
        label='Recall')
plt.plot(['10 Last PCs','10 Last BTs'],
    [NT10PCPrecision,NT10BTPrecision],'*',color='blue',
        label='Precision')
plt.plot(['10 Last PCs','10 Last BTs'],
    [NT10PCF1,NT10BTF1],'*',color='green',label='F1 score')
plt.plot(['10 Last PCs','10 Last BTs'],
    [NT10PCAccuracy,NT10BTAccuracy],'*',color='black',
        label='Accuracy')
plt.legend(prop={'size': 10})
plt.xlabel('Dimensionality Reduction Approach')
plt.ylabel('Statistical Metric')
plt.savefig('RWideasMetric.pdf')
```

Amirhossein Kazemi

NTNU
Norwegian University of
Science and Technology

DNV·GL