Master Thesis

Changepoint Detection in Smartphone Usage

conducted at the
Signal Processing and Speech Communications Laboratory
Graz University of Technology, Austria

in co-operation with
meemo-tec OG
Graz, Austria

by
Johanna Rock, 01130333

Supervisors:
Assoc.Prof. Dipl.-Ing. Dr. mont. Franz Pernkopf
Dipl.-Ing. BSc Christian Knoll

Graz, May 22, 2018

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

date	(signature)

Acknowledgments

First and foremost, I would like to thank the three founders of the company meemo-tec OG, *Dipl.-Ing. Ralph Gruber, Dipl.-Ing. Christian Pendl and Manfred Weiss, for giving me the chance to work on such an interesting topic, for their helpfulness in every matter and for letting me be a part of their wonderful team. I gratefully thank Assoc.Prof. Dipl.-Ing. Dr. mont. Franz Pernkopf and Dipl.-Ing. BSc Christian Knoll for their guidance, their patience and their fast and valuable support throughout this thesis. Finally, I would like to thank Gabriel Schanner, for showing interest in my work, for listening to my presentation test runs and especially for helping me out whenever I needed him.*

Abstract (English)

Psychiatric health care relies heavily on self assessments and -monitoring. The treatment's success of patients with bipolar disorder depends on an early detection of depressive and manic states. Objective behavioral measurements, representing relevant aspects of the mental illness, can be collected by smartphone applications.

This thesis investigates the application of changepoint detection algorithms to smartphone usage data, in order to recognize bipolar state changes. We pursue the goal to autonomously learn the user behavior, and detect changes therein, without requiring any manual configurations, user inputs or learning targets. This enables the application of the approach to new, unseen data.

We introduce a change detection process consisting of data collection and -processing, feature selection and -extraction, changepoint detection and evaluation. Expert knowledge is used to select disease relevant measurements, while unsupervised feature selection methods are used to further narrow the subset down to user relevant features. The changepoint detection algorithms *ChangeFinder* and *Bayesian Online Changepoint Detection* are implemented and evaluated according to the objectives of this thesis.

Results are presented on the basis of three datasets. Hypothetical data is used to evaluate the algorithmic performance according to different use cases, such as outliers and changepoints of various causes and amplitudes. We show the detection of incisive events in smartphone data of ordinary users, among them a ligament rupture and a changing relationship status. Usage data of bipolar patients, which originates from an ongoing clinical study, is used to demonstrate the detection of bipolar state changes. We conclude, that the *Bayesian Online Changepoint Detection* algorithm is better suited for our objectives. It incorporates prior knowledge about the domain. Thus it gives us the possibility to further improve the results and configure the algorithm to find certain types of changes, such as changes between bipolar states. No manual parameter selection is required and the performance on our data is promising. However, further evaluation on bipolar usage data is required.

Keywords: Changepoint Detection, Bayesian Changepoint Detection, Smartphone Usage Data, Bipolar Disorder, Mental Disease Monitoring, Time Series Analysis, Multivariate Time Series, Unsupervised Learning

Abstract (German)

Psychiatrische Gesundheitsversorgung ist stark auf Selbsteinschätzung und -beobachtung angewiesen. Der Behandlungserfolg von Patienten mit bipolarer Störung hängt von einer frühzeitigen Erkennung von depressiven und manischen Phasen ab. Objektive Verhaltensmessungen, welche relevante Aspekte der psychischen Erkrankung widerspiegeln, können via Smartphone Applikationen gesammelt werden.

Diese Diplomarbeit untersucht die Anwendung von Algorithmen zur Veränderungsdetektion (Changepoint Detection) anhand von Smartphone Nutzerdaten, um Phasenänderungen der bipolaren Störung zu erkennen. Das Benutzerverhalten, sowie dessen Änderungen, sollen automatisch gelernt werden, ohne die Zuhilfenahme von manuellen Konfigurationen, Benutzereingaben oder Lernzielen (Targets). Dies ermöglicht den Einsatz des Verfahrens für neue, ungesehene Daten.

Wir stellen einen Veränderungsdetektions-Prozess (Change Detection Process) vor, welcher aus Datensammlung und -verarbeitung, Feature-Auswahl und -Extrahierung, Veränderungsdetektion und Evaluierung besteht. Expertenwissen wird verwendet um krankheitsrelevante Messwerte auszuwählen, wobei unüberwachte Methoden der Feature-Auswahl dazu verwendet werden, das Ergebnis weiter auf benutzerrelevante Features einzuschränken. Die Veränderungsdetektions-Algorithmen *ChangeFinder* und *Bayesian Online Changepoint Detection* werden implementiert und anhand der Zielsetzungen evaluiert.

Ergebnisse werden anhand von drei Datensätzen präsentiert. Hypothetische Daten werden verwendet, um die algorithmische Leistung anhand von verschiedenen Anwendungsfällen zu bewerten. Darunter sind statistische Ausreißer und Änderungspunkte aufgrund von verschiedenen Ursachen und Ausschlagsgrößen. Wir zeigen die Erkennung von einschneidenden Ereignissen in Smartphone Daten von gewöhnlichen Personen, darunter ein Bänderriss und eine Veränderung des Beziehungsstatus. Nutzerdaten von bipolaren Patienten, welche von einer laufenden klinischen Studie kommen, werden verwendet, um Veränderungen zwischen bipolaren Phasen zu erkennen. Wir kommen zu dem Schluss, dass der *Bayesian Online Changepoint Detection* Algorithmus besser für unsere Zielvorgabe geeignet ist. Er lässt Vorwissen über die Domäne einfließen. Dies gibt uns die Möglichkeit, die Ergebnisse weiter zu verbessern und den Algorithmus für das Finden spezieller Änderungstypen, wie Änderungen in bipolaren Phasen, zu konfigurieren. Keine manuelle Parameterauswahl ist notwendig und die Ergebnisse sind vielversprechend. Allerdings sind weitere Auswertungen mit bipolaren Nutzerdaten notwendig.

Schlüsselwörter: Veränderungsdetektion, Bayes-Veränderungsdetektion, Smartphone Nutzerdaten, Bipolare Störung, Überwachung von psychischen Erkrankungen, Zeitreihenanalyse, Multivariate Zeitreihen, Unbeaufsichtigtes Lernen

Contents

1 Introduction

Bipolar disorder is a psychological illness, where patients cycle through three different states, namely depression, mania and the normal state. During these states the patients experience extreme variations in thinking, behavior and mood. The state characteristics are mostly in contrast to each other during depressive- and manic phases, while they often have moderate levels during the normal state. For instance, the mood and energy level of a person tend to decrease during a depressive phase while they are most likely increased during a manic phase and usually have a moderate amplitude during a normal phase of the bipolar disorder. An early detection of behavioral changes is crucial for the patient's health, because the treatment has to be adapted according to his or her current mental state. The earlier the treatment is administered to the patient's state, the more effective it is. If patients overlook early warning sings, they often end up visiting the doctor very late which might lead to severe measures and extended hospitalization. In order to avoid such situations it is very important to train bipolar affected people in closely observing their behavior and identifying early warning signs. This results in significant training effort, which is hard to finance and strongly depends on the patient's compliance and discipline. It often is impractical and in some cases might even be impossible to achieve. Especially for people with bipolar disorder it can be very challenging to objectively perceive their behavior, because awareness and capacity of detailed remembering might be affected by their current mental state [1].

In order to assist bipolar affected people with registering state changes, this thesis aims to investigate the applicability and performance of an automatic changepoint detection to smartphone usage data. The goal is to implement and evaluate a change detection process, that detects state changes in the bipolar disorder. The approach could then be used to detect early warning signs of state changes and react to them, such that countermeasures can be proposed in a timely manner.

The start-up *meemo-tec* develops a mobile application for Android, which is called the *Bip-Up*. It automatically tracks smartphone usage data and provides the user with an overview of his or her recorded activities. The available data contains the categories *physical activity*, with activities such as walking or cycling, *visited locations*, such as home or work, and *application statistics*, such as phone calls or messaging. No user interaction is required and purely objective data is used to capture the user's behavior. Additionally the mood can be entered, such that the user can observe correlations between his or her attitude and lifestyle. The application can be used by ordinary people, but it is developed particularly for people with bipolar disorder. The changepoint detection implemented within the context of this thesis, uses smartphone data, that is collected using the *Bip-Up*.

There are already various applications for mood tracking and digital diaries, among others the *eMoods Bipolar Mood Tracker*. They mostly rely on self reporting by the users though, and don't provide an objective way to track user behavior.

The *Monarca Project* [1] investigated smartphone based recognition of states and state changes in bipolar disorder. They identified the three aspects *social interaction*, *physical motion* and *travel patterns* to be most relevant for the mental disorder and use the four sensing modalities *speech*, *voice*, *acceleration* and *location* to represent these aspects. The research group introduces a system to detect states, and state changes. The state recognition is performed using a naive

Bayes classifier in a supervised manner. The state change recognition is performed using a one-class classification, where the "default" state is learned and data points falling outside of this model are interpreted as changes, which requires a manual threshold selection.

Note, that because of the data diversity between users, a supervised learning approach would require a learning phase of states per user, including clinical visits, in order to obtain targets. This is not realistic in a real world scenario. In the case of state change recognition no targets are required, but in the proposed approach the threshold is selected manually, which again is not realistic in a real word scenario with unseen user data.

The change detection process implemented in the context of this thesis, does not rely on supervised learning and labeled targets in any way. The targets used within this thesis are exclusively required for evaluation purposes but do not influence algorithmic parameters. The goal is, to find approaches which work autonomously, requiring as little manual configuration as possible. Also, we pursue the objective to minimize the detection lag, so the time between the real change and its detection, in order to optimize reaction times to state changes.

1.1 Outline of this Thesis

Chapter 2 contains background information about the bipolar disorder, smartphone data and the reflection of measured smartphone data to bipolar state characteristics. The principle of changepoint detection and its associated terms are introduced, and the benefits of an automated detection of bipolar state changes are discussed. This chapter also contains a short overview of the clinical study, which gives us access to smartphone usage data of bipolar patients collected using the *Bip-Up*.

Chapter 3 presents each step contained in the *Change Detection Process*. It introduces how data is collected and processed in order to receive plausible features. Using feature selection and -extraction, a relevant feature subset is selected or new features are created respectively. Two implemented changepoint detection algorithms, the *ChangeFinder* and the *Bayesian Online Changepoint Detection*, are introduced and evaluated.

Results with three different datasets are presented in chapter 4. Hypothetical data is used to compare the performance of the implemented algorithms and their variations. The applicability to real world data is shown by detecting incisive events in general usage data and by applying the algorithms to study data of bipolar affected people.

Chapter 5 gives concluding remarks, including a comparison of the implemented algorithms and proposed future improvements.

2 Background

This chapter contains important background information, that is associated with this thesis. General facts about the bipolar disorder (2.1) as well as an overview of smartphone sensors, and the data that can be captured by using them (2.2), are provided. We discuss, how smartphone usage data reflects to relevant aspects of bipolar states (2.3). Next, the goal of changepoint detection in general (2.4) as well as the benefits of an automated change detection in bipolar states (2.5) are discussed. Finally, we describe the clinical study (2.6), from which we obtain the datasets of bipolar patients and we state terms (2.7), which are throughout this thesis.

2.1 Bipolar Disorder

Bipolar disorder, formerly called manic depression or manic-depressive illness, is a serious mental disorder that causes extreme variations in thinking, mood, and behavior. Bipolar affected people experience periods of depression and mania. While most patients cycle through both periods regularly, some people experience more depressive than manic episodes and the other way round. In between those periods people experience short or long time spans without any symptoms or with unstable moods. The characteristics of depression and mania include severe mood shifts and changes in thinking, energy- and activity-levels, sleep patterns and behavior.

Symptoms of depressive mood episodes often include:

- persistently low mood, pessimism
- feeling of worthlessness, helplessness, guilt
- low self-esteem
- low energy
- loss of interest in hobbies
- increased need of sleep, problems sleeping
- change of appetite with weight loss or gain
- difficulty concentrating, remembering and decision-making
- thought of suicide

Symptoms of manic mood episodes often include:

- long periods of joy, excitement, or euphoria
- overconfidence
- disconnected, racing thoughts and way of speaking
- increased energy
- little need for sleep
- impulsive, high-risk, pleasure-seeking behavior

People with bipolar disorder often have problems in school, keeping employments and sustaining relationships. Unwise and risky business decisions as well as impulsive actions increasingly leads to financial and social drawbacks among bipolar patients. In very serious, untreated cases affected people might even attempt suicide.

The bipolar disorder can be categorized in several types according to the severity of the symptoms and the period course [2].

Bipolar I describes a variant where the patient suffers from mayor depression and manic periods, which last at least one week.

Bipolar II shows very similar patterns as Bipolar I, but the characteristics are more moderate. Especially the manic phases lack the more severe characteristics, these episodes are then called hypomania.

Rapid-Cycling is a supplemental diagnosis to bipolar disorder, it states that the patient experiences very frequent period changes such that he or she has four or more episodes of major depression, mania or hypomania within a year.

Bipolar disorder is considered a chronic illness that can not be cured, but in most cases it can be controlled with the right treatment. The early detection of period changes is considered crucial in order to adapt the treatment as fast as possible to the current patient's condition. Typically, a combination of medicine, such as mood stabilizer and antidepressant, as well as therapy and lifestyle changes are used for treatment. Also solid social support is considered an important factor influencing the patients health.

Period durations and recurrence rates vary between each bipolar affected person. Episodes can last from weeks to several months, where depressive states generally last longer with an average of 5.2 months while manic and hypomanic states average to a duration of 3.5 months [3]. Statistical factors are similar throughout different diagnostic types, such as additional occurrence of prominent psychotic features. Recurrence rates of depression and mania average to about one episode a year, which makes an average of two episodes per year in total.

According to [4] around 1% of the population is affected by bipolar disorder, while estimates vary from 0.5% to 2%. These discrepancies are due to the perceptive nature of the illness and the patient's desire to conceal past episodes.

2.2 Smartphone Data

Smartphones have a high potential of aggregating data that is associated with the user behavior. The devices are widely accepted by society; today most people use them extensively for everyday tasks and keep them at their side the majority of time. This enables the device to register its user's actions and behavior, not only while the user is actively interacting with the smartphone, but also while the device is in standby mode by recording and analyzing in the background, as long as it is kept in close range. Device sensors, that observe a broad variety of environmental signals, are a prerequisite. Each smartphone model may include different sensor models. The types and qualities of sensors integrated in smartphones depend mostly on the hardware manufacturers, although the operating system requires a minimal set of sensors in order to support the hardware.

The access to this data from within the application level is regulated by the operating system. While some operating systems allow almost full access to the data (like Android OS), others limit the amount of data accessible to third party applications more strictly, which is the case for iOS.

The following list gives an overview of commonly used physical sensor types integrated in smartphones.

The accelerometer is a motion sensor, that measures the acceleration of the device fall, i.e. the orientation and speed along three axis

The gyroscope perceives the objects orientation by measuring the rotational velocity along three axes.

The magnetometer measures magnetic fields and thus provides information about the objects orientation according to cardinal directions.

GPS uses satellite signals to calculate the objects position in the world.

The barometer measures atmospheric pressure.

The proximity sensor uses an infrared LED and IR light detector to measure the distance to the closest object facing its front.

The ambient light sensor measures the brightness of the surrounding light.

The thermometer measures the temperature in- and outside the device.

The humidity sensor measures the humidity.

The pedometer is dedicated to count the number of steps that the user has taken.

Most smartphone operating systems provide data from software sensors, in order to provide additional and more accurate information that complements the data measured directly by physical sensors. Orientation data might be available to applications, which is a combination of the accelerometer, gyroscope and magnetometer. Most mobile operating systems also use more complex data processing in order to supply applications with basic detection providers, such as physical activity event discovery.

Furthermore, mobile devices are capable of collecting huge amounts of data concerning the usage of applications. The smartphone operating system traces every interaction between user and phone. Thus it has information about when and how long applications are used, what actions the user performs, what messages the user writes, as well as audio and video material.

The operating system limits the access from third party applications to prevent data abuse and it grants permissions depending on the scope and user preferences. These regularizations differ for each operating system.

2.2.1 Application Level Data Access in Android OS

Android OS offers a relatively wide range of sensor data and usage information to third party applications. Some physical sensors are categorized to have non-sensitive data and thus don't need explicit access permissions from the user, these include the accelerometer, the proximity- and the ambient light sensor. Sensors and usage data containing more privacy relevant data require so called dangerous permissions, which have to be granted by the user in form of a system dialog at runtime. Location, microphone, phone, activity recognition and SMS are among those dangerous permission groups [5]. Additionally the Android OS provides access to application statistics, which includes events for each user- or background interaction with any application on the phone. In order to receive these usage events the application has to be granted a special permission which has to be activated in the system settings themselves.

2.2.2 Application Level Data Access in iOS

In iOS access to most of the physical sensors and privacy relevant information behaves similarly to the Android equivalent. Non-sensitive data does not require any permission while a list of more privacy relevant information needs explicit permissions granted by the user [6]. In contrast to Android OS there is no possibility to receive system wide usage events from other applications in iOS.

2.3 Reflection of Smartphone Data to Bipolar States

User behavior and lifestyle is reflected by a wide range of data accessible by mobile applications. Physical activity can be approximated by using activity recognition events, provided by the smartphone operating system. Location services provide raw data that can be used to model behavioral patterns, such as daily routines and a rough estimate of the user's occupation. Combined with user input, such as location labeling, this information gets even more valuable. Application statistics grant extensive knowledge about user interactions with applications. When it is combined with information about application groups, this data represents the involvement in certain activities, such as listening to music, social interaction or shopping.

Physical activity corresponds to the energy level and urge to action of a person. A depressed person is likely to stay very close to his or her home, avoid unnecessary physical strain and refrain from actively participating in physical exercises. A person in a manic episode, however, would most likely feel restless and full of energy; he or she would therefore keep moving around and exercise more.

Location data represents daily routines and occupations. A depressed person is likely to sleep longer, stay at home for the majority of time and maybe even stay home from work. Characteristics of manic episodes on the other hand would be to have less need for sleep, move around a lot, visit new places, go out in the evenings and spend only a fraction of the time at home.

Application usage data can represent a multitude of behavioral factors, depending on application groups. A depressed person might not like to listen to music, because there is no interest in it. A manic person might listen to music excessively though, because every song seems great and is experienced with joy. Most depressive people don't participate much in shopping, and would not spend much time in shopping associated applications. People in manic episodes however are known to overly participate in shopping activities and make risky financial decisions. One aspect captured very well by mobile application usage data is social activity. Not only SMS and phone calls are used to communicate with other people, but also a variety of social media and messenger applications. Tracking the time a user spends with phone calls, messages and other digital social interactions allows for a relatively good representation of the overall investment in communication can be made. Social activity is known to be a key characteristic of depressive and manic periods. While a depressive person is likely to avoid social contact and not respond to communication attempts in a timely manner, people in a manic period will most likely behave contrary. A manic person might seek extensive social interaction, with familiar people as well as with new acquaintances, and have a high talkativeness.

All this combined information gives a fair coverage of relevant characteristics for bipolar disorder.

The monarca project [1],[7] investigated the relevance of wearable and smartphone-based sensor data to the symptoms of bipolar disorder and how this data can be used in order to detect states and state changes. Social interaction, physical motion and travel patterns count to the most relevant behavioral aspects according to their medical personnel. In particular they investigated phone call patterns, voice analysis, location- and mobility-patterns and physical motion parameters, all on a daily basis. They used speaker-turn and voice analysis to detect emotions, which are strongly influenced by the mental state. In order to perform speaker-turn analysis the audio material, recorded during phone calls, is examined in matters of speaker turns (where the person in observation is talking), speaker short terms (where the person in observation is talking, but only using short reaction phrases, such as "okay" or "right") and non-speaking segments (which are either pauses or turns of the counterpart). Functionals like duration average, duration standard deviation and the count of these properties, as well as the number of speaker turns per minute and percent of speaking time from the total conversation are used as features. For the voice analysis the audio material is split into same-sized fragments and analyzed in matters of low level descriptors, such as rms (root-mean-square) frame energy and harmonic-to-noise ratio - both giving information about how clearly, loudly and energetic a person is talking. Functionals are then applied to these frame-wise low-level descriptors and used as features. GPS data is used to represent location and mobility patterns, while acceleration data is used to reflect the amount, forcefulness and speed of motion. They also found out, that it can be useful to split the day into frames and analyze the individual features within such a time frame instead of a daily basis, because the user behavior may only shift along time and not change altogether. They used the above features, fused by weights, to learn a default model of the user's behavior. The naive bayes classifier was used to identify a variation from the default model and therefore detect state changes.

2.4 Changepoint Detection

Changepoints are time steps within a data sequence, which indicate the delimitation from one generative process to the next. The sequence of measurements over time between two changepoints is called a segment, it contains data produced by the same underlying model. The behavior, described by the parameters of the underlying model, can change over time due to external events or internal systematic changes. Changepoint Detection [1] is occupied with finding abrupt changes, the changepoints, in a data sequence, but ignoring trends which slowly change the data.

While the off-line changepoint detection can be seen as a segmentation algorithm, that tries to find the best division of the whole data sequence into a fixed or variable amount of segments, the on-line variant tries to identify any changepoint in time. Therefore on-line algorithms take only past, already seen data points into account and aim to identify a change as fast as possible after its occurrence. Here, not only the accuracy of changepoint detection is of importance, but also the number of time steps that passed until the change is detected, which we call the detection lag. Often the accuracy of detected changes and the detection lag are in a trade-off relation. Therefore, the off-line changepoint detection represents a variation which favors the accuracy over the lag by completely neglecting the lag.

Changepoint detection is used in a broad variety of application areas such as medical condition monitoring, climate change detection, speech-, image- and human activity analysis. In medical condition monitoring it can be used for automated real-time observation of physiological parameters such as heart rate, electroencephalogram (EEG), and electrocardiogram (ECG) in order to investigate issues such as sleep problems or epilepsy. Here the time series consists of real valued measurements. Image analysis uses the digital encoding of an image at each time step [8].

2.5 Benefits of Automated Changepoint Detection for Bipolar State Changes

The mental state is currently identified by a clinical rating scales, that are mainly based on self reports. These questionnaires are subjective and they highly depend on the patient's perception of their own behavior. Especially for people with bipolar disorder it can be very hard to answer questions about their behavioral changes truthfully, because awareness and capacity of detailed remembering might be affected by their current mental state.

It is crucial for the patient's health to detect state changes and their warning signs in an early stage, because the pharmacotherapy has to be adapted to the current situation. The earlier the treatment is administered to the patient's state the more effective it is. If patients overlook

[1] In this thesis also referred to as Change Detection

early warning sings, they often end up visiting the doctor very late which might lead to severe measures and extended hospitalization. In order to avoid such situations it is very important to train bipolar affected people in closely observing their behavior and identifying early warning signs. This requires significant training effort, which is hard to finance and strongly depends on the patient's compliance and discipline. It often is impractical and in some cases might even be impossible to achieve [1].

Autonomous and objective behavior records would benefit patients and doctors in order to better identify early warning sings and current mental states. An automated change detection of bipolar states would further help patients to observe their behavior and might signal the necessity to initiate further steps, such as a visit to the doctor. It could by no account replace the care of a doctor, but rather support the patient with self-monitoring and identifying critical changes in his or her state. This allows for fast reaction by the patient and subsequently a fast treatment by the doctor, which is a crucial factor to the effectiveness of the treatment the patient receives.

2.6 Clinical Study

A clinical study regarding bipolar affective disorder and the aid of smartphone data is currently conducted by the Special Outpatient Clinic of the Department of Psychiatry and Psychotherapeutic Medicine (Spezialambulanz der Universitätsklinik für Psychiatrie und Psychotherapeutischer Medizin). This study aims to evaluate the data acquisition via the smartphone application *Bip-Up* regarding sleep-, physical activity- and communication durations as well as the mood. On the other hand it focuses on the evaluation whether the in-software identified behavioral patterns, based on the collected data, sufficiently reflect to the mood and allow for early warning signs of depression and (hypo) manic phases. The acceptance of the application and compliance by the patients is also investigated, in order to evaluate whether the application is applicable for the analyzed patient group. The acquired data of this study is used for data analysis and evaluating the changepoint detection within the context of this thesis.

The study is planned to comprise 24 bipolar affected participants in the patient group and 24 people without any psychological illnesses in the control group. A duration of six months per participant is sought, while in total the study is scheduled to last two years. Participants are instructed to use their smartphones as usual. The application acquires data automatically without requiring any user interaction except for the mood, which is prompted to enter in constant intervals (such as two times a day).

In total four medical visits are scheduled. In the beginning of each participant's attendance, the application is installed on his or her smartphone and the Clinical Global Impression (CGI) is evaluated. The following medical visits are scheduled after one month, after two months and after six months. The data is then transmitted from the participant's smartphone to a local database. At every visit a structured clinical interview is performed by the medical personnel. Also, the participants fill in validated questionnaires about physical activity and daily routines in the last months, and their current psychological condition. This allows for labeling the data to bipolar states and serves as ground truth.

Participants are chosen according to the following prerequisites:

1. Age between 18 and 70 years
2. Knowledge about handling smartphones
3. Patient groups with diagnosed bipolar affected disorder
4. Control groups without psychological illnesses

2.7 Terms

Following key terms and definitions will be used throughout this thesis.

Time series A time series is a sequence of observations with a constant time interval between measurements. It is denoted by $\boldsymbol{X}^{1:t} = \{\boldsymbol{x}_i : i = 1, ..., t\}$, where t is the time and every x_i is a d-dimensional real valued vector.

Changepoint A changepoint is a time index, which partitions a time series into two segments where each segment results from a different underlying stochastic process.

Stationarity In a stationary time series statistical properties, such as mean and variance, are constant over time. Many statistical procedures used for time series analysis require stationarity, that requires pre-processing of stationary real world data.

Seasonality Seasonal trends in time series are variations in the data according to patterns in regular intervals, such as weekly, monthly or quarterly.

Skewness Skewness is an asymmetry measure of a probability distribution of a random variable about its mean.

Kurtosis The kurtosis describes the form of a probability distribution regarding the thickness of its tail.

Jumping mean A changepoint through jumping mean is a change due to an abrupt shift of the model mean.

Jumping variance A changepoint through jumping variance is a change due to an abrupt shift in variance.

Offline algorithms Offline algorithms consider a finite time series and deal with the segmentation of the entire sequence into a predefined or best-fitting amount of segments.

Online algorithms Online algorithms are concerned with the real time identification of change points at the current time step. The algorithms only consider past, already seen data points and are evaluated each time step in order to find the next change point as soon as it occurs.

Detection lag The detection lag terms the amount of time steps passing between the actual changepoint and the time step in which the change is detected.

Hazard function The hazard function describes the instantaneous rate of occurrence of the event (e.g. a changepoint). In other words it is the rate of an event occurring in a specific time step when it did not occur before that time step.

3 Change Detection Process

The change detection process implemented within the scope of this thesis consists of multiple sequential steps as shown in Figure 3.1. In this chapter we introduce each of these steps and discuss the mechanisms as well as the actual implementations in detail.

In the first step (Section 3.1), the data has to be recorded on the smartphone, using the provided operating system APIs. This raw data, collected by the mobile application, is processed such that time spans and events are created from all available sensor inputs. The preprocessed data is saved to the local database and sent to a server which makes the data accessible to the actual change detection process. There the data is again imported and processed, such that a variety of features are constructed from the available sensor data. The feature selection step (Section 3.2) selects the most essential features: according to either one of the implemented feature selection algorithms or expert knowledge. Instead or in addition, features can be extracted (Section 3.3) such that the size of the resulting feature set is reduced, but it covers most of the feature space. The actual changepoint detection (Section 3.4) is based upon the selected and extracted features from the previous step. Two different state of the art change detection algorithms were implemented and evaluated, including several variations with respect to feature representation and weighting. Finally, the results are evaluated (Section 3.5) with respect to our data, in order to compare the effectiveness and suitability of feature sets and algorithms.

We chose Android OS as medium because it provides advantages over iOS with respect to the amount of data made available to mobile applications. The data collection on the mobile devices was not part of this thesis and is described only for completeness's sake. The actual implementation of the change detection process comprises all steps onward the import of the preprocessed data and was accomplished in the python programming language.

Only a very limited amount of labeled data was available during the development process. This data comes from self-monitoring in form of dairy based notes from regular participants and some very limited notes and assessments of bipolar affected people through medical personnel on the other hand. This lack of labeled data and the great variety of user data between different people, does not admit a supervised approach for the change detection. Instead, we only use the labeled data for evaluating the change detection at development time. The algorithm is developed to work on unseen data from unknown users, and does not require any prior data labeling or learning phase.

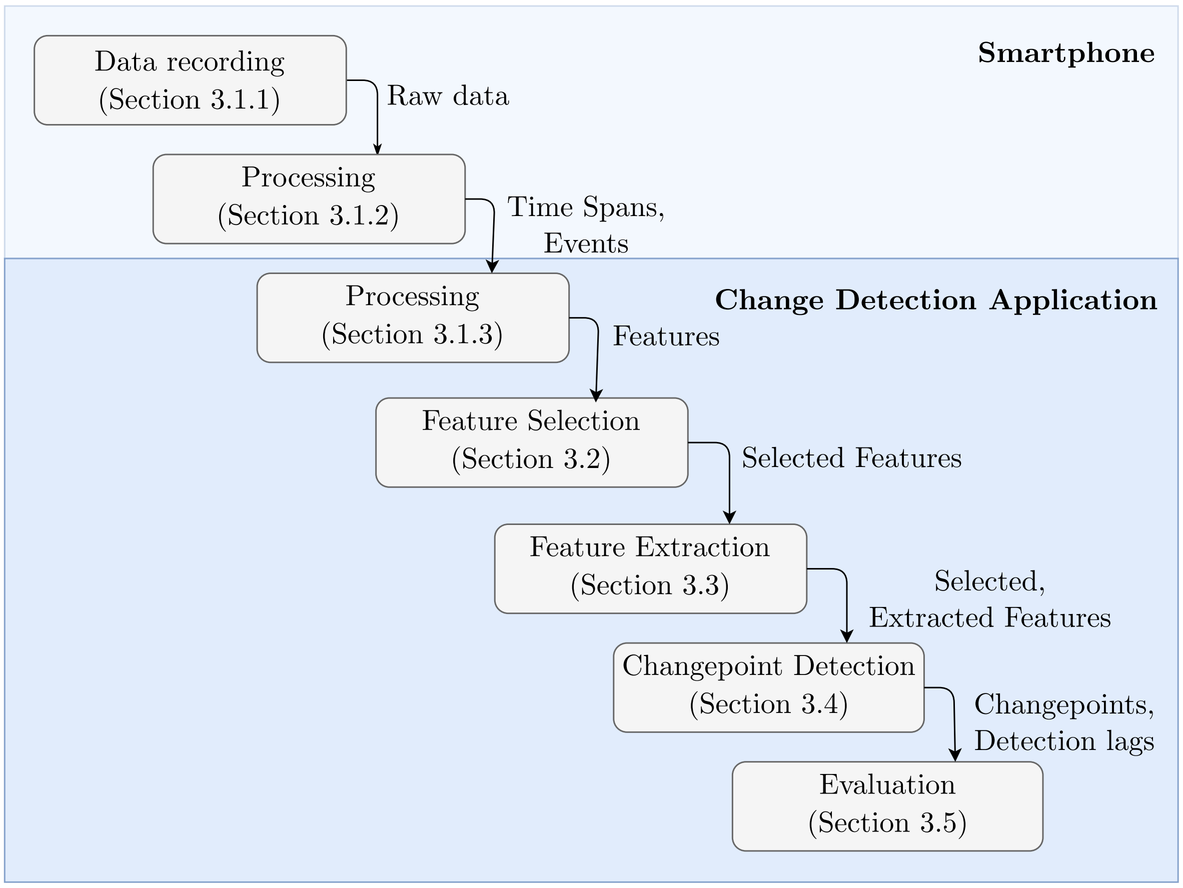

Smartphone
Data recording (Section 3.1.1)
Raw data
Processing (Section 3.1.2)
Time Spans, Events
Change Detection Application
Processing (Section 3.1.3)
Features
Feature Selection (Section 3.2)
Selected Features
Feature Extraction (Section 3.3)
Selected, Extracted Features
Changepoint Detection (Section 3.4)
Changepoints, Detection lags
Evaluation (Section 3.5)

Figure 3.1: Overview of the change detection process.

3.1 Data Recording and Processing

In this section we describe the first steps, required for the change detection process. First, the structure of raw data on Android phones is introduced. Then we describe the data processing within the mobile application as well as the change detection process.

Data recorded by mobile devices, that represents the user behavior, exhibits great diversity between different users. While one person may extensively use certain means of communication, such as phone calls or social media applications, another person might use totally different communication media, such as SMS or messenger applications, or even communicate through the mobile device very limited in the first place. This makes it very hard to compare data between users, which also limits data to learn from, both, in general or related to a certain state. Also the user behavior can strongly vary from one day to the next, as the time spent with certain activities or at certain locations does only depend indirectly and in complex patterns on the particular measurements of past days. For example a person might have spent a lot of time at work the last days and does not go to work at all the next day, which is a perfectly normal behavior.

3.1.1 Raw Data Structure

In order to capture all available data needed to best reflect the user behavior, a variety of raw data is recorded by the Android application.

The physical activity is received through the *ActivityRecognitionClient*, which allows to request activity updates. Once subscribed to receive such events, a handler is called whenever a new activity is detected. The provider wakes up the phone according to the detection interval or specific events received through different sensors, and reads short bursts of sensor data. It is then analyzed by the provider in order to determine if the user is currently on foot (with distinction of walking and running), in a vehicle, on a bicycle, still or tilting. The result, consisting of the detected activity and a confidence indication, is notified to the application handler. The confidence value represents the likelihood of the given type and can have values between zero and one hundred, where larger values represent a higher likelihood of the given type. Physical activities are not mutually exclusive and thus the confidence of all activities summed up does not have to be smaller than 100. Some types have a hierarchical relation, so ON_FOOT could have a very high confidence of 100 and at the same time WALKING could also have a quite high confidence, such as 95. Physical activities require the ACTIVITY_RECOGNITION permission. [9]

Application usage statistics are received through the *UsageStatsManager*. The application can request a list of application usage events that occurred within a certain time window. The application requests, for instance, a list of all events that happened within the last fifteen minutes. Such an event contains details about the respective application, such as the Android Activity and package identifier, as well as the user interaction type, such as MOVE_TO_FOREGROUND, MOVE_TO_BACKGROUND or USER_INTERACTION. In order to use these services the PACKAGE_USAGE_STATS permission is required. [10]

Location data is received via the *FusedLocationProviderClient*. Location updates are requested using the *requestLocationUpdates* method, where the interval of active- and passive location sensing can be configured. The active location sensing rate defines the minimal required update interval, received through active measurements, while the passive sensing rate defines the maximal required update interval, when the actual sensing is initiated through other applications. The device location consists of longitude, latitude, altitude, speed and accuracy, and it is notified to the handler whenever a location update within the configured interval takes place. The accuracy states the radius in meters around the given coordinates, in which the device is located with a probability of 68%. The accuracy estimate only considers the horizontal accuracy and is not concerned with altitude or speed confidence estimates. The use of location services requires the ACCESS_COARSE_LOCATION or ACCESS_FINE_LOCATION permission, depending on the desired accuracy. [11], [12]

3.1.2 Preprocessing on the Smartphone

The outcome of the preprocessing step on the smartphone are time spans and events, that represent the raw data received by the operating system sensor managers.

Physical activity events contain the estimated type and a confidence indication. Whenever the physical activity handler is called with a new detection entry, the application decides about the plausibility of the new activity by means of the confidence indication and constructs time spans for activity types. A received entry is stored in the database if the confidence is large enough and the type is different from the previous one. If such a new, plausible activity type is detected, the last physical activity time span is marked as completed.

Application usage events are used to create time spans of actively used applications. Whenever the event list is retrieved, a timespan is created for every MOVE_TO_FOREGROUND and its corresponding MOVE_TO_BACKGROUND event. The time span entry consists of start time, end time, Android Activity and package name.

Location events are used to create time spans representing the user's residence. Further, using user input for labeling long lasting locations, they are linked to each other according to their purpose of residence, such as *Home*, *Work*, *Sports* or *Shopping*. Whenever a new location event is received, it is linked to an already existing location time span or it is interpreted as a new point on the user's movement path. This is decided by comparing the old location point, including its accuracy radius, with the new location point and its accuracy radius. If the distance between the accuracy circles is longer than a predefined threshold, for instance fifty meters, the point is considered as a new one and the previous time span is marked as completed. If two successive location events are close according to the criteria described above, a new time span is created. The current center is given by the location point with the smallest accuracy radius, i.e, the most precise location point of a certain time span. The start- and end times of the time span are the time of the first- and last received location points for that time span respectively. Only time spans lasting longer than a predefined threshold, such as five minutes, are kept. See Figure 3.2 for a visual demonstration of location time span calculations. These procedures are required, because coordinates in the received location events vary considerably, even if the device remains at the exact same physical location.

Location points are clustered once every day and considered location time spans from the last seven days. Hereby, the *Density-Based Spatial Clustering of Applications with Noise (DBSCAN)* [13] clustering algorithm is applied, it provides clustering results with a variable number of clusters. Every resulting cluster center is then compared to already existing cluster centers from previous runs. If the distance is less than a predefined threshold, it is considered the same cluster and is linked to the cluster label; if the label exists, it is given by the user. Novel clusters are only considered, if the sum of all associated time spans exceeds a threshold, for instance ten hours. The user is prompted to label only unlabeled and relevant clusters the next time he or she resides there.

Sleep time is estimated by using a combination of the above information according to several rules. The calculation of sleep time spans assumes: the labeled home location, a STILL physical activity, no user interaction and additional time constraints. All labeled home location time spans that intersect with STILL physical activity time spans serve as initial estimates. Then, all recorded user interactions, represented through the USER_INTERACTION application usage event, are used to further split the initial sleep time estimate. Additionally, some timely constraints, such as a minimal sleep time of two hours and a time frame between midday of two succeeding days, are applied. Finally, the longest remaining sleep time span is used.

These timely constraints and the labeled home location constraint restrict the target audience, for which the sleep time estimate can work properly, but are necessary in order to increase the estimate quality for typical cases.

The mood is the only user input, that is not automatically measured or estimated on the basis of other information. It is prompted from the user at predefined times, such as twice a day, one time in the morning and another time in the evening. The mood is represented by a number between minus three and plus three, where minus three corresponds to a very low mood (depressed) and plus three is a very elevated mood (manic).

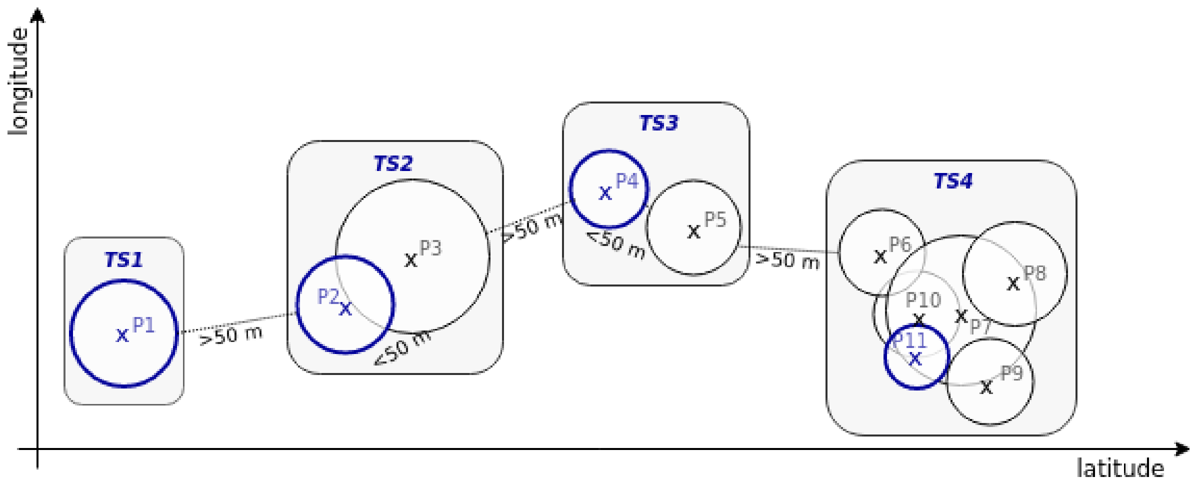


Figure 3.2: Visual demonstration of location time span calculations. TS_i are the time spans, P_i are location points and the circles around each location point indicate the accuracy of the measurement. Circles with a smaller radius indicate a more precise location.

3.1.3 Processing within the Change Detection Process

In the Processing step of the Change Detection Process the input from the smartphone is transformed into features that can be used by the algorithms. A feature contains the duration that the user spent with a certain activity during a certain time frame within a day. Typical time frames are: *morning*, *afternoon* or *whole day*. See Figure 3.3 for an example of feature construction according to input data. In this case the two walking time spans are used to construct a single feature with the interpretation: ”Duration of walking in the morning”.

Example Input

8:23;WALKING;5 min
8:47;WALKING;12 min

Example Output

MORNING;WALKING;17 min

Figure 3.3: Exemplary feature construction.

Additionally, data refinement is applied to the data set in order to remove implausible data samples. First, elements with only zero values are removed. These were potentially caused by an application blackout, such as an empty battery. Second, a duration that exceeds a threshold might also be implausible and is removed for certain activity types. For instance it would be implausible to have a walking activity in the *whole day* time frame, that lasts longer than 22 hours. On the other hand it would be plausible to have a home location in the *whole day* time frame that exceeds 22 hours. Third, normalization is performed for the dataset, so that all features are scaled between zero and one.

Feature Construction in the Change Detection Application

In the Change Detection Application each data category is represented through a data model and its data is loaded into the respective repository as soon as it is accessed by the application. The base class of all repositories contains methods to initiate data structures for holding time series data including their date membership as well as the database session and some abstract methods for accessing data with a common interface. The two repository types, for time span data and for event data, extend that base repository and implement the data access methods. Both repositories return two lists, the values and their corresponding dates. With the *get_data* method the data can be limited, filtered and combined by functionals with the following parameters:

day_time filters the data according to the given time of day. The following partitions are available:

- **all day** : 00:00:00 - 23:59:59
- **morning** : 06:00:00 - 8:59:59
- **before noon** : 09:00:00 - 11:59:59
- **noon** : 12:00:00 - 13:59:59
- **afternoon** : 14:00:00 - 17:59:59
- **evening** : 18:00:00 - 23:59:59
- **night** : 00:00:00 - 05:59:59

arithm_func defines the function that is applied to the duration of each time span per selected day_time. Typically this could be the sum, average, median or standard deviation.

time_span limits the time frame considered for the query.

mtype defines the measurement type, or feature type, used for the query, it depends on the actual model type. For the physical activity repository, and the physical activity model, this would be vehicle, cycling, on_foot, still, unknown, tilting, walking or running.

filter_func filters the data according to its date. The function takes a date as input and returns a boolean whether to include the datum.

Each extended feature repository implements the *process_data* method, which queries the data directly from the database by using the corresponding data model. Models are implemented with the same hierarchy as repositories. The feature model is the base and defines only common properties, such as an ID and owner information used to identify to whom the data belongs. The *TimeSpan* model extends from this base model and implements all methods used for time span handling and the partitioning of data according to day_time filtering. So it contains methods to conveniently work with start- and end times, get the duration within a certain day_time partition or get dates belonging to a time span. The concrete models, such as the location-, activity- or app_stats- models are only concerned with defining specific properties, such as latitude and longitude, and dealing with the measurement type memberships, such as home or work.

Loading helpers implement methods for convenient data loading by specifying feature types (= measurement types), a time window of interest and some properties, such as continuousness versus validity of data, which are important for certain data analysis methods or algorithms.

3.2 Feature Selection

This section contains an overview of feature selection as well as detailed information about selected methods, that were also implemented during this thesis. We discuss *feature selection through expert knowledge* (Section 3.2.1) and two unsupervised mechanisms, the feature selection through *similarity* (Section 3.2.2) and *squared correlation* (Section 3.2.3).

Feature selection is concerned with selecting a relevant subset of the original features. The resulting feature set is intended to contain all features important for the machine learning task, while rejecting redundant or irrelevant features. Feature selection is used to reduce the input dimension and thereby optimize the training time, make it easier to interpret the outcomes and enhance generalization by reducing over-fitting. Many algorithms are sensitive to the use of redundant features, because they lead to a bias in the learned model. Also, irrelevant features can result in learning a pattern that might have developed by coincidence; this is particularly relevant for datasets with a small amount of data.

Feature Selection algorithms typically include a technique for searching new feature subsets, and an evaluation measure that scores these subsets. In the supervised case the evaluation is done by using target class labels. In the unsupervised case, where no labeled targets are available, an objective function is maximized, and represents the performance according to some criterion. Most unsupervised feature selection methods can be classified into two categories. The first category consists of methods that attempt to maximize clustering performance according to some index. The second category contains methods that consider feature dependency and relevance, while various dependence measures are used [14].

3.2.1 Feature Selection Through Expert Knowledge

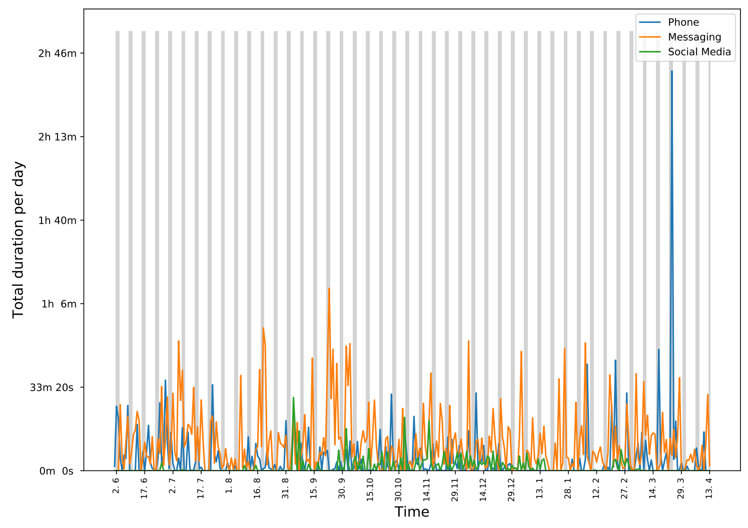
Manual feature selection is reasonable, when sufficient expert knowledge about the domain is available. Experts select, rank or weight individual features according to their relevance for the considered problem. Additionally, it can be used in combination with automatic feature selection, where first the expert selects relevant features and then an algorithm selects a subset of these features in an automated manner.

Social interaction, physical motion and travel patterns are assessed to be the most relevant categories in the case of bipolar state changes. See Section 2.3 for more detailed information on how smartphone data reflects on bipolar states and what the researchers of the *monarca project* [1] identified to be the most relevant data.

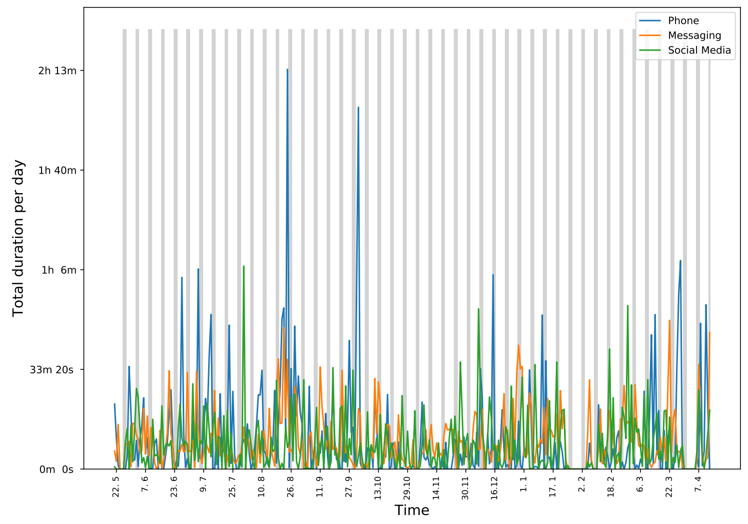
For our implementation, only pre-selected (according to expert knowledge) measurements are observed in order to minimize memory requirements and save battery life. The measurements are processed to a multitude of features as described in Section 3.1.3. In the subsequent step the final features are selected by automated feature selection mechanisms and are used by the change detection algorithm. This automated feature selection helps to reduce the redundancy of relevant features and adjust to the user specific relevance of them.

Figure 3.4 shows the Phone, Messaging and Social Media features of two distinct test subjects. Note, how different the measured data of the two users is.

See Figure 3.5 for a visual display of the correlation between various features of the same subject, that are generally assumed to be relevant.



(a) Subject 1



(b) Subject 2

Figure 3.4: Comparison of Phone, Messaging and Social Media usage data from two different subjects.

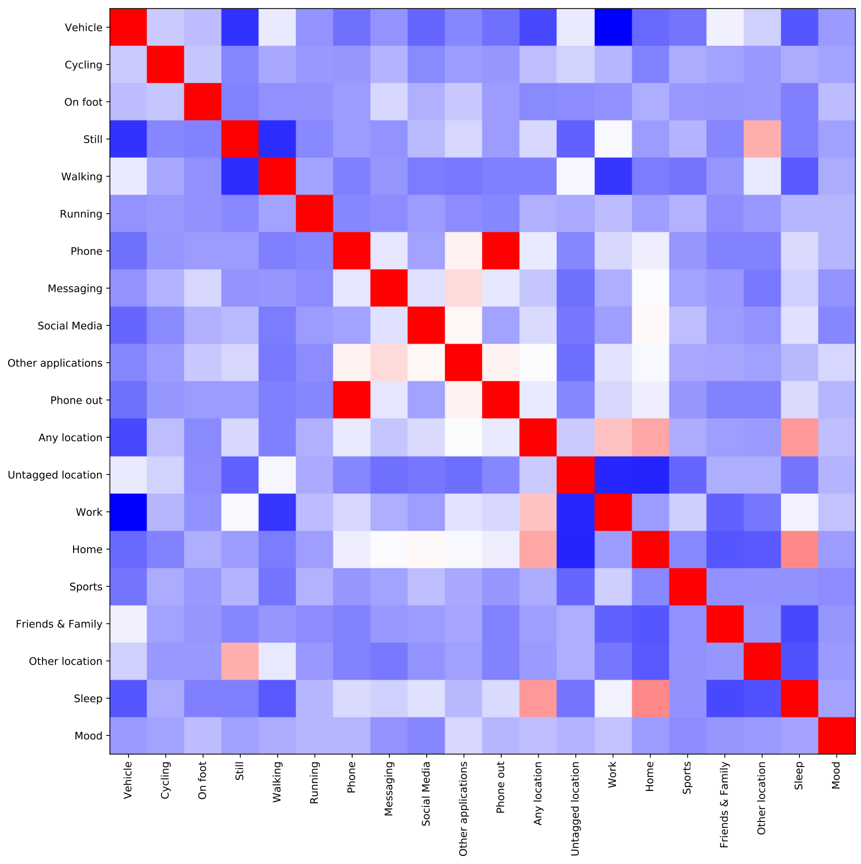


Figure 3.5: Visual display of the correlation between various features. Red coloring indicates a high positive correlation, blue coloring indicates a high negative correlation and white means, that the data is uncorrelated.

3.2.2 Feature Selection Through Feature Similarity

The authors of [14] propose an algorithm, which is based on measuring similarity between features and removing redundancy within them. This is especially suitable for high-dimensional datasets with many samples, because of its computational efficiency. Further, its generic nature makes it applicable for a wide range of real world data sets. The method partitions the original features into clusters of similar features and selects one feature for each cluster. Feature similarity is measured by the *maximum information compression index* [14] (described below). Both, the clustering algorithm and the similarity measure, pursue the goal of minimizing the information loss while minimizing the redundancy in the selected feature subset.

Similarity Measures

The *correlation coefficient* measures the similarity between two one-dimensional random variables $\boldsymbol{x}$ and $\boldsymbol{y}$, it is given by:

$$\rho(\boldsymbol{x},\boldsymbol{y}) = \frac{cov(\boldsymbol{x},\boldsymbol{y})}{\sqrt{var(\boldsymbol{x})var(\boldsymbol{y})}}, \tag{3.1}$$

where $var(\boldsymbol{x})$ is the variance of the random variable $\boldsymbol{x}$ and $cov(\boldsymbol{x},\boldsymbol{y})$ is the covariance between the two variables $\boldsymbol{x}$ and $\boldsymbol{y}$. If $\boldsymbol{x}$ and $\boldsymbol{y}$ are linearly dependent, then $\rho(\boldsymbol{x},\boldsymbol{y}) = \pm 1$, if they are uncorrelated, then $\rho(\boldsymbol{x},\boldsymbol{y}) = 0$. Therefore $1 - |\rho(\boldsymbol{x},\boldsymbol{y})|$ can be used as a similarity measure with $0 \leq 1 - |\rho(\boldsymbol{x},\boldsymbol{y})| \leq 1$. This measure is symmetric, and invariant to scaling and translations of the random variables.

The *maximal information compression index* $\lambda_2(\boldsymbol{x},\boldsymbol{y})$ is introduced in [14], which is equivalent to the smallest eigenvalue of the covariance matrix of the univariate random variables $\boldsymbol{x}$ and $\boldsymbol{y}$. It is calculated by:

$$\lambda_2(\boldsymbol{x},\boldsymbol{y}) = \frac{var(\boldsymbol{x}) + var(\boldsymbol{y})}{2} - \sqrt{\frac{(var(\boldsymbol{x}) + var(\boldsymbol{y}))^2}{4} - var(\boldsymbol{x})var(\boldsymbol{y}) + cov(\boldsymbol{x},\boldsymbol{y})^2}. \tag{3.2}$$

The value of λ_2 is the eigenvalue of the second principal component of the feature pair $(\boldsymbol{x},\boldsymbol{y})$. See Section 3.3.1 for more information about Principal Component Analysis and the interpretation of the principal components. The following properties for λ_2 hold:

1. $0 \leq \lambda_2(\boldsymbol{x},\boldsymbol{y}) \leq 0.5(var(\boldsymbol{x}) + var(\boldsymbol{y}))$
2. $\lambda_2(\boldsymbol{x},\boldsymbol{y}) = 0$ iff $\boldsymbol{x}$ and $\boldsymbol{y}$ are linearly dependent
3. $\lambda_2(\boldsymbol{x},\boldsymbol{y}) = \lambda_2(\boldsymbol{y},\boldsymbol{x})$ (symmetric)
4. Sensitive to scaling
5. Invariant to translation

Feature Selection Algorithm

The feature selection method consists of the following steps: First, the original features are clustered into a number of similar subsets. This is done using the *k-nearest neighbors (k-NN)* principle and a similarity measure (e.g. λ_2). The distances $d(\boldsymbol{f}_i, \boldsymbol{f}_j)$ are calculated for every feature pair. We denote $\boldsymbol{R}_i = \{d(\boldsymbol{f}_i, \boldsymbol{f}_j) : i \neq j\}$ as the ordered set of distances from $\boldsymbol{f}_i$ to its neighbors. The distance to the k^{th} nearest neighbor for each feature $\boldsymbol{f}_i$ is denoted as r_i^k, it is the k^{th} entry of $\boldsymbol{R}_i$. The distance to the k^{th} nearest neighbor measures, how dense the cluster of k features around $\boldsymbol{f}_i$ is. The minimal distance to the k^{th} nearest neighbor, which is the minimal

r_i^k, is denoted as $r_i^{k\prime}$. The feature corresponding to this minimal distance is $\boldsymbol{f}_i'$. We "select" $\boldsymbol{f}_i'$ and remove all k nearest neighbors of the selected feature.

These steps are repeated; in every iteration the cluster size k is decremented by one. Only, if the minimal distance $r_i^{k\prime}$ is smaller than the initial minimal distance ϵ (of the first iteration), the feature is selected and its k nearest neighbors are removed. Otherwise k is further reduced. The cluster size k is initialized with some value less than the number of dimensions d of the original feature set $\boldsymbol{O}$.

See Algorithm 3.1 for a step by step description in pseudocode of the feature selection algorithm and Listing 3.2 for implementation details.

```
1   Choose initial value of $k \leq d-1$
2   $\boldsymbol{R} = \boldsymbol{O}$
3   $\epsilon = \infty$
4   $r_i^{k\prime} = \infty$
5
6   if $k >$ cardinality($\boldsymbol{R}$) − 1:
7     $k =$ cardinality($\boldsymbol{R}$) − 1
8   if $k = 1$:
9     stop
10  For each feature $\boldsymbol{f}_i$ in $\boldsymbol{R}$:
11    compute $r_i^k$
12  Find feature $\boldsymbol{f}_i'$ for which $r_i^k$ is minimal
13  if $r_i^{k\prime} > \epsilon$:
14    $k = k - 1$
15    go to line 6
16  "select" $\boldsymbol{f}_i'$
17  remove k nearest features of $\boldsymbol{f}_i'$
18  if $\epsilon = \infty$:
19    $\epsilon = r_i^{k\prime}$
20  go to line 6
```

Algorithm 3.1: Algorithm "Unsupervised feature selection using feature similarity".

```python
def calculate_knn_distances(data, i, k, dissimilarity_fun):
  feature = data[:, i]
  distances = []
  for j in range(data.shape[1]):
    if i != j:
      distance = dissimilarity_fun(feature, data[:, j])
      distances.append((j, distance))
  distances = sorted(distances, key=lambda tup: tup[1])
  knn = []
  for i in range(k):
    knn.append(distances[i][0])
  return distances[k - 1][1], knn


def feature_selection_using_feature_similarity(data, feature_names,
        dissimilarity_fun):
  # Initialize
  # Choose some k <= D-1
  k = max(1, math.ceil(data.shape[1] / 3))
  r = data.copy()
  feature_names = feature_names.copy()
  e = sys.maxsize

  # Stop and return selected features when k == 0
  while k > 0:
    # Find feature with min r_ik (minimal information loss when removing k-NNs)
    knn_distances = []
    for i in range(r.shape[1]):
      r_ik, knn_indices = calculate_knn_distances(r, i, k,
            dissimilarity_fun=dissimilarity_fun)
      knn_distances.append((i, r_ik, knn_indices))
    sorted_dist_list = sorted(knn_distances, key=lambda tup: tup[1])

    r_min = sorted_dist_list[0][1]
    min_r_knn_indices = sorted_dist_list[0][2]

    # set e in first iteration
    if e == sys.maxsize:
      e = r_min

    if r_min > e:
      k -= 1
      # Attention:
      # In paper k == 1:
      #   break
      # --> 1-nn clustering would never be performed
      # --> but then two equal features would not be reduced
      if k == 0:
        break
      continue

    # Remove k nearest neighbors of feature with min r_ik
    min_r_knn_indices = sorted(min_r_knn_indices, reverse=True)
    for i in min_r_knn_indices:
      r = np.delete(r, i, 1)
      feature_names = np.delete(feature_names, i)

    # Correct k if it is larger than D-1
    if k > r.shape[1]-1:
      k = r.shape[1]-1

  return r, feature_names
```

Listing 3.2: Implementation of the feature selection algorithm using feature similarity.

3.2.3 Feature Selection Through Squared Correlation

Another way to select features [15] uses the *squared correlation coefficient*. The proposed algorithm selects and ranks features by successively selecting new features into the feature subset $\boldsymbol{S}^m$. In each iteration m one of the remaining (not yet selected) features is added. This feature is chosen using the squared correlation coefficient, such that it most contributes to the representation of the remaining features. This means, that after each iteration of the algorithm, the new feature subset $\boldsymbol{S}^m$ contains the most representative features. Note, that the order in which features are added to $\boldsymbol{S}^m$ indicate the ranking of the feature importance.

The *squared correlation coefficient* between two random variables $\boldsymbol{x}$ and $\boldsymbol{y}$ with N realizations is given by:

$$sc(\boldsymbol{x},\boldsymbol{y}) = \frac{(\boldsymbol{x}^T\boldsymbol{y})^2}{(\boldsymbol{x}^T\boldsymbol{x})(\boldsymbol{y}^T\boldsymbol{y})} = \frac{(\sum_{i=1}^N x_i y_i)^2}{\sum_{i=1}^N x_i^2 \sum_{i=1}^N y_i^2} \tag{3.3}$$

In order to find the most significant feature, the squared-correlation coefficient is calculated for every feature pair $sc(\boldsymbol{f}_i,\boldsymbol{f}_j)$. Let c be a $n \times n$ matrix of all squared correlation coefficients, with

$$c_{i,j} = sc(\boldsymbol{f}_i,\boldsymbol{f}_j) : i,j = 1,...,n. \tag{3.4}$$

The capability of a feature $\boldsymbol{f}_i$ to represent all other n features $\boldsymbol{f}_j$ is given by:

$$\bar{c}_j = \frac{1}{n}\sum_{i=1}^{n} c_{i,j} \tag{3.5}$$

The feature $\boldsymbol{s}_1$, that best represents all the others is then selected as $\boldsymbol{s}_1 = \boldsymbol{f}_{l_1}$, where

$$l_1 = \underset{1\le j\le n}{\arg\max}\{\bar{c}_j\}. \tag{3.6}$$

In each step m, a new feature $\boldsymbol{s}_m$ is selected and added to the subset of the previous step $\boldsymbol{S}^{m-1}$, such that the new subset $\boldsymbol{S}^m$ is the most representative compared with any other subsets formed by adding a candidate feature to $\boldsymbol{S}^{m-1}$. Thus, in the first step $m = 1$, the first selected and most significant feature $\boldsymbol{s}_1$ explains the variation in the overall features with the highest percentage. In each successive step $m = 2, ..., n$ each candidate feature $\boldsymbol{u}_j \in \boldsymbol{S} - \boldsymbol{S}^{m-1}$ (each feature, that is not yet selected) is transformed into its orthogonal representation using the orthogonalized selected features in $\boldsymbol{S}^{m-1}$, while the first associated orthogonal feature is chosen as $\boldsymbol{q}_1 = \boldsymbol{s}_1$. The transformation of the candidate features $\boldsymbol{u}_j \in \boldsymbol{S} - \boldsymbol{S}^{m-1}$ is performed by:

$$\boldsymbol{q}_j^{(m)} = \boldsymbol{u}_j - \frac{\boldsymbol{u}_j^T \boldsymbol{q}_1}{\boldsymbol{q}_1^T \boldsymbol{q}_1} \boldsymbol{q}_1 - \cdots - \frac{\boldsymbol{u}_j^T \boldsymbol{q}_{m-1}}{\boldsymbol{q}_{m-1}^T \boldsymbol{q}_{m-1}} \boldsymbol{q}_{m-1} \tag{3.7}$$

The m^{th} most significant feature $\boldsymbol{s}_m$ is then calculated in the same manner as in the first iteration, but the squared-correlation coefficient is now computed between $\boldsymbol{f}_i$ and $\boldsymbol{q}_j^{(m)}$.

$$c_{i,j} = sc(\boldsymbol{f}_i, \boldsymbol{q}_j), i = 1, ..., n; j = 1, ..., n - m - 1 \tag{3.8}$$

The m^{th} most significant feature is then chosen by $\boldsymbol{s}_m = \boldsymbol{f}_{l_m}$, where l_m is calculated according to Equation 3.6. The associated orthogonal vector is chosen as $\boldsymbol{q}_m = \boldsymbol{q}_{l_m}^{(m)}$. This process is repeated until the selected subset can explain the variation in the overall features with a sufficiently high percentage TH. The performance is measured by the *error reduction ratio (ERR)*. The k^{th} ERR states how much of the overall reconstruction error can be reduced by including $\boldsymbol{s}_k$ to the selected features $\boldsymbol{S}^{k-1}$. It is given by:

$$ERR(k) = \frac{1}{n} \sum_{j=1}^{n} sc(\boldsymbol{f}_j, \boldsymbol{q}_k) = \frac{1}{n} \sum_{j=1}^{n} \frac{(\boldsymbol{f}_j^T \boldsymbol{q}_k)^2}{(\boldsymbol{f}_j^T \boldsymbol{f}_j)(\boldsymbol{q}_k^T \boldsymbol{q}_k)} \tag{3.9}$$

The *sum of error reduction ratio (SERR)* in step m states the percentage of reconstruction error, that can be reduced by selecting the subset $\boldsymbol{S}^m$.

$$SERR(m) = \sum_{k=1}^{m} ERR(k) \tag{3.10}$$

See Algorithm 3.3 for a step by step description in pseudocode and see Listing 3.4 for implementation details.

```
$m = 1$
For every feature pair $(\boldsymbol{f}_i, \boldsymbol{f}_j)$:
  Calculate the squared-correlation coefficient $c_{i,j} = sc(\boldsymbol{f}_i, \boldsymbol{f}_j)$
For every feature $\boldsymbol{f}_i$:
  Calculate the mean squared-correlation coefficient $\bar{c}_j = \frac{1}{n}\sum_{i=1}^{n} c_{i,j}$
Calculate $l_1 = \arg\max_{1 \le j \le n}\{\bar{c}_j\}$
Choose first significant feature $\boldsymbol{s}_1 = \boldsymbol{f}_{l_1}$
Determine orthogonalized first significant feature $\boldsymbol{q}_1 = \boldsymbol{s}_1$
For every selected feature $\boldsymbol{s}_k \in \boldsymbol{S}_m$:
  Calculate $ERR(k) = \frac{1}{n}\sum_{j=1}^{n} \frac{(\boldsymbol{f}_j^T \boldsymbol{q}_k)^2}{(\boldsymbol{f}_j^T \boldsymbol{f}_j)(\boldsymbol{q}_k^T \boldsymbol{q}_k)}$
Calculate $SERR(m) = \sum_{k=1}^{m} ERR(k)$
If $SERR(m) > TH$:
  stop
$m = m + 1$
Determine candidate features $\boldsymbol{S}_\alpha = \boldsymbol{S} - \boldsymbol{S}_{m-1}$
For each candidate feature $\boldsymbol{\alpha}$ in $\boldsymbol{S}_\alpha$:
```

```
17    Ortogonalize $\boldsymbol{\alpha}_j$ by $\boldsymbol{q}_j^{(m)} = \boldsymbol{\alpha}_j - \frac{\boldsymbol{\alpha}_j^T \boldsymbol{q}_1}{\boldsymbol{q}_1^T \boldsymbol{q}_1}\boldsymbol{q}_1 - \cdots - \frac{\boldsymbol{\alpha}_j^T \boldsymbol{q}_{m-1}}{\boldsymbol{q}_{m-1}^T \boldsymbol{q}_{m-1}}\boldsymbol{q}_{m-1}$
18  For every candidate- and original feature pair $(\boldsymbol{f}_i, \boldsymbol{q}_j)$:
19    Calculate the squared-correlation coefficient $c_{i,j} = sc(\boldsymbol{f}_i, \boldsymbol{q}_j)$
20  For every candidate feature:
21    Calculate the mean squared-correlation coefficient $\bar{c}_j = \frac{1}{n}\sum_{i=1}^{n} c_{i,j}$
22  Calculate $l_m = \arg\max_{1 \le j \le n}\{\bar{c}_j\}$
23  Choose most significant candidate feature $\boldsymbol{s}_m = \boldsymbol{f}_{l_m}$
24  Go to line 9
```

Algorithm 3.2: Algorithm "Feature Subset Selection and Ranking for Data Dimensionality Reduction".

```python
def err(data, s_m, q):
  n = data.shape[1]
  m = s_m.shape[1]

  err_k = np.zeros(n)
  err = np.zeros(m)
  for k in range(m):
    for j in range(n):
      err_k[j] = math.pow(data[:, j].transpose().dot(q[:, k]), 2) / (data[:, j].transpose().dot(data[:, j]) * q[:, k].transpose().dot(q[:, k]))
    err[k] = sum(err_k)/n
  return sum(err)


def squared_correlation(x, y):
  sc = math.pow(x.transpose().dot(y), 2) / (x.transpose().dot(x) * y.transpose().dot(y))
  if math.isnan(sc):
    sc = 0
  return sc


def orthogonalize(alpha_j, q):
  q_j = alpha_j.copy()
  for q_i in q.T:
    q_j -= (alpha_j.transpose().dot(q_i)/q_i.transpose().dot(q_i)) * q_i
  return q_j


def feature_selection_using_squared_correlation(data, feature_names, threshold = 0.9):
  feature_names = feature_names.copy()
  num_cols = data.shape[1]
  C = np.zeros((num_cols, num_cols))
  C_hat = np.zeros(num_cols)
  q = np.zeros((len(data), 1))
  s_m = np.zeros((len(data), 1))
  alpha = data.copy()

  for i in range(num_cols):
    for j in range(num_cols):
      C[i,j] = squared_correlation(data[:, i], data[:, j])
    C_hat[i] = np.average(C[i,:])

  l1 =np.argmax(C_hat)
  s_m[:,0] = data[:, l1]
  q[:,0] = data[:, l1]
  alpha = np.delete(alpha, l1, 1)

  selected_feature_names = [feature_names[l1]]
  feature_names = np.delete(feature_names, l1)

  m = 1
  while err(data, s_m, q) < threshold:
    num_alpha = alpha.shape[1]
    C = np.zeros((num_cols, num_alpha))
    C_hat = np.zeros(num_alpha)
    for j in range(num_alpha):
      q_j = orthogonalize(alpha_j=alpha[:,j], q=q)
      for i in range(num_cols):
        C[i,j] = squared_correlation(data[:, i], q_j)
```

```
            C_hat[j] = np.average(C[:,j])
        l1 = np.argmax(C_hat)
        m += 1
        s_m = np.column_stack((s_m, alpha[:,l1]))
        q = np.column_stack((q, orthogonalize(alpha_j=alpha[:,l1], q=q)))
        alpha = np.delete(alpha, l1, 1)

        selected_feature_names.append(feature_names[l1])
        feature_names = np.delete(feature_names, l1)

    return s_m, selected_feature_names
```

Listing 3.4: Implementation of the feature selection algorithm using squared-correlation coefficients.

3.2.4 Discussion

It is advisable to combine different feature selection mechanisms, in order to obtain good features for smartphone usage data. We used expert knowledge to find a rather extensive set of features, or measurement signals, that represent all aspects of interest. The mobile application only collects relevant data, so that memory requirements are kept at a minimum and battery life is optimized. In the case of smartphone usage data, we cannot exclusively rely on expert knowledge to select meaningful features, because every user has a very distinctive behavior. We need feature selection mechanisms, that select relevant features for each user individually on the basis of previously unseen data. Therefore, we apply one of the automatic mechanisms on top of the pre-selected features.

The two introduced algorithms from Sections 3.2.2 and 3.2.3 both require a rather large set of original features in order to yield a good choice of selected features. The algorithm based on the maximum information compression index (3.2.2) is simple and easy to interpret. The initial cluster size k is selected manually; it indirectly specifies how many features are selected and how many are removed. Note, that there is no measurement of how well the subset represents the original features; the stopping criterion depends on the error threshold which, in turn, depends on the initial value of k. If, for instance, all original features are relevant, this algorithm would still remove some of them.

The algorithm based on the squared correlation coefficient (3.2.3) selects features, until the selected subset exceeds a pre-selected percentage of accountability for the original features. If, for instance, all features were relevant, this algorithm might select all of them. However, on the considered data this algorithm does not perform particularly well.

The feature selection through feature similarity (3.2.2) seems to be a valid choice for our data, although it requires a large set of original features in order to perform well.

In the results in Chapter 4 manual feature selection is performed, because it leads to better results. Features are picked by hand, according to prior knowledge about the targets and a visual analysis of the data signal. Automated feature selection is still essential for the application on unseen data.

3.3 Feature Extraction

This section contains a short overview of feature extraction in general and the method *Principal Component Analysis*.

Feature extraction is, in contrast to feature selection, exclusively concerned with dimensionality reduction by creating new features. The original features are extracted and combined into another feature set, such that the resulting set contains less features but the loss of information is minimal. In this case, it is hard to interpret the newly obtained features, because they do not correspond to original features but are instead a combination of all of them.

3.3.1 Principal Component Analysis

Principal Component Analysis (PCA) is one of the most well known feature extraction techniques in statistical data analysis and machine learning [16]. It converts a set of possibly correlated values into a set of linearly independent values, which are called the principal components. The number of principal components is determined by the minimum of the number of features or the number of samples minus one. Principal components are ranked according to their importance to represent the data. The first principal component accounts for as much of the variability in the data as possible, and each succeeding component accounts for as much of the remaining variability as possible. See Figure 3.6 for a visual representation of the PCA. The resulting features are calculated by projecting the original data to the principal components. PCA is mainly used to perform dimensionality reduction, which can be done by selecting the $k \leq n$ first principal components obtained by a $f \times s$ data set, where s is the number of samples, f is the number of features and $n = min(f, s - 1)$.

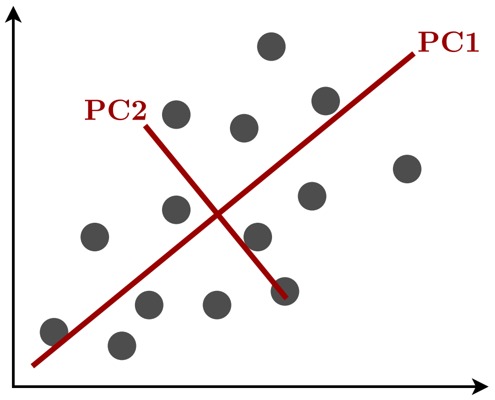


Figure 3.6: Visual representation of PCA.

3.4 Changepoint Detection

This section is concerned with changepoint detection algorithms. First, we address algorithm independent issues. Second, we summarize the two changepoint detection algorithms *ChangeFinder* (Section 3.4.1) and *Bayesian Online Changepoint Detection* (Section 3.4.2). Each algorithm is first introduced in a theoretical manner, as reported by their authors, and further explained according to their implementation for this thesis. Third, we discuss different statistical models, and their characteristics, for learning the observed data (Section 3.4.3). Our smartphone usage data is analyzed according to these characteristics and we introduce some more decision criteria for choosing a suitable model.

Both implemented algorithms, the ChangeFinder (CF) and the Bayesian Online Changepoint Detection (BOCPD), learn the signal distribution of the current segment and try to infer the change likelihood at a past or current time step. Independent of the used algorithm the problem of changepoint detection in multivariate smartphone data can be approached using different methods of feature fusion. In total there are around 175 features available consisting of eight activity types, six usage statistic types, nine location types, mood and sleep, each within one of the seven day slices, such as *all day* or *in the morning*. The input data contains any number of features, selected through feature selection mechanisms described in Section 3.2, per day, which makes the data a univariate (one feature) or multivariate (multiple features) time series. In order to represent the user behavior well, there is obviously more than one feature needed. This raises the question of whether to use one model representing all features at once or multiple models representing each of the features individually.

In the case of one model learning a multivariate signal there would be one change detection algorithm, which processes one feature vector per day. The multivariate variation not only considers the distribution of each feature, but additionally captures the relation between features. So, for instance, if two features are normally distributed according to some parameters and at some point the correlation between them changes, but the parameters of each distribution stay the same, a multivariate model could detect a change whereas two univariate models for each feature could not. In order to learn a multivariate model more data samples are required than when learning two separate univariate models, because of the additional complexity it contains.

In the case of multiple univariate models learning the underlying distribution of each feature, the change likelihood of each feature is combined in some way in order to determine the overall change likelihood and decide whether a changepoint occurred. The results of the individual change detections per feature can be combined using a logical AND or a logical OR operation by means of detected changes (true or false valued), or more meaningful by a weighted combination of the individual change likelihoods. The assignment of weights is again another optimization problem. Within this thesis considered strategies for choosing weights include equal weighting of all features, weights depending on expert knowledge or according to data availability.

Depending on the used model distribution there is a certain preliminary lead time required in order to give an estimate about the underlying process that generated the observed signal. Each detector takes *lead_t*, the lead time, as a parameter. Before the lead time elapses, the model parameters are updated but no change can be detected. As soon as the lead time is expired, the algorithms start with the changepoint detection and update the models in every step. As soon as a changepoint is detected the model parameters are reset and the lead time countdown starts again, in order to learn a new, unbiased model of the underlying process of the new segment.

The *OnlineSimulator* class is used in the change detection phase to reproduce the knowledge scenario at the current time step, thus the to the algorithm currently available data. It takes a detector object, the data set and optionally the dates for plotting purposes. The detector contains the actual algorithm. Methods to update the available data and to extract the detected changepoints are accessible via an interface. The *OnlineSimulator* iterates through the data signal and calls the appropriate detector methods to simulate the data signal at each time step. Additionally it tracks detector properties which are updated at every time step, they are used to display the results after the run has finished. The *ChangeDetector* is used as a base class for all implemented change detection algorithms. It holds common variables, such as detected changepoints and the time steps in which the changepoints were detected, which gives the detection lag. The abstract method *next(x)* is called at every time step to provide the current input datum to the algorithm and perform the detection calculations. *Check_for_change_point()* is called at every time step to do the actual verification whether a changepoint occurred and *plot(data, dates)* is used to plot detector specific properties in the end.

3.4.1 Change Finder

An algorithm, that reduces the problem of changepoint detection to outlier detection is introduced in [17], where the authors describe a framework that is used to address both problems in the same manner. It is a real-time method for detecting significant changepoints, which at the same time filters single outliers. This is achieved by applying a two-stage algorithm, called *ChangeFinder*.

It first calculates outlier scores based on the deviation of the new observation to the learned model. Then it averages these outlier indicators over a fix-sized window. The first sage is repeated with the averaged scores as a basis, such that a change score is the result. The framework is designed to fit one model in each stage to the current signal value and its averaged outlier score respectively. This is a cost efficient approach compared with similar concepts that try to find the most probable changepoint at each time step by best fitting each adjacent segment to a stochastic process.

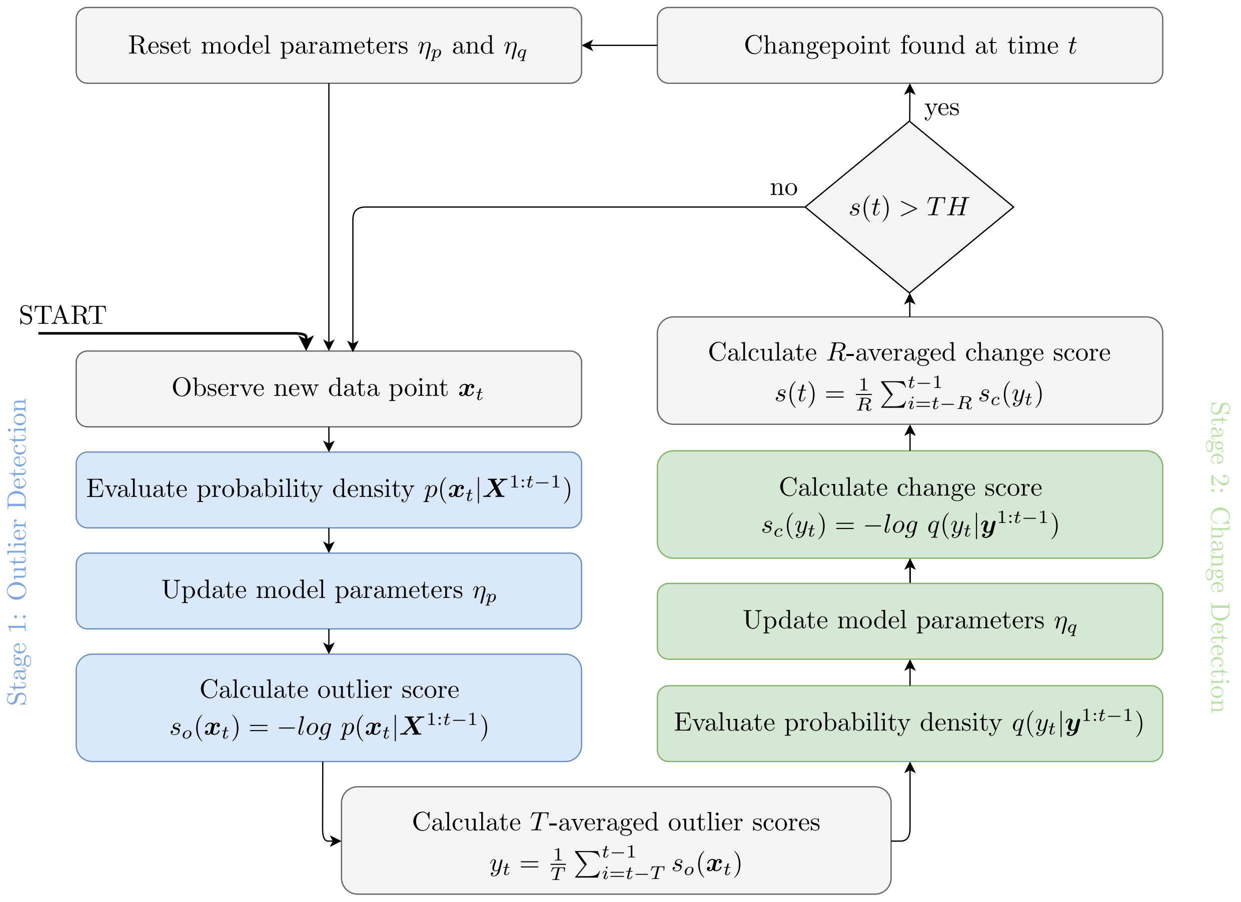

Reset model parameters η_p and η_q
Changepoint found at time t
yes
no
$s(t) > TH$
START
Observe new data point $\boldsymbol{x}_t$
Calculate R-averaged change score $s(t) = \frac{1}{R}\sum_{i=t-R}^{t-1} s_c(y_t)$
Evaluate probability density $p(\boldsymbol{x}_t|\boldsymbol{X}^{1:t-1})$
Calculate change score $s_c(y_t) = -log\ q(y_t|\boldsymbol{y}^{1:t-1})$
Update model parameters η_p
Update model parameters η_q
Calculate outlier score $s_o(\boldsymbol{x}_t) = -log\ p(\boldsymbol{x}_t|\boldsymbol{X}^{1:t-1})$
Evaluate probability density $q(y_t|\boldsymbol{y}^{1:t-1})$
Calculate T-averaged outlier scores $y_t = \frac{1}{T}\sum_{i=t-T}^{t-1} s_o(\boldsymbol{x}_t)$
Stage 1: Outlier Detection
Stage 2: Change Detection

Figure 3.7: Flow chart of the ChangeFinder algorithm.

See Figure 3.7 for a stepwise representation of ChangeFinder. In the first stage the outlier score is calculated. At each time step the conditional probability density function of $\boldsymbol{x}_t$ given $\boldsymbol{X}^{1:t-1} = \boldsymbol{x}_1, ..., \boldsymbol{x}_{t-1}$, denoted as $p(\boldsymbol{x}_t|\boldsymbol{X}^{1:t-1})$, is evaluated. It gives the probability density of the currently observed data vector to be generated by the underlying process, estimated using the already seen data sequence. The stochastic process is learned incrementally at each time step by updating the process of the last step by the currently observed data point. There are many ways to learn this process (e.g. with a model based on the normal distribution), the suitability highly depends on the data though. The conditional probability density for the currently observed data point is used to calculate a loss score, which expresses the prediction loss for $\boldsymbol{x}_t$ relative to the learned probability density function $p(\boldsymbol{x}_t|\boldsymbol{X}^{1:t-1})$. A logarithmic loss function can be used to calculate this score; it is defined as

$$s_o(\boldsymbol{x}_t) = -log\ p(\boldsymbol{x}_t|\boldsymbol{X}^{1:t-1}). \tag{3.11}$$

A higher loss score indicates that the observed data point is an outlier with a higher probability.

The sequence of outlier scores from the first stage is the basis for the second one. We slide a window with a constant size T over the outlier scores and construct the *T-averaged score* as

$$y_t = \frac{1}{T} \sum_{i=t-T+1}^{t} s_o(\boldsymbol{x}_i), \tag{3.12}$$

where $s_o(\boldsymbol{x}_i)$ is the loss score calculated according to 3.11. For all scores $s_o(x_i)$ with $i \leq T$ default values are assumed to calculate the *T-averaged score.* The process of averaging over a window of ourlier scores is needed to smooth out single outliers and keep only bursts of outliers. When a changepoint occurs, the averaged outlier score will start to increase and further rise up within the next time steps. The duration of an elevated averaged outlier score depends on the window size chosen for the averaging process. When an isolated outlier occurs the averaged score will rise slightly, but then decrease again at the next time step.

The resulting sequence of *T-averaged outlier scores* is then used in the second stage to learn the underlying probability density function. The *R-averaged score at time t* is then calculated using a loss function (e.g. 3.11) and the learned model η_q of the averaged outlier score as

$$s(t) = \frac{1}{R} \sum_{i=t-R+1}^{t} s_c(y_i). \tag{3.13}$$

A higher score $s(t)$ indicates that a changepoint occurred with a higher probability. The two moving average processes are used to remove isolated outliers. When the moving average window size R is small, changepoints and outliers are detected immediately after they occur, but it is hard to distinguish between a single outlier and a changepoint. In the case where R is large, the detection lag for changepoints increases, but outliers are filtered successfully and only significant changepoints are identified.

Implementation Details

The ChangeFinder implementation uses the detector base class and therefore implements the common methods *next(x)* and *check_for_change_point().* An SDAR model [17], which is based on an autoregressive model, is used to learn the *T-averaged score* y_t. Initialization parameters can be used to configure the SDAR model, such as the influence of more recent and older samples for the probability density function, or window sizes for the moving average calculations. When a new value is observed, the *next(x)* method is called, which contains the score calculations. First a white Gaussian noise is added to the data signal in order to prevent numerical problems, which might occur for all-zero slices in the data sequence.

The outlier score is calculated using a model based on the *StudentT* distribution. The assumption that data in the time series linearly depends on previously observed values is not necessarily valid for smartphone usage data. The StudentT distribution resembles the normal distribution when sufficient data is available, though it performs relatively well even with only a small amount of available observations and is capable of updating its parameters without storing all past data vectors. See Section 3.4.3 for a more detailed discussion about the suitability of models and distributions. The change score is calculated using a SDAR model, because in this case there certainly exists an autoregressive property of the time series. This is because of the averaging step over the window from the first stage of the algorithm, which results in a dependency of the value in time step t on the values in the time steps $t - r$ to $t - 1$, where r is the window size.

The outlier- and change scores are calculated according to the algorithm, while window sizes for the averaging steps can be configured. When the scores are used to identify changepoints, the lead time is used to prevent premature detections. A changepoint is detected if $s(t)$ exceeds a configured threshold. The changepoint times are then saved and the models are reset in order to prevent a biased start for the next segment in the time series.

3.4.2 Bayesian Online Changepoint Detection

Another changepoint detection algorithm [18] performs exact inference of the most recent changepoint using a Bayesian approach. It is a message-passing algorithm, that is concerned with estimating the run length, or the time since the last changepoint.

The data within each segment is assumed to be i.i.d. (independent and identically distributed) from some probability distribution $P(\boldsymbol{x}_t|\eta_p)$, while also the distribution parameters $\eta_p, p = 1, ..., m$ between different segments are presumed to be i.i.d. The *a priori* probability for a changepoint, that is based on expert knowledge, is denoted as $P_{AP}(\tau)$. The length of the current run, or segment, at time t is denoted as r_t. The set of observations associated with this run is $\boldsymbol{X}_t^{(r)}$. The length of the current run is zero, whenever a changepoint occurs; $\boldsymbol{X}_t^{(r)}$ can consequently be empty.

To find the posterior distribution

$$P(r_t|\boldsymbol{X}^{1:t}) = \frac{P(r_t, \boldsymbol{X}^{1:t})}{P(\boldsymbol{X}^{1:t})}, \tag{3.14}$$

the joint distribution over the run length r_t and the observed data sequence $\boldsymbol{X}^{1:t}$ are examined in a recursive form.

$$\begin{aligned} P(r_t, \boldsymbol{X}^{1:t}) &= \sum_{r_{t-1}} P(r_t, r_{t-1}, \boldsymbol{X}^{1:t}) \\ &= \sum_{r_{t-1}} P(r_t, \boldsymbol{x}_t|r_{t-1}, \boldsymbol{X}^{1:t-1}) P(r_{t-1}, \boldsymbol{X}^{1:t-1}) \\ &= \sum_{r_{t-1}} P(r_t|r_{t-1}) P(\boldsymbol{x}_t|r_{t-1}, \boldsymbol{X}_{t-1}^{(r)}) P(r_{t-1}, \boldsymbol{X}^{1:t-1}), \end{aligned} \tag{3.15}$$

The distribution $P(\boldsymbol{x}_t|r_{t-1}, \boldsymbol{X}_{t-1}^{(r)})$ only depends on the set of observations associated to the current run $\boldsymbol{X}_{t-1}^{(r)}$. This means that a recursive message-passing algorithm can be used to calculate the joint distribution over the current run length and the observed data, which is based on the calculations of the prior over the current run length r_t given the run length in the last time step r_{t-1} and the predictive distribution over the newly-observed data point, given the data since the last changepoint $P(\boldsymbol{x}_t|\boldsymbol{X}_{t-1}^{(r)})$.

The conditional prior on the changepoint $P(r_t|r_{t-1})$ describes the structure of the algorithm. It can only have two outcomes, either the run length continues to grow and $r_t = r_{t-1} + 1$ or a changepoint occurs and the run is truncated, which means the run length r_t is zero.

$$P(r_t|r_{t-1}) = \begin{cases} H(r_{t-1} + 1) & \text{if } r_t = 0 \\ 1 - H(r_{t-1} + 1) & \text{if } r_t = r_{t-1} + 1 \\ 0 & \text{otherwise} \end{cases} \tag{3.16}$$

The probability mass of either case is defined through the hazard function $H(\tau)$. It describes the probability, that a changepoint occurs at time τ, considering the application background. It is given by:

$$H(\tau) = \frac{P_{AP}(\tau)}{\sum_{t=\tau}^{\infty} P_{AP}(t)} \tag{3.17}$$

The predictive probability $P(\boldsymbol{x}_t|\eta_t^{(r)})$, gives the probability, that the observed data point belongs to the learned model of the current segment.

The growth probability gives the probability, that a run length in the last time step r_{t-1} is incremented and the segment is ongoing.

$$P(r_t = r_{t-1} + 1, \boldsymbol{X}^{1:t}) = P(r_{t-1}, \boldsymbol{X}^{1:t-1}) P(\boldsymbol{x}_t|\eta_t^{(r)})(1 - H(t)). \tag{3.18}$$

The change probability gives the probability, that the segment is truncated and the run length drops to zero.

$$P(r_t = 0, \boldsymbol{X}^{1:t}) = \sum_{r_{t-1}} P(r_{t-1}, \boldsymbol{X}^{1:t-1}) P(\boldsymbol{x}_t|\eta_t^{(r)}) H(t). \tag{3.19}$$

The run length with the maximal likelihood is used to detect new changepoints. It is given by:

$$r_t^{max} = \underset{r \in 1:r_t}{\arg\max}\, P(r_t|\boldsymbol{X}^{1:t}), \tag{3.20}$$

with

$$P(r_t|\boldsymbol{X}^{1:t}) = \frac{P(r_t, \boldsymbol{X}^{1:t})}{\sum_{r_t} P(r_t, \boldsymbol{X}^{1:t})}. \tag{3.21}$$

In every time step the message-passing algorithm first evaluates the predictive probabilities $P(\boldsymbol{x}_t|\eta_t^{(r)})$ of the newly-observed data point given the learned model parameters for each past time step. Then the hazard function (3.17) is evaluated for all potential run lengths, $\tau = 1, ..., t$. The algorithm then calculates the growth probabilities (3.18) by scaling the probabilities of each possible run length from the last time step by the predictive probabilities and the conditional changepoint prior. In order to calculate the change probability (3.19), the probabilities of each possible run length from the last time step are scaled by the predictive probabilities and the conditional changepoint prior, which is $H(\tau)$ in this case. These probabilities are then summed up and give the probability, that the run length drops to zero and a changepoint occurs. In the next step the most likely run length (3.20) at the current time step is evaluated. It is the run length hypothesis with the maximal probability mass. A new changepoint is detected, if the run length with the maximal probability does not trace back to the last changepoint. The model parameters for the current time step $\eta_t^{(r)}$ are updated if the segment continues and they are reset if a new changepoint occurs.

See Algorithm 3.5 for step by step calculations in form of pseudo code and see Figure 3.8 for an illustration of a changepoint model described in terms of run lengths. Figure 3.8(a) shows a univariate data sequence, which can be divided into three segments. The changepoints c_1 at time $t = 7$ and c_2 at time $t = 12$ occur due to a mean-shift in the data. The changepoints separate the data sequence into segments with lengths $s_1 = 6$, $s_2 = 5$ and an uncompleted segment with an undetermined length $s_4 \geq 4$. Figure 3.8(b) shows the actual run length r_t as a function of time. The run length is increased by one in every time step where the current segment continues, and it drops to zero as soon as a changepoint occurs. Figure 3.8(c) illustrates the message-passing algorithm. It shows the path on which the algorithm propagates the run length probability mass recursively between time steps. Solid lines indicate the case where no changepoint occurs and the probability mass is passed "upward" from the last time step. Dotted lines indicate the case where a changepoint occurs, the current run is truncated, and the probability mass is passed "down" to a run length of zero. Each circle represents a run length hypothesis, containing probability mass. At each time step the most likely run length is determined and used to detect new changepoints.

For initialization two different cases are considered. When a changepoint occurs immediately before the first observation, all the probability mass for the initial run length is placed at zero, thus $P(r_0 = 0) = 1$. This is used for applications where the last changepoint is known and the observation sequence then starts exactly at the beginning of the new segment. In the other case, where the observation sequence is most likely a recent subset of the segment, the prior over the initial run length is approximated by the normalized *survival function*

$$P(r_0 = \tau) = \frac{1}{Z} S(\tau), \tag{3.22}$$

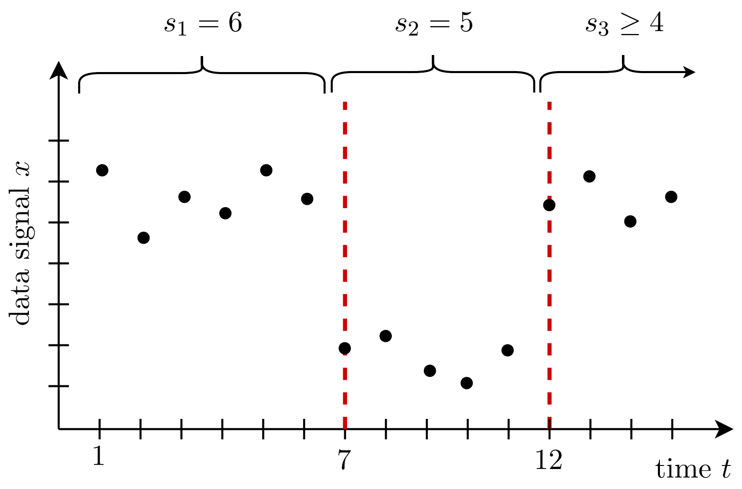
where Z is an appropriate normalizing constant, and $S(\tau)$ is the survival function.

$$S(\tau) = \sum_{t=\tau+1}^{\infty} P_{AP}(t) \tag{3.23}$$

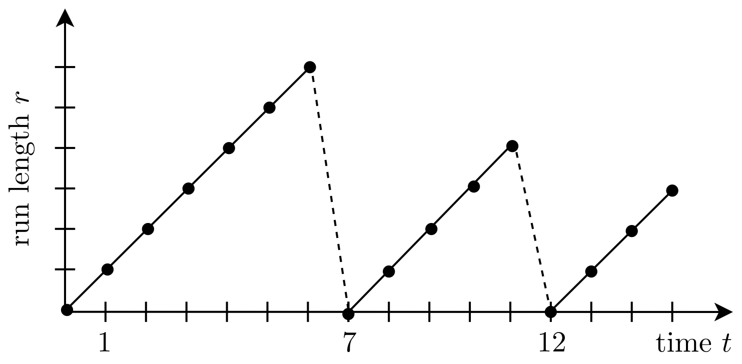
```
Initialize
  $P(r_0) = S(\tau)$  or  $P(r_0 = 0) = 1$
  $\eta_1^{(0)} = \eta_{prior}$
Observe New Sample $\boldsymbol{x}_t$
Evaluate Predictive Probability
  $\pi_t^{(r)} = P(\boldsymbol{x}_t|\eta_t^{(r)})$
Evaluate Hazard Function
  $h_t = H(1:t)$
Calculate Growth Probabilities
  $P(r_t = r_{t-1}+1, \boldsymbol{X}^{1:t}) = P(r_{t-1}, \boldsymbol{X}^{1:t-1})\pi_t^{(r)}(1-h_t)$
Calculate Change Probabilities
  $P(r_t = 0, \boldsymbol{X}^{1:t}) = \sum_{r_{t-1}} P(r_{t-1}, \boldsymbol{X}_{1:t-1})\pi_t^{(r)} h_t$
Calculate Evidence
  $P(\boldsymbol{X}^{1:t}) = \sum_{r_t} P(r_t, \boldsymbol{X}^{1:t})$
Determine Run Length Distribution
  $P(r_t|\boldsymbol{X}^{1:t}) = P(r_t, \boldsymbol{X}^{1:t})/P(\boldsymbol{X}^{1:t})$
Determine Maximal Run Length Likelihood
  $r_t^{max} = argmax_{r \in 1:r_t} P(r_t|\boldsymbol{X}^{1:t})$
Update Sufficient Statistics
  if changepoint occurred
    $\eta_1^{(0)} = \eta_{prior}$
  else
    $\eta_{t+1}^{(r+1)} = update(\eta_t^{(r)}, \boldsymbol{x}_t)$
Return to line 4
```

Algorithm 3.3: Overview of steps needed to perform the bayesian online changepoint detection.

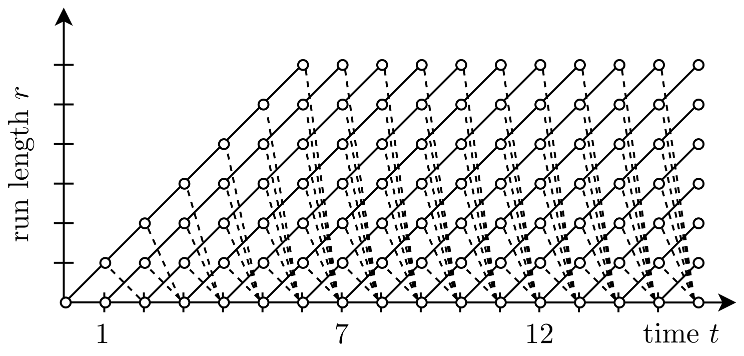
The conditional prior on the changepoint $P(r_t|r_{t-1})$ makes the algorithm efficient, because the probability mass function is non-zero in only two cases. Either the run length of the previous time step is increased by one, or it drops down to zero. In each time step all possible run length likelihoods of the previous time step and their model parameters are needed. The space- and time-complexity per time step are linear in the number of data points.



(a) Data sequence separated by changepoints.



(b) Run length.



(c) Trellis of run length estimate.

Figure 3.8: Changepoint model expressed in terms of run lengths.

Implementation Details

The Bayesian Online Changepoint Detection implementation uses the detector base class and therefore implements the common methods *next(x)* and *check_for_changepoint()*. Initialization parameters are used to configure the hazard- and survival functions, a survival normalizing constant, the model used to estimate the observation likelihoods and the lead time. The prior over the initial run length is chosen to be the normalized survival function, as in equation 3.22. This is necessary, because the first datum is assumed to be recorded sometime within a segment rather than at the very beginning. Depending on the parameter *t_lead_active* a countdown for the lead time is initialized.

When a new value is observed, the *next(x)* method is called, which checks the lead time, if applicable, and performs the calculations. The wrapper for the likelihood observation models are able to handle initial updates from data received during the lead time as well as updates during the actual changepoint evaluation, where they keep model parameters for every time step. The implementation in python is shown in Listing 3.6.

```
# based on:
# http://hips.seas.harvard.edu/content/bayesian-online-changepoint-detection
# and https://github.com/hildensia/bayesian_changepoint_detection
def next(self, x):
    self._update_base_residuals(x)

    # Update model while in the lead time countdown
    if self.t_lead_countdown > 0:
        self.observation_likelihood.update_theta(x, initial=True)
        self.t_lead_countdown -= 1
        return

    if self.t_lead_countdown == 0:
        self.Ts.append(self.signal_size-1-self.t_lead0)
        self.t_lead_countdown -= 1

    self.t += 1

    R = np.zeros((self.t + 2, self.t + 2))
    R[0:self.t+1,0:self.t+1] = self.R
    self.R = R

    # Evaluate the predictive distribution for the new datum
    # under each of the parameters.
    predprobs = self.observation_likelihood.pdf(x)

    # Evaluate the hazard function for this interval
    H = self.hazard_func(r=np.array(range(self.t_lead0+1,self.t_lead0+self.t+2)))

    # Evaluate the growth probabilities:
    # Shift the probabilities down and to the right,
    # scaled by the hazard function and the predictive probabilities.
    self.R[1:self.t+2, self.t+1] = self.R[0:self.t+1, self.t] * predprobs * (1-H)

    # Evaluate the changepoint probabilities:
    # Accumulate the mass back down at r = 0.
    self.R[0, self.t+1] = np.sum( self.R[0:self.t+1, self.t] * predprobs * H)

    # Renormalize the run length probabilities for improved numerical
    # stability.
    # Run Length Distribution = Runlength Probabilities / Evidence
    self.R[:, self.t+1] = self.R[:, self.t+1] / np.sum(self.R[:, self.t+1])

    # Update the parameter sets for each possible run length.
    self.observation_likelihood.update_theta(x, False)

    # Evaluate the maximal run length likelihood for the current time t
    self.maxes.append(self.R[:, self.t+1].argmax())
```

Listing 3.6: Implementation of the Bayesian Online Changepoint Detection Algorithm.

The method *check_for_changepoint()* uses the *maxes* list, which contains the indices of the run length with the maximal likelihood per time step. Using the current time step t and the run length with the currently maximal likelihood, the time step of the last changepoint is calculated. If a new changepoint was detected, the time step in which it occurred and additionally the time step of detection are preserved. The distribution models and the lead time are then reset. So after the lead time has elapsed, there exists a clear model, learned from only the data after the last changepoint.

Hazard function

The *hazard function* defines the prior knowledge over the changepoint frequency in the domain.If no prior knowledge is available, one could choose a constant. This assumes, that the occurrence of a changepoint at a certain time step t does not depend on t and is given by:

$$H(\tau) = 1/\lambda \tag{3.24}$$

Alternatively, a more suitable hazard function can be used if prior knowledge exists. In the case of bipolar state changes, we choose $P_{AP} = \mathcal{N}(\mu = 120, \sigma = 45)$. This models, that around 68% of bipolar affected people experience a state change after 75 to 165 days of their previous state [3]. The hazard function is given by the density function divided by the survivor function. For numerical stability, this is calculated by exponentiating the difference of the logarithmic density and the logarithmic survivor likelihood. In the case of the assumed normal distribution, it can be calculated as follows in python:

```python
def normal_hazard(mu, si, r):
  return exp(norm.logpdf(r, loc=mu, scale=si) - norm.logsf(r, loc=mu, scale=si))
```

Listing 3.7: Implementation of the normally distributed hazard function.

Feature Fusion Versions In order to deal with the numerous features relevant for the multivariate smartphone data, there are two versions regarding feature fusion, as already described in Section 3.4. Both versions were implemented: the BOCPD with a multivariate model, and the BOCPD with multiple univariate models that are combined using weights. To accomplish this, the *BayesDetector* was extended to activate / deactivate the lead time and supply functions to reset certain properties and retrieve run length likelihoods. For the multivariate version one detector is instantiated and supplied with a multivariate distribution model. This detector is then directly used by the *OnlineSimulator* to simulate the data sequence and call the appropriate methods for every time step.

In case of the weighted version, a wrapper was implemented. It receives hazard- and survival functions, distribution models, normalizing constants for the survival functions, weights and a lead time. All parameters except the lead time are arrays of the same size, which is again the same size as the input vector x in the *next(x)* step. The wrapper then creates a *BayesDetector* for each feature, with the parameter elements respectively. Here univariate distribution models are used, where each feature might use its particular distribution model. For hazard- and survival functions the same distribution with same parameters is used for all features, because it represents the changepoint frequency and is independent of the distribution of individual features. The same lead time is used for all concrete detectors, where it is deactivated and instead handled in the wrapper. When the *next(x)* method is called, the concrete detectors are iterated and their appropriate methods are called, so that the wrapper in the end holds the run length likelihoods for the current time step t and weights them according to the *weights* parameter. The resulting fused run length likelihoods are then used to determine the most likely run length and further the last changepoint in the same manner as the *BayesDetector*.

3.4.3 Model Selection and Data Characteristics

The models that are used to describe the underlying stochastic process of the time series data are crucial for the performance of the change detection algorithm. First, one can distinguish between two types of models, the *independent models* which don't take the chronological course of the time series into account and *time series models*, or *regression models*, which are based on the linear dependency of data points on previously observed data. The suitability of these models strongly depends on the data, but is also influenced by the algorithms that use them.

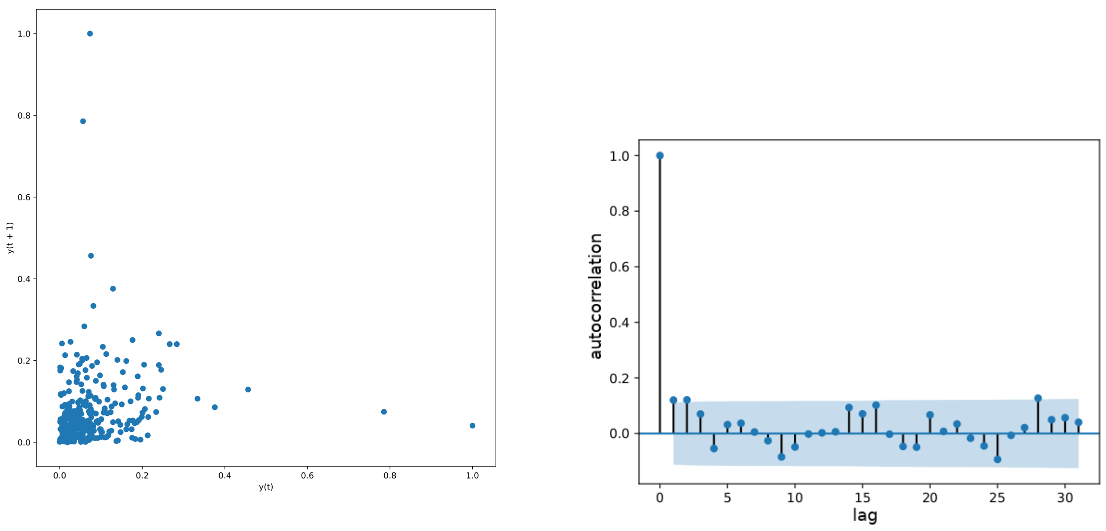
Time Series Model

A time series model is conceptually a linear regression of the value at time step t against prior values of the series $\boldsymbol{X}^{1:t-1}$. Commonly used time series models are the *autoregressive (AR)* models and the *moving average (MA)* models as well as combinations of both. Within the context of this thesis only the autoregressive model was investigated regarding its suitability due to its popularity in time series literature. The autoregressive model for univariate data is defined by

$$x_t = \delta + \sum_{i=1}^{k} \phi_i x_{t-i} + \epsilon, \tag{3.25}$$

where k is the order of the model, ϕ_i is the model parameter for the predecessor at distance i, δ is the process mean weighted by $(1 - \sum_{i=1}^{k} \phi_i)$ and ϵ is white noise [19].

In order to determine the suitability of the AR model for the smartphone usage time series, the auto regression properties of the data were analyzed using visual methods. Figure 3.9 shows autoregressive visualizations of the *Walking* feature. Figure 3.9(a) shows the measurement difference of that feature from one day to the next, where the x-axis contains the value at time step t and the y-axis represents the data value at time step $t + 1$. Figure 3.9(b) shows the correlation of data values to previously observed data with a fixed distance (lag). This visual representation indicates, if there are linear dependencies and if applicable, suitable orders for a auto regressive model. In smartphone usage data most of the features don't seem to behave according to an AR model.



(a) Data plot with $lag = 1$.

(b) Autocorrelation.

Figure 3.9: Autoregressive analysis of physical activity feature Walking.

Models for Time-Independent Samples

Independent models fit some assumed distribution to the relevant data, weighting every data point equally, independent of its distance to the current time step. There are various distributions available [20], differing among others in

1. Continuity
2. Symmetry
3. Upper- / lower limits
4. Frequency of outliers

In order to choose a distribution, the smartphone data was analyzed regarding these characteristics. In our case the data is continuous with upper and lower limits, the only exception is the *mood*, which is a discrete value between minus three and plus three. Features with the time frame *whole day* for instance can reach duration values between zero seconds and twenty-four hours. The data representing activity durations is mostly clustered around a central value, where positive and negative derivations are approximately equally likely. This characteristic highly depends on the user him- or herself though. A user might, for instance, not use social media applications very often so the mayor cluster of usage durations per day would be at or slightly above zero, the lower limit. When the user now and then uses this application group, the distribution would gain a skewness toward the positive. The frequency of extreme values in our smartphone usage data is rather high, which results in a high kurtosis (or positive excess kurtosis, which is related to the normal distribution).

The *normal distribution* is one of the most commonly used continuous distributions in probability theory. It is often used to model unknown distributions of random variables in the natural and social sciences. It is popular because of its broad applicability; the *central limit theorem* states, that the sum of realizations of random variables tends toward a normal distribution, even if the original variables are not normally distributed. The normal distribution is described by only two parameters, the mean and the standard deviation. This makes model fitting to the normal distribution easier and more efficient than with most other distributions. The normal distribution is characterized by a strong tendency for data to take on a central value while positive and negative derivations are equally likely, so it is symmetric. The frequency of deviations falls off rapidly as we move further away from the central value, so it has a flat tail modeling few extreme values.

Hence, the smarphone data does not fit the normal distribution particularly well. Alternatively, one can use the *logistic distribution* or the *Cauchy distribution*, which are both symmetric but have a higher kurtosis than the normal distribution. This means, that these distributions have a heavy tail. The Cauchy distribution has a scale parameter to determine how heavy the tail of the distribution is.

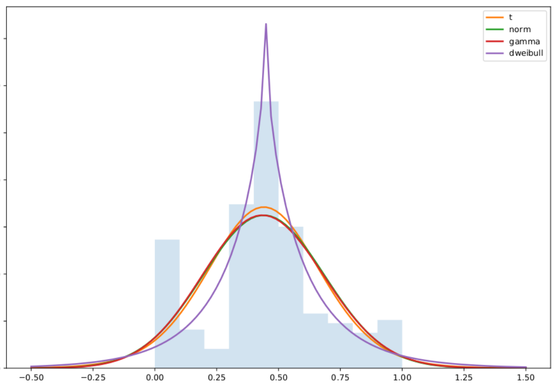
The *Student's t-distribution* or *t-distribution* is another symmetric and bell-shaped distribution, that has heavy tails. It is especially useful when a normal distribution is assumed and there is only a small sample size available, while mean and standard deviation are unknown. The t-distribution resembles the normal distribution when sufficient data is available, though it performs relatively well even with only a small amount of available observations and is capable of updating its parameters without storing all past data vectors.

Independent of the distribution's form, it is advisable to use distributions that allow a piece-wise update of their parameters. Another aspect to the distribution choice is the amount of data required to successfully represent the stochastic process. The more complex the model, the more data is required to learn the process. A more complex model might be able to represent the data better than a simpler one, but if there is not enough data to properly fit the model, it might perform worse in comparison to the simple model.

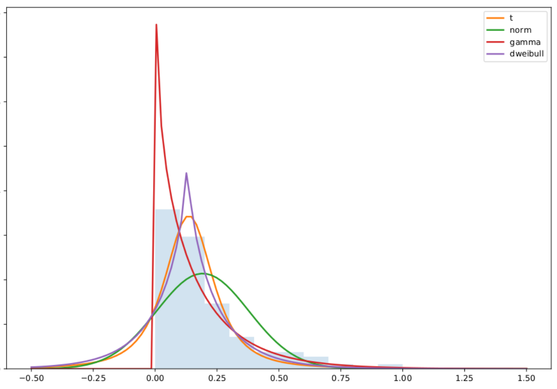
We can either use multiple univariate models that are combined, or one multivariate model. An advantage of the multivariate model is, that additionally to the individual distributions of the features it models the relationship between them. A weighted model consisting of two features could not detect a situation where the features change their relation to each other while a multivariate model could. For the weighted case expert knowledge is required in order to properly choose the importance of the individual components. The models themselves are simpler and might yield better results for less learning data.

Within the context of this thesis, models based on the *t-distribution*, *multivariate t-distribution*, *mixture of t-distributions*, *multivariate normal distribution*, *mixture of Gaussians*, *multivariate mixture of Gaussians* were applied to learn smartphone usage data and visually analyzed [21], [22], [23].

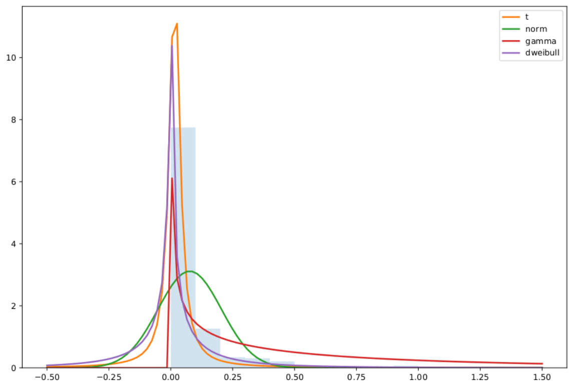
See Figure 3.10 for exemplary visual analysis of the distributional fit of univariate smartphone usage data to some of the most relevant distributions.



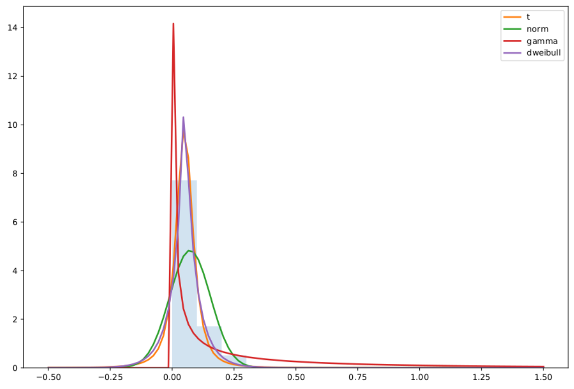
(a) Location *Home*.



(b) App Usage *Messaging*.



(c) App Usage *Phone*.



(d) Physical Activity *Walking*.

Figure 3.10: Various distributions to model smartphone usage data.

3.5 Evaluation

This section is concerned with the evaluation of the algorithmic performance. First, we explain how the evaluation of the different datasets is performed and then we give an introduction to the *F-score* calculations, in principal and how it can be applied to changepoint detection.

In order to evaluate the different algorithms with distinct distributions and various configurations, a consistent scheme of quality estimation is required. The fact, that only very limited and incomplete data for changepoint targets is available, makes the evaluation process difficult. In order to verify the correct implementation and get a first insight of the performance of the two algorithms, some synthetic data is generated. The performance is measured by means of the F_1 *score*, which is a well known and widely used measure of accuracy in statistical analysis of binary classification. See Section 3.5.1 for a detailed description of the *F-score* and how it can be applied to evaluate the accuracy of detected changepoints while also considering the *detection lag*.

In order to evaluate the performance of different distribution models and feature selection mechanisms it is necessary to apply the algorithms to real smartphone usage data. For this purpose the recorded usage data of multiple people was examined, including self-recorded targets in form of a coarse diary. So the subjects themselves provided records of extraordinary events and changes in their lifestyle. In this case, the evaluation was done manually by executing the change detection on the data and tracing back the detected changepoints to actual events that are listed in the participant's notes.

Actual data from bipolar affected people is acquired through the study, which is described in Section 2.6. Because the study is still ongoing and at the time of this thesis there are only few participants with relatively short time series of recorded smartphone usage data, there is only very limited data available to evaluate the actual use case of the change detection process. The available data is used in the same manner as the smartphone usage data, that originated by the group of ordinary participants. In this case the targets result from the interviews and questionnaires made by the clinical personnel during the medical visits of the study, as described in Section 2.6. They are used for the detection evaluation by means of a manual review and individual discussion.

3.5.1 F-score

The *F-score* is widely used as a measure of accuracy in statistical analysis of binary classification. It expresses the relation between *precision* and *recall*. Precision p describes the amount of selected elements that are relevant, which is calculated by the number of correct positive results (true positives) divided by the number of all by the classifier as positive categorized results (true positives + false positives). Recall r represents the amount of relevant items that are selected, which is calculated by the number of correct positive results (true positives) divided by the number of all truly positive, or relevant, samples(true positives + false negatives). See Figures 3.11 and 3.12 for a visual representation of the precision and recall calculations.

The general formula for the *F-score* is:

$$F_\beta = (1+\beta^2)\frac{pr}{(\beta^2 p)+r}, \tag{3.26}$$

where β is a positive real value, which emphasis either precision or recall. A $\beta < 1$ weights recall lower than precision, such as the $F_{0.5}$ measure, and a $\beta > 1$ weights recall higher than precision, such as the F_2 measure. The F_1 score is the traditional or balanced F-score, which is the harmonic mean of precision and recall. It is given by:

$$F_1 = 2\frac{pr}{p+r} \tag{3.27}$$

Perfect precision and recall result in an F_1 score of 1 and a value of 0 means that every element was classified the wrong way.

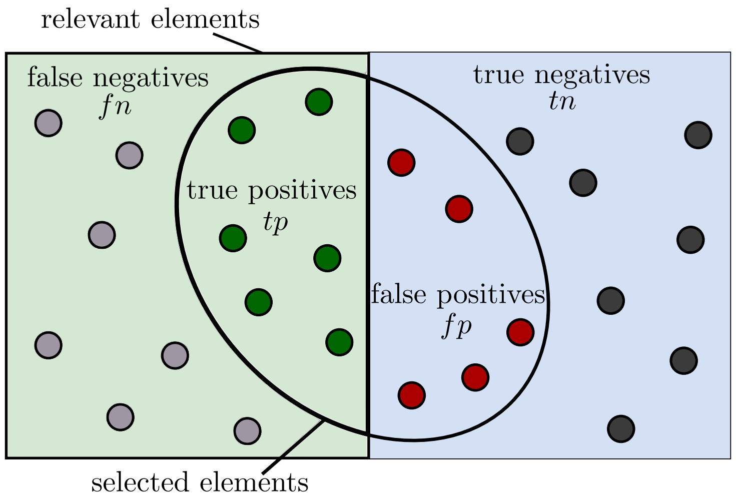
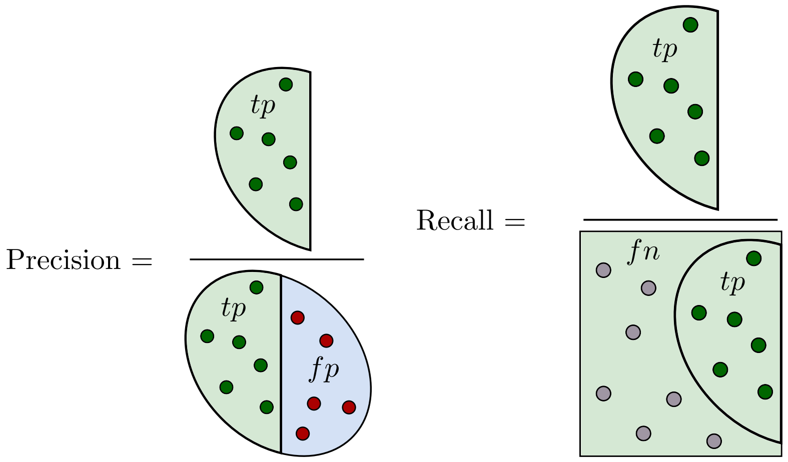


Figure 3.11: True positives tp, false positives fp, true negatives tn, false negatives fn.



(a) Precision: How many selected items are relevant?

(b) Recall: How many relevant items are selected?

Figure 3.12: Precision and Recall.

Application of the F_1 score to Changepoints

The F_1 score can be applied to evaluate the performance of detected changepoints in various ways. First, one can only consider the time step of the detected changepoint and ignore the detection lag for the evaluation process. Then there is the assignment of detected changepoints to real changepoints, which influences the calculation of the true positives, false negatives, false positives and true negatives. So a detected changepoint could for example be accepted as true positive only when it was detected on the very same day as the original changepoint, or it could be accepted if it was within a window around the real changepoint. Also, there could be a score, indicating how close the detected changepoint was to the real one. The same considerations could be made for the detection lag, which would introduce either two acceptance windows or two scores describing how close the detected changepoint is to the real one and how long it took to detect this change.

We chose to calculate the true positives tp and further the true negatives $tn = m - tp$ using

$$tp = \sum_{i=1}^{n} (1 - \alpha_L)^{\mathbb{1}_{CP}} + \alpha_L^{\mathbb{1}_L}, \tag{3.28}$$

where n is the number of real changepoints and m is the number of detected changepoints. The weight for the detection lag is denoted as α_L, the weight for the changepoint is $1 - \alpha_L$. We use $\mathbb{1}_{CP}$ as the indicator function for changepoints, where $\mathbb{1}_{CP} = 1$ if the changepoint is within the window and $\mathbb{1}_{CP} = 0$ otherwise. We denote $\mathbb{1}_L$ as the indicator function for detection lags, where $\mathbb{1}_L = 1$ if the lag is within the lag-window and $\mathbb{1}_L = 0$ otherwise.

In our case an accurate detection as well as a short detection lag are relevant, event though the accuracy of the detection is considered more important. We chose equal window lengths for the changepoint and the detection lag. The detected changepoint and the time of detection must be within a window around the real changepoint, in order to be counted as correct. When the changepoint is detected correctly but the lag is too large, only $(1 - \alpha_L)$ is added to the true positives (tp) and α_L is added to the true negatives (tn).

4 Results

In this chapter the two algorithms *ChangeFinder (CF)*, as described in Section 3.4.1, and *Bayesian Online Changepoint Detection (BOCPD)*, as described in Section 3.4.2, are evaluated and discussed. General algorithmic performance is examined in Section 4.1 by using synthetic data. Various cases, such as mean-shifts, variance-shifts and outliers, are investigated with univariate- and bivariate data. Each case is evaluated using the F_1 *score*, as described in Section 3.5.1, and its visualization is discussed individually.

The results and applicability to real world data is studied in Sections *General change detection in user behavior* (4.2) and *Detection of state changes in Bipolar Disorder* (4.3). Here the data originates from the *Bip-Up*, it is acquired and preprocessed as described in Section 3.1. The feature selection was performed manually, by picking appropriate features by hand.

4.1 Synthetic Data

In this section the data is drawn from normal distributions; the parameters are indicated in the particular cases.

4.1.1 Univariate Data

The univariate data signal $\boldsymbol{x}^{1:t}$ is drawn from one of the following models

$$p_1(x) \sim \mathcal{N}(\mu = 0.6, \sigma = 0.02)$$

$$p_2(x) \sim \mathcal{N}(\mu = 0.1, \sigma = 0.02)$$

$$p_3(x) \sim \mathcal{N}(\mu = 0.3, \sigma = 0.02)$$

$$p_4(x) \sim \mathcal{N}(\mu = 0.35, \sigma = 0.02)$$

$$p_5(x) \sim \mathcal{N}(\mu = 0.35, \sigma = 0.15)$$

The time series is segmented by two mean-shifts at time steps $CP_1 = 50$ and $CP_2 = 100$. For the BOCPD a *constant hazard* function (see Section 3.4.2) was used as prior and the *student-T* distribution as observation model. For the *ChangeFinder* the *student-T* distribution was used as observation model and the SDAR model ($r = 0.02$ and $k = 5$) was used for learning the Outlier Score with window sizes of $T = 5$ and $R = 5$.

Mean-Shift

The data in the first segment is drawn from p_1, the data in the second segment is drawn from p_2 and the data in the third segment is again drawn from p_1.

Figure 4.1 shows a visualization of the *Bayesian Online Changepoint Detection (BOCPD)* algorithm and its key values. The top diagram shows the data signal in blue, while the detected changepoints are indicated by a red dashed line. The diagram in the middle visualizes the run length likelihood $P(r_t|\boldsymbol{X}^{1:t})$ (see Section 3.4.2) per time step. Darker coloring implies a higher likelihood and lighter coloring means a lower likelihood. Thus, dark diagonal lines imply a probable path for the run length development, and therefore a potential continuous segment. A gradient within one time step along the y-axis, that reaches from dark to light, indicates an outlier or changepoint. The bottom diagram shows the run length with the maximal likelihood in every time step. During the *lead time*, the run length is not tracked and thus it remains at zero. After the algorithm starts its calculations, the run length grows until the next changepoint is detected, where it then drops back down to the most probable run length for the current time step.

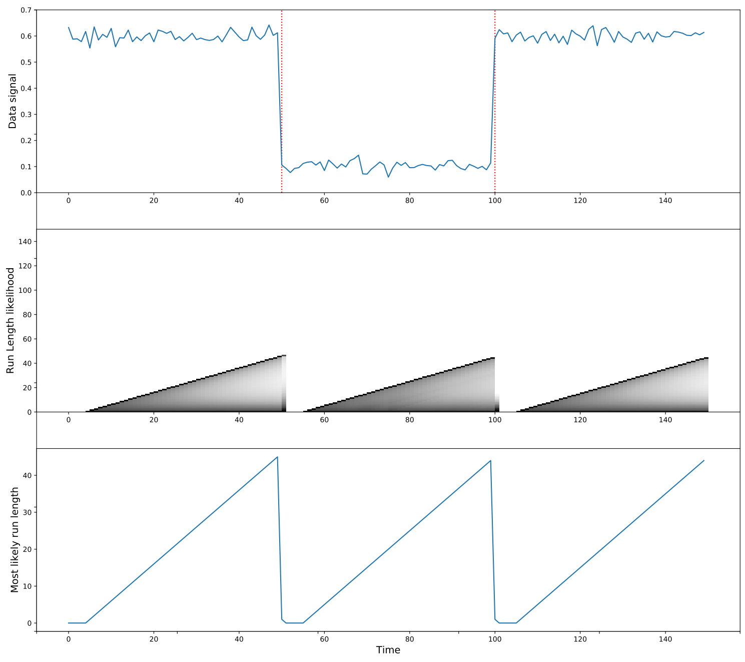


Figure 4.1: Bayesian Online Changepoint Detection (BOCPD); $F_1 score = 1.0$

Figure 4.2 shows a visualization of the *ChangeFinder (CF)* algorithm and its key values. The top diagram shows the data signal in blue, while the detected changepoints are indicated by a red dashed line. The next diagram contains the *Outlier Score*, calculated according to equation 3.11, using a *studentT* model. The third diagram shows y_t, the *T-averaged Outlier Score*, which is calculated according to equation 3.12 with a window size $T = 5$. The next diagram shows the *Change Score*, which is again calculated according to equation 3.11 by using the Outlier Score and the model learned from it. The bottom diagram visualizes the *R-averaged Change Score*, which is calculated according to equation 3.13 with a window size of $R = 5$. Here the ChangeFinder detector is configured with a *threshold* $= 4$, and SDAR model parameters $r = 0.02$ and $k = 5$.

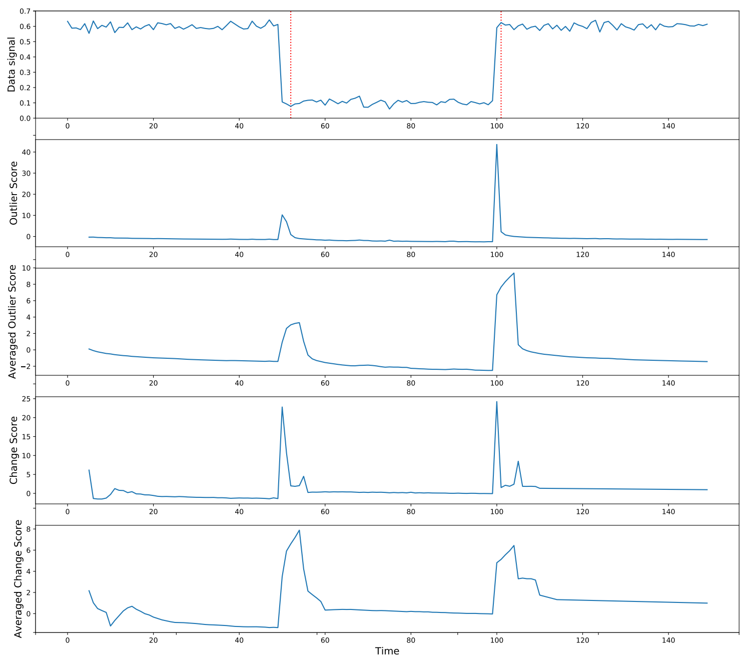


Figure 4.2: ChangeFinder (CF); $F_1 score = 1.0$

Strong Mean-Shift with Outliers

The data is drawn from p_1 in the first segment, from p_2 in the second one and again from p_1 in the third segment. An outlier, which is a single extreme value, occurs at time step $t_o = 25$.

Figure 4.3 shows, that the BOCPD algorithm immediately reacts to changepoints and outliers that differ strongly from the learned model. Here the algorithm is already very certain about a change at the time step of its occurrence. This results in a very low detection lag ($lag = 0$), which is optimal. In case of outliers this is a problem though, because in the first step an outlier does not differ from a changepoint and thus every outlier with such a high deviation from the model is interpreted as a changepoint.

In Figure 4.4 one can see, that the *ChangeFinder (CF)* algorithm generally handles single outliers very well, even if they have extreme values. Due to the averaging process, single samples don't carry weight. For the same reason the detection lag is higher, even if the change is very thoroughgoing. So there is a trade-off relation between the detection lag and the outlier resistance, which can be configured using the window size R of the averaging step.

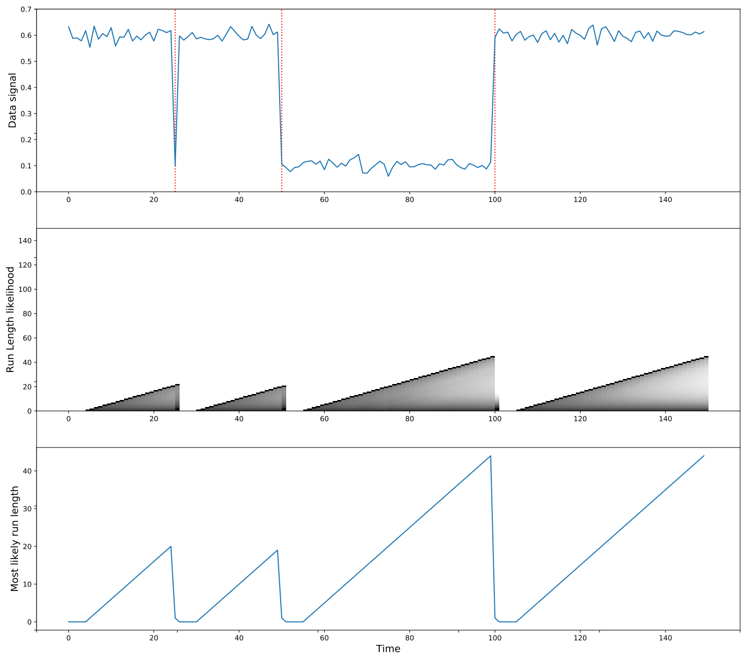


Figure 4.3: Detection of strong mean-shifts with outliers; $F_1 score = 0.8$

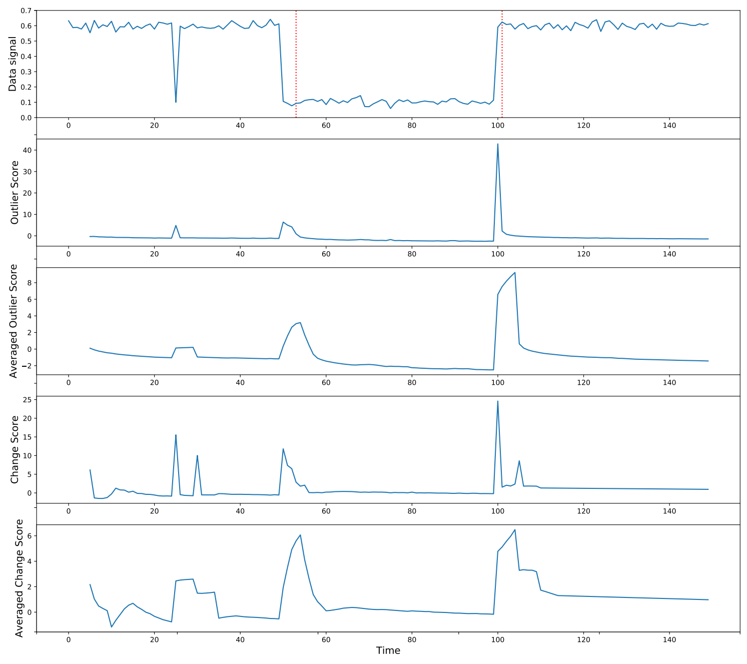


Figure 4.4: Detection of strong mean-shifts with outliers; CF; $F_1 score = 1.0$

Weak Mean-Shift with Outliers

The data is drawn from p_1 in the first segment, from p_3 in the second one and again from p_1 in the third segment. Note, that in this case the mean of p_3 is closer to the one of p_1 than in the case above. An outlier occurs at time step $t_o = 25$.

Figure 4.5 shows, how the BOCPD algorithm reacts to outliers and changepoints with a moderate deviation of the learned model. One can see how uncertainty arises when an outlier occurs at time step $t = 25$. In the run length likelihood diagram a darker diagonal line evolves and slowly fades away again around time step $t = 40$. This happens, because the outlier did not differ enough from the learned model in order to immediately trigger a change detection and with every subsequent non-outlier sample the likelihood of a change further decreases again. At time step $t = 50$ another extreme value occurs and the uncertainty about the run length rises. In subsequent time steps further extreme values occur according to the learned model. With each time step the likelihood of a change at time step $t = 50$ increases until it outperforms the likelihood of a connected segment since time step $t = 0$, and a change is detected after a lag of 3 time steps.

In Figure 4.6 we can see, how the CF algorithm handles moderate changes. It handles outliers very well and correctly detects moderate changepoints. Here the detection lag is further increased in comparison to the above case with strong mean-shifts.

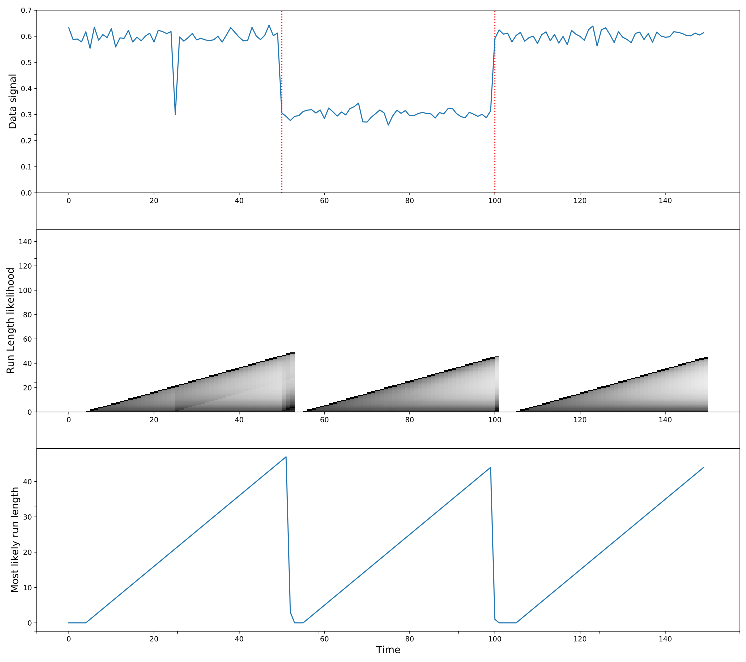


Figure 4.5: Detection of weak mean-shifts with outliers; BOCPD; $F_1 score = 1.0$

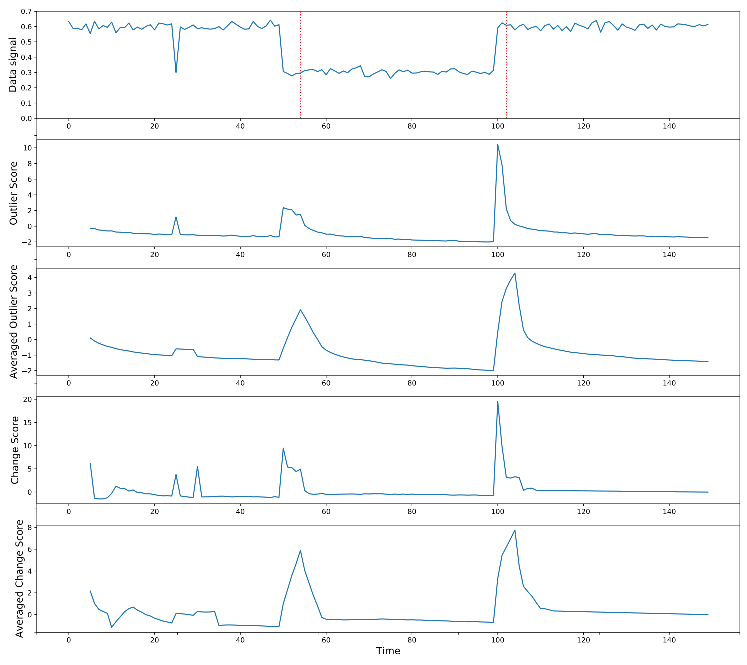


Figure 4.6: Detection of weak mean-shifts with outliers; CF; $F_1 score = 1.0$

Variance-Shift

The data is drawn from p_4 in the first segment, from p_5 in the second one and again from p_4 in the third segment.

Figures 4.7 and 4.8 show that both, the BOCPD and the CF algorithms, don't handle variance shifts well. Uncertainty arises but the correct changepoints are not found.

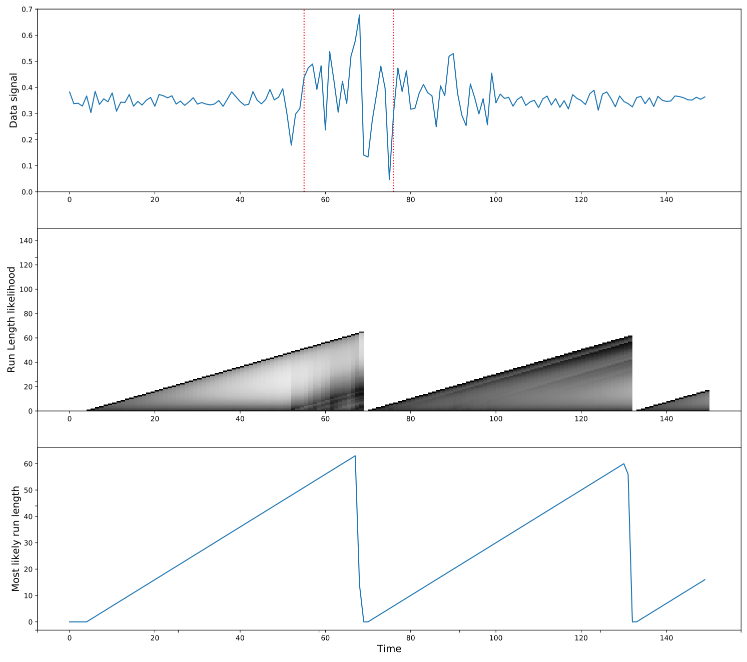


Figure 4.7: Variance-shifts; BOCPD; $F_1 score = 0.0$

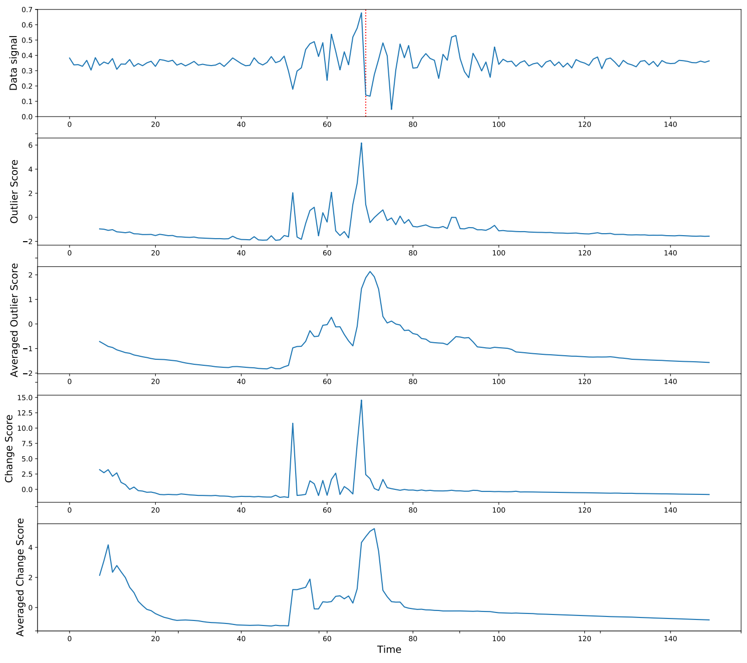


Figure 4.8: Variance-shifts; CF; $F_1 score = 0.0$

4.1.2 Bivariate Data

The bivariate data signal $\boldsymbol{X}^{1:t}$ is generated by drawing samples from two of the following models

$$p_1(x) \sim \mathcal{N}(\mu = 0.6, \sigma = 0.02)$$

$$p_2(x) \sim \mathcal{N}(\mu = 0.1, \sigma = 0.02)$$

$$p_3(x) \sim \mathcal{N}(\mu = 0.3, \sigma = 0.02)$$

Unless indicated otherwise, a *constant hazard* function (see Section 3.4.2) was used as prior for the BOCPD and the multivariate *student-T* distribution as observation model. For the *ChangeFinder* the multivariate *student-T* distribution was used as observation model and the SDAR model ($r = 0.02$ and $k = 5$) was used for learning the Outlier Score with window sizes of $T = 5$ and $R = 5$.

Mean-Shift with Outliers

The data of both features is drawn from p_1 in the first segment, from p_2 in the second one and again from p_1 in the third segment. The mean-shifts occur at time steps $CP_1 = 50$ and $CP_2 = 100$ and an outlier occurs at $t_O = 25$. This gives us sample sizes of $s = 50$ per segment.

Figure 4.9 shows the BOCPD with multiple univariate features, that are combined using weights. We can see, that this algorithm detects changes very well in the given data with a small detection lag of $lag = 0$ (optimal). It is also sensitive to outliers, which we can see at time step $t = 25$, where a single extreme value is detected as change in the first time step after its occurrence.

Figure 4.10 shows the BOCPD with one multivariate model. This algorithm is less sensitive to outliers, but it is also less sensitive to changepoints. This leads to higher detection lags of $lag_1 = 20$ and $lag_2 = 25$. Note, that the accuracy in both cases is very high.

Figure 4.11 shows the CF algorithm, which does not detect any of the two changes. We can see the Change Score in the bottom diagram, which indicates the first changepoint, but does not trigger a changepoint detection.

Both BOCPD algorithms, with weighted features and with one multivariate model, detect the changes correctly. The weighted version has a zero-lag, but it also detects an outlier as a changepoint. The multivariate version does not detect the outlier, but has higher lags for the changepoints. The CF algorithm on the other hand does not detect any change, but we can see an indication for the first changepoint in the Change Score.

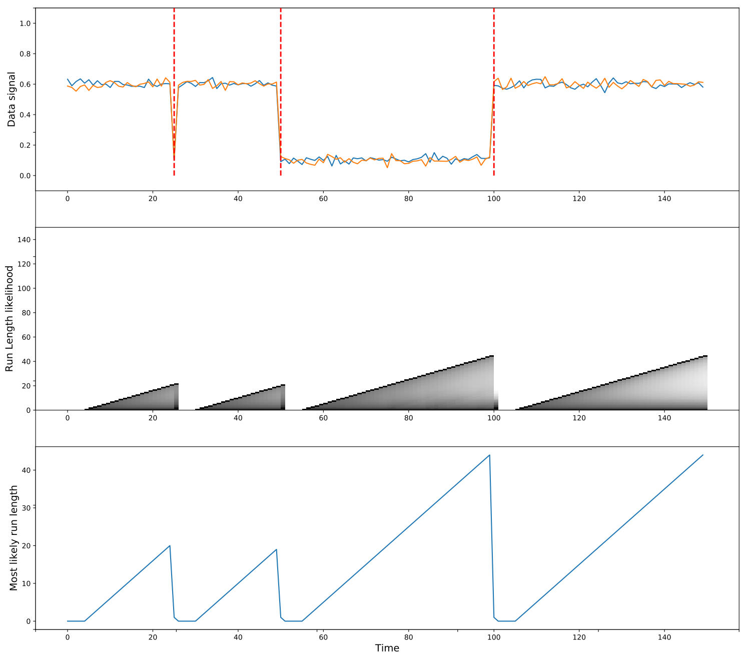


Figure 4.9: Mean-shifts with outliers; BOCPD (weighted univariate features); $s = 50$; $F_1 score = 0.8$

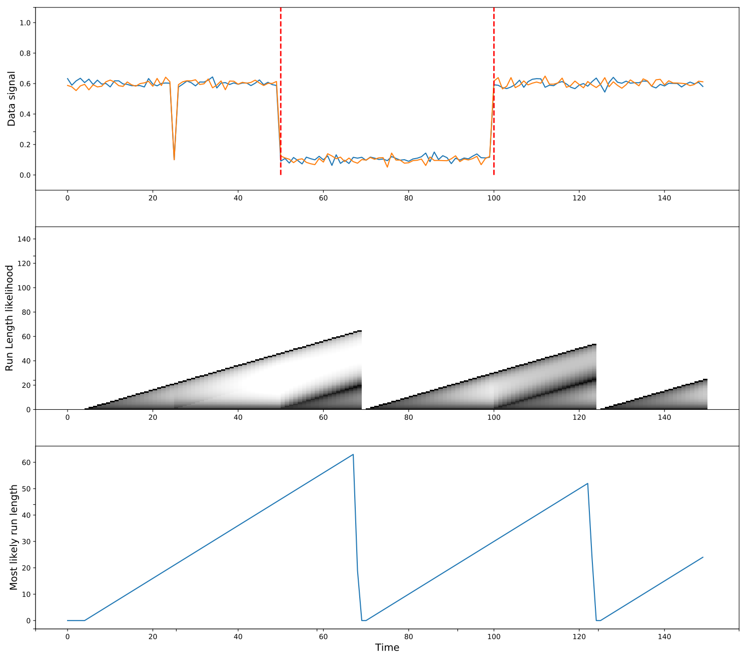


Figure 4.10: Mean-shifts with outliers; BOCPD (multivariate); $s = 50$; $F_1 score = 0.7$

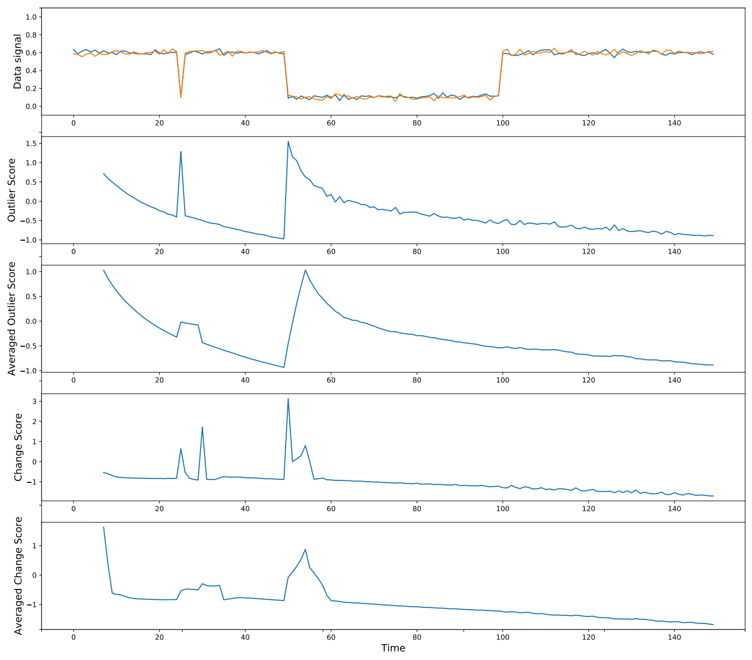


Figure 4.11: Mean-shifts with outliers; CF; $s = 50$; $F_1score = 0.0$

Mean-Shift with Outliers and a Higher Sample Size of $s = 200$

As above, the data of both features is drawn from p_1 in the first segment, from p_2 in the second one and again from p_1 in the third segment. The mean-shifts in the first feature occur at time steps $CP_1 = 200$ and $CP_2 = 400$ and an outlier occurs at $t_O = 100$. Note, that in this case the segments have sample sizes of $s = 200$.

Figure 4.12 shows the weighted BOCPD. As in the case with segment sizes of $s = 50$, this algorithm detects the changepoints correctly with a minimal lag, but it also detects the outlier as a changepoint.

In Figure 4.13 we can see, that the multivariate BOCPD gets more sensitive to changepoints with larger sample sizes per segment, as the detection lag decreases.

Figure 4.14 shows, that the CF algorithm performs very well with higher sample sizes. It still handles outliers very well, while changepoints are detected successfully.

Both BOCPD algorithms get more sensitive to changepoints and outliers, when the segments last longer. This can have positive effects (in the case of changepoints and detection lags) and negative effects (in the case of outliers). The CF algorithm on the other hand clearly profits from larger sample sizes and, in this case, performs very well in terms of changepoints and outliers.

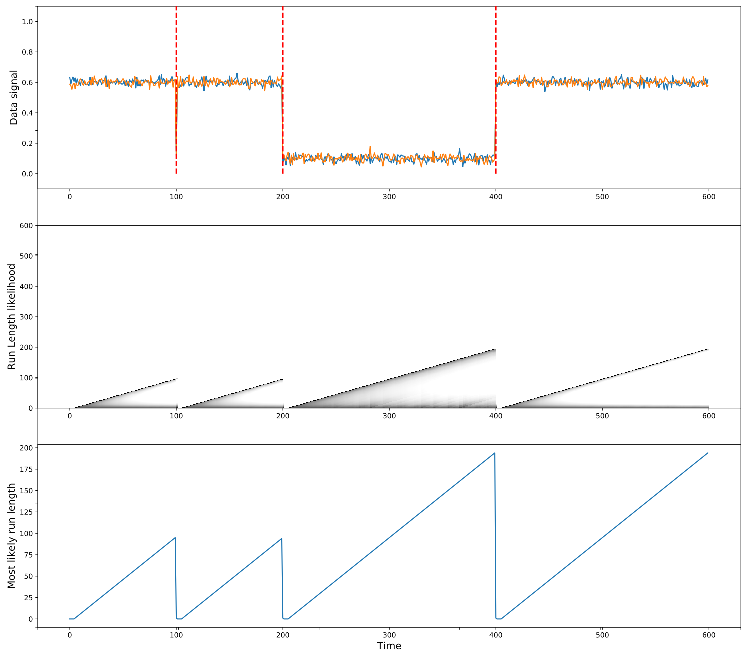


Figure 4.12: Mean-shifts with outliers; BOCPD (weighted univariate features); $s = 200$; $F_1 score = 0.8$

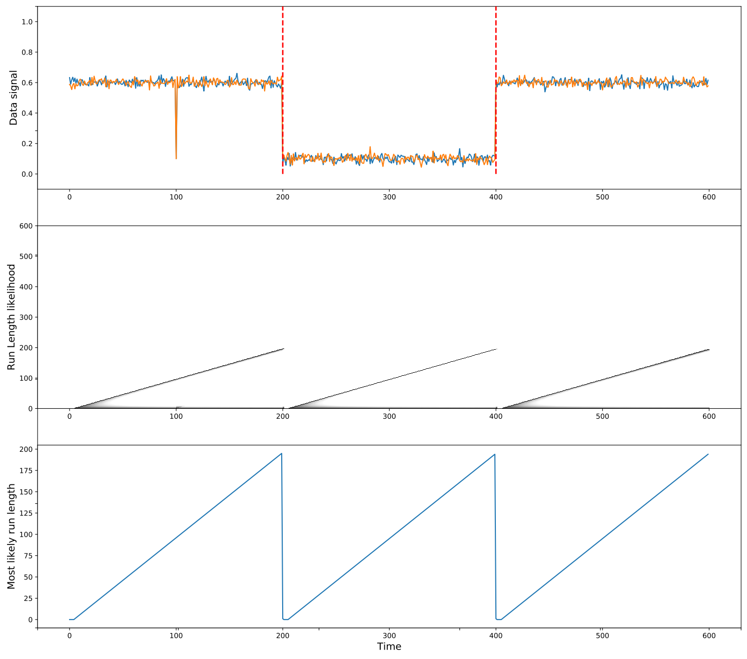


Figure 4.13: Mean-shifts with outliers; BOCPD (multivariate); $s = 200$; $F_1 score = 1.0$

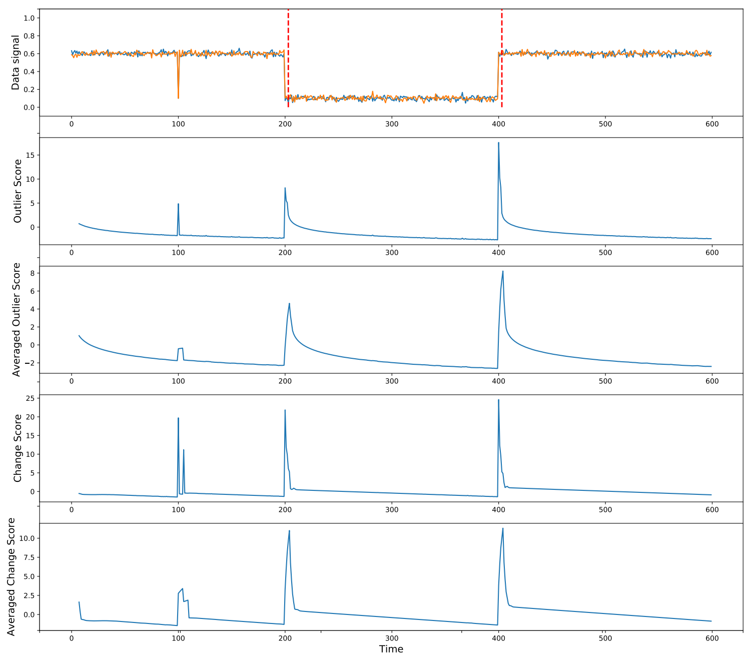


Figure 4.14: Mean-shifts with outliers; CF; $s = 200$; $F_1score = 1.0$

Mean-Shift with Outliers in One Feature

The data of the first features is drawn from p_1 in the first segment, from p_2 in the second one and again from p_1 in the third segment, while the data of the second feature is drawn from p_1 in all three segments.

In the middle diagram of Figure 4.15, we can see that the weighted BOCPD algorithm captures the segment structure very well, but the change likelihood never outperforms the likelihood for a continuous segment starting at time step $t = 0$ and thus, no changepoint is detected.

Figure 4.16 shows, that also the multivariate BOCPD contains some of the segmental structures, though not as clearly as in the weighted case. Again, no changepoints are detected.

The CF algorithm, as shown in Figure 4.17, does not yield any valuable results.

All three algorithms have problems to detect changepoints, that only result from one of two features. This case also shows the importance of feature selection, because the performance increases when irrelevant features are removed from the dataset.

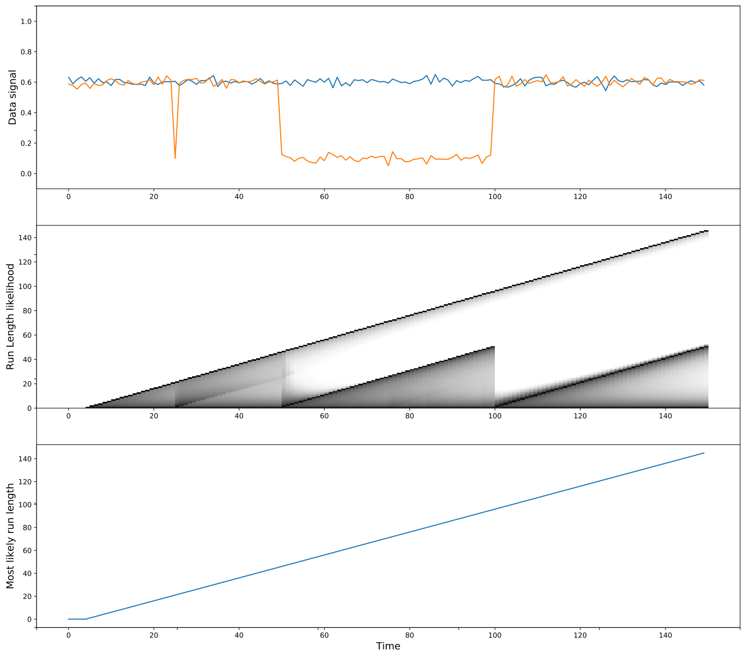


Figure 4.15: Mean-shifts with outliers in one feature; BOCPD (weighted univariate features); $s = 50$; $F_1 score = 0.0$

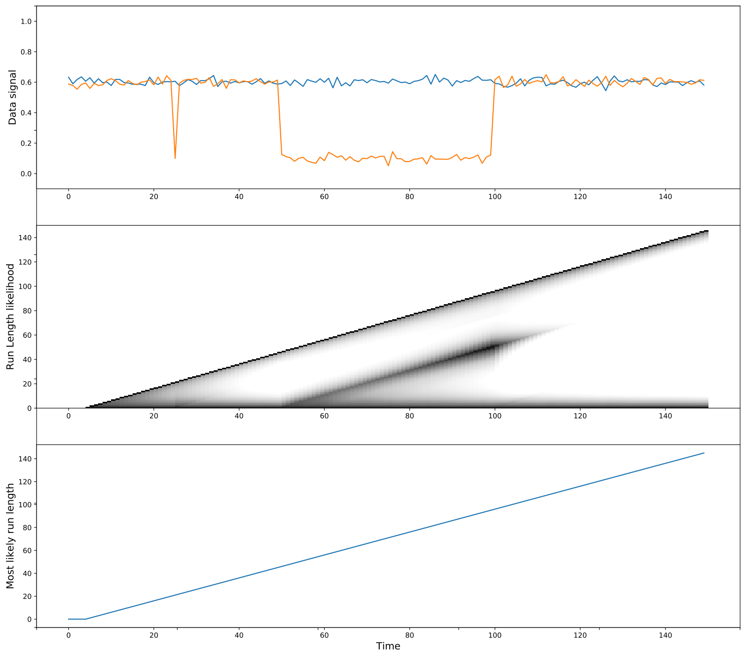


Figure 4.16: Mean-shifts with outliers in one feature; BOCPD (multivariate); $s = 50$; $F_1 score = 0.0$

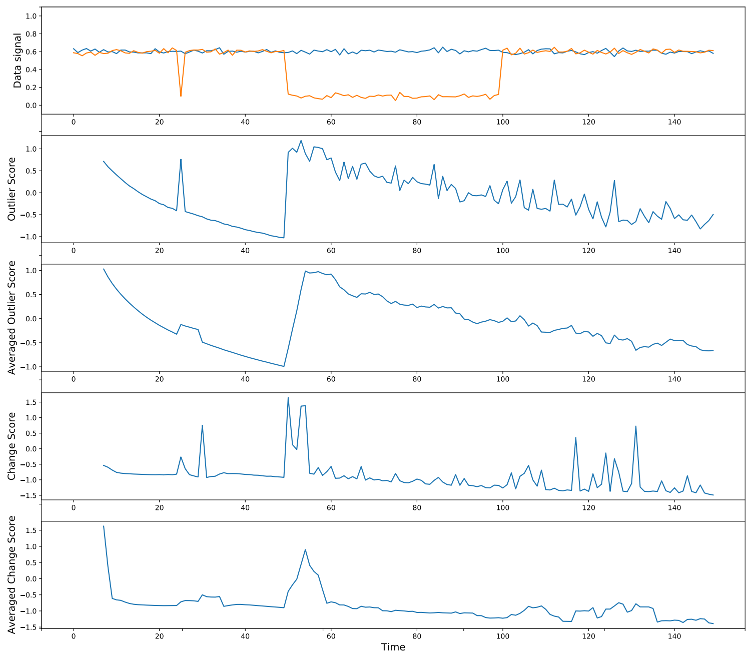


Figure 4.17: Mean-shifts with outliers in one feature; CF; $s = 50$; $F_1 score = 0.0$

Weak Mean-Shift with Outliers

The data of both features is drawn from p_1 in the first segment, from p_3 in the second one and again from p_1 in the third segment. Note, that we have a smaller mean-shift in this case.

Figure 4.18 shows, that the weighted BOCPD handles weak mean-shifts very well. The changepoints are detected with a small lag of $lag_1 = 3$ and $lag_2 = 0$, while the outlier is successfully overlooked.

In Figure 4.19 we can see, that the weighted BOCPD is less sensitive to weak mean-shifts and does not detect any changepoints.

Figure 4.20 shows, that the CF algorithm does not produce valuable output in the given case.

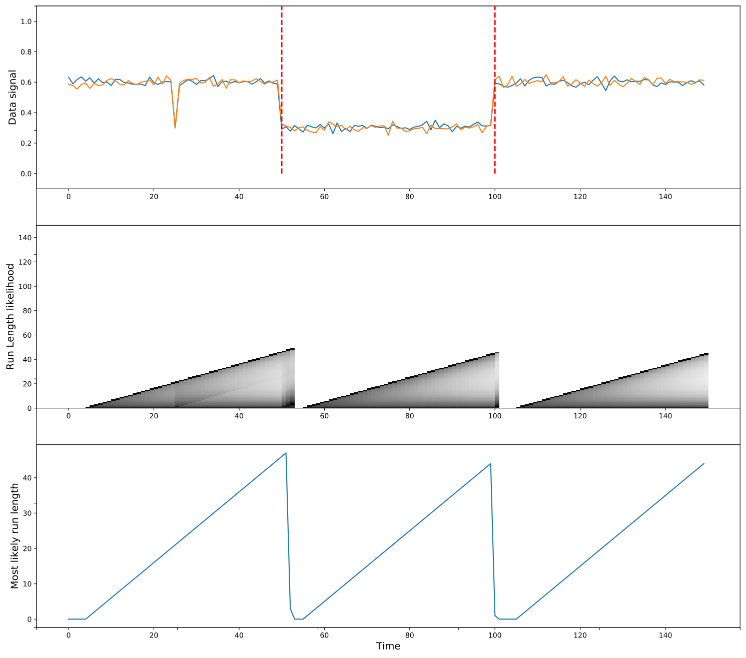


Figure 4.18: Weak Mean-shifts with outliers; BOCPD (weighted univariate features); $s = 50$; $F_1 score = 1.0$

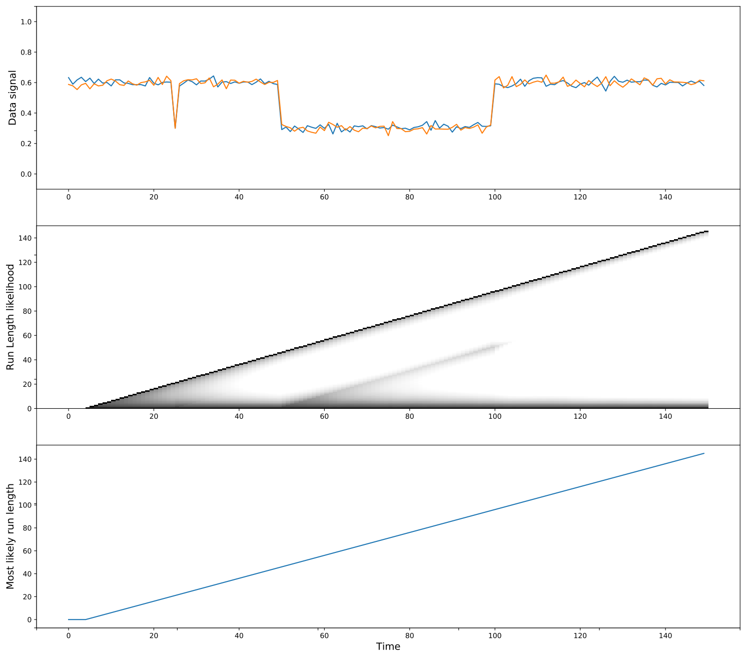


Figure 4.19: Weak Mean-shifts with outliers; BOCPD (multivariate); $s = 50$; $F_1 score = 0.0$

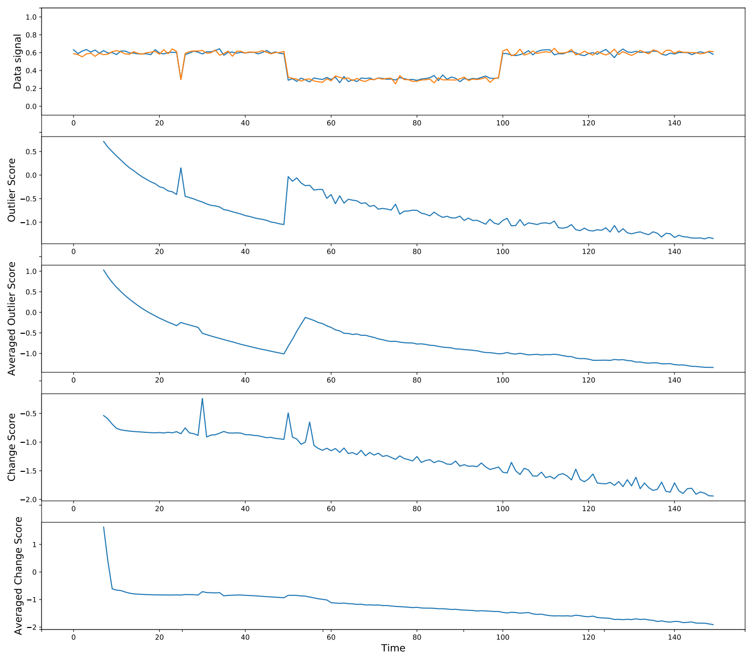


Figure 4.20: Weak Mean-shifts with outliers; CF; $s = 50$*;* $F_1 score = 0.0$

BOCPD with Improved Hazard Function

The data of both features is drawn from p_1 in the first segment, from p_2 in the second one and again from p_1 in the third segment.

Figures 4.21 and 4.22 show the two BOCPD approaches with an improved hazard function (see Section 3.4.2). It assumes the changepoint occurrence to be normally distributed with $\mu = 50$ and $\sigma = 10$. The prior knowledge, which is applied through the hazard function, leads to an improved outlier handling in the case of the weighted BOCPD, and to a reduction of the detection lag in the case of the multivariate BOCPD.

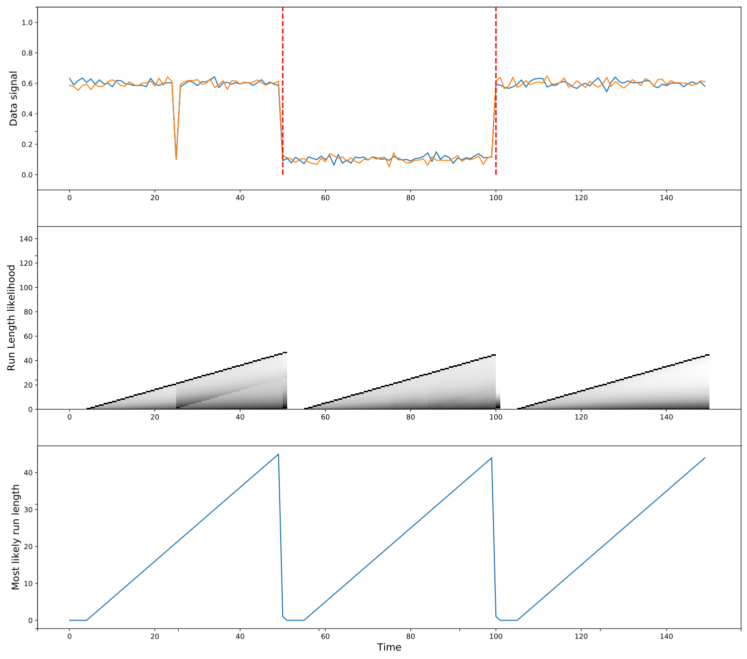


Figure 4.21: BOCPD (weighted univariate features); normal hazard; F_1*score* $= 1.0$

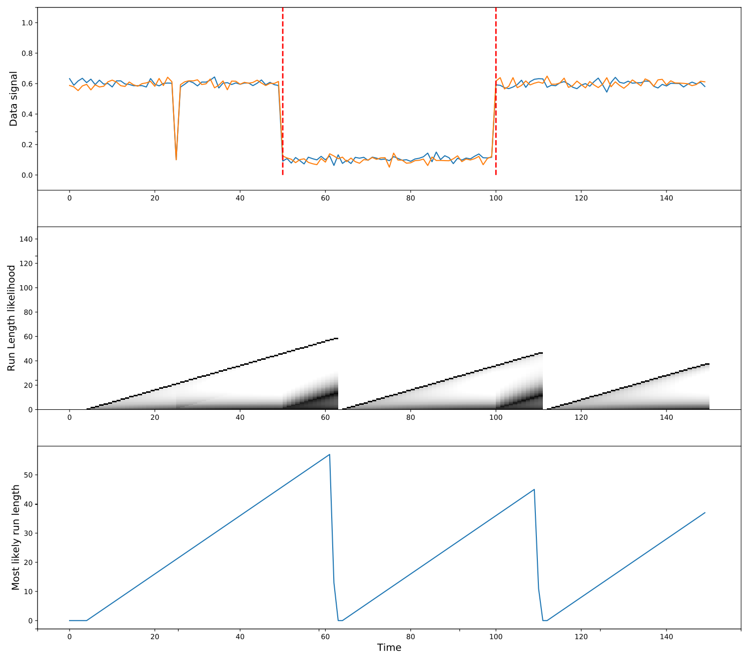


Figure 4.22: BOCPD (multivariate); normal hazard; F_1*score* $= 0.7$

ChangeFinder with Different Window Sizes R

The data of both features is drawn from p_1 in the first segment, from p_2 in the second one and again from p_1 in the third segment.

Figures 4.23 and 4.24 show the CF algorithm with different window sizes $R_1 = 2$ and $R_2 = 10$. We can see, that a small window size of $R = 2$ leads to the detection of mere outliers as changepoints as shown in Figure 4.23. Large window sizes of $R = 10$ on the other hand lead to a large detection lag as shown in Figure 4.24.

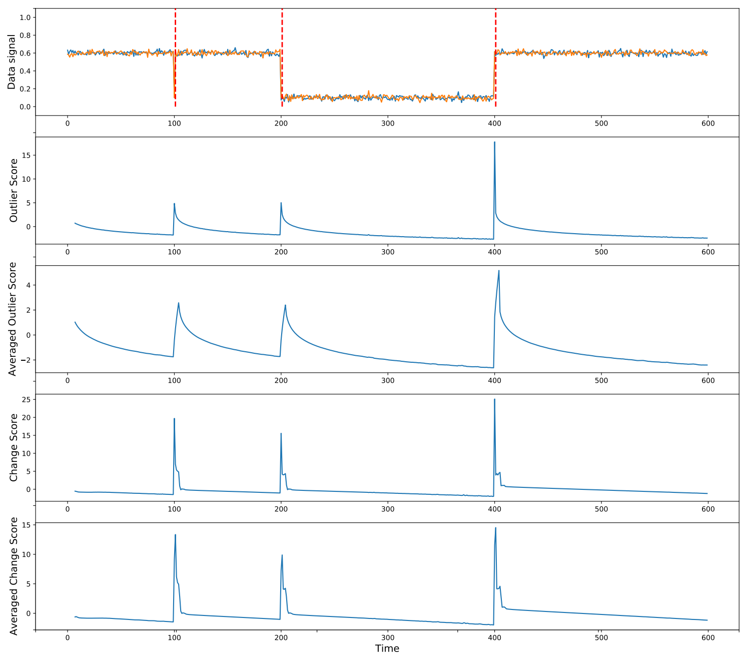


Figure 4.23: CF; $R = 2$; $F_1 score = 0.8$

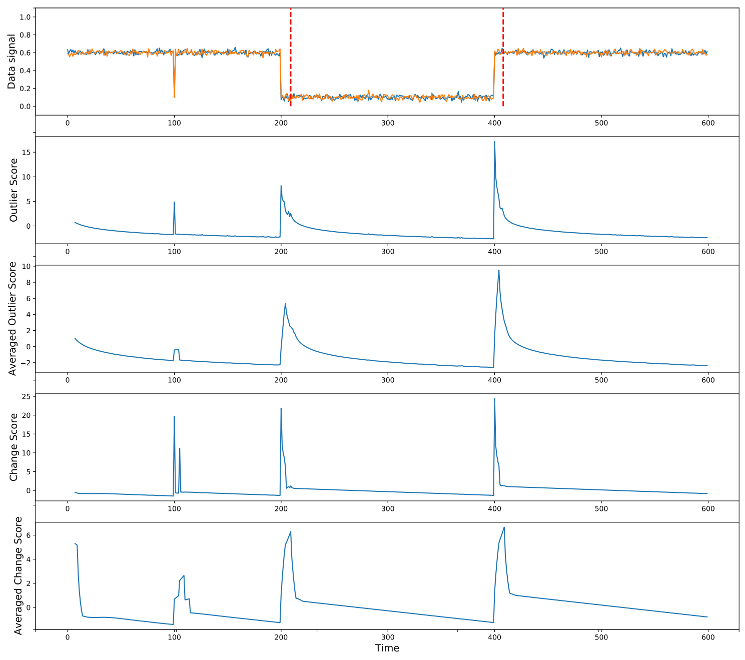


Figure 4.24: CF; $R = 10$*;* $F_1 score = 0.0$

4.1.3 Discussion

The analysis of the presented use cases and parameter configurations leads to the assumption, that the *ChangeFinder* is particularly useful for datasets, where large segment sizes are expected. It handles outliers very well, but has difficulties with small amounts of data. The detection lag and the outlier resistance are in a trade-off relation, which depends on the window size R. The threshold parameter, which ultimately decides whether a change occurred, has to be selected manually, which would be a huge problem for unseen real world data. Thus, the ChangeFinder algorithm is not expected to perform well on the considered data.

The Bayesian approaches perform both very well for the considered cases, even with small samples sizes. Even subtle changes are detected, but with a higher detection lag. The weighted BOCPD is more sensitive to changes and outliers. In all considered cases, the weighted version performed similarly or better than the multivariate BOCPD in terms of accuracy and detection lags. However, the multivariate BOCPD is more resistant to outliers in the given cases. The *hazard* function can be used to improve the performance regarding changepoints and outliers, if prior knowledge about the domain exists.

4.2 General Changepoint Detection in User Behavior

The data discussed in this section is acquired using the *Bip-Up*. It originates from people without any psychological disorders. The aim of analysis and change detection within this data, is to identify events and instants of time where some kind of change in the user behavior occurred. Causes for such a behavioral change might include holidays, relocations, employer changes, the beginning or ending of a relationship, or accidents.

In the following change detection examples the data is loaded and preprocessed in the same manner. The feature selection is performed manually. Features that are assumed to be relevant for a given target and visually appear important, are selected. Additionally, further simple feature selection mechanisms are performed, in order to ignore faulty sensor measurements and features with insufficient data. Samples with all-zero- or extreme values, are removed from the dataset. All features with a zero-variance or a zero-valued eightieth percentile are also removed. The remaining data is then scaled to values between zero and one.

In this section the BOCPD algorithms, the weighted as well as the multivariate version, use a constant hazard function $H(\tau) = \lambda$, because in this scenario no prior knowledge is available. For the same reason the BOCPD with weighted features uses equal weights for all features, so $w = \frac{1}{m}$ where m is the number of features.

4.2.1 Holiday

10.06.2017 - 17.06.2017	Dog visit
29.07.2017 - 05.07.2017	Holiday
22.08.2017 - 03.09.2017	Dog visit
14.09.2017 - 18.09.2017	Holiday
29.12.2017 - 31.01.2018	Holiday

Table 4.1: Targets for data in Figures 4.25, 4.26 and 4.27.

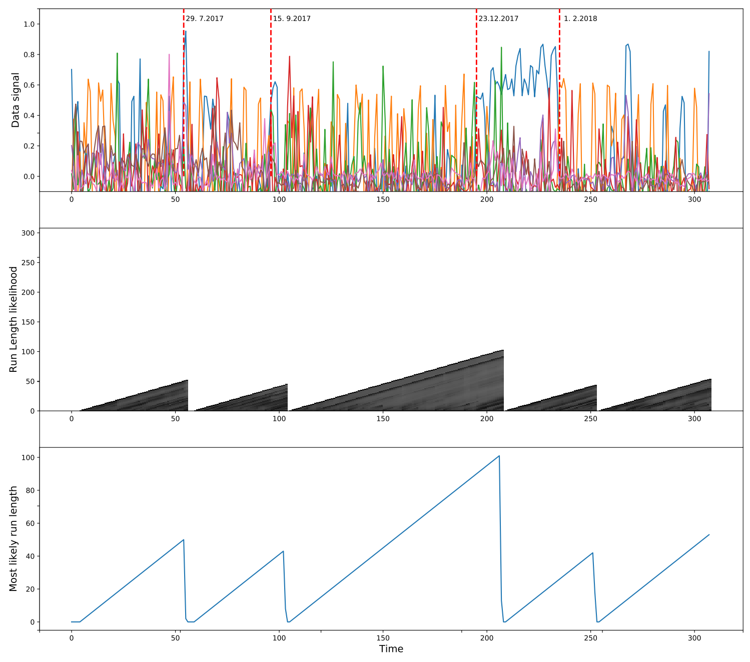


Figure 4.25: Detection of holidays; BOCPD (weighted univariate features); PCA; Features: Work, Home, Any known location, Sleep, Walking, Running, Cycling, Vehicle, Messaging and Phone calls.

Figures 4.25, 4.26 and 4.27 show the successful detection of long-lasting holidays using the weighted BOCPD, the multivariate BOCPD and the CF respectively. All configurations use the same ten features: Work, Home, Any known location, Sleep, Walking, Running, Cycling,

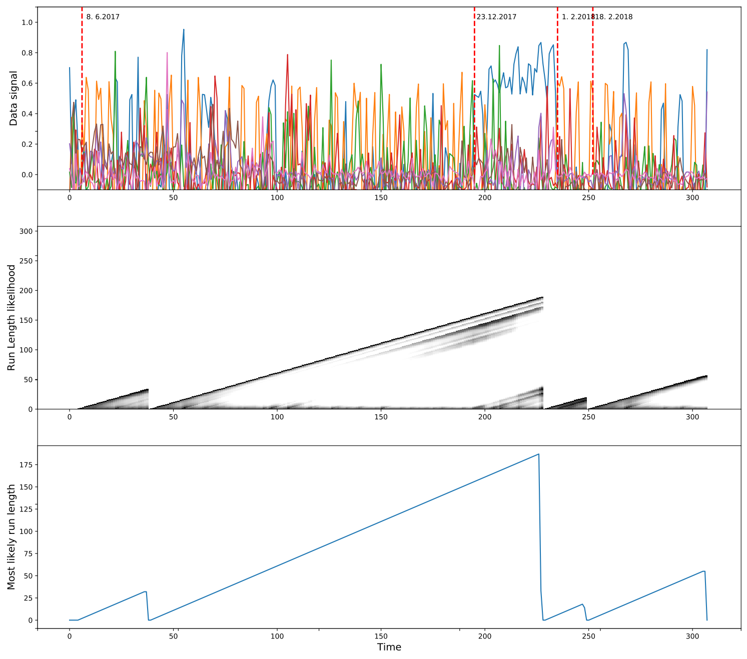


Figure 4.26: Detection of holidays; BOCPD (multivariate); PCA; Features: Work, Home, Any known location, Sleep, Walking, Running, Cycling, Vehicle, Messaging and Phone calls.

Vehicle, Messaging and Phone calls. PCA is applied to the filtered, scaled data. The targets for holidays, and also times where a dog was visiting the user's home, are listed in Table 4.1. One can see, that none of the algorithms detected all holidays and dog visits, but all of them detected the longest holiday and some of the other dates. The BOCPD with weighted features (4.25) detects most of the relevant dates with all the holiday beginnings and the end of the longest one. Note, that the lead time makes is very hard, or even impossible, for the algorithm to detect subsequent changes after a short segment, because the first five days after a changepoint are solely used to learn model parameters and not before then the algorithm again starts to detect changes. The multivariate BOCPD (4.26) detects a change on the very beginning, which might originate from the dog visit beginning at the 10^{th} of June. The last detected changepoint in this figure has no obvious cause. The ChangeFinder (4.27) also detects the large holiday in January as well as the dog visit in June. Additionally the holiday in August is detected. These evaluations suggest, that the BOCPD with weighted features performs best in the given case.

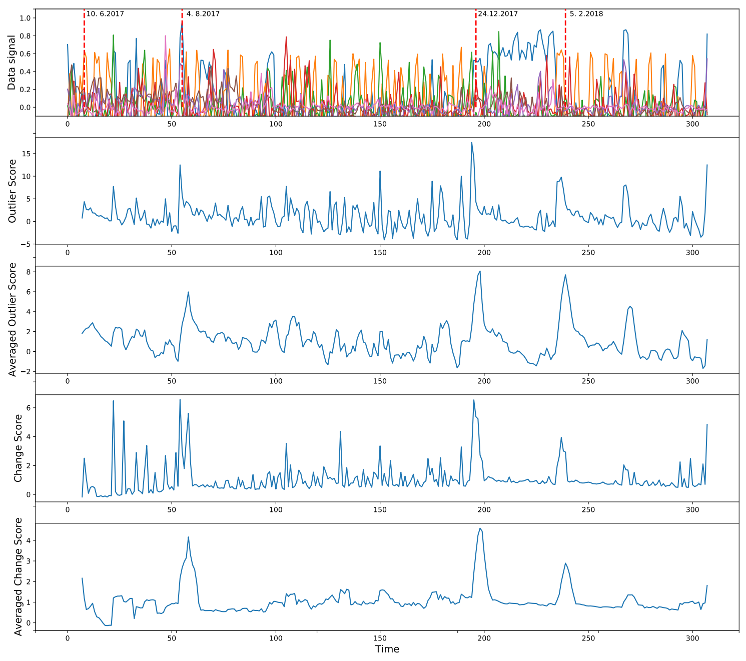


Figure 4.27: Detection of holidays; CF; PCA; Features: Work, Home, Any known location, Sleep, Walking, Running, Cycling, Vehicle, Messaging and Phone calls.

4.2.2 App Release

28.03.2018	App Release

Table 4.2: Targets for data in Figures 4.28, 4.29 and 4.30

Figures 4.28, 4.29 and 4.30 show the detection of an application release date. It is linked to a period of two to three weeks of intense, stressful and overly long working behavior, right before the release date. Here the three features Work, Home, and the application usage type *Other* were used in order to give focus on the relevant data. Table 4.2 contains the target date. Figures 4.28, 4.29 and 4.30 show the output of the weighted BOCPD, the multivariate BOCPD and the CF respectively. The multivariate BOCPD does not detect any changes in the given space of time. Both, the BOCPD with weighted features and the CF, detect a change around the release date. While the weighted BOCPD is closer to the real date than the CF, it also has a rather high detection lag. The BOCPD additionally detects a change 22 days before the release date, which might originate from the beginning of the intense pre-release phase. Again, the BOCPD with weighted features seems to perform best for this use case.

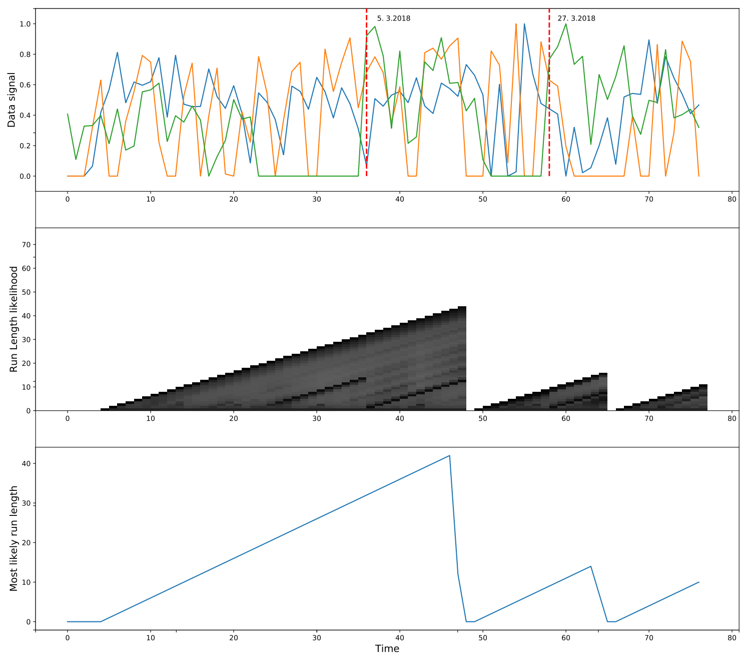


Figure 4.28: Detection of an App Release date; BOCPD (weighted univariate features); Features: Work, Home and Application type "Other".

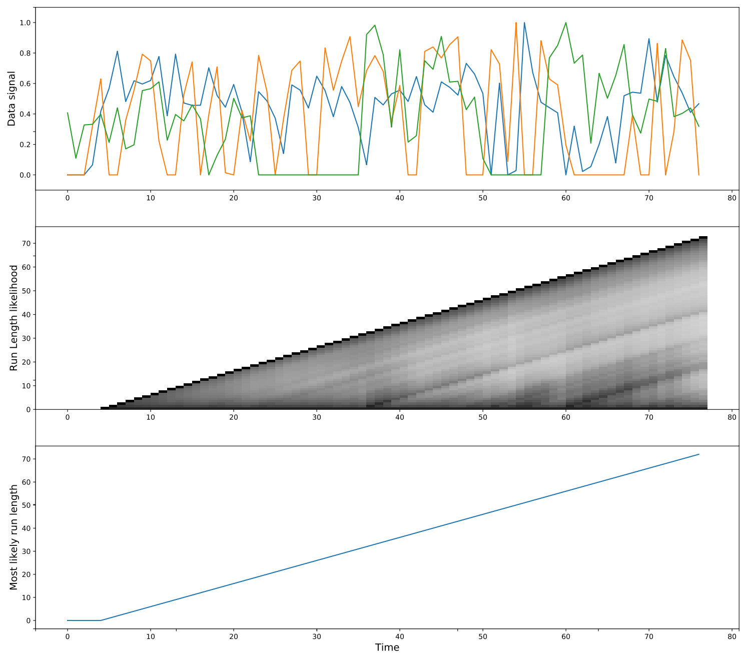


Figure 4.29: Detection of an App Release date; BOCPD (multivariate); Features: Work, Home and Application type "Other".

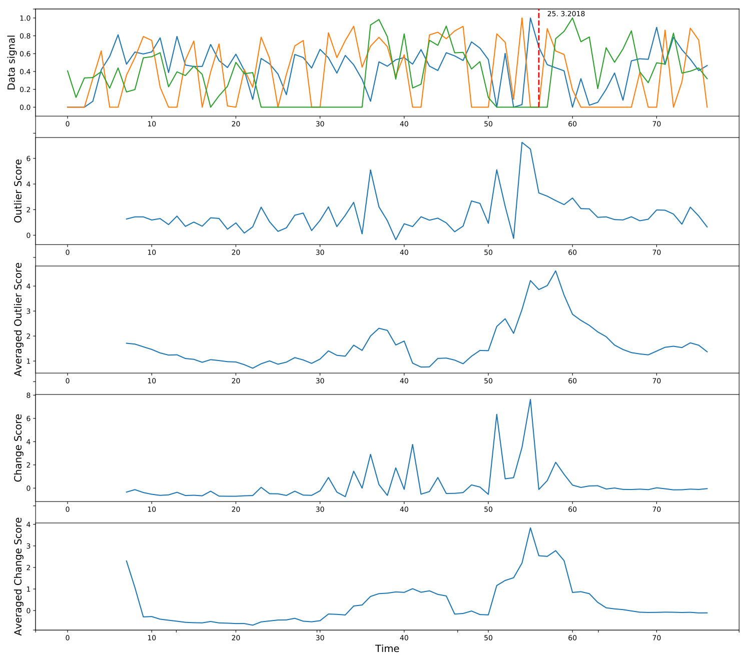


Figure 4.30: Detection of an App Release date; CF; Features: Work, Home and Application type "Other".

4.2.3 Relationship

17.09.2017	Beginning of relationship
6.11.2017	Breakup

Table 4.3: Targets for data in Figure 4.31

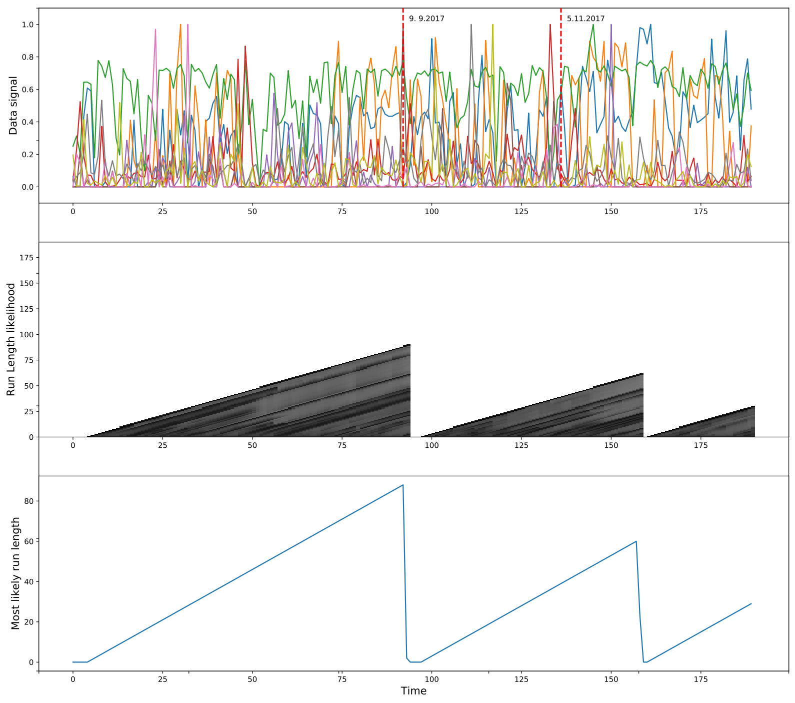


Figure 4.31: Detection of the start and end of a relationship; BOCPD (weighted univariate features); Features: Home, Work, Any known location, Walking, Running, Cycling, Vehicle, Messaging and Phone calls.

Figure 4.31 shows the detection of a relationship, from its beginning to the breakup. The nine features Work, Home, Any known location, Walking, Running, Cycling, Vehicle, Messaging and Phone calls were used. Table 4.3 contains target dates. The BOCPD with weighted features was the only one of the evaluated algorithms to identify these dates. A changepoint was detected one day after the target breakup date. Another changepoint was detected around one week prior to the target date for the beginning of the relationship, while the user indicated, that already around that time the relationship began to form.

4.2.4 Ligament Rupture

Figures 4.32 and 4.33 show the detection of a ligament rupture alongside the detection of holidays. The eight features Home, Work, Walking, Running, Vehicle, Messaging, Phone call and Social Media were used. Table 4.4 contains the ligament rupture and holiday dates. Figure 4.32 shows the BOCPD with weighted features, it detects a change one day after the ligament rupture and two changes at the beginning and the end of holidays. Figure 4.33 shows the CF

21.08.2017	Ligament rupture
10.09.2017 - 15.09.2017	Holiday
26.10.2017 - 01.11.2017	Holiday
27.01.2018 - 04.02.2018	Holiday
31.03.2018 - 07.04.2018	Holiday

Table 4.4: Targets for data in Figures 4.32 and 4.33.

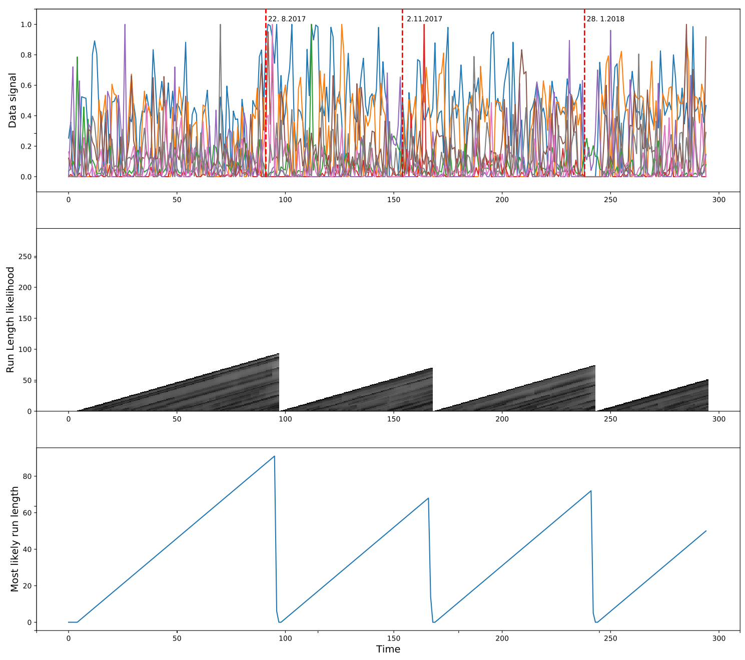


Figure 4.32: Detection of a ligament rupture; BOCPD (weighted univariate features); Features: Home, Work, Walking, Running, Vehicle, Messaging, Phone call and Social Media.

algorithm applied to the same dataset. It detects a change two days after the ligament rupture and additionally three changes around the beginning or ending of holidays, and one change that has no obvious cause.

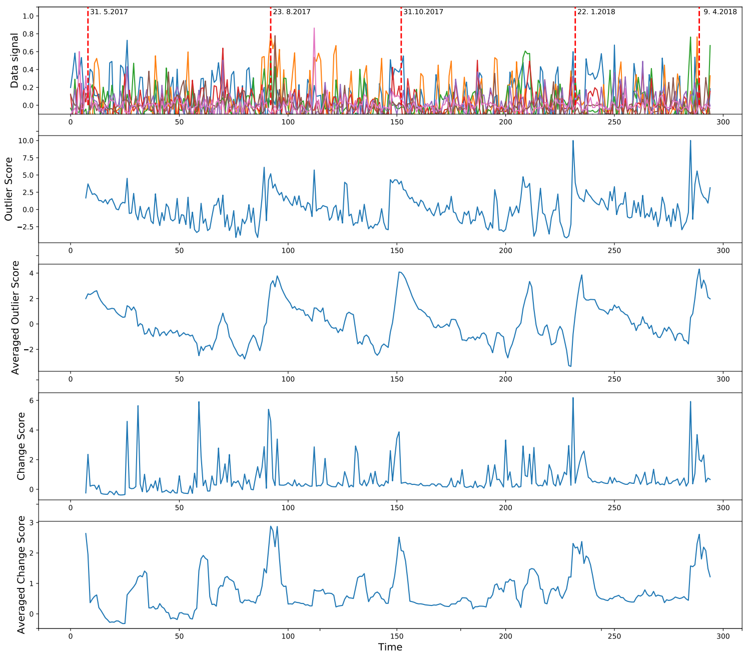


Figure 4.33: Detection of a ligament rupture; CF; PCA applied; Features: Home, Work, Walking, Running, Vehicle, Messaging, Phone call and Social Media.

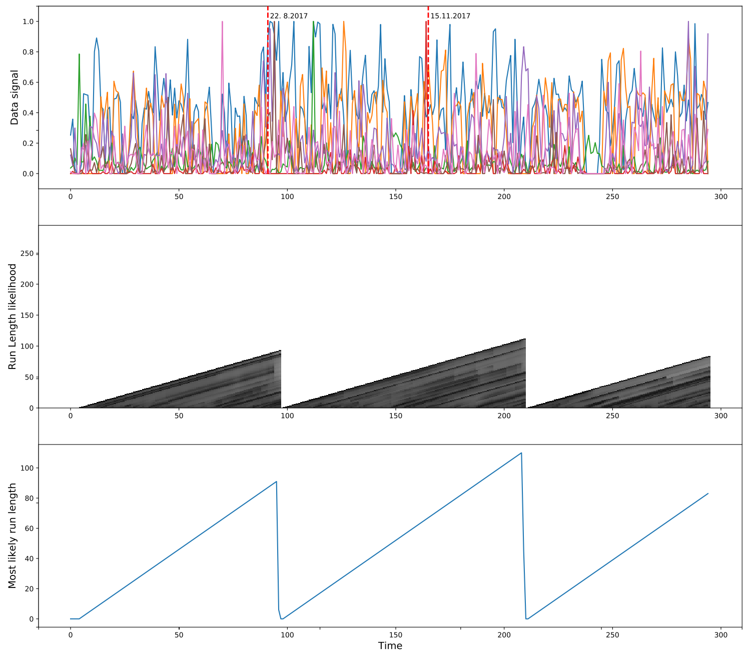


Figure 4.34: Detection of a ligament rupture without feature Vehicle; *BOCPD (weighted univariate features).*

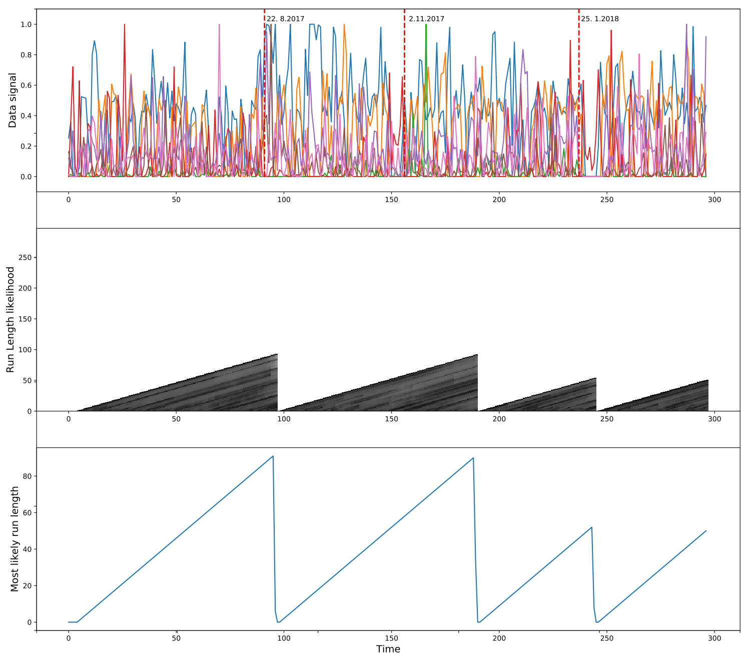


Figure 4.35: Detection of a ligament rupture without feature Walking*; BOCPD (weighted univariate features).*

Figures 4.34, 4.35 and 4.36 contain the same dataset as before and show the BOCPD with weighted features in every subfigure. It displays experiments with the features needed for correct detection of the ligament rupture. Figures 4.34 and 4.35 show the outputs when the features *Vehicle* and *Walking* are excluded respectively. Both configurations yield a changepoint at the same instant of time as before, which is one day after the ligament rupture. When all physical activity features are excluded from the dataset, no changepoint associated with the ligament rupture, but instead another change with no obvious cause, is found.

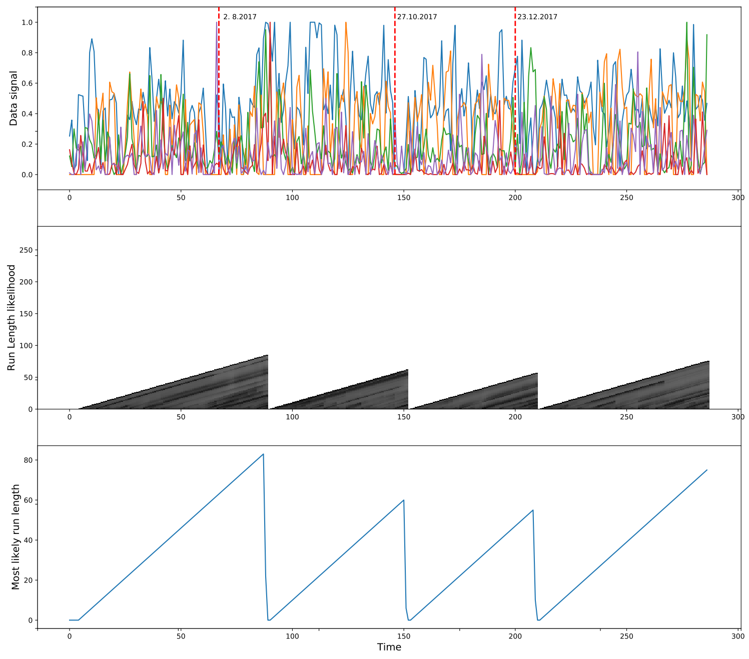


Figure 4.36: Detection of a ligament rupture without any physical activity features; BOCPD (weighted univariate features).

4.3 Detection of State Changes in Bipolar Disorder

The data discussed in this section is acquired using the *Bip-Up* in the context of a clinical study (described in Section 2.6). The data originates from people with psychological disorders, in particular the bipolar disorder. The aim of analysis and change detection within this datasets is to identify bipolar disorder state changes, such as the start or end of a depressive-, manic- or normal phase. The data is loaded and processed in the same manner as in the previous section.

The feature selection is performed manually. Features that are assumed to be relevant for a given target and visually appear important, are selected. Additionally, further simple feature selection mechanisms are performed, in order to ignore faulty sensor measurements and features with insufficient data. Samples with all-zero- or extreme values, are removed from the dataset. All features with a zero-variance or a zero-valued eightieth percentile are also removed. The remaining data is then scaled to values between zero and one.

In this section the BOCPD with weighted features uses equal weights for all features, so $w = \frac{1}{m}$ where m is the number of features. The hazard function for both Bayesian approaches is based on a normal distribution, in order to add prior knowledge about the frequency of state changes in bipolar disorder.

So far, the study is in progress for more than eight months (of two years in total) and comprises ten participants. However, the test phase for each participant starts in stages. Because the first few participants were from the control group (without any psychological disorders), there is only very limited relevant data available at this point. All bipolar patients have only datasets for around one month. This is a very limited amount of data for the investigated algorithms, because they require some time to learn the user behavior of the current state and according to that, detect changes. Additionally, only datasets containing relevant state changes according to the received targets can be used to investigate the algorithmic performance. This makes the evaluation very difficult and the results sparse. The targets for bipolar state changes originate from the results of questionnaires and clinical assessments of the patients during one of the doctor visits.

4.3.1 Successful Detection of a Manic State Change

08.03.2018 - 20.03.2018	Depression
21.03.2018 - 23.03.2018	Normal
24.03.2018 - 25.03.2018	Mania
26.03.2018 - 04.04.2018	Normal

Table 4.5: Targets for data in Figure 4.37.

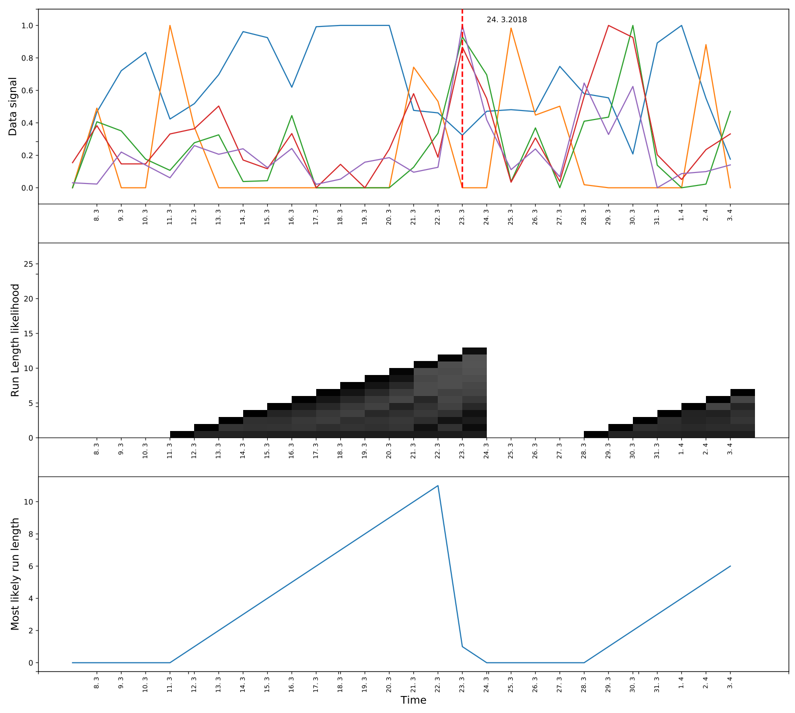


Figure 4.37: Detection of a manic state; BOCPD (weighted); Features: Work, Home, Untagged locations, Walking and On foot.

Figure 4.37 shows the detection of the beginning of a short manic phase. The data contains all three states of the bipolar disorder, as indicated in Table 4.5. The recording started on 08.03.2018 and is available until 04.04.2018. The depressive phase starts at the very beginning of the recording, on 08.03., and lasts until 20.03. There is a very short normal state until 24.03., where the patient changes to a manic state. After 25.03. the participant again was evaluated to reside in a normal state. The weighted BOCPD found a change at 24.03., the beginning of the manic phase. Due to the early start of the depression phase (at the first day of recording) and the quick state changes, no other changes were detected. Here the features Work, Home, Untagged locations, Walking and On foot were used.

4.3.2 Undetected Depressive State Change

14.03.2018 - 22.03.2018	Depression
14.03.2018	Positive event *Other*: Internship
20.03.2018	Positive event *Finance*
02.04.2018	Positive event *Celebration*
04.04.2018	Negative event *Conflict*

Table 4.6: Targets for data in Figures 4.38, 4.39 and 4.40.

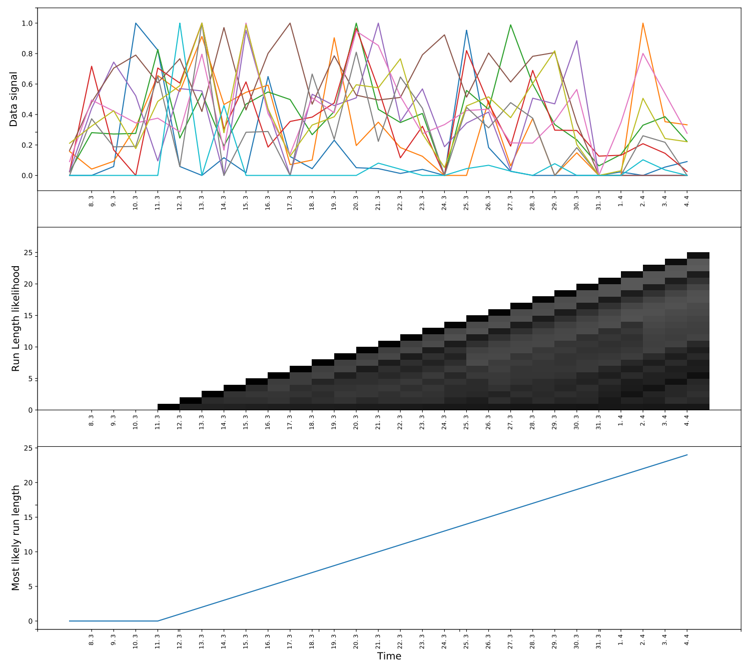


Figure 4.38: No changepoints detected; BOCPD (weighted); Features: Untagged locations, Friends & Family, Walking, On foot, Vehicle, Cycling, Phone, Social Media, Messaging and Other applications.

Figures 4.38, 4.39 and 4.40 show a case, where none of the three investigated algorithms found any changes. Table 4.6 contains all relevant states as well as positive- and negative events, according to the medical assessment. The depressive phase lasts from 14.03.2018 until 22.03.2018 and various incisive events, spread within the observation period, were noted. Note, that also no optical recognition of changes or events are possible according to the measured data. The variety of events and the short observation period make it very difficult to properly learn the user behavior in the current state and detect any changes.

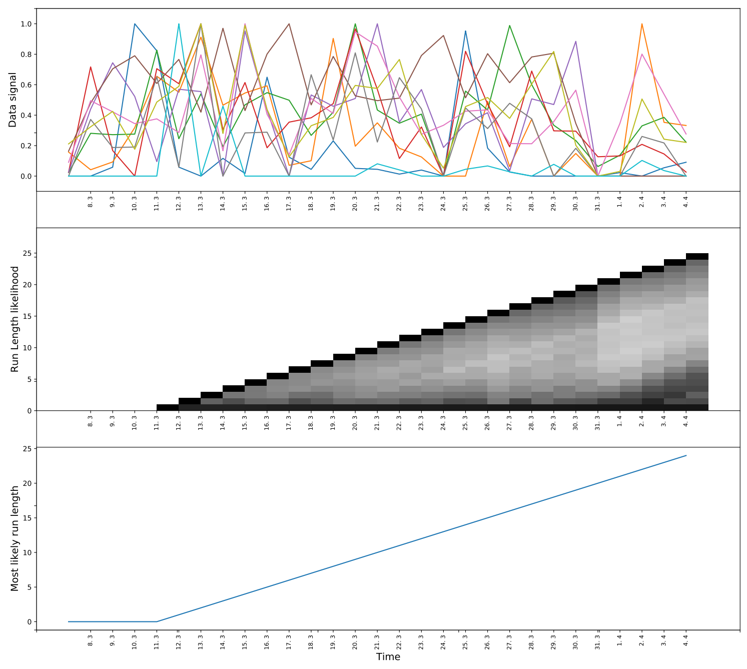


Figure 4.39: No changepoints detected; BOCPD (mv); Features: Untagged locations, Friends & Family, Walking, On foot, Vehicle, Cycling, Phone, Social Media, Messaging and Other applications.

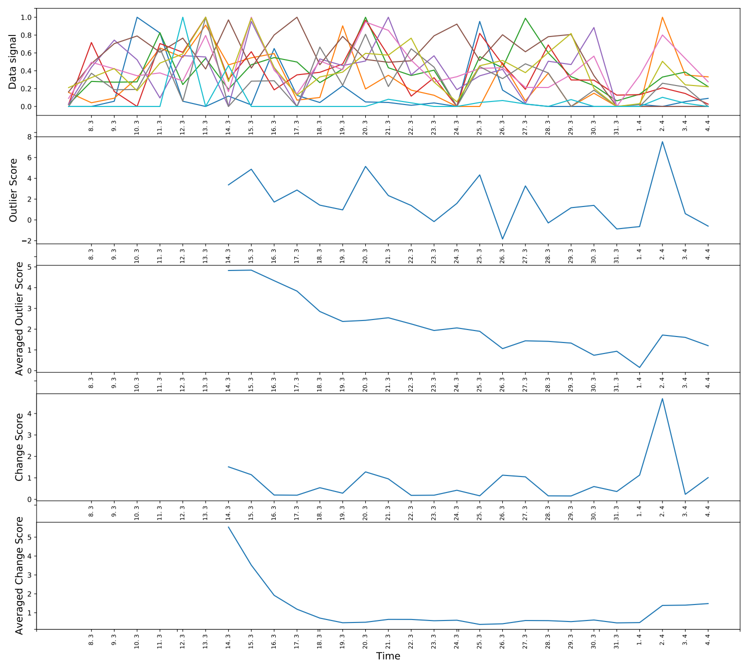


Figure 4.40: No changepoints detected; CF; Features: Untagged locations, Friends & Family, Walking, On foot, Vehicle, Cycling, Phone, Social Media, Messaging and Other applications.

4.4 Discussion

The results obtained by smartphone usage data of ordinary people (Section 4.2) and bipolar patients (Section 4.3) are promising.

As already expected, based on the results of specific use cases for synthetic data (Section 4.1), the CF algorithm only performs well on long time series data with long segments. Also the hyperparameter selection states a problem for the considered data. The *threshold* parameter had to be chosen manually, were the optimal value differs strongly for each evaluated dataset.

The BOCPD algorithm performs quite well on the considered cases. The weighted version finds most of the targets and usually has small to moderate detection lags. The multivariate version of the BOCPD often seems to capture the structure of segments, but does not always detect the changepoints correctly. For the BOCPD no selection of hyperparameters is required in order to obtain viable results. The possibility to improve the algorithmic performance using prior knowledge about the domain is a huge advantage. It is used to give focus on the bipolar state changes in Section 4.3. Also the feature selection is considered to be very important; the less irrelevant features are contained within the dataset, the better are the results. We have also seen, that a small lead time is important to find short segments, because during the lead time no changepoints can be detected.

The weighted BOCPD clearly appears to be the best choice for detecting changes in smartphone usage data. For the generic detection of changes in the user behavior a constant hazard is appropriate, while a hazard that is based on a normal distribution, improves the results for bipolar state changes. PCA can be applied if many relevant features are contained in the dataset, but otherwise I would recommend using the original ones.

5 Conclusion

The aim of this thesis was to automatically detect state-changes of people with bipolar disorder by considering smartphone usage data.

Data of the three categories physical activity, such as walking or cycling, labeled locations, such as home or work, and application statistics, such as phone calls or messaging, is recorded by a mobile application for Android, which is called *Bip-Up*. It is essentially a digital diary, that enables users to obtain an objective overview of their daily activities. It can be used by ordinary people, but it was particularly developed for people with bipolar disorder. In order to react to state changes in this psychological illness, and therefore improve the treatment, it is essential to objectively assess behavioral changes. In order to address this problem, the application of an automatic change detection to smartphone usage data was investigated in this thesis. The application implemented within this context imports the data from the smartphone application and contains all further steps, required to perform an automatic changepoint detection. The data is processed to obtain plausible features, which are then selected and extracted to only represent relevant information within the data. The *ChangeFinder* and *Bayesian Online Changepoint Detection* algorithms were implemented and evaluated.

In principle both investigated algorithms can be applied to smartphone usage data and are capable of detecting certain changes within this data.

The *ChangeFinder* algorithm is fast and simple to implement with a clear interpretation of each intermediate step of calculations. Here, we have a trade-off relation between the detection lag, which is the time between the target changepoint and the detected one, and the resistance to detecting outliers. Varying the window size allows the configuration of this trade-off, in order to fit the algorithm to the considered application. The larger the window, the higher is the detection lag and the less is the risk to identify outliers as changes. Note, that the detection lag has an upper limit, i.e. the window size, because a change is only detected at the current time step in the ChangeFinder scheme. This algorithm performs poorly on short time series as well as data with short segments. Because of this, and especially because it requires manual parameter selection, for the change threshold as well as some model parameters, it is not expected to perform well on the considered data.

The *Bayesian Online Changepoint Detection* algorithm mostly performs better on our data, especially for recordings with shorter segments. With a larger number of samples per segment, the algorithm gets more sensitive to changepoints and outliers. Single extreme values can then lead to a change detection in the current time step, so it is more prone to outliers than the ChangeFinder. The BOCPD also detects subtle changes, but with higher lags. Because this algorithm is capable of detecting changepoints retrospectively, as soon as the evidence predominates, there is no upper limit for the detection lag. Here, prior knowledge about the domain and the type of changes that are of interest, can be used to influence the outcome. With more profound prior knowledge about the frequency of changepoints, the results are improved noticeable. Also, it can be used to configure the algorithm to preferentially find certain types of changes, according to a known frequency of such events. In our case, this can be used to tune the algorithm with regard to bipolar states changes. All in all the BOCPD appears to be suited better for our data and provides the possibility to give focus on specific changes using prior knowledge without requiring any manual parameter selection.

The change detection was applied to three different datasets. Synthetic data was used to assess the algorithmic performance according to predefined use cases, such as outlier handling and changepoint detection in the univariate- and multivariate case. The algorithms were further applied to real world data. First, they were applied to find changes in the user behavior of ordinary people without psychological illnesses. Incisive events that implicate a change of behavior, such as holidays, accidents or changes in the relationship status, were considered and successfully detected. The second dataset originates from a clinical study, that is still in progress at the time of this writing. It evaluates the correctness of recorded data as well as the applicability and acceptance of the *Bip-Up* through the target audience. The targets for dates and durations of bipolar states are evaluated by the clinical personnel, due to questionnaires and psychological assessments during medical visits of the patients. Due to the resent start of the study, only a limited amount of participants with long-lasting records is available. As expected, BOCPD works better for our data, especially the promising results in Sections 4.2 and 4.3 suggest that the implemented approach can be used to automatically detect bipolar state changes.

5.1 Outlook

For a more reliable application of the proposed change detection algorithm, several improvements are presented below. First, a profound evaluation of the algorithms with an application to extensive data of bipolar users including targets is required, such that the algorithmic performance can be estimated.

Further, the prior knowledge about the change frequency in bipolar disorder can be improved. This could be achieved by not only considering general expert knowledge, but additionally including the results of a user's self-assessment as well as medical assessments relating to a specific user. So the patient could be questioned about his or her medical history regarding the change frequency and duration of bipolar states. Additionally, state targets per user, if obtained by clinical personnel, could be used to further fine-tune the prior knowledge about changes for that specific user.

Manual and semi-supervised feature selection methods can be used to improve the relevance of the selected feature subset. A self-assessment of the user, according to his or her known early warning signs, could be used to further select, or rank, relevant features. Further, features could be analyzed, manually or automatically, according to state targets obtained by clinical personnel, if available. Features are considered more valuable, for which learned distributional parameters differ strongly when comparing samples that are close to the targets with samples that are far from them. Note, that a feature ranking could be used for choosing the weights of the BOCPD with multiple, univariate features.

Additionally the algorithms themselves could be improved by some simple modifications. The BOCPD can be prevented from detecting changes in the first step, which might be due to a single extreme observation. In order to do so, a change is detected only after the same new changepoint was suggested by the algorithm for a certain amount of steps, for instance three time steps. This results in a higher detection lag, though. In the case, where the change is found slowly in a retrospective manner, and the detection lag is already higher, this is not necessary and thus could only be applied in the case where the current detection lag would be smaller than some predefined threshold. In this way the BOCPD would be more resistant to outliers and only loose detection accuracy in controversial cases.

By remembering a set of previous sample values, the lead time could be shortened in some cases. When a changepoint is detected with some detection lag, these values could be used to learn the new model and reduce the downtime of the algorithm. This way, changepoints partitioning shorter segments could be detected.

Bibliography

[1] A. Grünerbl, A. Muaremi, V. Osmani, G. Bahle, S. Öhler, G. Tröster, O. Mayora, C. Haring, and P. Lukowicz, "Smartphone-based recognition of states and state changes in bipolar disorder patients," *IEEE Journal of Biomedical and Health Informatics*, vol. 19, no. 1, pp. 140–148, 2015.

[2] U. McCormick, B. Murray, and B. McNew, "Diagnosis and treatment of patients with bipolar disorder: A review for advanced practice nurses," *J Am Assoc Nurse Pract.*, vol. 27, no. 9, pp. 530–542, 2015.

[3] L. Tondo, G. H. Vazquez, and R. J. Baldessarini, "Depression and mania in bipolar disorder," *Current neuropharmacology*, vol. 15, no. 3, pp. 353–358, 2017.

[4] Y. B. R H Belmaker, "Bipolar disorder: Mania and depression," *Discovery Medicine*, vol. 4, no. 23, pp. 239–245, 2014.

[5] G. Inc. (2018) Android developer guide. [Online]. Available: https://developer.android.com/guide/topics/permissions/index.html

[6] A. Inc., *iOS Security*, 2018. [Online]. Available: https://www.apple.com/business/docs/iOS_Security_Guide.pdf

[7] A. Grünerbl, P. Oleksy, G. Bahle, C. Haring, J. Weppner, and P. Lukowicz, "Towards smart phone based monitoring of bipolar disorder," in *mHealthSys '12*, 2012.

[8] S. Aminikhanghahi and D. J. Cook, "A survey of methods for time series change point detection." *Knowledge and information systems*, vol. 51, no. 2, pp. 339–367, 2017.

[9] (2018) Google apis for android. [Online]. Available: https://developers.google.com/android/reference/com/google/android/gms/location/DetectedActivity

[10] (2018) Android developer's reference guide. [Online]. Available: https://developer.android.com/reference/android/app/usage/UsageEvents.Event.html

[11] (2018) Google apis for android. [Online]. Available: https://developers.google.com/android/reference/com/google/android/gms/location/FusedLocationProviderClient

[12] (2018) Google apis for android. [Online]. Available: https://developers.google.com/android/reference/com/google/android/gms/location/LocationRequest

[13] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *Kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[14] P. Mitra, C. A. Murthy, and S. K. Pal, "Unsupervised feature selection using feature similarity," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 3, pp. 301–312, March 2002.

[15] H. L. Wei and S. A. Billings, "Feature subset selection and ranking for data dimensionality reduction," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 1, pp. 162–166, Jan 2007.

[16] C. M. Bishop, *Pattern Recognition and Machine Learning*, ser. Information Science and Statistics. Springer, 2006.

[17] J. Takeuchi and K. Yamanishi, “A unifying framework for detecting outliers and change points from time series,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 4, pp. 482–492, April 2006.

[18] R. P. Adams and D. J. C. MacKay, “Bayesian online changepoint detection,” Cambridge, UK, 2007.

[19] R. Adhikari and R. K. Agrawal, *An Introductory Study on Time Series Modeling and Forecasting*. LAP Lambert Academic Publishing, 2013.

[20] A. Papoulis, *Probability & Statistics*, ser. Prentice-Hall international editions. Prentice Hall, 1990. [Online]. Available: https://books.google.at/books?id=HQ3vAAAAMAAJ

[21] R. P. Adams and D. J. MacKay, “Bayesian online changepoint detection,” http://hips.seas.harvard.edu/content/bayesian-online-changepoint-detection, 2017.

[22] J. Kulick, “Bayesian changepoint detection,” https://github.com/hildensia/bayesian_changepoint_detection, 2016.

[23] I. Lauzana, N. Figueroa, and J. Medina, “Bayesian online multivariate changepoint detection algorithm,” https://github.com/epfl-lasa/changepoint-detection, 2017.