Real-time unsupervised log event anomaly detection in public transportation

Felicia Segui
Andreas Timürtas

LUND UNIVERSITY

Department of Automatic Control

MSc Thesis
TFRT-6184
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Abstract

Detecting log data anomalies in real-time is useful since it makes it possible to apply logic that corrects the anomalies when they happen. This project presents a method for detecting public transportation bus event log data anomalies in real-time, without having a labeled data set. Initially, each unique bus trip is represented by the event frequencies, a representation that is not suitable for real-time. With a data set assumed to only contain normal data, an autoencoder, a PCA model and a clustering algorithm label each data point in the frequency domain, as normal or anomalous. The labeled data is split into sequences of events with a rolling window, a representation that is suitable for detecting anomalies in real-time. To separate the anomalous event sequences from the normal event sequences that occur, during the same bus trip as an anomalous event sequence, the event sequences together with their labels are grouped and counted. By comparing the frequency for each event sequence in anomalous trips with the frequency of the corresponding event sequence in normal trips, the sequences that are overrepresented in anomalous trips are detected and receive a final label being normal or anomalous. These labeled sequences are further used in the real-time detector. With the three base labeling models (autoencoder, PCA and clustering algorithm), different combinations of models are created. These models are either created by applying the union or the intersection of all anomalous labeled journeys. This results in 11 different models that are all tested and evaluated. The evaluation is performed by calculating the recall, precision and F1-score of experiments performed with a data set of assumed normal journeys, together with injected simulated anomalies. The evaluation is performed at two places within the method; one after the initial labeling and another after the real-time detector. The results obtained using this evaluation method show that the combination using the autoencoder and the clustering algorithm together through intersection is the best model combination, based on the F1-score calculated after the real-time detection. This combination scores a median recall and precision of 0.89 respectively 0.72, which results in an F1-score of 0.79.

Acknowledgements

We would like to thank Hans Ekström and Sara Olsson at Gaia as well as Johan Eker at the Department of Automatic Control, LTH, for continuously giving us valuable help and guidance. Without their support this thesis would not have been possible. Also, thank you to Linnea Mellblom at Gaia for providing us with detailed information about the data, as well as contributing with ideas when we encountered challenges. A special thanks to our opponents Isabelle Frodé and Viktor Sambergs for reviewing our thesis and giving us suggestions for improvements. Finally, we would like to thank our examiner Anton Cervin, for being very helpful whenever we had questions.

Contents

1. Introduction **9**
- 1.1 Background . . . 9
- 1.2 Outline of thesis . . . 10
- 1.3 Objective of project . . . 10
- 1.4 Related work . . . 11
- 1.5 Data . . . 14
- 1.6 Individual contributions . . . 16

2. Theory **17**
- 2.1 Scaling and normalization of data . . . 17
- 2.2 PCA . . . 19
- 2.3 Clustering algorithms . . . 20
- 2.4 Autoencoder . . . 29
- 2.5 Evaluation metrics . . . 30

3. Method **33**
- 3.1 Problem formulation . . . 33
- 3.2 Pipeline . . . 34
- 3.3 Evaluation with simulated anomalies . . . 49

4. Results **53**
- 4.1 Precision, recall and F1-score . . . 53
- 4.2 Result summary . . . 64

5. Discussion **65**
- 5.1 Frequency-based representation versus sequence-based representation . . . 65
- 5.2 Metrics and parameters . . . 66
- 5.3 Result analysis . . . 67
- 5.4 Future work . . . 71

6. Conclusions **73**

Bibliography **75**

1

Introduction

In this chapter background information needed to understand the motivation and the challenges of the thesis is presented, as well as previous work done within the field.

1.1 Background

Together with many other industries, public transportation is currently undergoing a major digital transformation. The manual work of creating timetables, announcing the next stop to the passengers and choosing suitable routes for the journeys is replaced with advanced logistic systems, automatic systems that manage the vehicle's audio callouts and displays as well as optimization algorithms that in real-time present the most efficient route to the next bus stop, for the driver. Together with the implementation of new digital tools come large amounts of data in different forms and for different types of use. Some data is almost static, such as available roads and timetables, and some data is highly dynamical, for instance the actual arrival or departure times to stops for an active vehicle.

The dynamical data is often stored in logs, in which a row corresponds to a certain happening that took place at a certain time on a certain journey. These happenings are called events, and when a new event occurs, a row is added to the log with information such as the time of occurrence, which journey it occurred on and varying event specific information. An event could be anything from internal state changes, to changing the displayed text at the front of the outside of the bus. This type of real-time streaming of events opens up for having a continuous dialog with the vehicle. One example of this could be that the vehicle automatically displays the next stop after departing from the previous stop, based on the geographical position of the vehicle, instead of changing display at a predetermined time based on the static timetable. The latter example would cause confusion every time the bus is late or early, in contrast to the first example of system which adapts to the current situation.

Another situation in which you want to be able to act directly during the trip is when something does not go as expected. For instance, this could be a vehicle that displays *Not in traffic* during a journey, or announcing the wrong next stop to the passengers. If these faults are detected in short time after they happen, it would be possible to fix the mistakes before they make any harm. Due to an increasing complexity of these kinds of systems, it can be difficult to know exactly what the root cause is and where in the software architecture it occurs. A natural first step in the work towards minimizing faulty behavior of the system would be to develop a model that notifies the user when something anomalous occurs. A challenge in creating a such model is to define what an anomaly is. The perfect model would be generic enough to capture anomalies that the user is not aware of, but not to the point that it also starts to label normal data as anomalous. How to define an anomaly for the model without specifying exactly what an anomaly is, makes the problem not only mathematically and statistically hard to solve, but also philosophically complex. A major part of contributing scientifically with something new is to be able to evaluate it. However, designing a suitable evaluation method for an anomaly detection model could feel like a paradox when the purpose of the model is to detect anomalies that no one knows exist.

1.2 Outline of thesis

This thesis starts with an introduction in Chapter 1 that includes a brief background of the project, what the authors aim to accomplish, the division of work as well as a section with related work that should give an idea of how this project contributes scientifically and how it relates to previous work. The chapter also includes the objective of the project together with the research questions that the thesis intend to answer, as well as an overview of the data used in the thesis. In Chapter 2 the theory behind the tools and algorithms used in the experiments is described. A thorough description of the problem is included in the beginning of the Chapter 3. This chapter also includes the method of the project and the problem formulation presented in Section 3.1 describes why the method is especially important for this this thesis. In Chapter 4 the results from the experiments are presented, in Chapter 5 the discussion and analysis are placed and Chapter 6 contains the conclusions.

1.3 Objective of project

As stated in Section 1.1, there is a need of detecting event log anomalies in real-time for public transportation vehicles. Gaia is one of many companies that wishes to find such anomalies in an early stage. This thesis aims to develop a method for real-time event log anomaly detection, with having access to a very limited amount of known anomalous event sequences. As mentioned earlier, a consistent and transparent evaluation of such a method is crucial. Consequently, the suitable choices that appear along the way of developing a method for this problem are constrained by the ability to evaluate the method as a whole. Within the limits for valid evaluation, we intend to develop, compare and analyze different variations of the method and finally present the optimal one.

1.3.1 Questions at issue

The thesis primarily aims to answer the following research questions:

- How can a real-time detector for sequences of log events be defined?
- How well does the above mentioned real-time detector perform, when only having access to an unlabeled data set?
- How can a small data set of known anomalous trips be used for tuning, in the pipeline belonging to the real-time detector, while keeping a generic nature of the detector where it also detects anomalies outside the known anomalies?

1.4 Related work

It is easy to find research articles considering one of the subjects of public transport, event logs, anomaly detection, real-time models and unsupervised learning. Even the majority of combinations of two of the subjects is earlier researched and written about. However, projects including three or four of the fields are definitely less common and no project considering all areas, as this thesis does, has been found while researching. In the following sections a selection of relevant articles is presented, with comments on how they differ and relate to this thesis.

1.4.1 What defines an anomaly?

An anomaly can be defined in a range of different ways, and in most cases it needs to be defined in context of the surrounding data. In the paper *On the nature and types of anomalies: a review of deviations in data* by [Foorthuis, 2021] an extensive literature study is presented and the different types of anomalies are categorized and analyzed. The paper defines an anomaly as "occurrences in a data set that are in some way unusual and does not fit the general patterns". Important in the definition is the lack of an interesting detail; anomalies does not have to be constructed from an underlying faulty structure, as long as the occurrence is either unusual or does not fit the normal pattern. However, an anomaly detector meant for applications similar to the one in this thesis aspires to detect anomalies originating from faulty systems. Therefore, an anomaly in this project is defined similarly to the definition of Foorthuis with the addition that the occurrence also needs to be unwanted, not only an outlier within the representation of the data. This leaves room for interpretation of what *unwanted* means and it is something that needs to be specified by the users that aim to develop an anomaly detector.

1.4.2 Log anomaly detection for streaming data in real-time

In the paper *Unsupervised real-time anomaly detection for streaming data* [Agha, 2017] a model is presented that is able to detect anomalies in real-time streaming data using unsupervised methods. The detector uses an algorithm called Hierarchical Temporal Memory, which is a type of online sequence memory algorithm. The paper also presents a novel benchmark used for anomaly detectors specified on streaming data called the *Numenta Anomaly Benchmark*. The main difference between the problem described in the paper and the problem described in this project is that our data is naturally divided into journeys. The data in the paper is fully streaming, meaning that no natural sub-sequences can be assigned. In our problem the data can be separated through the natural journeys a vehicle performs, a format that is incompatible with the previously mentioned benchmark. *Unsupervised real-time anomaly detection on streaming data for large-scale application deployments* [Jernbäcker, 2019] is another article within the same field that uses the *Numenta Anomaly Benchmark* for evaluating the detector. For detecting anomalies in both real and simulated data, a combination of *Hierarchical temporal memory*, *Restricted Boltzmann machines* and *Autoregressive integrated moving average* is used. The detectors were chosen because each one of them is specialized to detect a certain type of anomaly. With a similar intention, this thesis also combines, as well as compares, three different detector algorithms. However, due to the lack of labels in the thesis's data, as well as the data structure that is incompatible with the benchmark, the detector used in this thesis is supplemented with other tools that enable evaluation as well as detection in real-time.

1.4.3 Anomaly detection for sequence data

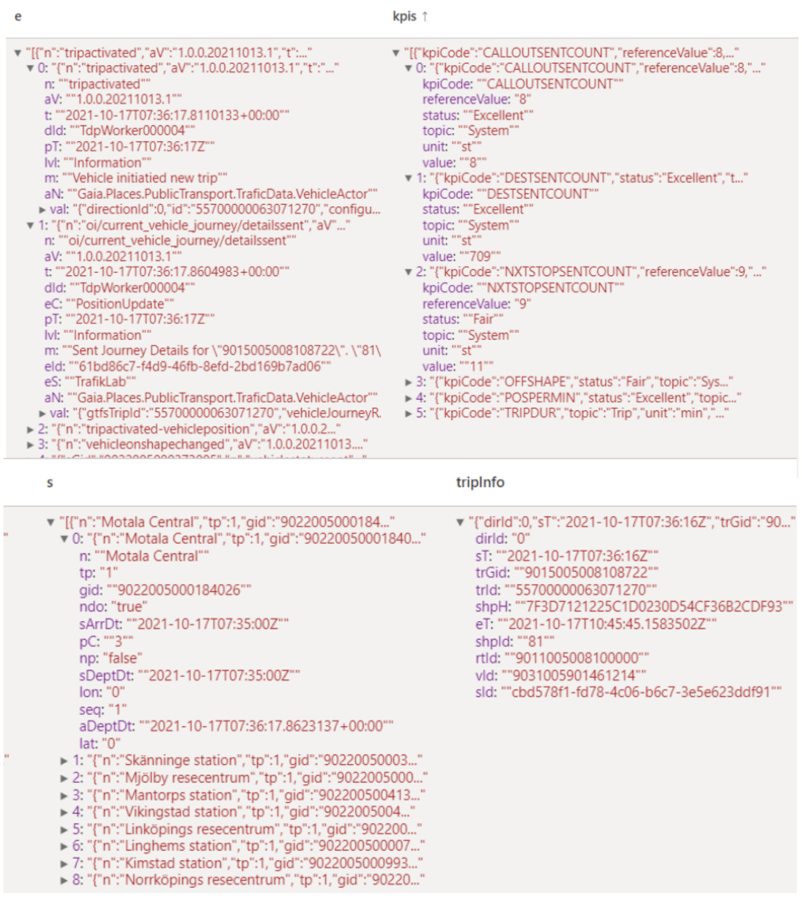
In the paper *Comparative Evaluation of Anomaly Detection Techniques for Sequence Data* [Kumar, 2008] different anomaly detectors are presented. These detectors are implemented to work on a range of different data sets. Similar to the problem in this thesis the data used in the paper has a sequential structure, where each sequence consists of a number of consecutive logs. The paper brings forth different techniques that are implemented to solve the problem. These techniques are divided into three different categories; *kernel based*, *window based* and *Markovian*. The *kernel based* techniques build on the idea that anomalies can be detected by comparing the similarities between sequences. *Window based* techniques however calculate an anomaly score for fixed length windows, in a sequence. The last category, the *Markovian*, calculates a probabilistic anomaly score for each event given the previous events. Connecting this article to our thesis, a sequence would correspond to an event log for a bus journey, and a window in a sequence would correspond to a number of consecutive events within a bus journey. The pipeline presented in this thesis includes a model similar to the *kernel based* algorithms as well as a model with a similar approach as the *window based* algorithms. The thesis's pipeline could be seen as an ensemble model, since the two previously mentioned models, in contrast to the algorithms described in the paper, operate after each other and are both necessary for the solution as a whole.

1.4.4 Evaluation of unsupervised models on unlabeled data

A common problem with unsupervised models when used on unlabeled data is that an evaluation of the model can be unreliable and difficult. One approach that is often used when a novel unsupervised model is presented is to train the model on unlabeled data and then evaluate the model using existing labels. An example of this is found in the article *Clustering-based real-time anomaly detection—A breakthrough in big data technologies* [Ariyaluran Habeeb et al., 2019] that proposes a real-time framework for detecting anomalies in big data, with clustering algorithms. Although the clustering algorithms handle unlabeled data, the performances of the algorithms are evaluated by comparing the predicted labels with the actual labels. However, to do this, the original data needs to be labeled, something that can not be assumed to be the case in real world applications. To manually label a data set is often very time-consuming, and the person that does the job needs to have a knowledge about the data as good as the ground truth. The paper *A Large-scale Study on Unsupervised Outlier Model Selection: Do Internal Strategies Suffice?* [Akoglu, 2017] investigates how accurate different existing evaluation approaches for unlabeled data performs on anomaly detector tasks. The paper compared 297 different models based on 8 different detectors on 39 detection tasks using 7 unrelated strategies to compare and evaluate the models. The 7 strategies can be divided into to two categories; stand-alone and consensus-based. Stand-alone strategies only require a single model and the corresponding output while consensus-based strategies rely on multiple models and the agreement between the models. The conclusion reached in the paper is that no strategy performs significantly better than to simply use the state-of-the-art detector with a random configuration. Furthermore, the stand-alone strategies did not perform significantly better than simply choosing a random model from the existing pool of models. The problem with evaluating a model with unlabeled data is present in the problem described in this thesis. While evaluation strategies utilizing internal and inherit data information are used in our thesis, the returned metrics are not enough for presenting how the model performs in a trustworthy way. To be able to present trustworthy results the final models are evaluated by using simulated anomalies.

1.5 Data

The provided data from Gaia consists of historical event logs from buses operating under the public transportation service Östgötatrafiken. The data of a journey is stored in a JSON file. There is a lot of information included in the logs, in addition to the *trip ID*, *timestamp* and *event name*. A screenshot of a small proportion of the nestled data can be seen in Figure 1.1.



```
e
▼ "[{"n":"tripactivated","aV":"1.0.0.20211013.1","t":..."
  ▼ 0: "{"n":"tripactivated","aV":"1.0.0.20211013.1","t":"..."
      n: ""tripactivated""
      aV: ""1.0.0.20211013.1""
      t: ""2021-10-17T07:36:17.8110133+00:00""
      dId: ""TdpWorker000004""
      pT: ""2021-10-17T07:36:17Z""
      lvl: ""Information""
      m: ""Vehicle initiatied new trip""
      aN: ""Gaia.Places.PublicTransport.TraficData.VehicleActor""
    ▸ val: "{"directionId":0,"id":"5570000063071270","configu...
  ▼ 1: "{"n":"oi/current_vehicle_journey/detailssent","aV"..."
      n: ""oi/current_vehicle_journey/detailssent""
      aV: ""1.0.0.20211013.1""
      t: ""2021-10-17T07:36:17.8604983+00:00""
      dId: ""TdpWorker000004""
      eC: ""PositionUpdate""
      pT: ""2021-10-17T07:36:17Z""
      lvl: ""Information""
      m: ""Sent Journey Details for \"9015005008108722\". \"81\
      eId: ""61bd86c7-f4d9-46fb-8efd-2bd169b7ad06""
      eS: ""TrafikLab""
      aN: ""Gaia.Places.PublicTransport.TraficData.VehicleActor""
    ▸ val: "{"gtfsTripId":"5570000063071270","vehicleJourneyR.
  ▸ 2: "{"n":"tripactivated-vehicleposition","aV":"1.0.0.2..."
  ▸ 3: "{"n":"vehicleonshapechanged","aV":"1.0.0.20211013...."

kpis ↑
▼ "[{"kpiCode":"CALLOUTSENTCOUNT","referenceValue":8,..."
  ▼ 0: "{"kpiCode":"CALLOUTSENTCOUNT","referenceValue":8,"..."
      kpiCode: ""CALLOUTSENTCOUNT""
      referenceValue: "8"
      status: ""Excellent""
      topic: ""System""
      unit: ""st""
      value: ""8""
  ▼ 1: "{"kpiCode":"DESTSENTCOUNT","status":"Excellent","t..."
      kpiCode: ""DESTSENTCOUNT""
      status: ""Excellent""
      topic: ""System""
      unit: ""st""
      value: ""709""
  ▼ 2: "{"kpiCode":"NXTSTOPSENTCOUNT","referenceValue":9,"..."
      kpiCode: ""NXTSTOPSENTCOUNT""
      referenceValue: "9"
      status: ""Fair""
      topic: ""System""
      unit: ""st""
      value: ""11""
  ▸ 3: "{"kpiCode":"OFFSHAPE","status":"Fair","topic":"Sys..."
  ▸ 4: "{"kpiCode":"POSPERMIN","status":"Excellent","topic..."
  ▸ 5: "{"kpiCode":"TRIPDUR","topic":"Trip","unit":"min","..."

s
▼ "[{"n":"Motala Central","tp":1,"gid":"9022005000184..."
  ▼ 0: "{"n":"Motala Central","tp":1,"gid":"90220050001840..."
      n: ""Motala Central""
      tp: "1"
      gid: ""9022005000184026""
      ndo: "true"
      sArrDt: ""2021-10-17T07:35:00Z""
      pC: ""3""
      np: "false"
      sDeptDt: ""2021-10-17T07:35:00Z""
      lon: "0"
      seq: "1"
      aDeptDt: ""2021-10-17T07:36:17.8623137+00:00""
      lat: "0"
  ▸ 1: "{"n":"Skänninge station","tp":1,"gid":"90220050003..."
  ▸ 2: "{"n":"Mjölby resecentrum","tp":1,"gid":"9022005000..."
  ▸ 3: "{"n":"Mantorps station","tp":1,"gid":"902200500413..."
  ▸ 4: "{"n":"Vikingstad station","tp":1,"gid":"9022005004..."
  ▸ 5: "{"n":"Linköpings resecentrum","tp":1,"gid":"902200..."
  ▸ 6: "{"n":"Linghems station","tp":1,"gid":"902200500007..."
  ▸ 7: "{"n":"Kimstad station","tp":1,"gid":"9022005000993..."
  ▸ 8: "{"n":"Norrköpings resecentrum","tp":1,"gid":"90220..."

tripInfo
▼ "{"dirId":0,"sT":"2021-10-17T07:36:16Z","trGid":"90..."
    dirId: "0"
    sT: ""2021-10-17T07:36:16Z""
    trGid: ""9015005008108722""
    trId: ""5570000063071270""
    shpH: ""7F3D7121225C1D0230D54CF36B2CDF93""
    eT: ""2021-10-17T10:45:45.1583502Z""
    shpId: ""81""
    rtId: ""9011005008100000""
    vId: ""9031005901461214""
    sId: ""cbd578f1-fd78-4c06-b6c7-3e5e623ddf91""
```

Figure 1.1 Small proportion of the nestled log data, accessed from JSON files.

In Table 1.1 and Table 1.2 below the information fields used for this project are presented. Since the raw structure of the data is nestled, the parent field is included in the name when the child field is explained. For the project, data from 14th to 20th October 2021 was extracted and used.

Table 1.1 A collection of the data fields used in this project. e[x] represents the x:th event in a journey and s[x] represents the x:th stop in a journey.

Field	**Description**
e	The log events for one bus trip. Each entity of e contains a nestled numbered list of the events in the bus trip.
kpis	Statistics obtained after the journey ended.
s	The journey's bus stops, with information about the scheduled and actual arrival and departure times.
tripInfo	General information about the trip.
tripInfo.trId	Also called trip ID, which is an ID for the journey that is unique for each bus line and time of the day. This means that every journey on the same line and time of the day have the same trip ID, even when the journeys take place on different dates.
tripInfo.rtId	Also called route ID, which is unique for each bus line.
kpis[0].referenceValue	Also called CallOutRefValue, which is the expected number of times of callout executions. The callout is the sound announcement that notices the passengers about the next bus stop.
kpis[0].value	Also called CallOutValue, which is the actual number of times that a callout is executed during a trip. The callout is the sound announcement that notices the passengers about the next bus stop.
e[x].val.metresToNextStop	The distance to next stop, only available for events that are related to the next stop.
kpis[4].value	Also called TripDurationMinutes, which is the time in minutes from the beginning of the logging of the journey, to the last log message of the journey.
kpis[3].value	Also called PosPerMin, which is the number of position updates of the bus per minute.
e[x].t	The timestamp of event number x.
e[x].n	The event name of event number x.
e[x].m	The event message for event number x.

Table 1.2 A collection of the data fields used in this project. e[x] represents the x:th event in a journey and s[x] represents the x:th stop in a journey.

Field	Description
e[x].val.metresToNextStop	The time to next stop, only available for events that are related to the next stop.
e[x].val.fromState	A field that contains the state that the bus changes from. Only available for events that represents changes of the state, for instance when a bus changes state from *Arriving* (to the bus stop) to *Departing* (from the bus stop).
e[x].val.toState	A field that contains the state that the bus changes to. Only available for events that represents changes of the state, for instance when a bus changes state from *Arriving* (to the bus stop) to *Departing* (from the bus stop).
e[x].val.onShape	A boolean field that tells if the bus deviates from the planned geographical route or not.
e[x].val.VehicleState	A field with the current state of the bus, in relation to the previous or next bus stop.
e[x].val.stopId	A unique ID of the bus stop, only available for events that are directly related to a bus stop.
e[x].val.isFinalDestination	A boolean field that is true when the bus reached the final destination, only available for events that are directly related to a bus stop.
e[x].val.EventStopName	The name of the stop, only available for events that are directly related to a bus stop.
e[x].val.signage_v1.text	A field for the display message shown at the display outside the bus. Only available for events related to that display.
s[x].aDeptDt	The timestamp for the actual departure for the x:th bus stop.
s[x].gid	The ID for the x:th bus stop.
s[x].n	The name of the x:th bus stop.

1.6 Individual contributions

In this thesis both authors have contributed equally to the majority of the parts of the project. Since both authors started the project with a similar background of studying Machine Intelligence, the most natural way of working was side by side. In the development part Andreas was responsible for the autoencoder, while Felicia was responsible for the clustering algorithms.

2 Theory

The intention with this chapter is to give the reader background information and theory that is valuable for the research as well as for the experiment part of this thesis. In Section 2.1 the data scaling and normalization are presented, while the model algorithms are presented in Section 2.2, 2.3 and 2.4. The last section, Section 2.5, describes the evaluation metrics used in the thesis.

2.1 Scaling and normalization of data

When working with a data set that has multiple columns it is often beneficial to standardize or normalize the values, before training a Machine Learning model [Yildirim, 2022] or performing data analysis such as PCA [Scikit-learn, 2022a]. If the data is not scaled, the value range of the individual columns will affect the result of the Machine Learning model or data analysis, as a column with a larger value range will generally have a bigger impact on the model or analysis than a column with a smaller value range [Yildirim, 2022]. This is especially intuitive with the PCA case, as the most important columns are those that best represent the overall variance of the data set. As soon as one column varies more than the other, it will be considered as more essential for the data set. Without scaling the result will therefore be misleading, as columns with larger value ranges not necessarily represents the variance of the data set best [Scikit-learn, 2022a].

Note that a row of the input data set $\mathbf{x}$ is written as $\mathbf{x}^i$, while a column of the input data set $\mathbf{x}$ is written as $\mathbf{x}_j$, in the following definitions.

2.1.1 Unit Norm Scaling

With Unit Norm Scaling, the data is scaled to have a norm of 1. The most common norm used in this scaling is the L2 norm.

$$\mathbf{x}^{i\prime} = \frac{\mathbf{x}^i}{||\mathbf{x}^i||} \tag{2.1}$$

In Equation 2.1 the calculation needed for Unit Norm Scaling is presented, where the values of the i:th row $\mathbf{x}^i$ are divided by the norm of the i:th row $||\mathbf{x}^i||$, which results in the norm scaled $\mathbf{x}^{i\prime}$ [Scikit-learn, 2022e].

2.1.2 Minmax Scaling

Minmax scaling is used to push the data into a range between a lower limit a and a higher limit b. The often used default range is having $a = 0$ and $b = 1$. In Equation 2.2 below, the Minmax Scaling calculations are presented.

$$\mathbf{x}'_j = a + \frac{(\mathbf{x}_j - min(\mathbf{x}_j))(b-a)}{max(\mathbf{x}_j) - min(\mathbf{x}_j)} \tag{2.2}$$

Each column in a data set is scaled independently according to Equation 2.2 above, where $\mathbf{x}_j$ is the values for the j:th column, $min(\mathbf{x}_j)$ is the minimum value of the j:th column, $max(\mathbf{x}_j)$ is the maximum value of the j:th column and $\mathbf{x}'_j$ is the resulting scaled values of the j:th column [Scikit-learn, 2022d].

2.1.3 Standard Scaling

By scaling with a Standard Scaler the resulting data set will have a mean value of 0 and a unit variance. This is achieved by performing the calculations presented in Equation 2.3 below.

$$\mathbf{x}'_j = \frac{\mathbf{x}_j - \mu_j}{\sigma_j} \tag{2.3}$$

The mean value μ_j and the standard deviation σ_j are calculated for each column j. After calculating the statistics for the columns independently, the mean is subtracted from the values of the j:th column, $\mathbf{x}_j$, and the standard scaled value $\mathbf{x}'_j$ is obtained by dividing with the standard deviation [Scikit-learn, 2022g].

2.1.4 Robust Scaling

The scaling of data containing outliers with methods that uses the mean, variance or the whole data range, will be affected by the outliers. To prevent the outliers from influencing the scaling, a Robust Scaler can be used. Instead of scaling with the mean and a fixed range or the complete data range, a Robust Scaler uses the median together with the range between the 25:th percentile and the 75:th percentile.

$$\mathbf{x}'_j = \frac{\mathbf{x}_j - median(\mathbf{x}_j)}{Q_{3j} - Q_{1j}} \tag{2.4}$$

Robust Scaling is performed column wise. In Equation 2.4 above, Q_{3j} and Q_{1j} represent the 75:th percentile and the 25:th percentile, for the j:th column in the data set. Following, $median(\mathbf{x}_j)$ is the median of the j:th column of the data set and $\mathbf{x}'_j$ is the scaled values of column j [Scikit-learn, 2022f].

2.2 PCA

Large data sets are commonly represented in higher dimensions. This can be problematic when the data is analyzed, since you can not plot the data in more than three dimensions. One method to tackle this problem is the Principal Component Analysis (PCA), a technique that reduces the dimension of a data set while still keeping as much information as possible. It is done by finding new variables that maximizes the variance at the same time as they are uncorrelated.

Given a data set $\mathbf{X}$ with p dimensions and n observations where $\mathbf{x}_j$ is the j:th column of $\mathbf{X}$. We aim to define the linear transformation

$$\sum_{j=1}^{p} a_j \mathbf{x}_j = \mathbf{X}\mathbf{a}$$

that maximizes the variance with a vector of constants $\mathbf{a}$. The variance is then equal to $var(\mathbf{X}\mathbf{a}) = \mathbf{a}'\mathbf{S}\mathbf{a}$, as any such linear transformation, where $\mathbf{S}$ is the covariance matrix of the observations. Our aim is then to maximize the expression $\mathbf{a}'\mathbf{S}\mathbf{a}$ but to make sure that the solution is well defined we need another restriction. Here different variants of PCA can be created, the normal restriction of choice is that the vector $\mathbf{a}$ needs to be unit-norm, meaning that $\mathbf{a}'\mathbf{a} = 1$. To maximize the expression under the restriction we can define the Lagrange function $L(\mathbf{a}, \lambda)$ as

$$L(\mathbf{a}, \lambda) = \mathbf{a}'\mathbf{S}\mathbf{a} - \lambda(\mathbf{a}'\mathbf{a} - 1) \tag{2.5}$$

where λ is the Lagrange multiplier. Maximizing function 2.5 can be done by differentiating, with respect to $\mathbf{a}$, the expression and setting the result equal to the zero-vector. Doing this gives us the expression

$$\mathbf{S}\mathbf{a} - \lambda\mathbf{a} = \mathbf{0} \iff \mathbf{S}\mathbf{a} = \lambda\mathbf{a}.$$

The problem is now reduced to finding the eigenvectors of the covariance matrix $\mathbf{S}$. Due to the restriction that $\mathbf{a}$ should have unit-norm, an interesting relation between the variance and the eigenvalues can be found:

$$var(\mathbf{X}\mathbf{a}) = \mathbf{a}'\mathbf{S}\mathbf{a} = \lambda\mathbf{a}'\mathbf{a} = \lambda.$$

This means that the largest eigenvalue λ_1 will maximize the variance, and therefore the corresponding eigenvector $\mathbf{a}_1$ is the vector that we are searching for. The constant vector $\mathbf{a}_1$ can then be used to transform the data set $\mathbf{X}$ from a dimension of p to a dimension of 1, while keeping the maximal variance. Due to the covariance matrix $\mathbf{S}$ being symmetric and positive semi-definite the eigenvectors will be orthogonal, we can therefore show that the projections, or principal components $\mathbf{Xa}_j$, are uncorrelated. We show this by calculating the correlation between two principal components

$$corr(\mathbf{Xa}_i, \mathbf{Xa}_j) = \mathbf{a}_i'\mathbf{S}\mathbf{a}_j = \lambda_j \mathbf{a}_i'\mathbf{a}_j = 0$$

if $i \neq j$.

The projection $\mathbf{Xa}_j$ is called the j:th Principal Components or PC_j. The Principal Components can be used to visualize the data in lower dimensions by plotting with the first PC:s. For example if you want to visualize the data in two dimensions the first two Principal Components, PC_1 and PC_2, can be used. Because the eigenvectors $\mathbf{a}$ are ordered based on the which can represent the most variance of the data the first Principal Components are most suited for lower dimension visualization. [Jollife and Cadima, 2016]

2.3 Clustering algorithms

The overall idea for clustering algorithms is grouping data with respect to the samples' feature values. How the groupings are made differs between the algorithms and chosen hyperparameters. To perform clustering, at least a data set with objects O and a belonging distance function $d : O \times O \rightarrow R^+$, for instance Euclidean or Manhattan, are needed. Usually each object O is d-dimensional containing real valued points, $O \subset R^d$, where the points are sampled from $p(x)$, an unknown probability density function.

The majority of the clustering algorithms aim to minimize the dissimilarity within the clusters while maximizing the dissimilarity between different clusters, by using the distance function d. These kinds of clustering algorithms assume that the density of the data set as a whole is the result of data points sampled from k number of probability density functions p_i that each belongs to one of the clusters and mathematically to a parametric family such as a Gaussian distribution. The number of clusters, k, needs to be set before the clustering starts. During the clustering the variable parameters related to p_i are tuned to represent the density of the belonging cluster as good as possible.

Another group of clustering methods that are non-parametric, in contrast to the clustering algorithms described above, is called density-based clustering. When using density-based clustering algorithms the number of clusters, k, is not predefined and the algorithms are not based on assumptions about the density function $p(x)$. The goal for this type of clustering algorithms does not include having the lowest dissimilarity with respect to the distance function d, within the clusters. Instead, clusters form at the high density areas of $p(x)$, which allow clusters with a big variety of appearances to be found. In contrast to the parametric clustering algorithms a point assigned to a cluster with a density-based clustering algorithm can be much closer to points that are assigned to another cluster, than to points within its own cluster. The intuitive explanation for this is that the density within the clusters respectively is higher, than the density in the areas between them. Because of the distinction between high- and low-density areas, density-based clustering algorithms label points as noise when they occur between the clusters. [Sander, 2010]

2.3.1 DBSCAN

Density-Based Spatial Clustering of Applications with Noise, DBSCAN, is a well-known density based clustering algorithm designed for finding clusters of arbitrary shape in large spatial data sets, without demanding the user to have much knowledge about the data domain.

For explaining the DBSCAN clustering method the Euclidean distance function will be used and denoted as $d(p,q)$, and for simplifying the visualization p and q will be two arbitrary points in the 2D-space. In Figure 2.1 DBSCAN clustering applied on a small data set is presented, with the two detected clusters marked.

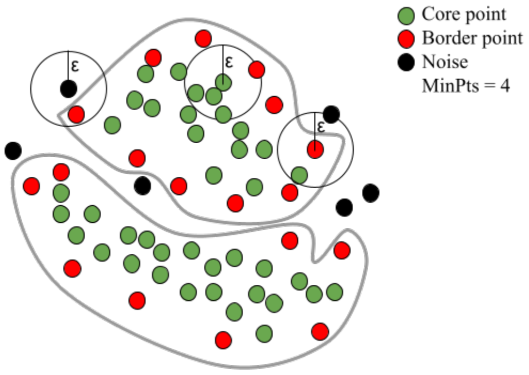


Figure 2.1 An example of a clustering with DBSCAN where two clusters are found. The green points represent the core points, the red points represent the border points and the black points represent the noise. The minimal number of neighboring points for a core point is 4.

Before the DBSCAN clustering begins, there are two parameters that need to be set: *MinPts* and *Eps*. In the figure $MinPts = 4$ and $Eps = \varepsilon$. Both these parameters are used to calculate the neighborhood $N_{Eps}(p)$ of each point p, which is a central part of this clustering method and defined in Equation 2.6 below.

$$N_{Eps}(p) = \{q \in D | dist(p,q) \leq Eps\} \tag{2.6}$$

The neighborhood $N_{Eps}(p)$ for the point p, is the set of points q that come from the same data set D as p and lie within a distance of *Eps* from p. If $|N_{\varepsilon}(q)| \geq MinPts$, i.e. the number of the neighboring points are greater than or equal to *MinPts*, the point is considered to be a core point. All core points that can be reached from going from one neighboring core point to another are assigned to the same cluster. A point that lies in the neighborhood of at least one core point, without having enough neighboring points to reach the requirement for being a core point itself, is considered to be border point. The border point is assigned to the same cluster as its neighboring core point. It is possible for a border point to have several core points in its neighborhood that belong to different clusters. To avoid soft clustering, the default in DBSCAN clustering is to assign that border point to the first approaching cluster, as the algorithm iterates from point to point starting with a core point. The points that do not reach the requirement for either a core point or a border point is labeled as noise. In Figure 2.1 core points are green, border points are red and noise points are black. Because of the condition of having at least *MinPts* number of neighboring points to be a core point, it is also required that the data set has more than *MinPts* for a cluster to be formed by the algorithm. [Xu, 1996]

2.3.2 OPTICS

Ordering Points To Identify the Clustering Structure, OPTICS, is a clustering algorithm that can be seen as an extension of the DBSCAN clustering algorithm. In comparison with DBSCAN, the OPTICS clustering algorithm does not require the clusters within a data set to have similar densities. As the OPTICS algorithm considers the intrinsic clustering structure, the information obtained from one run corresponds to many runs with different input parameters when using other density based clustering algorithms. This is often helpful as a set of real-data clusters coming from the same data set seldom fit under fixed global input parameters, for instance because the local densities can differ a lot. In Figure 2.2 below an example of a data set containing clusters with different densities is visualized. For a clustering algorithm that uses a global fixed density condition it could be difficult to both discover the three smaller high density clusters, encircled by green lines, and the two bigger low density clusters, encircled by blue lines, in the same run. If the clustering algorithm with a fixed global density condition manages to capture the three smaller clusters and define them as separate, there is a risk that the samples belonging to the bigger clusters will be labeled as noise. However, if the bigger clusters are found by the clustering algorithm with a fixed global density condition, it is possible that the density limit is too low to distinguish the empty area between the smaller high density clusters and instead are seen as one cluster, which in the figure is encircled by a pink line.

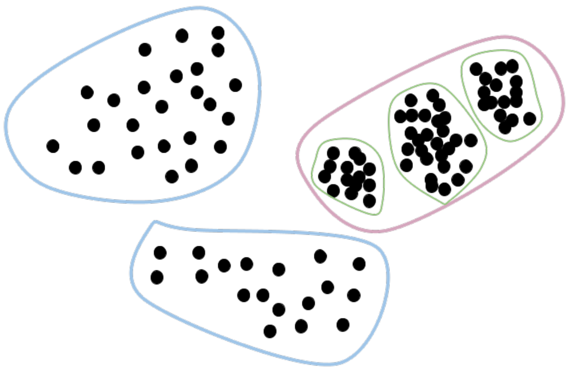


Figure 2.2 An example that shows a data set containing clusters with different densities.

To explain OPTICS, a suitable density based clustering algorithm that can distinguish clusters with different densities within the same data set, the DBSCAN approach will first be extended. In the previous section the parameters *MinPts* and *Eps* were defined, together with the neighborhood $N_{Eps}(p)$ of a point p. In summary each point has a number of samples in its neighborhood, see Equation 2.6, and if the number of points in its neighborhood is equal to or exceeds *MinPts*, the requirement for a cluster is fulfilled and the point together with its neighboring points are assigned to the same cluster. These kinds of points are considered to be core points, while points with at least one core point as neighboring point but a neighbor number less than *MinPts* are border points. The points that do not fulfill the conditions for being a core point or a border point are labeled as noise. The first step of extending DBSCAN towards being capable of finding clusters with different densities is allowing exploration of multiple values for *Eps* in the same run. All points in a high density neighborhood will be captured in a cluster both with their corresponding, smaller, *Eps* as well as with a bigger *Eps*. In contrast, the points in lower density neighborhoods will be labeled as noise for values of *Eps* that corresponds to high density areas. For producing consistent results with the extended approach, an infinite number of different *Eps* values are explored, from the smallest possible value for a cluster to be defined by the algorithm, to the maximum limit specified by the user, according to expression 2.7 below.

$$0 \leq \varepsilon_i \leq \varepsilon \tag{2.7}$$

The OPTICS algorithm works like the extended DBSCAN algorithm, except that it does not assign the points to the clusters directly. Instead, two new values are introduced and stored together with the data point. The first value is the core-distance, see definition 2.8 below. When $|N_\varepsilon(p)| \geq MinPts$, i.e. when the number of neighboring points is greater than, or equal to *MinPts*, the point's core-distance is the smallest *Eps* value that is needed for the point to be a core point, hence *MinPts-distance* $=$ *Eps* where *Eps* corresponds to the value needed for $|N_\varepsilon(p)| = MinPts$ to be fulfilled.

$$core\text{-}distance(p) = \begin{cases} \text{UNDEFINED} & \text{if } |N_\varepsilon(p)| < MinPts \\ MinPts\text{-}distance(p) & \text{otherwise} \end{cases} \tag{2.8}$$

In addition to the core-distance, reachability-distances are also calculated, see definition in Equation 2.9 below. When the resulting value is not UNDEFINED which happens under the same circumstances as for the core-distance, the reachability-distance is the greatest value of the point's core-distance and the distance between the point and another neighboring point, *o*.

$$r\text{-}distance(p,o) = \begin{cases} \text{UNDEFINED} & \text{if} |N_\varepsilon(p)| < MinPts \\ max(core\text{-}distance(p), distance(p,o)) & \text{otherwise} \end{cases} \tag{2.9}$$

Furthermore, a cluster-ordering is performed by an iterative process where reachability-distances are calculated, ordered and stored. In this paper, the exact pseudo code will not be presented or explained, since it will not contribute to the understanding of why OPTICS is a suitable and interesting algorithm for the experimental part of this thesis. After producing a cluster-ordering with the reachability-distances, a reachability plot is made. An example of a such plot can be seen in Figure 2.3 below.

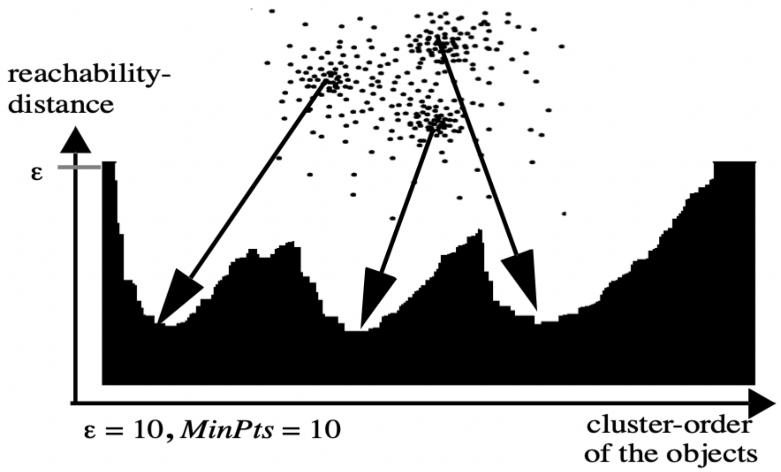


Figure 2.3 An example, retrieved from the article *OPTICS: ordering points to identify the clustering structure* [Ankerst et al., 1999] that shows a reachability plot, where the y-axis values are the reachability-distances and the x-axis values are the cluster-order of the objects.

In Figure 2.3 above only data points with defined core-distance and reachability-distance values are shown, i.e. data points that are assigned to clusters. The samples with $UNDEFINED$ core-distance and reachability-distance need a bigger value for Eps than the max value set by the user (ε in the figure) to not be labeled as noise. Each valley in the reachability plot represents a cluster and each cluster gets separated from the next one when there is a sharp decline in reachability-distance, as you move along the cluster-ordering axis. [Ankerst et al., 1999]

2.3.3 HDBSCAN

Hierarchical Density-Based Spatial Clustering of Applications with Noise or HDBSCAN is a clustering algorithm that extends DBSCAN, where the parameter *Eps* is not required to be specified. The algorithm can be divided into 5 parts where the first step is to define a new space. Similar to DBSCAN the algorithm distinguishes noise from in-cluster samples, an ability that makes an important difference in a situation where noise risk to be seen as a connection between clusters. The HDBSCAN algorithm will in the first step distance low-density areas further from high-density areas. This is done by defining a *core* distance for each point in the database. The $core\text{-}distance(p)$, see Equation 2.8, for the point p is defined as the distance to the k:th nearest neighbor for the point p. With this a new metric the mutual reachability distance, $d_{mreach-k}(p,q)$, is defined as

$$d_{mreach-k}(p,q) = \max\{core\text{-}distance(p), core_k(q), d(p,q)\} \tag{2.10}$$

where $d(p,q)$ is the previously mentioned distance function which should be chosen to the most suitable for the data, for instance the Euclidean or Manhattan distance.

The result of this transformation is a new space where highly dense areas remain highly dense while low-density areas are separated.

When the mutual reachability distances are calculated the algorithm proceeds to the next part: the creation of a minimum spanning tree. In this part the highly dense areas are located, taking into account that the concept of highly dense areas varies between data sets. This is done by connecting all data points, with weights at the edges equal to the mutual reachability distance of the connected points. Further, the optimal process is to create a hierarchy of connected points, from iteratively looping through a range of thresholds and dropping every edge with a weight larger than that threshold. The hierarchy would span from having all points connected by edges, to having no edges at all. Unfortunately this is an operation demanding high computational power and is unreasonable to implement. What instead is done is defining a minimum spanning tree where every point is reachable from every other point, while no edge can be replaced to lower the sum of mutual reachability distances. In Figure 2.4 an example of a minimum spanning tree is presented with the mutual reachability distance as weights.

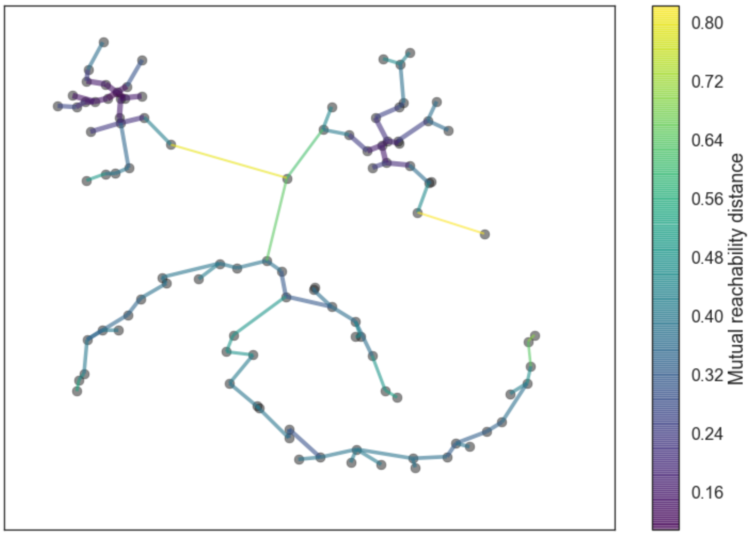


Figure 2.4 An example that shows a minimum spanning tree with weights defined by the mutual reachability distance. The figure is retrieved from the article *How HDBSCAN works* [McInnes et al., 2017].

With the minimum spanning tree the algorithm continues to the third part; creating a hierarchy of connected points. This is done by connecting the edges, one at a time, starting with the lowest weighted edge and increasing through the edges ordered by the weights. This creates a cluster hierarchy and an example is plotted in Figure 2.5. A main difference between DBSCAN and HDBSCAN is that the former would use the input parameter *Eps* to cut the cluster hierarchy. Everything above the cut is considered noise while the separated branches created under the cut are considered clusters. In HDBSCAN however, the cut is more data driven and varies depending of the look of the hierarchy.

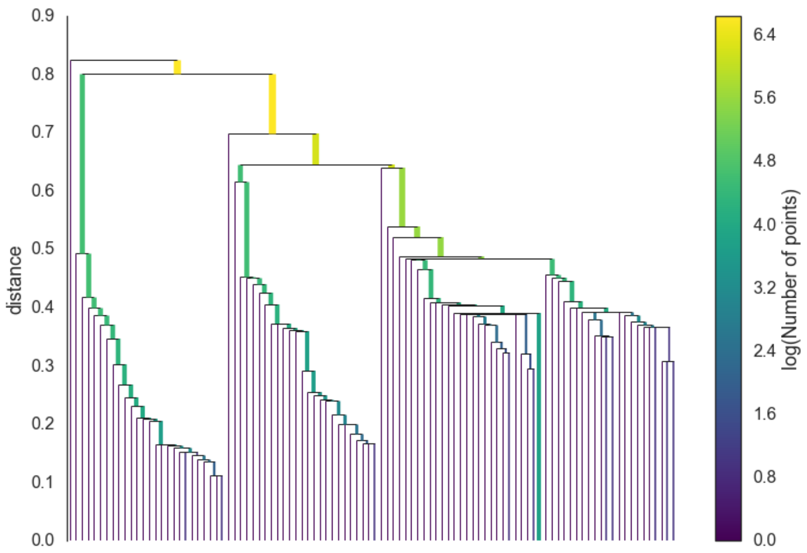


Figure 2.5 Example of a cluster hierarchy with y-axis representing the mutual reachability distance and the color as number of points in a branch. The figure is retrieved from the article *How HDBSCAN works* [McInnes et al., 2017].

In the fourth part of the algorithm, the parameter *MinPts*, i.e. minimum cluster size, which is specified when running HDBSCAN, is used. With a larger value of *MinPts*, the cluster hierarchy is simplified to larger but fewer clusters, similar to how the *Eps* parameter would affect the result for DBSCAN. In HDBSCAN this is done by iterating through all the splits, from top to bottom of the hierarchy, and applying a logic to the splits. The logic returns one of two cases; either the split results in a new cluster or the branch is considered a part of the current cluster. The logic investigates if the branch in question contains less than the *MinPts*, if the answer is yes then that branch will be considered a part from the parent cluster. If in the other hand all the branches from a split contain more than the *MinPts*, the parent cluster will end and each branch will form new clusters. The resulting tree has fewer nodes with more points in each node. An example is seen in Figure 2.6, retrieved from the same example as Figure 2.5.

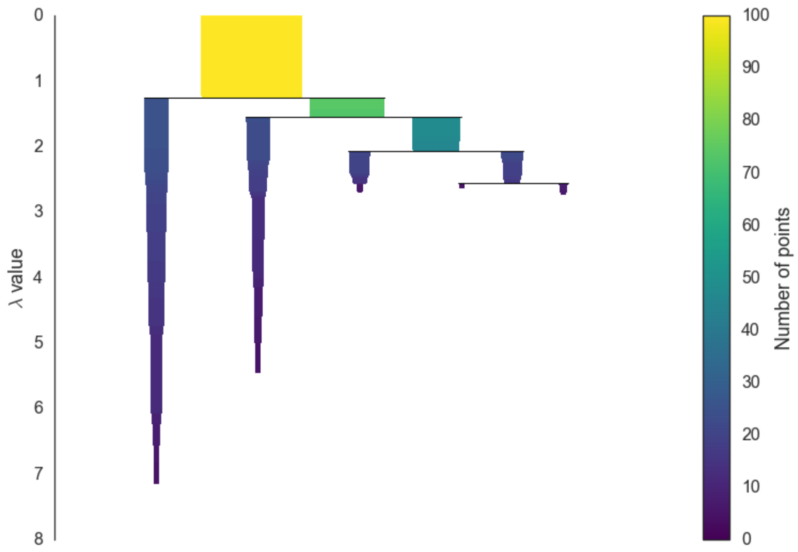


Figure 2.6 Example of a dendrogram where the hierarchy from Figure 2.5 has been used with the logic. The figure is retrieved from the article *How HDBSCAN works* [McInnes et al., 2017].

In the last step of the algorithm the final clusters are extracted. To do this a new metric called *stability* is defined, as well as a new measurement $\lambda = \frac{1}{distance}$. For each cluster the λ where the cluster start is also calculated and called λ_{birth}. The following formula is used for calculating the *stability*:

$$\sum_{p \in cluster} (\lambda_p - \lambda_{birth}) \tag{2.11}$$

where λ_p is the λ-value where the point p drops out from the cluster. The *stability* is increasing if a cluster is lasting longer before a split, while a high amount of points is included in the cluster. To decide which clusters should be considered to become the final clusters, the leaf nodes start off as the first clusters. The algorithm moves up along the tree and continuously checks if the sum of stability from the child nodes is greater than the parent stability. If this is the case, the parent stability is set to the sum of the stability from the children and the child nodes are kept as the current clusters. If in the other hand the parent stability is greater, the child nodes are discarded as the current clusters, and the parent node is chosen as the current cluster. This selection process is performed until we reach the root and the current clusters will be considered the final clusters. [McInnes et al., 2017]

2.4 Autoencoder

An autoencoder is an artificial neural network with a specific type of architecture. The network contains three parts; an encoder network, a decoder network and a latent vector. In Figure 2.7 an example of a simple autoencoder is presented. The example network has an input and output size of 20 each, with first and third hidden layers of size 10. The hidden layer in the middle has a size of 5 and this layer is called the latent vector. An autoencoder aims to reproduce the input sample, meaning that an ideal autoencoder returns a sample that is identical to the input. Between the input and output stage, the sample is reduced to a smaller dimension: the latent vector. The intelligence in an autoencoder lies in the ability to keep as much information of the sample as possible while reducing its dimension and then being able to reproduce the sample in the decoder network, where the sample is transformed back to the original dimension.

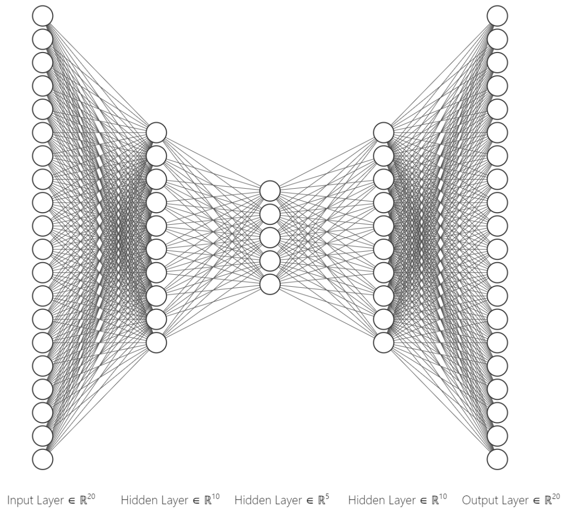


Figure 2.7 Example of an fully connected autoencoder with the latent vector with dimension 5 as the second hidden layer.

The metric used for optimization is called *loss* and the goal is to minimize it. Given the j:th input sample $\mathbf{x}_j$ and the corresponding predicted output $\mathbf{y}_j$, the loss for that sample, l_j, is calculated with the equation

$$l_j = ||\mathbf{x}_j - \mathbf{y}_j||_{mse}$$

where $||\cdot||$ is the mean squared error, calculated with the equation

$$||\mathbf{x}||_{mse} = \frac{1}{n}\sum_{i=1}^{n} x_i^2.$$

After the autoencoder is trained with only normal data, the idea is that it will perform well on new normal data, i.e. that the corresponding loss is low. In contrary the autoencoder will reproduce abnormal data poorly and hence return a large loss. By setting a threshold for the loss, the autoencoder can be used as an anomaly detector. For instance if the threshold is set to one standard deviation of all losses in the input data set, all input samples that return a loss greater than that threshold will be labeled as anomalous. The poor reconstruction of an abnormal sample is a result of that the autoencoder does not know how to decode the corresponding latent vector. As mentioned earlier, the training of the autoencoder needs to be done with data containing close to no anomalies. Except for demanding a data set of normal data no labels are necessary, which makes the approach semi-supervised. [Kuo, 2019]

2.5 Evaluation metrics

For evaluating the performance of a model in a representable and consistent way it is important to choose suitable metrics. In the binary classification field the majority of the metrics is derived from four values; true positives, true negatives, false positives and false negatives. In Figure 2.8 below a chart of the four values is presented. It is meant to show the relationship between the values and the data set's actual labels and predicted labels.

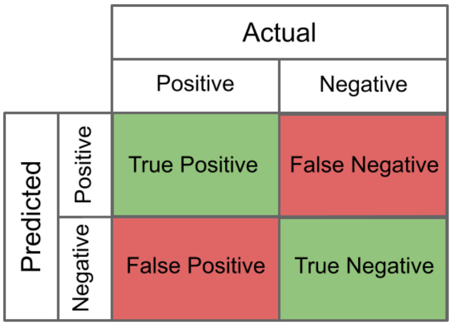


Figure 2.8 A chart that explains where true positives, false, negatives false positives and true negatives come from, in relation to the actual labels and the predicted labels, when performing binary classification.

For simplifying the explanation of the values presented above and at the same time relate to the values that will be used in the thesis, an *Actual Positive* sample will correspond to an anomaly and an *Actual Negative* sample will correspond to a normal data point. Consequently, a true positive is an anomalous sample that is labeled as an anomaly while a true negative is a normal sample labeled as normal. A false positive is therefore an actual normal sample, labeled as anomalous and a false negative is an actual anomalous sample, labeled as normal. Henceforth these values will be referenced to as the count of all samples within each of the four categories and denoted as TP, TN, FP and FN respectively. [Dilmegani, 2019]

2.5.1 Precision

The precision is a measurement of a model's ability to not label an actual negative sample as positive. In Equation 2.12 below the calculation needed to retrieve this metric is presented.

$$Precision = \frac{TP}{TP+FP} \tag{2.12}$$

The maximum and optimal value is 1 which corresponds to only having actual positives labeled as positives. The minimum value is 0, which corresponds to only having actual negatives labeled as positives, or no samples at all labeled as negative. [Scikit-learn, 2022b]

2.5.2 Recall

How well a model finds all the actual positive samples and labels them as positives can be measured by a metric called recall. In Equation 2.13 below the recall expression is presented.

$$Recall = \frac{TP}{TP+FN} \tag{2.13}$$

The maximum and optimal recall score is 1 and it corresponds to having all actual positive samples labeled as positive. Contrarily, a value of 0 is the smallest possible recall value, which corresponds to a model that does not label any actual positive sample correctly. [Scikit-learn, 2022c]

2.5.3 Accuracy

Classification accuracy is a commonly used metric when evaluating a classifying Machine Learning model. It is calculated by dividing the number of correctly labeled samples with the number of total samples, as shown in Equation 2.14 below.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{2.14}$$

This metric is preferable to use when having balanced classes, i.e. when there is equal, or close to equal, number of samples in each class. A high accuracy score for a data set with unbalanced classes does not by itself imply that the model performs well. [Mishra, 2018]

2.5.4 F1-score

Precision, recall and accuracy are valuable metrics for evaluation but each alone does not paint the whole picture. A model can obtain a great score with the recall metric and still be an incapable model for a given task. This could happen if the model predicts a great part of the data as positive, making sure that it captures all the actual positives and classifying a great part of actual negative samples at the same time. Similarly, a model can achieve a high score in precision by prediction a few but certain samples as positive while ignoring the rest of the actual positive samples.

Accuracy can be a great metric in certain types of data sets but in cases where the number of actual positive and negative samples is skewed the metric holds minimal information. For example in cases where the actual positive samples are a fraction of the data set, a high accuracy can easily be obtained by labeling all samples as negative. To overcome these problems a different metric can be used; the F1-score. In Equation 2.15 the formula to calculate the metric can be seen. This metric ranges between 0 and 1, where 1 is the optimal score and is reached when both recall and precision are at their optimal. [Korstanje, 2021]

$$F1 = 2 \cdot \frac{Precision \times Recall}{Precision + Recall} \tag{2.15}$$

3

Method

In this chapter the problem formulation and the method are presented. Because of the absence of data labels, a big part of the scientific contribution of this thesis lies within the method. The way of using different data representations for extracting as much information as possible, is a novel suggestion for handling unlabeled data. With this in mind, the method should not only be seen as a manual that can be used for reproducing the experiment, but also as the actual result of the thesis. Firstly in the method part, the pipeline is presented, which includes everything from pre-processing of the data to the real-time anomaly detection model. After comes an explanation of the evaluation used in the thesis.

3.1 Problem formulation

The problem this thesis aims to solve is to detect event log anomalies in real-time for public transportation bus data, without having access to a labeled data set. As earlier described, in 1.1 Background, each bus journey, has an event log that is extended every time an event occurs during the trip. Occasionally the bus produces sequences of events that are not expected and are caused by faulty behavior of the bus, external disturbances or a bug in the software. The challenge is to detect these faulty sequences using the event logs, and to detect them during the journey instead of after. One of the first questions that arises is; What defines an anomalous event sequence? As mentioned in Section 1.4.1, an outlier is not necessarily an anomaly, as an anomaly also needs to be unwanted. This definition makes the problem more complex than just finding abnormalities in the data set with respect to the representation of the data.

Because the data set provided for this thesis is unlabeled and no specification about the characteristics of a faulty sequence exists, the solution has to be versatile and implemented for unsupervised data. What is nevertheless provided is a subset of journeys that are anomalous, since they satisfy the conditions for at least one out of three known anomaly types. The known anomalous trips can be used in the implementation of the solution. However, we can not assume that the anomalous subset are the only anomalies, due to the generic nature of the method and the aim of creating a method with abilities of detecting unknown anomalies. Therefore, the method has to be able to not only detect the specific anomalies but also anomalies unfamiliar to both the method and the company Gaia.

The specifications surrounding the problem that the thesis aims to solve can be presented as following; A trip T can be described as a sequence of events e_i, $T = \{e_1, e_2, e_3, \ldots, e_n\}$, that varies in length n. The events belonging to the trip T can be partitioned into sub-sequences. Thus, a sub-sequence $S = \{e_i, e_{i+1}, e_{i+2}, \ldots, e_{i+j}\}$, with length j and starting at event e_i, contains a varying number of consecutive events. In a data set of trips, M, we assume that 10% of the trips contain anomalies, denoted as M_a, where $M_a \subset M$. An anomalous trip is denoted T_a and $T_a \in M_a$. These anomalies come from anomalous underlying structures that result in sub-sequences of events that are not to be expected in normal behavior. The project aims to create a real-time detector $D(S)$ that, given a belonging sub-sequence from a trip T, determines if the sub-sequence S is anomalous and therefore deciding if $T \in M_a$. Since the model uses the event sub-sequences of a trip for identifying anomalous behavior, it can operate in real-time while the bus runs. Except for a small subset of known anomalous trips $T_{known\_a} \in M_{known\_a}$, where $M_{known\_a} \subset M_a$, no prior information about the anomalous underlying structures is available. Due to the desired generic nature of the model, M_{known_a} does not represent the whole subset of anomalous trips, i.e. $M_{known\_a} \neq M_a$. Instead, it is a subset of the anomalous trips where $|M_{known\_a}| < |M_a|$.

3.2 Pipeline

The central part of the method presented for the Master's thesis is the pipeline, visualized in figure 3.1. With the unlabeled data a supervised model was not feasible. Instead, an unsupervised method was implemented that used techniques that are most commonly used in semi-supervised applications. To be able to use models that depended on having a data set of normal data available, a heavy filtration of the original data set was required. After the filtration the data left was the previously mentioned data set M. Thereafter, M was sorted into two data sets; *clean data* and *residue data*. As a consequence of the sorting, *residue data* contained all anomalous trips, i.e. $M_a \in$ *residue data*. To represent a single journey a frequency-based representation was implemented where each journey was mapped to a single vector. The goal with the pipeline was to implement a detector that detected anomalies in real-time, $D(S)$, with a sub-sequence of events, S, as input. However, no restrictions for including the corresponding finished historical journey, T, in the training of the detector existed. The pipeline contained two different labeling processes where the first process used complete journeys, T, while the last worked as a real-time detector using sub-sequences of events included in T. As mentioned above, the complete journeys used in the first labeling process were represented by their event-frequencies. In the first part, three different models were implemented to perform the labeling; clustering, PCA and autoencoder. Each of the models labeled every journey T as either normal or anomalous. The methods were implemented using the two data sets, *clean data* and *residue data*, obtained by the filtering. In addition to comparing the performance of the three labeling models one by one, combinations of them were also explored.

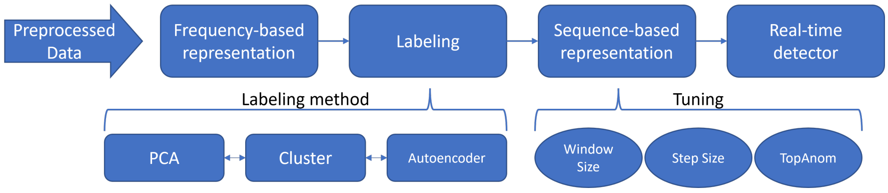


Figure 3.1 Illustration of the different steps in the pipeline. The data is first preprocessed before it is represented using the frequency of each event. The process proceeds to the part where the three labeling models are used. Before the real-time detector, sequences are extracted from the journeys and tuning of the three parameters (*Window Size*, *Step Size* and *TopAnom*) is performed.

After the labeling was performed by the three models, the real-time detector part came next. Because of the real-time aspect of the detector, complete journeys could not be used here. Instead, the trips were divided into sequences of events, each called S, that were classified as normal or anomalous one by one. The sequence-based representation was obtained by using a sliding window. Length of window as well as step size were determined by a tuning where *specific anomalies* (i.e. M_{known_a}), a subset of known anomalous trips, were used to compare the performance of different window lengths and step sizes. Added to each event sequence S was the label retrieved from the frequency-based labeling, corresponding to the journey T that the event sequence was included in. For each unique event sequence, the ratio between the number of anomalous labeled journeys that contained the sequence and the number of normal labeled journeys that contained the sequence, was calculated. The sequences were sorted based on the ratio and a top percentage of the sequences were marked as the final anomalies. This enabled real-time alarming, as the detector got the possibility to alarm about an anomalous event sequence as soon as it was extracted from the logs and exposed to the detector. In Figure 3.1 an overview of the pipeline can be seen.

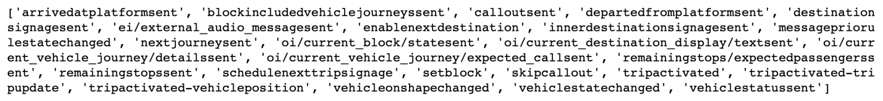
3.2.1 Preprocessing of data

Before models were trained, the data needed to be transformed, filtered and cleaned. These processes are explained in detail in the following section. The first step in the data preprocessing was to extract the data relevant for this project, M, from the event logs. Thereafter, the data set was divided into two; the *clean data* and the *residue data*. This was necessary as the models used in the first labeling process were evaluated with methods that required a data set that was close to *non-anomalous*. In addition, an autoencoder can only be used as an anomaly detector if it is trained with only normal data, i.e. the *clean data*. The deriving of the *specific anomalies*, M_{known_a}, previously mentioned in Section 3.2, is also explained in this section.

3.2.1.1 Filtering of data

Before using the data, the relevant data M needed to be extracted. In the following section the filters that were applied to the whole data set are presented.

Vehicle type
Within public transportation several types of vehicles are operating. The event patterns differ a lot between the different types of vehicles, why it is preferable to have separate models for each vehicle type. As mentioned earlier in the report, this thesis focuses on the buses and the data was filtered accordingly. The resulting data set consisted of event logs only from bus journeys.
Events
Not all vehicles were capable of providing event logs with all events, since some events came from devices that not all buses had. All events belonging to the devices were removed, since it was desirable to only use samples of data that came from the same circumstances. In addition, some events were copies of other events, but with a different name. The reason was that the system transitioned into a new way of formatting the events. All the events in the new format were removed to avoid having redundant events. Out of the 58 unique original events, the 25 events presented in Figure 3.2 were kept:



```
['arrivedatplatformsent', 'blockincludedvehiclejourneyssent', 'calloutsent', 'departedfromplatformsent', 'destination
signagesent', 'ei/external_audio_messagesent', 'enablenextdestination', 'innerdestinationsignagesent', 'messageprioru
lestatechanged', 'nextjourneysent', 'oi/current_block/statesent', 'oi/current_destination_display/textsent', 'oi/curr
ent_vehicle_journey/detailssent', 'oi/current_vehicle_journey/expected_callsent', 'remainingstops/expectedpassengerss
ent', 'remainingstopssent', 'schedulenexttripsignage', 'setblock', 'skipcallout', 'tripactivated', 'tripactivated-tri
pupdate', 'tripactivated-vehicleposition', 'vehicleonshapechanged', 'vehiclestatechanged', 'vehiclestatussent']
```

Figure 3.2 The 25 events that are used for the real-time detector.

CO_diff
After a journey is finished, statistics are obtained and added to the *kpis* field in the event log. Two of the values are *kpis[0].referenceValue*, further called *CallOutRefValue* and *kpis[0].value*, further called *CallOutValue*. These values correspond to the total number of times that the *calloutsent* event is expected to occur during the

journey and the total number of times it actually occurs. Equation 3.1 below shows a calculation made in the preprocessing that extracts the absolute value of the difference between the expected and actual number of *calloutsent* during a journey.

$$CO\_diff = |CallOutRefValue - CallOutValue| \tag{3.1}$$

The *CO_diff* value was added to all journeys, as an extreme such value possibly indicated anomalous behavior. An important note is that this value was only added to data used in the part of the pipeline where the data was represented in a frequency manner. Since it is impossible to know the *CallOutValue* before the journey is finished, this addition of entity cannot be made in a real-time situation, in contrast to the operations described previous in this section.

3.2.1.2 Cleaning of data

To retrieve a data set that corresponded to bus trips that only contained normal event data, a harsh filtering was applied. In addition to the resulting data set, which is called *clean data*, the filtering process also returned a data set that contained the data that did not pass one or more of the filtering steps, called *residue data*. To maximize the percentage of normal data in the *clean data*, it was expected that the *residue data* consisted of both anomalies, M_a, and normal data, due to the harsh filtering applied. The division of the data set M is depicted in figure 3.3 below.

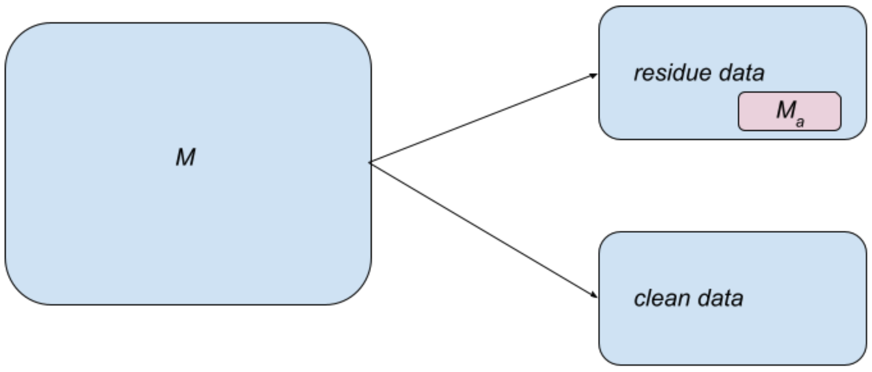


Figure 3.3 The data set M is splitted into two subsets; *clean data* and *residue data*.

Following, the filtering steps used are presented.

Calloutsent
The *calloutsent* event is the event that is logged when the bus passengers are noticed

about the next stop with an audio announcement. The callout should be executed either 30 seconds before the predicted arrival time to the next bus stop, or 200 meters before the next bus stop. To exclude journeys with callouts executed too early or too late from *clean data*, only journeys with *calloutsent* values that fulfill expression 3.2 below, were included.

$$\begin{aligned} 100 \leq \mathit{MetresToNextStop} \leq 300 \\ 20 \leq \mathit{TimeToNextStop} \leq 40 \end{aligned} \tag{3.2}$$

Journeys that included at least one *calloutsent* event that either had a *MetresToNextStop* value or a *TimeToNextStop* value that did not satisfy the above expression, were included in the *residue data*.

PosPerMin

After finishing each journey a *PosPerMin* value is added to the *kpis* field. This value corresponds to the mean of number of position updates per minute that has been made during the journey. Usually the position of the vehicle is updated every other second. Since the aim was to create *clean data*, a data set with as few abnormalities as possible, all journeys with a *PosPerMin* value that did not satisfy expression 3.3 below, were excluded from *clean data* and included in *residue data*.

$$25 \leq \mathit{PosPerMin} \tag{3.3}$$

A low *PosPerMin* value indicates that the vehicle has lost connection which possibly could have caused disturbances in the logs. Hence, the journeys that potentially had log disturbances were excluded from *clean data*.

TripDurationMinutes

The majority of all journeys took between 10 and 50 minutes. Since logs that continued logging after the bus arrived to the last stop existed, as well as logs that stopped logging before the journey was over, only logs that corresponded to journeys within the range of 10 to 50 minutes from first log event to last were included in *clean data*. After each journey is finished the time from the first log event to the last is saved as *TripDurationMinutes* in the *kpis* field. Only journeys with *TripDurationMinutes* values that satisfy condition 3.4 below were included in *clean data*. The rest were included in *residue data*.

$$10 \leq \mathit{TripDurationMinutes} \leq 50 \tag{3.4}$$

CO_diff

When a journey has a non-zero *CO_diff* value it has either been too few or too many callouts, i.e. audio announcements, during the journey. The number of callouts is predetermined as one callout should be executed ahead of each bus stop. Only journeys with a *CO_diff* value that satisfied expression 3.5 below were included in *clean data*.

$$CO\_diff = 0 \tag{3.5}$$

Journeys with a non-zero *CO_diff* value were included in *residue data*.

Not in traffic
During a journey with passengers, which is the type of journey analyzed in this thesis, the bus should never display *Ej i trafik*, *Not in traffic* in English. When this happens the event *destinationsignagesent* is logged together with a value *signage_v1.text* that says *Ej i trafik*. All journeys that included a such event, with a value of *Ej i trafik* were excluded from *clean data*. Journeys that only contained values that satisfy Equation 3.6 below were included in *clean data*.

$$signage\_v1.text \text{ != "Ej i trafik"} \tag{3.6}$$

Wrong callout
In addition to the event log, all actual arrivals and departures to bus stops along the journey are logged separately. The event log and stop log were partitioned by *tr_dt_id*, the unique journey ID, and concatenated row by row. After they were ordered by time, each bus stop ID in rows originating from the stop logs was compared with the most previous bus stop ID belonging to a *calloutsent* event. If the IDs matched, the belonging callout had announced the correct bus stop. Journeys that contained at least one mismatch of these entities were included in residue data, while journeys with no mismatches, i.e. journeys where relationship 3.7 was satisfied for all bus stops, were included in *clean data*.

$$stop\_log.stop\_id_j = calloutsent.stop\_id_j \tag{3.7}$$

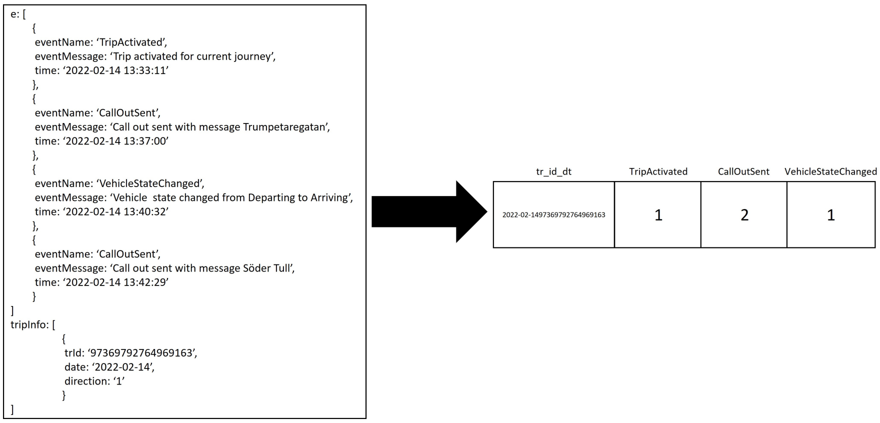
3.2.1.3 Specific anomalies

For deciding value of *window size*, *step size* and top percentage of anomalous sequences to label as anomalous, different combinations of such values were compared by evaluation metrics based on *specific anomalies*. During the initial data analysis, different structural abnormal behaviors emerged in some journeys. Without extracting specific anomalous sequences, a subset of anomalous trips were formed, M_{known_a}, and confirmed by the company Gaia. M_{known_a} are also called *specific anomalies*, and they were considered anomalous because of at least one out of the following three reasons: having extremely many *Ej i trafik*-values or *TripActivated*-events or having an extreme *CO_diff* value. $T \in M_{known\_a}$ if any of the trip's corresponding, and previously mentioned, values were in the highest top 1% out of all such values.

3.2.2 Frequency-based representation

As mentioned earlier, the first labeling part used the whole journey, T, represented by its event frequencies. Below, the method for transforming the data into a frequency-based structure is presented.

The data for a single trip is stored in a nestled JSON file, where each logged event is stored as a dictionary with information such as *eventName* and *eventMessage*. Each journey was transformed into a frequency-based representation, where a single trip was mapped to a vector of integers. The length of the vector was equal to the number of unique events. Each integer in the vector was equal to the number of times that the event, belonging to that column, occurred in the journey. In Figure 3.4 an example of how a trip was represented with this approach is displayed. Included in the vector was the *trId* combined with the date of the journey, as they together formed a unique ID for the current trip.



```
e: [
      {
       eventName: 'TripActivated',
       eventMessage: 'Trip activated for current journey',
       time: '2022-02-14 13:33:11'
      },
      {
       eventName: 'CallOutSent',
       eventMessage: 'Call out sent with message Trumpetaregatan',
       time: '2022-02-14 13:37:00'
      },
      {
       eventName: 'VehicleStateChanged',
       eventMessage: 'Vehicle  state changed from Departing to Arriving',
       time: '2022-02-14 13:40:32'
      },
      {
       eventName: 'CallOutSent',
       eventMessage: 'Call out sent with message Söder Tull',
       time: '2022-02-14 13:42:29'
      }
]
tripInfo: [
            {
             trId: '97369792764969163',
             date: '2022-02-14',
             direction: '1'
            }
]
```

tr_id_dt	TripActivated	CallOutSent	VehicleStateChanged
2022-02-1497369792764969163	1	2	1

Figure 3.4 Example of how a frequency-based representation looked like. The data from the example journey is stored in a JSON file, and the resulting representation is an vector with a column for each unique event with an additional column for the unique ID of the journey.

The last step of the creation of a frequency based representation, was scaling with the Scikit-learn Robust Scaler. As mentioned in the 2.1.4 Background subsection, the Robust Scaler is suitable for data sets containing outliers. The scaling of normal data points is unaffected by the outliers, since the median is used together with the range between the 25:th and 75:th percentile of the vector. In an early stage of this project the Unit Norm Scaling, Minmax Scaling and the Standard Scaling were also analyzed with a result that confirmed that the Robust Scaler was the most suitable.

3.2.3 Labeling

After the frequency-based calculations for each trip, T, were made the resulting representation was used to label the journey as either normal or anomalous. This was done using three different models; a clustering algorithm, an analysis of a PCA result and an autoencoder model. These three models were then tested individually and by combining them together. The combination was tested both by choosing the union of the results and the intersection. By testing all combinations and individual models the optimal one was obtained. If each labeling model would find different kinds of anomalies, we argued that the union of the models would probably perform well as it would include all detected anomalies. The intersection of the models on the other hand, would possibly perform well if the individual models labeled too many trips as anomalous, but agreed on the actual anomalies.

For selecting the best model, metrics to compare with needed to be defined. Due to the fact that our data was unlabeled simple metrics like accuracy or F1-score were not suitable and instead we relied on the filtering of the data, i.e. the *clean data*. With the assumption that close to no anomalies were included in the *clean data* a first condition for choosing the optimal model was defined, see expression 3.8 below. *CDLN* is short for *Clean Data Labeled Normal* and can be interpreted as the recall value for *clean data*. With that said, there was probably a significant amount of normal data in *residue data* as well, so no conclusions about the actual recall value for normal data could be made.

$$CDLN \geq 0.90 \tag{3.8}$$

In accordance with the expression above, only models that labeled at least 90% of the *clean data* as normal were considered. An equivalent value as *CDLN*, was calculated for the *residue data* and called *RDLN*, standing for *Residue Data Labeled Normal*. To further distinguish the best model from the remaining candidates with a *CDLN* value of at least 0.90, another metric had to be used. With the assumption that close to no anomalies existed in the *clean data*, a consequential assumption followed: that close to all anomalies were in the *residue data*. As mentioned earlier, the filtering applied for creating the *clean data*, was enough strict to ensure that *clean data* only contained *non-anomalous* data, hence the *residue data* contained all the anomalous samples, M_a. A new metric was created, the ratio $CDLN/RDLN$, and the best model was extracted by choosing the one with the highest such ratio among the models that passed the filtering presented in Equation 3.8 above. The usage of that metric was motivated with that we already had extracted all models that performed well on classifying *clean data* as normal. By using the ratio $CDLN/RDLN$ the *CDLN* value was maximized while the *RDLN* value was minimized. The reason for wanting to minimize the *RDLN*, after extracting the models with high *CDLN* values, was that the proportion of normal data within *residue data*, was less than the proportion of normal data in *clean data*.

3.2.3.1 Clustering

The cluster labeling of finished journeys represented by their event frequencies was made with the best performing clustering algorithm of DBSCAN, OPTICS and HDBSCAN, together with the most suitable hyperparameters. Due to the aim of creating a generic model that detects a broad variety of anomalies, rather than creating a model that recognizes a certain predetermined anomaly, clustering algorithms that separate noise from "in-cluster-data" were used. Clustering algorithms that assign every data point to a cluster are more suitable when the anomalies are expected to follow a certain pattern, as the most intuitive approach would be to label whole clusters as anomalous. Instead, we assumed that the majority of the data was normal and that the anomalies would be classified as noise.

As described in the Background and Theory section, see Section 2, all the three different clustering algorithms have a parameter called *MinPts*, i.e. minimum cluster size. In addition, DBSCAN and OPTICS also have the *Eps* parameter which is mandatory to predetermine for the DBSCAN algorithm but optional for the OPTICS algorithm, for which infinity is the default *Eps* value. For deciding which parameters to use together with each of the clustering algorithms, we systematically trained the models with all suitable parameters and possible combinations of them and saved the results in a table. A data point assigned to a cluster was labeled as normal and a data point that was, by the clustering algorithm, classified as noise was labeled as an anomaly. In Figure 3.5 below, the labeling by a DSBCAN clustering algorithm is visualized. Since the data naturally has the same dimension as number of unique events, PCA is used in the visualization to reduce the dimension to the first and second principal components. As mentioned above, the journeys classified as noise were labeled as anomalous and the journeys assigned to a cluster were labeled as normal. The clustering algorithm presented below distinguished one cluster, marked in blue with *False*. Even if more clusters had been found, all data within different clusters would have been labeled the same; as normal.

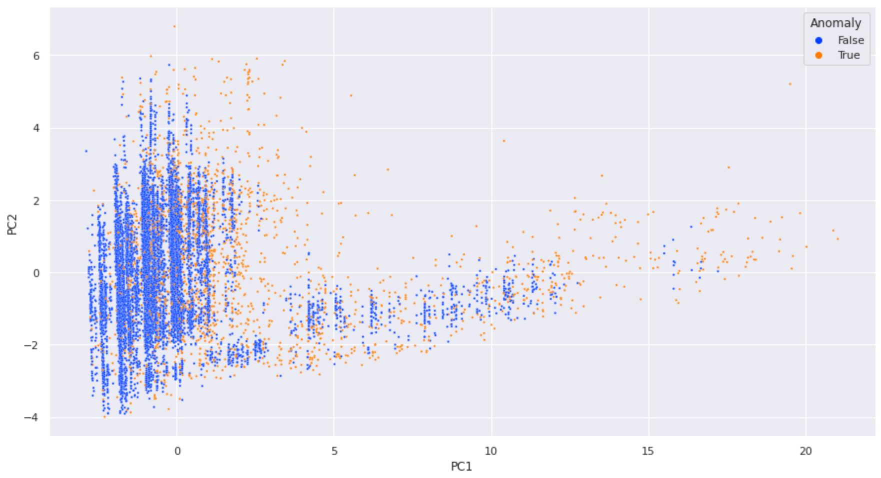


Figure 3.5 The labeling with one DBSCAN clustering algorithm, visualized with the first and second principal components.

During the systematic search for the optimal clustering algorithm and hyperparameter values, a range of values between 10 and 10 000 was used as *MinPts*, where the value was increased with 10 between 10 and 100, with 100 between 100 and 1000 and with 1000 between 1000 and 10 000. The values that were used for *Eps* ranged between 0.01 and 3, where the value was increased with 0.01 between 0.01 and 0.1 and with 0.1 between 0.1 and 3. Larger values for both *MinPts* and *Eps* were tested while the project was still in the experiment phase, however the result was consequently poor with either a *CDLN* value less than 90% or a *CDLN/RDLN* ratio way below the top results. Each row of the stored result table corresponded to a unique combination of clustering algorithm and hyperparameter values. An example of a result table is shown in Table 3.1 below.

Table 3.1 An extract from the result table for the OPTICS clustering algorithm.

Algorithm	NumberClusters	Noise%	RDLN%	CDLN%	CDLN/RDLN	MinPts	Eps
OPTICS, RobustScaler	27	0.976411	0.019704	0.026476	1.343715	10	0.01
OPTICS, RobustScaler	2	0.997110	0.002995	0.002812	0.938786	20	0.01
OPTICS, RobustScaler	0	1.000000	0.000000	0.000000	0.000000	30	0.01
OPTICS, RobustScaler	0	1.000000	0.000000	0.000000	0.000000	40	0.01
OPTICS, RobustScaler	0	1.000000	0.000000	0.000000	0.000000	50	0.01
...	...	...	...	...	...	...	...

The first two columns, *Algorithm* and *NumberClusters*, are self-explanatory, where *Algorithm* is only used for marking each row with the corresponding algorithm. *NumberClusters*, which is the number of clusters that the algorithm finds, was used to filter the result table before extracting the run with largest *CDLN/RDLN* ratio and a *CDLN* value of at least 90%. Only combinations of clustering algorithms and parameters with $NumberClusters > 0$ were considered as suitable, since having zero clusters is equal to only having noise. Although, that filtering was redundant as there was also a limit of maximum 30% noise, which corresponds to the *Noise%* column. After this filtering, only the clustering represented by the second row in Table 3.1 would get through. The rest of the columns; *RDLN%, CDLN%, CDLN/RDLN, MinPts* and *Eps* correspond to the metrics and parameters already described. As written earlier, the final decision of the optimal algorithm and parameters was made by choosing the combination with a *CDLN* value of at least 90% and the highest *CDLN/RDLN* ratio.

3.2.3.2 PCA

PCA is commonly used as a visualization tool when the data is represented in a higher dimension. In the pipeline however, the tool was used differently. The idea of including PCA in one of the base models for labeling the journeys arose when anomalies were found after analyzing the most extreme samples, that became visible when plotting the principal components of the frequency-based vectors.

By extracting the first three principal components of each sample, the journeys were classified as either normal or anomalous, in the labeling part of the pipeline. This was done by finding two borders for each component axis, creating a six-faced cuboid. For the j:th sample the classification was performed according to the following logic,

$$L_j = \begin{cases} Normal & \text{if } S_j \in \{(b_1^- \leq S \leq b_1^+) \cap (b_2^- \leq S \leq b_2^+) \cap (b_3^- \leq S \leq b_3^+)\} \\ Abnormal & \text{otherwise} \end{cases} \tag{3.9}$$

where b_i^+ and b_i^- were the positive and negative borders for the i:th component and S was the set of samples. To find the optimal borders, the previously mentioned *CDLN*, *CDLN/RDLN* and the percentage labeled as anomalies (*Noise*) were used. The optimal borders were found by optimizing each border separately, saving the 5 best borders and then testing each combination of borders together. The single border searches were done by iterating through each *clean data* point and calculating the *CDLN*, *RDLN* and *Noise* values. The reason for only iterating through *clean data* points is that an optimal border would exclude every residue data point of possible to obtain a better ratio. *Noise* was used in the condition that a maximum of 10% of the data set should be considered anomalous, a border that classifies more as anomalous was discarded. Similarly, *CDLN* was used to discard borders where more than 5% of clean data were labeled anomalous. These two conditions secured the search to only consider reasonable borders. Lastly the different values of a single border was ordered in descending order by the ratio *CDLN/RDLN* and the five best were used in the combination search.

After all single searches were performed the *CDLN/RDLN* was calculated for each combination of borders. The borders with the highest ratio were considered the final borders. In Figure 3.6 an example of how the borders looked like, is visible in two dimensions. Anomaly labeled data points are marked in orange and normal labeled data points are marked in blue. The red box represents the borders obtained from the previously described border search, for the first and second principal components. The orange dots inside the borders have third principal components outside the corresponding borders, and if the visualization had considered the third dimension as well, the orange samples would have been outside the border box in the plot.

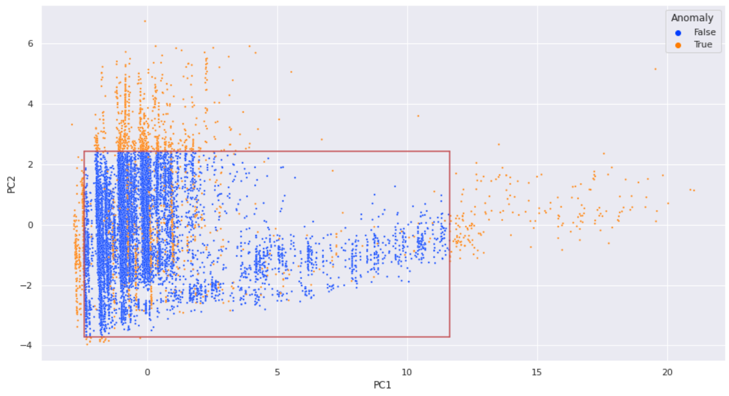


Figure 3.6 A plot over the journeys using PCA to visualize in 2 dimensions. The red square is the border that the labeling model produces.

3.2.3.3 Autoencoder

As explained in Section 2.4, an autoencoder can be used to detect anomalous samples by comparing the input vector with the output vector. If the difference is larger than a certain threshold the sample is labeled as anomalous. In our pipeline the autoencoder had a similar role, with the previous described frequency-based vectors as input vectors. The data set used for autoencoder training was *clean data*, in which close to no anomalies existed. The idea here is that when an anomalous sample passes through the model, the network will not be able to reconstruct the input as good as it reconstructs normal data. With this approach a labeling system was created where a loss was calculated for each sample passing the model. If the loss exceeds a certain threshold, the corresponding sample was labeled anomalous. In Figure 3.7 a flowchart of the labeling procedure for the autoencoder is displayed.

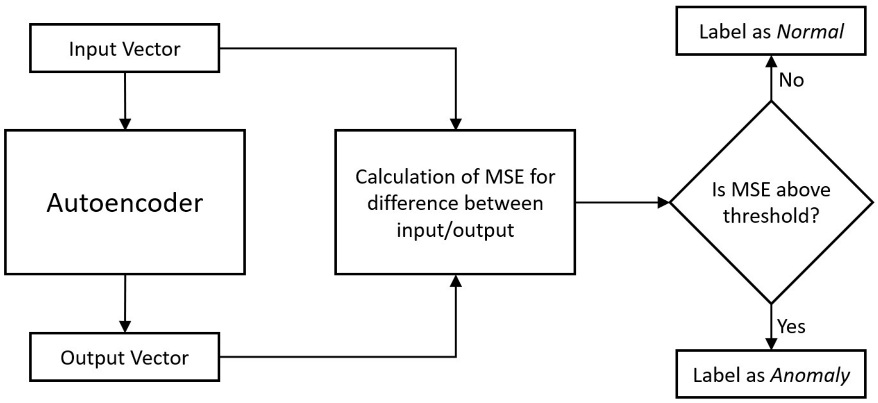


Figure 3.7 Flowchart of the labeling procedure with the autoencoder. Given a input vector the autoencoder tries to reconstruct the initial vector after first reducing the dimension. The difference between the two vectors is obtained and the Mean Square Error is calculated. If the resulting error is higher than a given threshold the journey is labeled as an *anomaly*, otherwise as *normal*.

For our autoencoder we chose to use two hidden encoder layers and two hidden decoder layers with a latent vector of size 8. The activation function used for all layers was the ReLU function and the loss function was the mean square error. The tunable hyperparameters were the number of neurons in each hidden layer, the number of epochs as well as the learning rate. These parameters were selected using a hyperparameter-tuning job with a Bayesian search. The values allowed for the search ranged for the neurons in the hidden layers between 20 and 500 and considered only integers. The values for the number of epochs ranged between 50 and 300 and the learning rates considered were 0.01, 0.005, 0.001 and 0.0001. The metric used to optimize the hyperparameters was the mean-square error for *clean data*.

The optimal threshold was found by using *CDLN* and the ratio *CDLN/RDLN*. *CDLN* was used for the condition that at least 90% of *clean data* should be labeled normal. From the remaining thresholds the optimal value was chosen as the one that maximized the ratio *CDLN/RDLN*. In Figure 3.8 an example of labeled journeys done by the autoencoder using this procedure can be seen. The multi-dimensional journeys are visualized in two dimensions using PCA.

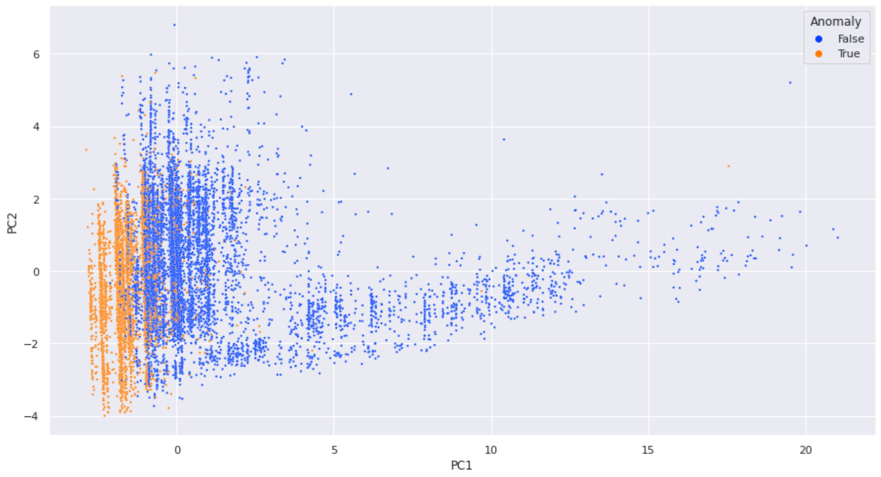


Figure 3.8 A plot over the journeys using PCA to visualize in 2 dimensions. The colors represent the labeled anomalies an autoencoder detects where orange dots represent anomalies and blue dots normal journeys.

3.2.4 Sequence-based representation

After labeling the journeys with the models and combination of models described above, the part where the data is represented by event sequences comes. The reason for using sequences instead of whole trips was, as mentioned earlier, that the sequences can be sent to the detector while a journey is ongoing and works therefore as input to the real-time detector, $D(S)$. This is not possible for the frequency-based representation, as a fair representation only can be made when a journey is finished. A sequence S consists of j consecutive events where j is the size of our sliding window. Included in the sequence was the unique key *tr_dt_id*, which connected the sequence S with the journey T that it is included in and further also the label, that the journey received from the frequency-based model. The sequencing method used two adjustable parameters; the *window size* with value j and the *step size* with value s, to partition the journey $T = \{e_1, e_2, ..., e_n\}$, into sequences $\{S_1, S_2, ..., S_k\}$, where $k \leq j$ and $S_i = \{e_{s(i-1)+1}, e_{s(i-1)+2}, ..., e_{s(i-1)+j}\}$. In other words, the *window size* value was used to define how many consecutive events e a sequence should contain while the *step size* value defined how many events to skip before extracting a new sequence, after extracting the previous sequence. In Figure 3.9 an example of how the sliding window works can be seen. Notice that the last sequence did not take a step of 2, but instead only one step. This is due to the length of the journey. When the window does not fill all positions with the sequence it moves backwards until the sequence reaches a length of n.

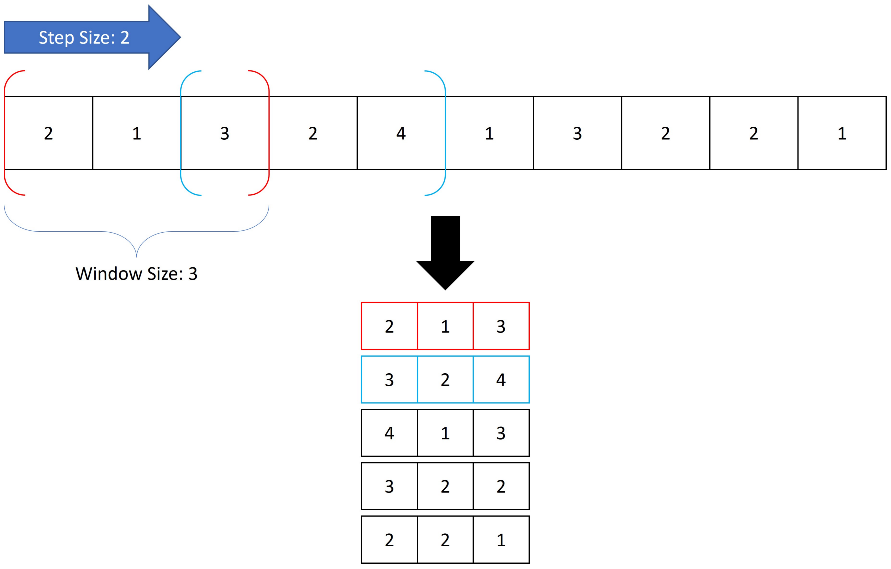


Figure 3.9 Example of how sequences can be constructed from a journey with the window size 3 and step size 2.

The optimal sizes were found by performing a grid search, using the real-time detector, described in the next section. The window size ranged between 2 and 9, while the step size depended on the size of the window. For a single step the minimum length allowed was 1 while the maximum was equal to the current *window size* value. This ensured that no events in the journey was skipped. To be able to compare different combinations of the two parameters and the real-time detector, a way to evaluate the models was implemented. The real-time detector $D(S)$ is explained in detail in Section 3.2.5, but in short it labeled each event sequence S as normal or anomalous. To evaluate, the F1-scores for the journeys containing *specific anomalies*, M_{known_a} described in Section 3.2.1.3, were used. A true positive classification represented a journey that both included at least one specific anomaly and at least one event sequence labeled as anomalous by the real-time detector. The combination that returned the highest score was used as the final combination. If two or more combinations returned the same F1-score, the combination that labeled the least number of journeys as anomalies was chosen, to prevent retrieving many false positives.

3.2.5 Real-time detector

The final step of the pipeline is the real-time detector itself, $D(S)$. Here we used the sequences extracted from the previous part to detect anomalies, as we needed to know where in the trip the anomalies occur. The idea is that an actual anomalous journey, T_a, will contain some sequences that are not common or non-existing in normal trips that are the reason for, or an indication of, that the journey is anomalous as a whole. To find sequences that were overrepresented among the anomalous trips, a new metric was introduced. It is a ratio calculated for each unique event sequence called *Ratio Estimation on Anomaly Share Over Normal Share* or simply *REASONS*. The expression for *REASONS* is presented in Equation 3.10 below, where S represents a unique event sequence. The numerator is the number of trips the unique sequence S is found in that are labeled anomalous, $|\{T \mid S \subset T \in M_{label\_a}\}|$, plus one, divided by the total number of anomalous labeled trips, $|M_{label\_a}|$. The denominator is the number of trips the unique sequence S is found in that are labeled normal, which can be expressed as $|\{T \mid S \subset T \in (M_{label\_a} \cap M)^C)\}|$, plus one, divided by the total number of normal labeled trips, $|(M_{label\_a} \cap M)^C|$. By adding ones we avoided division by zero, as well as getting a *REASONS* value of zero. This was done to be able to compare the different values, percentage wise.

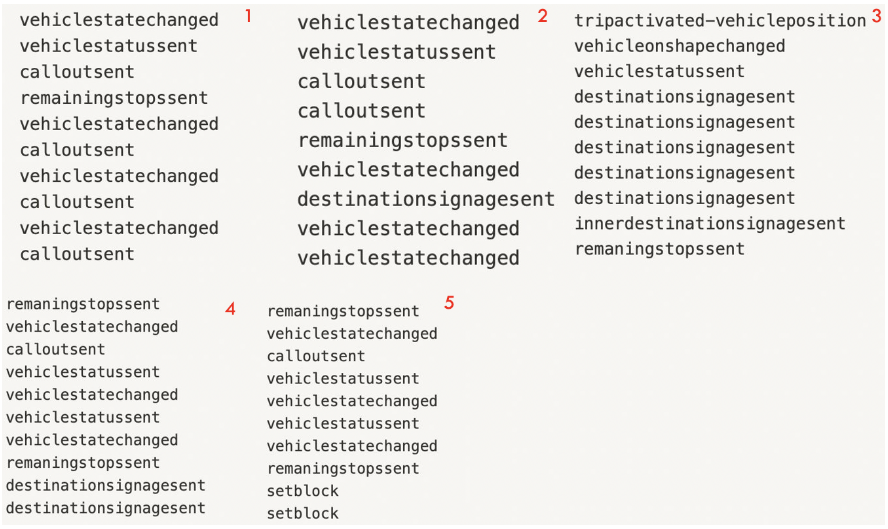
$$REASONS_S = \frac{(|\{T \mid S \subset T \in M_{label\_a}\}| + 1)/|M_{label\_a}|}{(|\{T \mid S \subset T \in (M_{label\_a} \cap M)^C)\}| + 1)/(|(M_{label\_a} \cap M)^C|)} \tag{3.10}$$

A sequence that overwhelmingly occurred in anomalous labeled journeys compared to normal labeled journeys returns a high ratio. The unique sequences were ordered in descending order of their *REASONS* value, so that the event sequences that occurred the most in anomalous trips compared to their occurrence in normal trips, came first. The detector chose a top percentage and marked them as the final anomalies. These were then used in the real-time detector $D(S)$ where an alarm is sent when one of these anomalous sequences is detected.

Important to note is how the top percentage was chosen. In the grid search described in the previous section, 3.2.4 Sequence-based representation, the top percentage value was also determined. The tuning of top percentage was performed simultaneously as the tuning of the window parameters and the optimal value was returned together with belonging window parameters. The values tested for the parameter *TopAnom* in the grid search ranged between 5% and 30% with 5 percentage point steps, which gave the search 6 alternatives to try.

3.3 Evaluation with simulated anomalies

Due to the fact that the pipeline used unlabeled data and therefore alternative metrics for tuning the models, the evaluation of the whole method differed from a conventional machine learning evaluation. As in other unsupervised problems, labels were only used for an observational evaluation and not for tuning the models. To produce an evaluation data set five simulated anomalous event sequences were provided from Gaia, which were randomly injected into 10% of a data set of *clean data*, hereafter denoted as M_{a_eval}. As anomalous event sequences can occur several times during the same trip, each randomized injection was independent of the other injections. This means that the same trip could be randomly chosen for an injection of a simulated anomaly multiple times. The simulated and injected anomalous sequences are presented in Figure 3.10 below.



```
1
vehiclestatechanged
vehiclestatussent
calloutsent
remainingstopssent
vehiclestatechanged
calloutsent
vehiclestatechanged
calloutsent
vehiclestatechanged
calloutsent
```

```
2
vehiclestatechanged
vehiclestatussent
calloutsent
calloutsent
remainingstopssent
vehiclestatechanged
destinationsignagesent
vehiclestatechanged
vehiclestatechanged
```

```
3
tripactivated-vehicleposition
vehicleonshapechanged
vehiclestatussent
destinationsignagesent
destinationsignagesent
destinationsignagesent
destinationsignagesent
destinationsignagesent
innerdestinationsignagesent
remaningstopssent
```

```
4
remaningstopssent
vehiclestatechanged
calloutsent
vehiclestatussent
vehiclestatechanged
vehiclestatussent
vehiclestatechanged
remaningstopssent
destinationsignagesent
destinationsignagesent
```

```
5
remaningstopssent
vehiclestatechanged
calloutsent
vehiclestatussent
vehiclestatechanged
vehiclestatussent
vehiclestatechanged
remaningstopssent
setblock
setblock
```

Figure 3.10 The five simulated anomalous event sequences, numbered from 1 to 5.

The simulated event sequences above can be divided into three different categories: having multiple *calloutsent* events close to each other, having multiple *destinationsignagesent* events close to each other and having events that usually occur in the beginning of a trip, after a bus stop. Event sequence number 1 and 2 belong to the category of having several *calloutsent* events closely, with number 1 being more extreme than number 2. Both are anomalies as the event is logged when an audio callout notices the passengers that the bus is soon at a bus stop, which only should be done once before each bus stop. Event sequence number 3 and 4 belong to the category of having multiple *destinationsignagesent* event closely, with number 3 being more extreme than number 4. This is an anomaly as the *destinationsignagesent* event is logged when the bus changes display name on the outside, which should not be done many times during a short amount of time. The last event sequence, number 5, is anomalous as the *setblock* event comes after an event sequence that is logged when the bus approaches a bus stop, stops at the bus stop and then departures from the bus stop. The normal occurrence of the *setblock* event is in the beginning of the journey, before any bus stops have been approached.

To evaluate pipeline runs that are true to the real case, a data set with *residue data* was created, called *residue data*$_{eval}$, that included the trips with injected anomalies, called M_{a_eval}. In addition to the trips within M_{a_eval}, normal trips were added to the $residue\ data_{eval}$ so that the proportion of $clean\ data_{eval}$ and $residue\ data_{eval}$ was the same as for the real case. In Figure 3.11 the data set partitioning is presented. M is the data set that included all the trips and by the earlier described heavy filtering, it was divided into *residue data* that included all anomalous trips, M_a, and *clean data*, that contained only normal trips. The just mentioned process of injecting anomalies into the data set and further creating $clean\ data_{eval}$ and $residue\ data_{eval}$ is depicted to the right in the sketch.

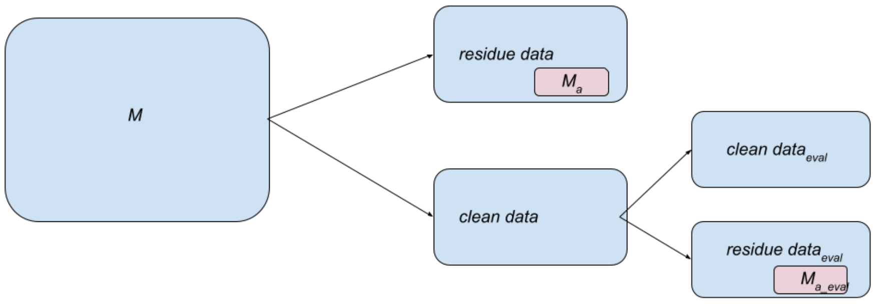


Figure 3.11 A sketch over the origin of the evaluation data set. The data marked with *eval* belong to the final evaluation data set.

After creating the two data sets, the pipeline proceeded exactly as in the real case, with additional evaluation, first after the initial classification with the clustering, PCA and autoencoder and secondly with the real-time detector. Recall, precision and F1-score were used for evaluating. In addition, the recall value for each simulated anomaly was calculated to enable further analysis of the method's strengths and weaknesses. In Figure 3.12 an overview of the pipeline is displayed that also marks where the evaluations are performed. As can be seen, the initial evaluation was done after the first labeling while the second evaluation was performed after the real-time detector.

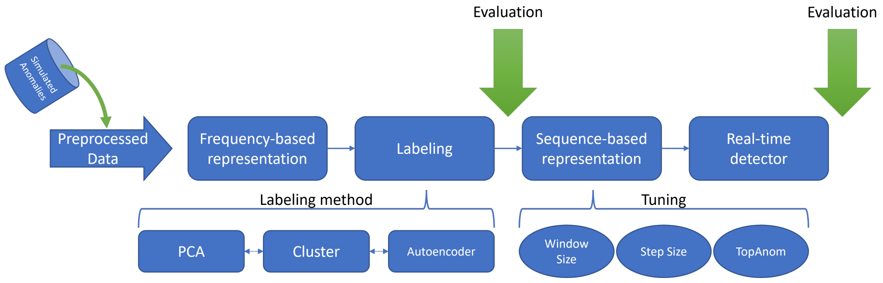


Figure 3.12 The same pipeline presented in Figure 3.1 with further illustrations on where the two evaluations are performed.

Which model combination that was evaluated depended on the performance of all model combinations in the initial labeling. For each new data set, tuning of the PCA border values as well as the autoencoder's and cluster algorithms' hyperparameters needed to be done. As described in Section 3.2.3, every intersection and union combination of the three models were tested as well as the base models on their own. This resulted in 11 different models to evaluate: 3 single models, 6 combinations of two of the models and 2 combinations of all models. Each combination was also tested on 10 different randomly injected data sets.

4

Results

In the following chapter, results from evaluation of 10 experiments are presented. In Section 4.1, the evaluation metrics for the frequency-based labeling as well as the real-time detector are presented. Throughout the chapter, the results from different combinations will be presented. The name for each combination follows a structure; for the base models the names *PCA*, *Cluster* and *Autoencoder* are used. For the combinations with all three models the names *Union* and *Intersection* refer to the combination where either union or intersection was used to combine the models. The remaining combinations are created using two base models through either union or intersection.

4.1 Precision, recall and F1-score

As described in Section 3.3 the evaluation was performed using simulated anomalies provided by Gaia. During the injection of the simulated anomalies only *clean data* was used, to ensure that all samples except the simulated anomalies were normal. Totally 10% of all trips in *clean data* received one or more simulated anomalies, forming the subset M_{a_eval}. Two data sets, $clean\ data_{eval}$ and $residue\ data_{eval}$, with the same proportions as the original *clean data* and *residue data* were then created by splitting the anomaly injected *clean data*. The whole M_{a_eval} was included in the $residue\ data_{eval}$, together with normal data, similar to the real-life case.

This randomized injection was performed 10 times producing 10 different data sets. The evaluation was performed on each data set with each model. As described in Section 3.3, the models tested were the autoencoder, PCA and clustering algorithm by themselves, as well as all combinations of intersection and union of them. This means that in total 110 evaluations were performed. The metrics used for the evaluation were recall, precision and the F1-score where a true positive classification mapped to an actual anomaly, labeled as an anomaly. The evaluation was performed both after the initial frequency-based labeling, where complete journeys were labeled, and also after the real-time detector, $D(S)$, where sequences of events, *S*, were labeled. As the thesis aimed to develop a real-time anomaly detector, the latter evaluation was of highest priority.

The results are presented in boxplots, in which the data is displayed as a box with whiskers. The box represents where 50% of the data is located and has its edges at the first and third quartile. The whiskers represent the highest and lowest value. The median score from the 10 experiments for each combination is also presented in two tables, on for the initial evaluation and another for the real-time evaluation.

4.1.1 Evaluation of initial labeling

The results from the initial frequency-based evaluation can be seen in the figures 4.1, 4.2 and 4.3 using the metrics F1-score, recall and precision. The figures have the evaluation score on the *y*-axis and the corresponding model combinations at the *x*-axis. All metrics range between 0 and 1, where 1 indicates optimal model performance. In general higher values of the three metrics are desirable, however as mentioned in Section 2.5.4, the F1-score is suitable as primary metric for a problem using data with unbalanced classes. Precision and recall do not separately give a fair picture of the overall performance of a model. Nevertheless, they are valuable for the analysis of the models, as they give the user more insight in the weaknesses and strengths of the models. In Table 4.1 the mean scores from the 10 evaluation runs for each combination can also be seen.

The experiment tested how well the initial labeling, meaning the PCA model, autoencoder and clustering algorithm, labeled the simulated injected samples as anomalies. Each box in the figures represent one of the 11 different combinations the three initial labeling models form.

In Figure 4.1 below, the recall is presented for the different model combinations tested. As mentioned in Section 2.5.2, the recall is the fraction of actual anomalies that are correctly labeled as anomalies. *PCA*, *Intersection*, *Intersection - Auto/PCA* as well as *Intersection - Cluster/PCA* did all receive a similar recall value distribution, both considering the median values of between 0.16 and 0.17, as seen in Table 4.1, and the variances shown by the whiskers and quartile makers that are plotted at the same places. *Cluster*, *Intersection - Auto/Cluster* and *Union - Cluster/PCA* did all receive a mean recall value between 0.69 and 0.73 as seen in Table 4.1. However, the variances differ, with the smallest belonging to *Union - Cluster/PCA* while the largest belongs to *Intersection - Auto/Cluster*. The rest of the models all have a median recall value close to 0.98, although the variance is greatest for *Autoencoder* and *Union - Auto/PCA*, while the tests belonging to *Union* and *Union - Auto/Cluster* resulted in a smaller range of recall values.

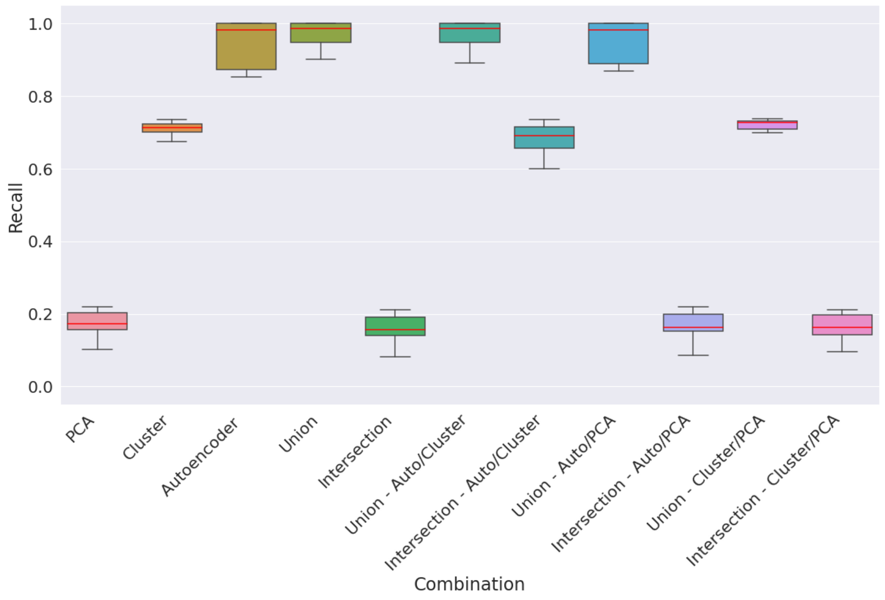


Figure 4.1 Boxplot with the different combinations evaluated on the initial frequency-based experiment using the recall metric. The *y* axis ranges between 0 and 1 while each box represent a combination.

Displayed in Figure 4.2 is the precision, calculated for the 10 experiments and 11 combinations of models, after the initial frequency-based labeling. Compared to the recall plotted above, the precision values are generally lower. As described in Section 2.5.1, precision measures the fraction of anomaly labeled samples, that actually are anomalies. All of the tests done with only *Cluster* resulted in precision values just above 0.45, which gives the box the appearance of a line. The variance of the precision value for *PCA* and *Intersection - Cluster/PCA* is very similar, however the distribution belonging to the latter combination is located at greater precision values. *Intersection - Auto/Cluster* has the largest median precision value of 0.61, seen in Table 4.1. *Intersection* has the second largest median precision value of 0.51. The values belonging to *Union* and *Union - Auto/PCA* are similarly distributed, while the remaining combinations have closely located median values, yet differing variances.

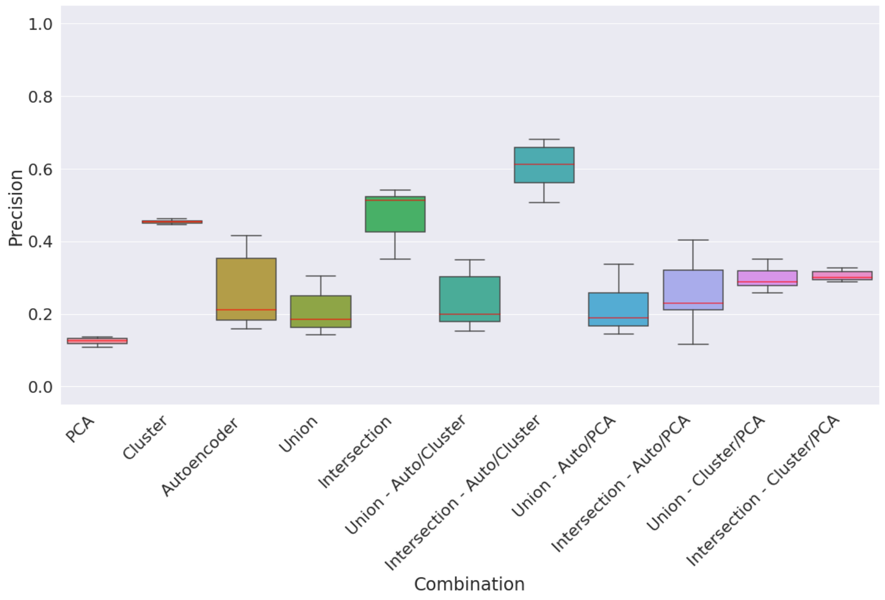


Figure 4.2 Boxplot with the different combinations evaluated on the initial experiment using the precision metric. The *y* axis ranges between 0 and 1 while each box represent a combination.

In Figure 4.3 below, the F1 values for all combinations and experiments are presented. As the F1-score both considers the precision and recall and at the same time works well with data sets containing unbalanced classes, it is suitable for determining the overall performance for the combination models in this thesis. Actually, no new information needs to be gathered to calculate the F1-score, yet it is useful as it would be difficult to, in a systematic way, manually rank the models taking both precision and recall into account. The best performing combination, regarding the F1-score, was *Intersection - Auto/Cluster* with a median score of 0.65. Both distributions of F1-scores belonging to *PCA* and *Cluster* have small variances, although the median values differ significantly. *Intersection - Auto/PCA*, *Union - Cluster/PCA* and *Intersection - Cluster/PCA* all have boxes with whiskers of similar appearances, where the median for *Intersection - Auto/PCA* and *Intersection - Cluster/PCA* are 0.19 respectively 0.21, while *Union - Cluster/PCA* scores a median value of 0.41. The rest of the combinations all have values that span over an approximately equally big range, measuring from the lowest to the largest values, with median F1-scores between 0.23 and 0.35.

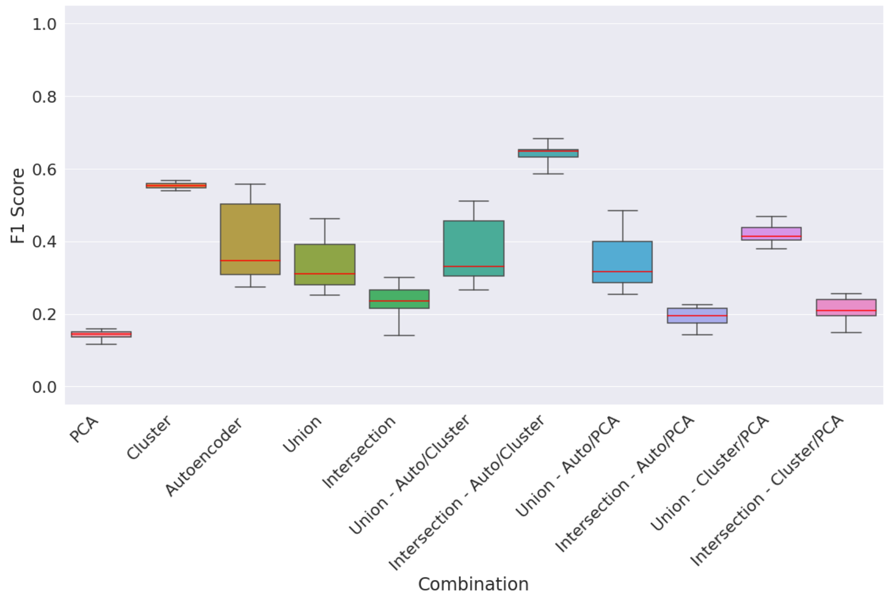


Figure 4.3 Boxplot with the different combinations evaluated on the initial experiment using the F1-score. The *y* axis ranges between 0 and 1 while each box represent a combination.

	Recall	Precision	F1-score
PCA	0.17	0.13	0.14
Cluster	0.71	0.45	0.55
Autoencoder	0.98	0.21	0.35
Union	0.99	0.18	0.31
Intersection	0.16	0.51	0.23
Union - Auto/Cluster	0.99	0.20	0.33
Intersection - Auto/Cluster	0.69	0.61	0.65
Union - Auto/PCA	0.98	0.19	0.32
Intersection - Auto/PCA	0.16	0.23	0.19
Union - Cluster/PCA	0.73	0.29	0.41
Intersection - Cluster/PCA	0.16	0.30	0.21

Table 4.1 Median scores for the initial labeling on each combination. The metric presented are recall, precision and F1-score and can be seen in three separate columns.

4.1.2 Evaluation of real-time detector

The results from the evaluation of the real-time detector are presented in three boxplot figures; one for the recall score, see Figure 4.4, one for the precision score, see Figure 4.5, and one for the F1-score, see Figure 4.6. Each box represents the score obtained on 10 different randomized data sets on one of the 11 combination of initial labeling models. The median scores for each metric on each combination can also be seen in Table 4.2.

The recall values for injected anomalies, after the detection performed by the real-time detector, are plotted in Figure 4.4 below. All intersection combinations including *PCA*, i.e. *Intersection*, *Intersection - Auto/PCA* and *Intersection Cluster/PCA*, as well as *PCA* itself, did not manage to produce any true positives. Hence, the corresponding recall scores are all zero. All the other combinations produced at least one test which labeled all injected anomalies as anomalies. Among those combinations, tests with *Union - Auto/Cluster* resulted in the recall score distribution with the least variance and tests with *Union* returned the second-smallest variance. The *Autoencoder*, *Intersection - Auto/Cluster*, *Union - Auto/PCA* and *Union - Cluster/PCA* have a larger span of recall values, from around 0.50 to 1.0. Yet, the largest span of recall values belongs to the tests using *Cluster*. In Table 4.2 we can see that all union combinations including *Cluster*, as well as *Cluster* itself, reaches a median score of 1.0.

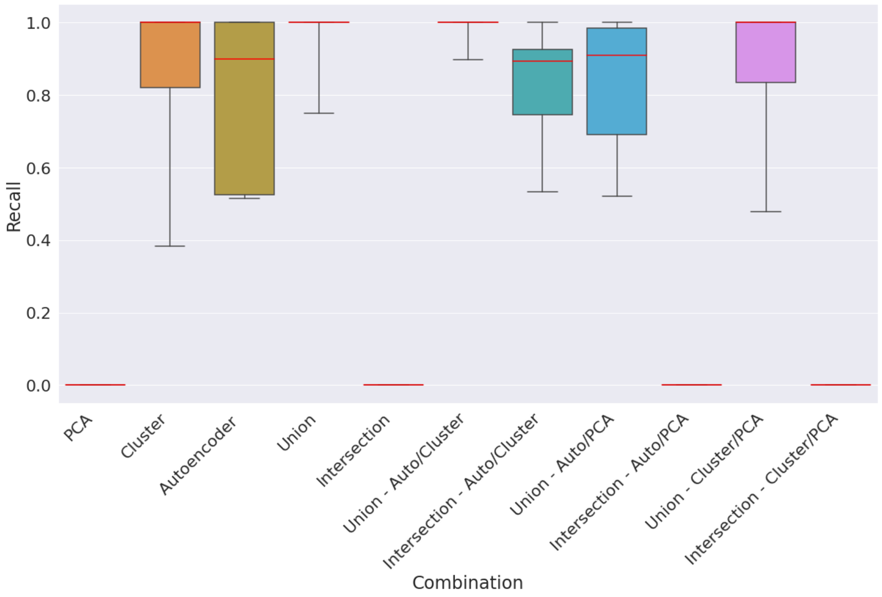


Figure 4.4 Boxplot with the different combinations evaluated using the recall metric with the real-time detector. The y axis ranges between 0 and 1 while each box represent a combination.

As in the evaluation of the initial frequency-based labeling, the precision values are in general less than the recall values for the real-time detector. The precision scores are displayed in Figure 4.5 below. Similar as for the recall values, *PCA* and all intersection combinations including *PCA*, i.e. *Intersection*, *Intersection - Auto/PCA* and *Intersection Cluster/PCA*, did not manage to label one injected anomaly correctly, resulting in zero valued precision scores. The undoubtedly highest median precision value was given by *Intersection - Auto/Cluster* with a median score of 0.72 as seen in Table 4.2. *Autoencoder*'s tests gave many different precision scores, resulting in the largest span measuring from lowest to highest value. *Union - Auto/Cluster*, *Union - Auto/PCA* and *Union - Cluster/PCA* all got precision values with a median between 0.20 and 0.25, but having slightly different variances. The test corresponding to *Union* returned one high precision score of around 0.60, nevertheless, the other tests resulted in a median score of 0.11. Lastly the tests for *Cluster* resulted in the secondly highest median value of 0.51, seen in Table 4.2.

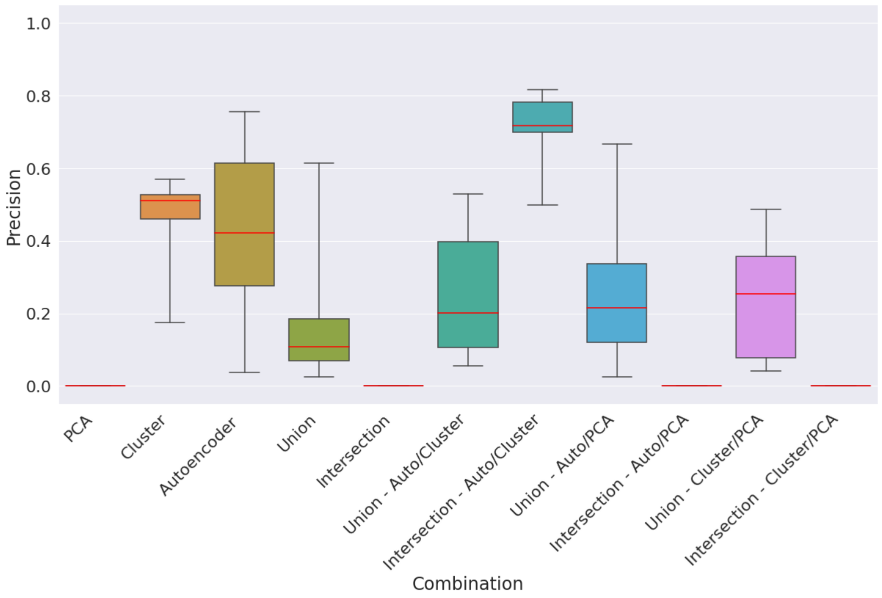


Figure 4.5 Boxplot with the different combinations evaluated using the precision metric with the real-time detector. The y axis ranges between 0 and 1 while each box represent a combination.

As mentioned previously, the F1-score is a combination of precision and recall and it does not consider any other information. In Figure 4.6 the F1-scores for the real-time model, using injected anomalies, are presented. The same combinations that result in zero valued precision and recall scores; *PCA*, *Intersection*, *In-*

tersection - Auto/PCA and *Intersection Cluster/PCA*, consequently return zero valued F1-scores. F1-score distributions belonging to *Union - Auto/Cluster*, *Union - Auto/PCA* and *Union - Cluster/PCA* show median values that all lie between 0.33 and 0.34, seen in Table 4.2, as well as ranges that only differs slightly. *Intersection - Auto/Cluster* has the highest F1-score, with a median value of 0.79, while the box belonging to *Cluster* contains the second greatest F1-score with a median value of 0.67, both values seen in Table 4.2. As for the precision plot above, *Autoencoder* has a score distribution with the largest range among the combinations, measuring from the smallest to the highest value. One of the tests performed with the *Union* combination returned an F1-score of almost 0.80, however the rest of the tests did not return as good scores with a median score of 0.20, seen in Table 4.2.

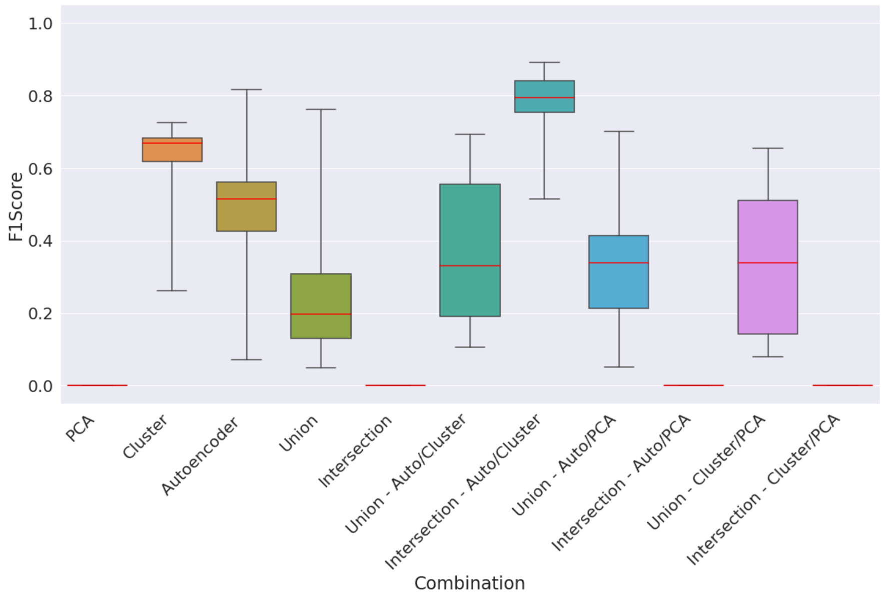


Figure 4.6 Boxplot with the different combinations evaluated using the F1-score with the real-time detector. The y axis ranges between 0 and 1 while each box represent a combination.

	Recall	Precision	F1-score
PCA	0.0	0.0	0.0
Cluster	1.0	0.51	0.67
Autoencoder	0.90	0.42	0.51
Union	1.0	0.11	0.20
Intersection	0.0	0.0	0.0
Union - Auto/Cluster	1.0	0.20	0.33
Intersection - Auto/Cluster	0.89	0.72	0.79
Union - Auto/PCA	0.91	0.22	0.34
Intersection - Auto/PCA	0.0	0.0	0.0
Union - Cluster/PCA	1.0	0.25	0.34
Intersection - Cluster/PCA	0.0	0.0	0.0

Table 4.2 Median scores for the real-time detector on each combination. The metric presented are recall, precision and F1-score and can be seen in three separate columns.

4.1.3 Evaluation on single injected anomalies

In Section 3.3 the five types of simulated anomalies were presented. In Figure 4.7 the recall scores for both the initial frequency-based labeling and the final real-time detection, $D(S)$, are presented, for each model combination on every simulated anomaly, $S \subset T \in M_{a\_eval}$. As mentioned earlier in the report, the recall is the percentage of the correctly labeled actual anomalies. Since we in this section are interested in each anomaly by itself, recall is the fraction of actual single injected anomalies of type number X, that actually are labeled as anomalous. The reason for using recall and not precision is that the precision does not say anything about how many of the actual anomalies that are labeled correctly, just the percentage of anomaly labeled samples that were correctly labeled. Using precision here would probably result in very low values as all the anomalies except for number X would be counted as false positives.

In the figure the blue bars represent the initial labeling using the frequency-based representation while the orange bars represent the real-time detection. The figures are meant to enable a deeper understanding of how the labeling and the detector, together with the different model combinations, perform on the different simulated anomalies.

As seen in the plots the model combination using only *PCA* or the intersection combinations containing *PCA*, all return a recall of 0 for the real-time detector. The corresponding precision is the second-lowest recall value for all anomalies and model combinations, with a large gap to the third-lowest value. The *Autoencoder*, *Union*, *Union - Auto/Cluster* and *Union - Cluster/PCA* all returns recall values of 1 for all injected anomalies, except for the fourth, for both the frequency-based labeling and the real-time detector. All simulated anomalies return similar looking recall bars for *Cluster*, *Intersection - Auto/Cluster* and *Union - Cluster/PCA*, where the recall values for the initial frequency-based labeling are lower than the recall values for the real-time detector. The exceptions among the recall values corresponding to the just mentioned model combinations and injected anomalies are the values returned by evaluating the performance of *Intersection - Auto/Cluster* on the fourth injected anomaly, that are significantly lower than the other recall values for that model combination, as well as the values for *Cluster* and *Union - Cluster/PCA* for the forth anomaly. Except for the values belonging to *PCA*, the recall values for the frequency-based labeling is equal to or lower than the same experiment's recall value for the real-time detector. The injected anomaly number 4 produces the lowest overall recall values with a good margin to the injected anomaly number 2 that produces the second overall recall values.

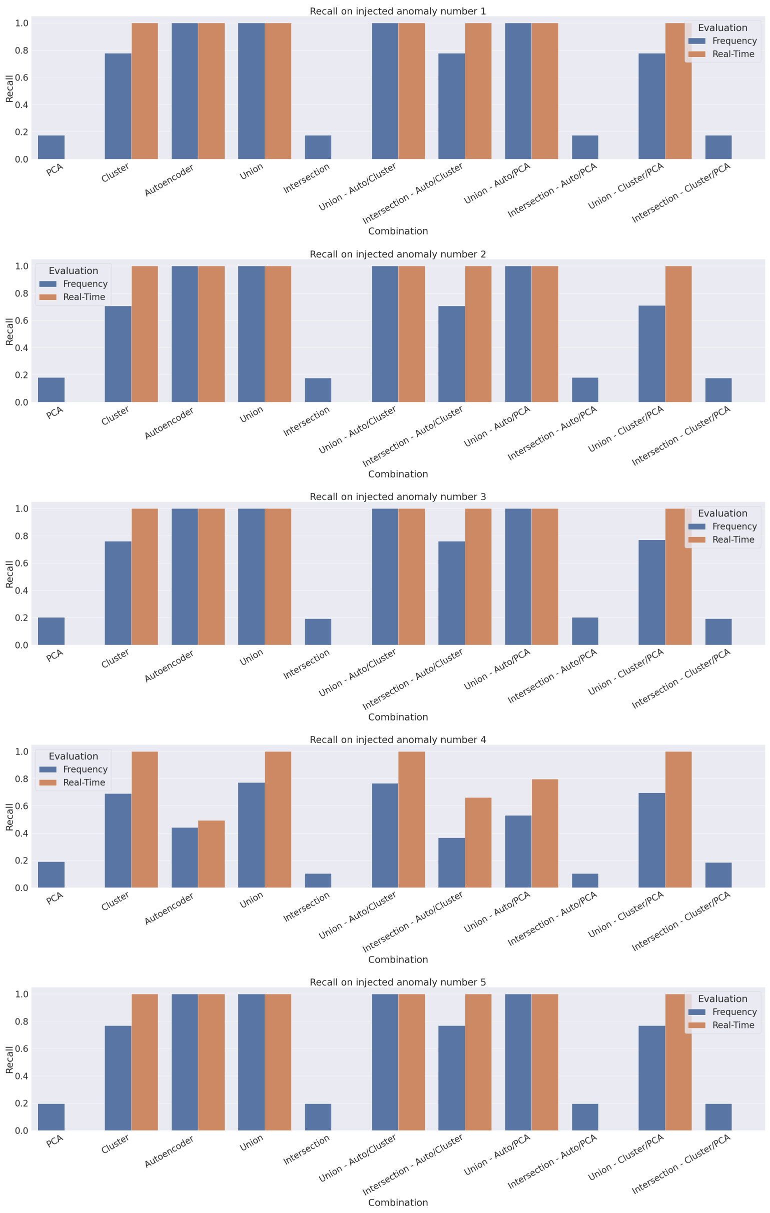


Figure 4.7 Recall for the initial and the real-time evaluations for each combination on each single simulated anomalies. The blue bars represent the initial labeling with the frequency vectors while the orange bars uses the real-time detector. The plots are ordered according to the five different simulated anomalies described in Section 3.3.

4.2 Result summary

The evaluation was performed with 10 runs for each of the 11 combinations. In the tables 4.1 and 4.2 the combination *Intersection - Auto/Cluster* can be seen scoring the highest F1-score, both after the initial labeling and the real-time detector $D(S)$. The median F1-score for the initial labeling reached a score of 0.65 and for the real-time detector a median F1-score of 0.79. In Figure 4.6 we can also see that the variance, measured with the range from the minimum and maximum score, for the combination *Intersection - Auto/Cluster* is relatively low compared to the remaining combinations (excluding the combinations that scored 0 in every evaluation run). This indicates that the combination is stable and reliable in scoring similarly to previous runs.

In Figure 4.7 the recall scores from the initial labeling and the real-time detector for each combination on each of the five simulated anomalies can be seen. The figure indicates that the first, second, third and fifth simulated anomaly are similarly difficult to detect while the fourth is significantly more difficult to detect. Every combination including the clustering algorithms with union scores 1.0 with the real-time detector $D(S)$ on each anomaly. However, the combinations with the PCA model using intersection scores 0.0 on the real-time detector on every anomaly.

5 Discussion

In the following chapter an analysis of the thesis's method and results is presented, as well as suggestions for future work. Firstly, the two different data representations are discussed, see Section 5.1 Frequency-based representation versus sequence-based representation, followed by a discussion about the metrics and tuning parameters, see Section 5.2. Thereafter, in Section 5.3 the results are analyzed and lastly, in Section 5.4, our suggestions for future work are presented.

5.1 Frequency-based representation versus sequence-based representation

To represent the journey's event logs by their event frequencies was an attempt to give the detector as much knowledge about the data set as possible, in a situation as this where labels were nonexistent. Since creating a detector that works in real-time is usually a greater challenge than providing it all historical and future data, it is intuitive to believe that first having access to all data would be advantageous. As mentioned in the method, see Chapter 5, the frequency-based representation was incompatible with real-time anomaly detection, as the journey needed to be finished for a fair calculation of each event frequency to be made. As the first classification was performed by giving the autoencoder, cluster algorithm and PCA model the event logs represented by the event frequencies per journey, the labeling was based on the number of times a certain event e occurred during a journey T, in relation to the number of times the other events occurred. In that phase, the models did not have the possibility to consider the order of the events.

Next, the labeled journeys were partitioned into sequences that were counted with respect to the event order and if they originated from anomalous or normal labeled journeys. Hence, the order of events had a significant role in that phase. The inability to discover fine event order deviations in the frequency-based labeling process and the limitation of not having access to future event occurrences in the real-time detector led to a discrepancy between which kind of anomalies that could be detected in the two stages. A journey that was easily labeled as anomalous in the frequency domain, probably also contained anomalous event sequences that the real-time detector could detect, for instance multiples of an event that usually occurred by itself. On the other hand, there was no indication of an anomaly for the frequency-based models when the order was shifted, if the total count of each event stayed the same. Theoretically, the event log for a journey could be reversed without being noticed by this detector. It is of high importance to be aware of that the real-time detector intelligence originates from a frequency-based labeling. Consequently, it should be approached as a detector that alerts about journeys that probably will deviate in terms of event frequencies, before the actual frequencies can be calculated. In contrary, the detector, or method as a whole, should not be approached as a model that detects small event order deviations.

5.2 Metrics and parameters

In the following section thoughts about the metrics and tuning parameters used in the method are presented. First comes Section 5.2.1 *CDLN* and *RDLN*, where the benefits and drawbacks regarding the metrics created specifically for this thesis are discussed. Thereafter, the Section 5.2.2 Metrics used for tuning parameters comes, which presents an analysis regarding that topic. Lastly the *window size*, *step size* and *TopAnom* are discussed in Section **??** *Window size* and *step size* and in Section **??** Top Anomaly percentage.

5.2.1 *CDLN* and *RDLN*

The metrics *CDLN*, *Clean Data Labeled Normal*, and *RDLN*, *Residue Data Labeled Normal*, were created to enable a solid sorting process when the most suitable model was to be chosen. Probably, the optimal metric would have been the F1-score, as it is suitable for classification problems with unbalanced classes. However, that metric was not accessible as the method of creating *clean data* needed *residual data* to both include normal and anomalous data (M_a), for ensuring that the *clean data* only included normal data. As a consequence, *true positives*, *true negatives*, *false positives* and *false negatives*, were not available. With the recall metric as inspiration, the fraction of *clean data* labeled normal was calculated, as well as the fraction of *residue data*, labeled normal. For the same reason as the recall by itself is not enough for representing the overall performance of a model, only using *CDLN* for extracting the most suitable model needed to be complimented with another metric. If not, a model that classified all samples as normal would achieve the highest score. Since the *residue data* included both normal and anomalous data, the F1-score was out of the question and instead the fraction $\frac{CDLN}{RDLN}$ was used, after extracting all models with a *CDLN* of at least 90%. As stated in Section 3.2.1.3 Labeling, that metric was used because it benefited maximization of *CDLN* and minimization of *RDLN*. This was, for the models with a $CDLN > 0.90$, wanted since we could assume that *clean data* only, or close to only, included normal data, while we also could assume that *residue data* included the existing anomalous trips M_a and therefore should $CDLN > RDLN$. A problem with having multiple conditions for extracting the best model could be that it is difficult to apply the same method on other, but similar problems. After the creation of *clean data* and *residue data*, which already demands the user to know a lot about the data, the user needs to set a limit for the lowest *CDLN* value, which also demands the user to know the data well. Possibly a metric that is more suitable than this combination of conditions can be created, nevertheless, it is not something that has been developed during this project.

5.2.2 Metrics used for tuning parameters

In the Sections 3.2.4 and 3.2.5 the process for tuning the three parameters, *window size*, *step size* and *TopAnom*, is described. The trips including *specific anomalies*, M_{known_a}, presented in Section 3.2.1.3 were used to find the optimal values. In the tuning search the F1-score of M_{known_a} was used, where a higher score was favorable. However, with this approach, a potential problem may occur. Because the F1-score was calculated by having the trips in M_{known_a} that were correctly labeled, as true positives, and a precision value that also only considered trips in M_{known_a} when dividing the true positives with all positives, it handled all other actual anomalies as actual normal data samples. Hence, an F1-score of this type that gets a value of 1 will probably not correspond to the most desirable result. The reason is that the result would correspond to having all actual specific anomalies labeled as anomalous, but not having any other samples labeled as anomalies. This would be the optimal result if no other anomalies existed, yet we are most certain that they did, as we could not expect the specific anomalies to represent every possible anomaly in the data set, i.e. $M_{known\_a} \subset M_a$ and $M_{known\_a} \neq M_a$ holds. Because the problem described in this thesis has a prerequisite of not knowing what kind of anomalies the data set contains, the optimal parameters with an F1-score of 1 are consequently not desirable. This concern was not considered in the implementation and could therefore be a realistic problem that may occur in some runs, when used in the future. However, during the evaluation the F1-score used in the tuning did not reach a problematic high value.

5.3 Result analysis

In the following section the results are discussed and analyzed. In particular, the result plots will be in focus. Firstly, Section 5.3.1 Result on the evaluation from the initial labeling, comes. Thereafter an analysis of the real-time detector comes, see Section 5.3.2 and lastly the Section 5.3.3 is presented.

5.3.1 Result on the evaluation from the initial labeling

From the results in Section 4.1.1 a clear observation can be made. The best combination after the evaluation runs was the intersection between the autoencoder and the clustering algorithm using the F1-score, that represents the overall performance best. While both models performed above average individually, the intersection of them elevated the score. This can be explained by the precision plot, see Figure 4.2, where the individual models show a lower score in comparison with the intersection of both. Meanwhile, the recall for the intersection combination seen in Figure 4.1 is lower than the recall for the individual models, especially compared to the autoencoder's score. However, the increase in precision contributed to produce a greater F1-score, enough to become the best result. This means that the intersection operation proportionally excluded more wrongly labeled normal trips, than correctly labeled anomalous trips, $T_a \in M_{a\_eval}$, from the subset of anomaly labeled journeys.

In further inspection concerning the combinations another observation is made. Focusing on the recall plot seen in Figure 4.1 we observe that combinations using union correlates strongly with the highest scoring individual model included in the combination. In the same plot we can similarly see that the combinations using intersection correlate strongly, with the individual model included in the combination that scores the lowest. For instance, the union of the autoencoder and the clustering algorithm scored similarly to the autoencoder, while the intersection between the PCA model and the clustering algorithm scored similarly to PCA individually. Furthermore, we can observe that combinations using union in general always scored higher compared to the individual models the combination originated from.

A similar looking pattern can be seen in Figure 4.2, for the combinations using either union or intersection. Given a combination of models, we observe that the precision presented in the plot is always higher for the intersection combinations, compared to the corresponding union combinations. This indicates that the journeys labeled as anomalies by two or all models often were true anomalies, i.e. T_a, and therefore the precision increased. In Figure 4.1, showing the recall values, we see that the union of combinations is always higher than the corresponding intersection combinations. Although the recall values are greater for the union combinations than for the intersection combinations, the intersection with the autoencoder and clustering algorithms received a median value of 0.69, indicating that it caught a majority of the simulated anomalies, $T_a \in M_{a\_eval}$.

5.3.2 Result on the real-time detector

In the figures in Section 4.1.2, the results from the evaluation runs for the real-time detector $D(S)$ are presented. As described in Section 3.2.5, the detector labeled sub-sequences of events S as normal or anomalous. When evaluating, all journeys that contained anomaly labeled sub-sequences were considered anomalous, i.e. if S_a

is a sub-sequence in the trip T then T is expected to belong to M_{a_eval}. The journeys that totally lacked anomaly labeled sequences of events, were considered normal. Consequently, the labeled journeys, and not the sub-sequences, were used for calculation of evaluation metrics. In the figures concerning the real-time detector a larger variety of metric values are seen, in comparison to the corresponding values coming from the evaluation of the initial labeling. Possibly, this could be an indication of that the detector either performed different on the different randomized data sets or that it was dependent of the tuning parameters' values.

In Figure 4.4 the recall scores for each combination are visualized. An observation can be made that the union is superior to the intersection for every pair of models. This is expected for the recall score, as the definition of union makes a recall value decrease impossible. However, the intersection between the autoencoder and the clustering algorithm reached a sufficiently good recall score, compared to the other intersection combinations. The explanation seems to be that both the clustering algorithm and the autoencoder reached good recall scores, which gave the intersection of the models a good chance of performing well. Furthermore, every combination using intersection with the PCA model produced a recall score of 0. This is due to the PCA model that did not classify a single simulated anomaly correctly. It is unexpected that the PCA model performs very poorly, however, it can be explained by the initial labeling results seen in the figures 4.1 and 4.2. In these figures, we observe that not only does the PCA model had difficulties finding the simulated anomalies, but also it classified a significant amount of normal data as anomalies. As seen in the figures, the poor performance could be suspected already in the initial frequency-based labeling since both the recall and precision scores are low. The combination of low scores on both recall and precision, together with the fact that sufficiently many samples were labeled anomalous in that phase, resulted in a large amount of false positives. Hence, the real-time detector's poor performance for PCA. An interesting analysis would be to investigate what kind of data the PCA model instead labeled as anomalies. The possibility still remains that the false positives somehow were anomalies, and that with a further data analysis, we would discover new types of anomalies that has not been acknowledged yet. Although, from the current point of view they are outliers data representation-wise but not anomalies, i.e. unwanted outliers.

In Figure 4.5 the precision for the real-time detector is displayed and we can again see that every combination with PCA through intersection got a score of 0. This followed from the recall for the same combinations being 0. However, a more interesting observation is the precision reached by the combination with the autoencoder and the clustering algorithm using intersection. This combination was the most capable in not incorrectly classifying normal data as anomalies and reached therefore the highest precision score. At the same time the combination reached a high recall score, even though it is not the best. These two values resulted in the best F1-score by all combinations, seen in Figure 4.6.

5.3.3 Result on evaluation of individually simulated anomalies

Presented in Figure 4.7 the recall values for each simulated anomaly on each combination are visualized. This illustrates the ability of correctly label the simulated anomalies individually, for the frequency-based labeling and for the real-time detector. As seen in the figure; the model combinations and real-time detector performed similar with all simulated anomalies except the fourth. With these individual anomalies, i.e the first, second, third and fifth, the model combinations follow an interesting logic. While inspecting the recall values corresponding to *PCA*, *Cluster* and *Autoencoder*, we observe that the recall values intersection combinations are always similar to the model with the lowest recall value. This holds true for both the initial labeling and the real-time detection. Furthermore, if the models are combined through union the resulting recall values are always close to the highest scoring base model. This could be an indication that the classified anomalies from the lowest scoring model, l, was a subset of the classified anomalies from the second highest, m, and highest, h, scoring models. Moreover, this also indicates that the classified anomalies from the second-highest model was a subset of the classified anomalies from the highest model. An approximation of this idea could be written as

$$M_{a_l} \in M_{a_m} \in M_{a_h}.$$

One simulated anomaly, presented in Figure 5.3.3, that differentiates itself from the remaining anomalies is the fourth one. It is especially clear that it has been harder to detect that anomaly, when analyzing the results corresponding to the autoencoder. The poor performance resulted in limited recall scores for the combinations containing the autoencoder. A particularly interesting observation can be made when we take a closer look at the combination *Union - Auto/PCA* and *Intersection - Auto/PCA*. In the previous cases the union between two models have resulted in the same score as the highest scoring of the two models. Similarly, the intersection of two models have produced a similar score to the worst model. Here we can instead observe that the union between the autoencoder and the PCA model returned a score higher than the scores from the two separated models, both for the initial labeling and the real-time detector. Likewise, the intersection of the two base models scored worse than both models separately. This indicates that the two sets of classified true anomalies from the two models did not fully overlap, but instead had some true positives outside the intersection of both sets.

5.4 Future work

During the project alternative ideas and approaches were discussed that did not fit within the scope of this Master's thesis, but could be interesting future work. One main part in the pipeline is the frequency-based representation that was used as the input for the three labeling models. The representation in the current implementation include only the value *CO_diff*, outside the frequencies. However, including more information about each completed journey in the representation, could possibly lead to an improvement of the detector. Important to remember is that the representation in the labeling phase does not have to be functional in real-time, why complete journeys as well as meta information about the journeys could be integrated.

Another alternative to represent a journey as a vector could be using techniques from the field of Natural Language Processing, *NLP*. As language is build on sentences constructed of words, an allegories could be made with the trips' event logs. The sentences in our case would then be the complete journeys and the words would be the events. A possible representation of the trips could be created using embedding techniques commonly used in *NLP*, where a sentence is mapped to a vector. The advantage with this approach is that the order of the events would have a significant impact on the labeling process, in comparison to the frequency-based labeling currently used.

In addition, an exciting analysis for future work would be to include other simulated anomalies to test the limits of the different combinations. The new anomalies would preferably originate from different structures. This would increase the understanding of which types of anomalies are better detected by the different combinations and models. In the future the pipeline could also be tested on newly discovered anomalies, giving an understanding if the detector is capable for similar anomalies.

The clustering algorithm used for the initial labeling uses the inherit ability of defining samples as outliers for the labeling. This technique however depends on the anomalies being isolated outliers, otherwise the clustering algorithms often recognizes a set of anomalies as a cluster. The problematic aspect of this is that systematic anomalies can occur in the data set without being labeled as anomalies. A possible improvement for this problem could be using a fourth base model that is specified in finding structural anomalies.

In addition to only testing intersection and union one by one, as done in this thesis, combinations of them could also be tested. An example can be made with three example sets: A, B and C. Currently we combine the sets either through $A \cap B \cap C$ or $A \cup B \cup C$. The sets could however also be combined using both methods, for instance by the combination $A \cup (B \cap C)$. If we then find a fourth base model that is specified for systematic anomalies, a combination can be constructed using the currently best combination *Intersection - Auto/Cluster* and the union with the fourth base model.

During the initial labeling in the pipeline some limits were set to be able to find the optimal PCA borders, clustering algorithm and threshold for the autoencoder. For the clustering algorithms any model that resulted in a value of *CDLN* less than 90% was discarded as well as models with more than 30% noise. The PCA model discarded borders with a *CDLN* value less than 95% and the autoencoder also had a limit of a *CDLN* value of 90% for the threshold. These limits where tested and chosen based on ideas on how we desired the method to work and with trial and error. Future work could however focus on finding a more careful tuning of the limits. This would enable the models to better label the journeys correctly and therefore increase the probability that the real-time detector detects actual anomalies.

5.4.1 Real-world applications

A model is only as good as the application the model will be used in. The real-time detector has great potential for improving public transportation applications that contain logs. With more confidence in that the data collected does not include anomalies, even more reliable models can be created. By implementing capabilities for handling the mistakes detected by the real-time detector, the application would both detect and correct anomalies when they happen. This would improve the overall passenger experience and make public transportation a more attractive choice.

In Section 1.1 the desired real-world application for the detector is discussed. With the results from the 10 evaluation runs we can confidently say that the detector with the combination *Intersection - Auto/Cluster* performed acceptable for the application. By studying the recall and precision results we see that almost every anomaly was detected while few false alarms occurred.

The detector can be implemented both as a model in the vehicles or in the cloud, since it does not require any heavy calculations to run in real-time. The implementation will simply be done by only include events used in the detector, as the stream of events is received. Continuously the stream of events is divided into suitable sequences according to the window size and then looked up in the detector table, to determine if the sequence is anomalous. However, an implementation for unknown sequences have to be made. The model could either mark the sequence with a third label, for example *New Sequence* or simple label the sequence as an *Anomaly*, depending on what Gaia finds as the best approach. The *New Sequence* labeling could also be an indication that the detector should be retrained and that a new underlying structure have been implemented in the logging of the events.

6
Conclusions

In this section summaries of findings related to the questions at issue, stated in Section 1.3.1, are presented.

How can a real-time detector for sequences of log events be defined?
We chose to create a real-time detector that relied on a labeling made with a data set containing log event frequencies of historical journeys. The use of event logs for whole trips instead of only sequences of events in the first phase, made it possible for the detector to make a final classification by comparing the individual event sequences' occurrences in normal labeled journeys versus anomalous labeled journeys. However, this would probably not have been the natural approach if the event sequences originally were labeled, hence the answer to this question is primarily applicable for problems including unlabeled data sets.

How well does the above mentioned real-time detector perform, when only having access to an unlabeled data set?
As presented in Section 4.1.2, Evaluation of the real-time detector, the model combination of the autoencoder and the clustering algorithm performed best, with a median F1-score of 0.79. In the method we motivated using union and intersection of the base models (PCA, autoencoder and clustering algorithm), with that the union combinations would perform best if the individual base models found different types of anomalies, while the intersection combinations would perform best if the actual anomalies were the anomalies that the base models agreed on. When comparing the clustering algorithm, the autoencoder and the combinations of them, the intersection statement holds; to label only their mutual anomalies, as anomalies, gives the best overall result. The combinations including PCA performed very poorly, as PCA did not manage to label any simulated anomalies correctly.

How can a small data set of known anomalous trips be used for tuning, in the pipeline belonging to the real-time detector, while keeping a generic nature of the detector where it also detects anomalies outside the known anomalies?
The tuning in question, regarded the *window size* value, the *step size* value and the

TopAnom percentage value. For deciding the best set of values, the metric used was the F1-score for M_{known_a}; the small data set with known anomalous trips. As discussed in 5.2.2 Metrics used for tuning parameters, despite the lack of a constraint, an F1-score of 1 would not be desirable as that would mean that no other anomalies except for the *specific anomalies* would be detected. However, the F1-score was never close to 1 in reality, probably because the labeling performed by the model combinations ensured to capture a suitable amount of varying anomalies, together with that the *specific anomalies* represented the distribution of actual anomalies well. We will not be able to grade the generality of the model, more than saying that the best model combination; intersection of the autoencoder and the clustering algorithm, performed well on four out of five simulated anomalies. However, what we can state with certainty, is that the real-time detector is generic in some sense, as it manages to to detect trips including the four simulated anomalies, even though the tuning is made with the *specific anomalies*.

Bibliography

Agha, S. A. A. L. S. P. Z. (2017). "Unsupervised real-time anomaly detection for streaming data". *Neurocomputing* **262**, pp. 134–147. DOI: https://doi.org/10.1016/j.neucom.2017.04.070.

Akoglu, M. Q. M. Y. Z. X. Z. L. (2017). "A large-scale study on unsupervised outlier model selection: do internal strategies suffice?" DOI: https://arxiv.org/pdf/2104.01422.pdf.

Ankerst, M., M. M. Breunig, H.-p. Kriegel, and J. Sander (1999). "Optics: ordering points to identify the clustering structure". In: ACM Press, pp. 49–60.

Ariyaluran Habeeb, R. A., F. Nasaruddin, A. Gani, M. A. Amanullah, I. Hashem, E. Ahmed, and M. Imran (2019). "Clustering-based real-time anomaly detection—a breakthrough in big data technologies". *Transactions on Emerging Telecommunications Technologies*, e3647. DOI: 10.1002/ett.3647.

Dilmegani, C. (2019). "Machine learning accuracy: true vs. false positive/negative". URL: https://research.aimultiple.com/machine-learning-accuracy/.

Foorthuis, R. (2021). "On the nature and types of anomalies: a review of deviations in data." *Int J Data Sci Anal* **12**, pp. 297–331. DOI: https://doi.org/10.1007/s41060-021-00265-1.

Jernbäcker, C. (2019). *Unsupervised real-time anomaly detection on streaming data for large-scale application deployments*. MA thesis. KTH Royal Institute of Technology.

Jollife, I. T. and J. Cadima (2016). "Principal component analysis: a review and recent developments". *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* **374** (2065). ISSN: 1364503X. DOI: 10.1098/RSTA.2015.0202. URL: https://royalsocietypublishing.org/doi/abs/10.1098/rsta.2015.0202.

Korstanje, J. (2021). "The f1 score". URL: https://towardsdatascience.com/the-f1-score-bec2bbc38aa6.

Kumar, V. C. V. M. V. (2008). “Comparative evaluation of anomaly detection techniques for sequence data”. *Eighth IEEE International Conference on Data Mining*.

Kuo, C. (2019). “Anomaly detection with autoencoders made easy”. URL: `https://towardsdatascience.com/anomaly-detection-with-autoencoder-b4cdce4866a6`.

McInnes, L., J. Healy, and S. Astels (2017). “Hdbscan: hierarchical density based clustering”. *The Journal of Open Source Software* **2**:11. DOI: `10.21105/joss.00205`. URL: `https://doi.org/10.21105%2Fjoss.00205`.

Mishra, A. (2018). “Metrics to evaluate your machine learning algorithm”. URL: `https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234`.

Sander, J. (2010). “Density-based clustering”. In: Sammut, C. et al. (Eds.). *Encyclopedia of Machine Learning*. Springer US, Boston, MA, pp. 270–273. ISBN: 978-0-387-30164-8. DOI: `10.1007/978-0-387-30164-8_211`. URL: `https://doi.org/10.1007/978-0-387-30164-8_211`.

Scikit-learn (2022a). “Importance of feature scaling” (). URL: `https://scikit-learn.org/stable/auto_examples/preprocessing/plot_scaling_importance.html` (visited on 2022-03-07).

Scikit-learn (2022b). *sklearn.metrics.precision$_s$core*. Scikit-learn. URL: `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html` (visited on 2022-04-07).

Scikit-learn (2022c). *sklearn.metrics.recall$_s$core*. Scikit-learn. URL: `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html` (visited on 2022-04-07).

Scikit-learn (2022d). *sklearn.preprocessing.MinMaxScaler*. Scikit-learn. URL: `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html` (visited on 2022-03-04).

Scikit-learn (2022e). *sklearn.preprocessing.Normalizer*. Scikit-learn. URL: `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Normalizer.html#sklearn.preprocessing.Normalizer` (visited on 2022-03-07).

Scikit-learn (2022f). *sklearn.preprocessing.RobustScaler*. Scikit-learn. URL: `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html` (visited on 2022-03-04).

Scikit-learn (2022g). *sklearn.preprocessing.StandardScaler*. Scikit-learn. URL: `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html` (visited on 2022-03-04).

Xu, M. E. H.-P. K. J. S. X. (1996). “A density-based algorithm for discovering clusters in large spatial databases with noise”. URL: `https://www.aaai.org/Papers/KDD/1996/KDD96-037.pdf`.

Yildirim, S. (2022). “Data preprocessing with scikit-learn: standardization and scaling”. *Towards Data Science* (). URL: `https://towardsdatascience.com/data-preprocessing-with-scikit-learn-standardization-and-scaling-cfb695280412` (visited on 2022-03-07).

Lund University **Department of Automatic Control** **Box 118** **SE-221 00 Lund Sweden**	*Document name* MASTER'S THESIS *Date of issue* May 2022 *Document Number* TFRT-6184
Author(s) Felicia Segui Andreas Timürtas	*Supervisor* Hans Ekström, Gaia System Aktiebolag Sara Olsson, Gaia System Aktiebolag Johan Eker, Dept. of Automatic Control, Lund University, Sweden Anton Cervin, Dept. of Automatic Control, Lund University, Sweden (examiner)

Title and subtitle

Real-time unsupervised log event anomaly detection in public transportation

Abstract

Detecting log data anomalies in real-time is useful since it makes it possible to apply logic that corrects the anomalies when they happen. This project presents a method for detecting public transportation bus event log data anomalies in realtime, without having a labeled data set. Initially, each unique bus trip is represented by the event frequencies, a representation that is not suitable for real-time. With a data set assumed to only contain normal data, an autoencoder, a PCA model and a clustering algorithm label each data point in the frequency domain, as normal or anomalous. The labeled data is split into sequences of events with a rolling window, a representation that is suitable for detecting anomalies in real-time. To separate the anomalous event sequences from the normal event sequences that occur, during the same bus trip as an anomalous event sequence, the event sequences together with their labels are grouped and counted. By comparing the frequency for each event sequence in anomalous trips with the frequency of the corresponding event sequence in normal trips, the sequences that are overrepresented in anomalous trips are detected and receive a final label being normal or anomalous. These labeled sequences are further used in the real-time detector. With the three base labeling models (autoencoder, PCA and clustering algorithm), different combinations of models are created. These models are either created by applying the union or the intersection of all anomalous labeled journeys. This results in 11 different models that are all tested and evaluated. The evaluation is performed by calculating the recall, precision and F1-score of experiments performed with a data set of assumed normal journeys, together with injected simulated anomalies. The evaluation is performed at two places within the method; one after the initial labeling and another after the real-time detector. The results obtained using this evaluation method show that the combination using the autoencoder and the clustering algorithm together through intersection is the best model combination, based on the F1-score calculated after the real-time detection. This combination scores a median recall and precision of 0.89 respectively 0.72, which results in an F1-score of 0.79.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title 0280-5316		*ISBN*
Language English	*Number of pages* 1-77	*Recipient's notes*
Security classification		

http://www.control.lth.se/publications/