

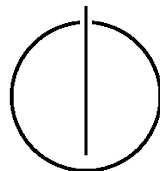
FAKULTÄT FÜR INFORMATIK

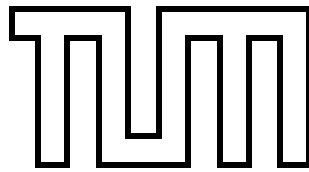
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis in Information Systems

**Applying Machine Learning Algorithms to
Support Root Cause Analysis –An
Experimental Study in Automotive
Engineering**

Duc Tien, Vu





FAKULTÄT FÜR INFORMATIK

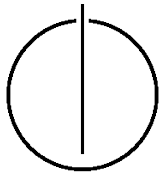
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis in Information Systems

**Applying Machine Learning Algorithms to Support
Root Cause Analysis –An Experimental Study in
Automotive Engineering**

**Anwendung von Maschine Learning Algorithmen, um
die Fehlerursachenanalyse zu Unterstützen –Eine
Experimentelle Untersuchung in der
Automobiltechnik**

Author: Duc Tien, Vu
Supervisor: Prof. Dr. Florian Matthes
Advisor: M.Sc. Martin Kleehaus
Dr. Valentin Solotych
Date: September 15, 2017



I assure the single handed composition of this master thesis only supported by declared resources.

Ich versichere, dass ich diese Masterarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 15. Oktober 2017

Duc Tien, Vu

Acknowledgments

First and foremost, I sincerely thank my advisors Dr. Valentin Solotych and Martin Kleehaus for their supervision of this thesis as well as the great discussions we had during the last six months. I am grateful to Prof. Florian Matthes for providing me the opportunity to write this thesis at the chair Software Engineering for Business Information System (SEBIS).

In addition to my advisors and supervisor, I want to give a special thanks to Dominik Guetermann for his ideas and supports that allowed the thesis to be started at all.

I am further grateful to my colleagues at IAV for providing such a wonderful working environment and for their supports: Andreas Merkl, Davide Passarelli, Tien-Dat Ho-Truong, Viet-Tiep Do, Michael Stappert and many more that I can't name all here.

Finally, I would like to thank my family and my girlfriend Hyunmin for being there supporting me whenever I needed. They encouraged and helped me to overcome my worst moments and helped indirectly the completion of this thesis.

Abstract

A shorter time to market is a clear business advantage for any company. Car manufacturers also thrive to bring their new cars to the market at their planned time. However, emission tests are heavy blockades for any new car. If a new car line cannot pass an emission test, the manufacturer cannot sell it. Finding the reasons why a car fails an emission test is hence an important task. Until now, to analyze the problem, the engineers must look at all signals produced from sensors built into the car. That process is thus very time intensive. In this thesis, we proposed a technique that ranks these signals based on their relevance to the emission outcome. The technique uses machine learning to automate the ranking process, thus helping the engineer to decrease the analysis time.

The thesis is divided in three main parts. In the first part, we conducted a literature review to assess the existing anomaly detection techniques. After that, three techniques are chosen to be implemented and benchmarked. We found that the IsolationForest technique performs best for automobile data. In the second part, we analyzed the problem proposed above and introduced a technique to solve it, using the anomaly detection techniques that we assessed in the previous part. In the last part, the proposed solution is implemented in form of a recommender system and gets evaluated using real data from emission tests.

Keywords: Anomaly Detection, Outlier Detection, Machine Learning, Automobile, Emission Test

Contents

Acknowledgements	vii
Abstract	ix
List of Tables	xiii
List of Figures	xv
Acronyms	xvii
1. Introduction	1
1.1. Motivation	1
1.2. Related Work	1
1.3. Research question	2
1.4. Outline	3
2. Foundations	5
2.1. Terminology	5
2.2. Type of anomalies	5
2.2.1. Point anomalies	6
2.2.2. Contextual anomalies	6
2.2.3. Collective anomalies	7
2.3. Introduction to Machine Learning	8
2.3.1. Machine learning tasks	8
2.3.2. Data sampling	9
2.3.3. Performance evaluation	9
3. Assessment of Anomaly Detection Techniques	15
3.1. A survey of the anomaly detection techniques	15
3.1.1. Probabilistic-based	15
3.1.2. Distance-based	17
3.1.3. Clustering-based	18
3.1.4. Reconstruction-based	19
3.1.5. Domain-based	22
3.1.6. Isolation-based	23
3.1.7. Information-theory-based	24
3.2. Comparison of Anomaly Detection Techniques	25
3.2.1. The techniques	25
3.2.2. The benchmark results	27

4. Experimental Environment	31
4.1. Problem Setting	31
4.1.1. Review of the task	31
4.1.2. The Dataset	31
4.1.3. Definition of our training and test data set	35
4.2. Solution proposal	36
4.2.1. Ranking based on anomaly percentage	36
4.2.2. Correlation between signal and emission	37
4.2.3. Combination of ranking systems	38
4.3. Architecture	39
4.4. Implementation	42
4.4.1. Data preparation	42
4.4.2. Building the ranking system	50
4.4.3. Visualizing the results	50
5. Evaluation	53
6. Conclusion and Future Work	57
Appendix	61
A. Raw data	61
Bibliography	63

List of Tables

4.1. Limits for passenger cars from regulatory frame EU5 (NEDC driving cycle)	35
---	----

List of Figures

2.1. A simple example of anomalies in a 2-dimensional data set [18]	6
2.2. A time series with contextual anomalies	7
2.3. Confusion matrix from a classification problem with two classes	10
3.1. Boxplot for a univariate data set [18]	16
3.2. Anomaly detection with linear regression	20
3.3. A neural network with two layers [21]	21
3.4. Contour graph created by Schölkopf's algorithm (OneClassSVM). The color bar represents the anomaly index of the contour graph. For anomaly index ≥ 0 , the region represents normal data, and for anomaly index < 0 , the region represents anomalous data. The lower the index value, the more likely the region contains anomalous data.	23
3.5. Example of random isolation from [55]	24
3.6. Speed benchmark of the techniques	28
3.7. Result of the accuracy benchmark	29
3.8. Accuracy test on a typical signal	30
4.1. Vehicle speed from a WLTC driving cycle	32
4.2. Vehicle speed from a NEDC driving cycle	33
4.3. General block structure of an MDF file [71]	34
4.4. Plot of emission change vs. signal change	37
4.5. Example for combining of ranking systems	38
4.6. Technology stack	39
4.7. Database schema	41
4.8. Data pipeline	42
4.9. Vehicle speed with error during test	45
4.10. A signal with trivial error	47
4.11. An example of mapping two time series using DTW, taken from [62]	49
4.12. Application interface	51
5.1. Emission outcome and searched signal from a problem set	54
A.1. Example of raw data from two signals with different resolution	61

Acronyms

AUC Area Under the Curve.

FNR False Negative Rate.

FPR False Positive Rate.

IQR Interquartile Range.

MDF Measured Data Format.

ML Machine Learning.

ROC Receiver Operating Characteristic.

TNR True Negative Rate.

TPR True Positive Rate.

TUM Technische Universität München.

1. Introduction

1.1. Motivation

Before going to mass production, a new line of car needs to be tested to assure it meets the required emission standards. A car consists of thousands critical electrical and mechanical components. When the result of the emission test is higher than the allowable norm, the car manufacturer must identify the reason for the failed test. Most of the time it is a wrong setting or a defective component that can breakdown a well-designed system.

The car houses a variety of sensors. The output from the sensor is called signal. Until now, the process of searching for the reason of a failed emission test, is done manually. The engineer uses his knowledge about the characteristic of each signal to diagnose them. That means he or she must look at each individual signal to find anomalies, which are related to the emission data.

However, there are thousands of signals in each test. Some tests have up to 5000 signals. Most of these signals show normal behaviour, which means that engineers spend much of their time analyzing normal signals. Moreover, the signals are from very different components. This means an emission test must be passed through many departments for analysis, even though only one department has the right knowledge to detect the relevant signals. This is not only time intensive, but also poses a big data security problem.

Therefore this thesis focuses on improving the analysis process by finding existing research about anomaly detection techniques, especially algorithms that are used for time series analysis. The algorithms will be assessed in terms of efficiency and accuracy, and compared with each other to find the best algorithm for solving the stated problem. The algorithms are used to build a ranking system that lists signals which influence the emission values in descending order. This will accelerate the whole process of finding the root-cause for a failed emission test. In the final step, the correctness and usefulness of the system will be evaluated with multiple data sets and the input of experienced engineering knowledge.

1.2. Related Work

Many researches have been conducted on detecting anomalies in time series. In health-care domain, Chuah et. al. proposed techniques to detect anomalous patterns from a ECG - a record of the heart beat pulses [22]. For consumer domain, Sheng Zhang et. al. used information theory to detect shilling attacks [74]. Shilling attack is a type of recommender system attack, in which attackers give false rating to products, in orders to

influence future recommendations. The authors used as input for the system the time series of rating for an item. Barford et. al. presented a signal analysis of network traffic to detect anomalies, that represent a failure in the system or an attack [11].

However, those researches only focus on detecting anomalies in one time series. In this thesis, the focus is on finding the root-cause of an anomalous time series, from multiple time series that can also contain anomalies. For a similar purpose, Chao Liu et. al. [54] proposed the two methods "sequential state switching" (S^3) and "artificial anomaly association" (A^3) to implement root-cause analysis for time series anomalies. In their work, the connection between components that produce time series data, are known. Meanwhile, in our data set, the connection between the components are unknown. Other researches about root-cause analysis that we found also require a graph structure of the components to find the root-cause node [34][19][25].

An root cause analysis tool is implemented by Florian Stoffel et. al for monitoring computer network [68]. However, the tool does not fully automate the process of finding root cause, but is instead a visual system that highlights the anomalies for analytical purpose.

1.3. Research question

In order to achieve the objectives which were discussed above, the following research questions are set as a guideline of the thesis:

Q1 : What are the related types of anomalies?

For a better understanding of the given problem, we first need to find the characteristics of anomalies. The type of anomaly plays a critical role in choosing the right algorithm for each data set. Having a basic knowledge is hence crucial for any research about anomaly detection in time series.

Q2 : Which algorithms can be used to detect anomalies in time series?

After the fundamental elements of the problem are explored, we conduct a literature review about anomaly detection techniques. Several sub-questions are posed in order to evaluate the algorithms further: Which techniques exist to detect anomalies? How are they compared to each other in terms of speed or accuracy? What are the advantages and disadvantages of each technique? Which technique is implemented in practice and ready to use in a production system?

Q3 : In which way can the root-cause analysis be supported?

In the final step, a solution is proposed using the knowledge gained from research question one and two. It consists of two parts: First, we need to find at least one feasible algorithm to apply on the data provided by a failed emission test. Second, we will

provide a way of visualizing the results so that the engineer can quickly use it for his analysis.

1.4. Outline

The remainder of this thesis is organized as follows:

In chapter 2, we will introduce the fundamentals of time series, anomaly types and machine learning. In chapter 3, we explore various existing anomaly detection concepts also called classes. Each class comprises of multiple anomaly detection techniques which are build on the same concept. Furthermore, we will briefly describe the idea behind each class and assess their strengths and weaknesses. A few techniques are then selected from multiple classes and benchmarked against each others, in order to find the most feasible one for our problem.

In chapter 4, a system that can help engineers, will be proposed and developed. We will first describe the problem of finding the root-cause in more detail. The technical aspects of the data sets and the requirements can be found here. Second, different machine learning steps for the solution will be described in this chapter. At the end of the chapter, we will show how the system is implemented.

The proposed algorithms are evaluated and the results are discussed in chapter 5. Finally, in chapter 6, we give concluding remarks and an outlook for future works.

2. Foundations

In this chapter, we present the knowledge base for this thesis. Starting with the basic definitions, we then describe the three types of anomalies in time series. We finish this chapter by giving a short introduction to machine learning.

2.1. Terminology

A time series is a collection of data points, where each data point consists of one or more values and a time stamp. It is common that the duration between two neighboring time stamps is constant. Time series is widely used in many domains. For example, in economics and finance, the financial indices, exchange rates and gross domestic product (GDP) are measured in the form of time series. In medicine, the heart rate and the temperature of patients are also shown in time series. In automobile industry, time series are often observed in outputs from sensors, such as vehicle speed, gas consumption and the emission outcome.

Anomaly detection is the identification of data points that differ from the rest of the data even without knowing what is the expected behavior of the data. These deviated data points are also referred to as outliers, exceptions, discordant observations, peculiarities, aberrations, surprises or contaminations in different domains [18]. In machine learning, the process of anomaly detection is similar to that of data classification. In data classification, the main goal is to label the data instances with multiple tags. Similarly, the task of anomaly detection is to label data with two tags, anomaly or normal.

Machine learning (ML) is a research field that involves building a model from data. The main goal of machine learning is to apply a model to solve problems related to new data. The process of building the model is called learning- or training step. The process of using the model to solve problems is named testing step. The data, which are used in training, are labeled as a training set while the new data is called test set. In section 2.3 different types of data sets will be discussed.

2.2. Type of anomalies

The type of anomaly is an important aspect when choosing the right detection techniques. The three categories of anomalies according to Chandola et al [18] are as followed.

2.2.1. Point anomalies

The simplest type of an anomaly is the point anomaly. For a group of data points G with high density, we define a normal area NA around G . When an individual data point P lies remarkably outside NA , P is a point anomaly with respect to G . Most of the anomaly detection techniques focus on this type of anomaly. In Figure 2.1, the data points are concentrated in two normal regions N_1 and N_2 . Two data points o_1 and o_2 are point anomalies with respect of N_1 and N_2 . Data points in O_3 form a collective anomaly, which will be described in section 2.2.3.

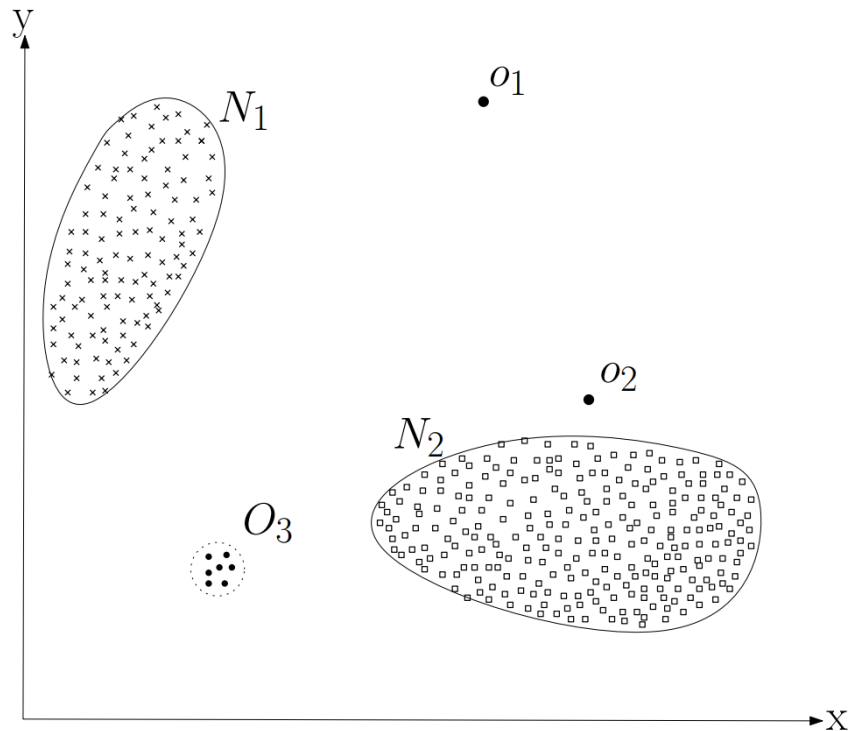


Figure 2.1.: A simple example of anomalies in a 2-dimensional data set [18]

2.2.2. Contextual anomalies

Contextual anomalies or conditional anomalies are data points, which are outstanding in a given context but are normal in a different one [67]. The context could be their surrounding data points or the time when they appear. These anomalies can only be detected, by having extended knowledge about the data-set as a whole. Each data instance must contain two attributes: the contextual and the behavioral attribute. Contextual attributes characterize the context of the instance. In a time series, the contextual attribute is the time stamp of the instance. If the time series contains periodicity, the contextual attribute is the position of the instance in each period. Behavioral attributes are the remaining non-contextual attributes. They are measured values of each point of the contextual attributes.

Figure 2.2 illustrates a time series with contextual anomalies. The marked data point is anomalous in context with its neighbors, while it is normal compared to data points in the first 400 seconds.

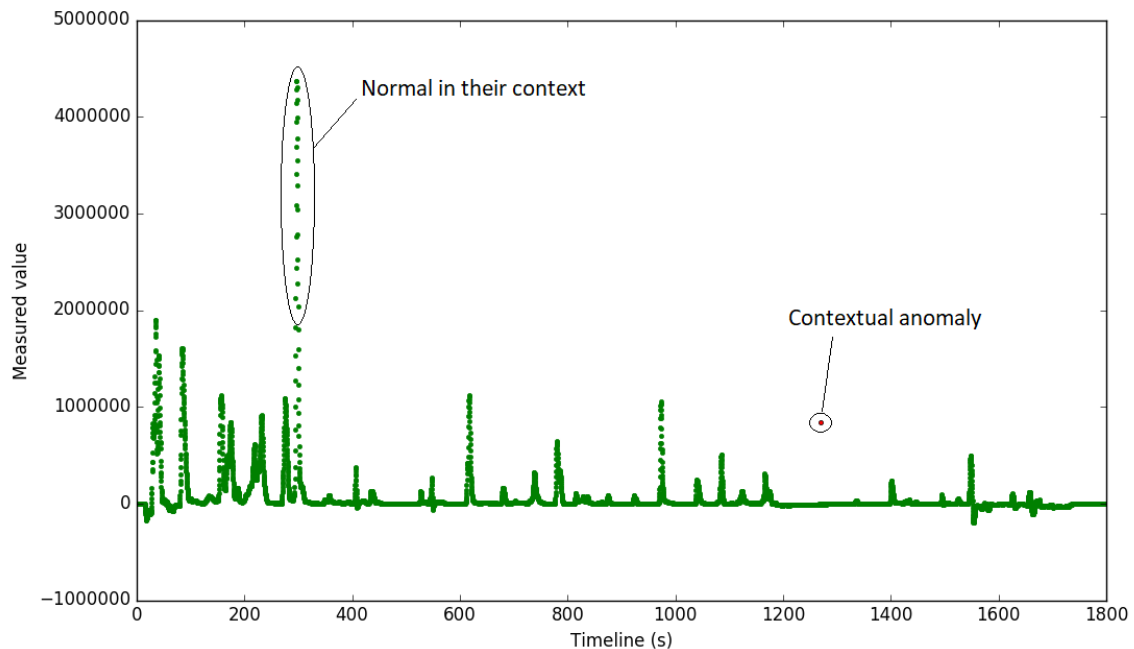


Figure 2.2.: A time series with contextual anomalies

2.2.3. Collective anomalies

We consider a group of data points G with all of the instances close to each other. G is called a collective anomaly if all the data points in G as a whole form an anomalous collection compared to another data point groups. The individual data points in G may not be anomalies alone, but all of them together form an anomaly group. In Figure 2.1, the data points in region O_3 are a collective anomaly.

Following is an example taken from the intrusive detection domain, to illustrate how an attack on a server can be exposed using collective anomaly detection [18].

... http-web, buffer-overflow, http-web, http-web, smtp-mail, ftp, http-web, ssh, smtp-mail, **http-web, ssh, buffer-overflow, ftp**, http-web, ftp, smtp-mail, http-web ...

Each individual action is not an anomaly. However, the highlighted sequence of events (buffer-overflow, ssh, ftp) forms a collective anomaly. This anomaly is from a typical web based attack. It starts with a request for a connection followed by copying data from the host computer to a remote destination via ftp.

2.3. Introduction to Machine Learning

In the following subsection, a brief introduction to machine learning is presented. For the used terminology in this chapter, the reader is referred to section 2.1.

2.3.1. Machine learning tasks

Machine learning techniques have been applied in a wide range of domains, such as natural language processing and computer vision. Some focus only on one domain, others on multiple domains (so called general-purpose techniques). Whether a technique is domain-focused or general-purpose, its main task can still be sorted into a certain category. Some of the most common categories are introduced below:

Regression

Given a set of data, with each data instance having an input vector of the form (x_1, x_2, \dots, x_n) and an output value y , regression tasks mainly deal with estimating the parameters of a function f so that $f(x_1, x_2, \dots, x_n)$ returns a similar result as y . In other words, the main goal is to minimize the error $|e = f(x_1, x_2, \dots, x_n) - y|$. Regression is widely used in the financial domain. For example, it is used to estimate the product price, the value of a stock and the housing price.

Classification

In this category, the task is to build a model that can label a data instance based on its attributes. The model is trained using a set of data instances with known labels. One of the most common use-cases is to use a classification model as a spam filter, i.e. marking an email as spam or not. Another example is the cancer prognosis in the health-care domain.

Clustering

Clustering is similar to classification since both the techniques' main task is to find the correct label of the data instances. However, the main difference is that clustering does not require the labels of the data to be known in advance. Furthermore, classification labels a data instance while clustering labels a group of instances that have similar properties. One example for clustering is targeted marketing, where the customer base is segmented to groups that bring the most or the least revenue.

Ranking

The main task of ranking is to build a model that correctly ranks a collection of items based on predefined criteria. It is also referred to as ordinal classification or ordinal regression, because the ranks have ordinal values. Search engines are the most common application of this category, whereas recommender systems also fall into this category and are gaining in popularity. In our prototype, the recommender system gives a ranking

of the signals based on their relevance to the emission.

Density estimation

Density estimation deals with finding the probability of the occurrence of an event. A rescaled histogram is the basic form of density estimation. Preventive maintenance is a popular example: the time of failure of a specific component is collected over a large number of samples. Based on that data, the life expectation of this component can be estimated. This way, the maintainer can decide when a component should be changed before it fails.

2.3.2. Data sampling

The core element of machine learning is the available data. We need to divide the data into different data sets for different purposes. The data set we acquire before a machine learning system goes into production phase, is divided into three sets: “training set”, “validation set” and “test set” [4].

Both “training set” and “validation set” are used for the training step of a classification algorithm. A training set is defined as the data set used to train the algorithm initially. In the training set, the outcome of each instance is known. Similar to training set, the outcome of a validation set is also known. We use the validation set to test the accuracy of the algorithm and tune the parameters of the algorithm. An important point to note is that the ML algorithm is only trained from the “training set” and not from the “validation set”. Validation set is used for testing and tuning only.

After a model is finalized from the training step, the accuracy of the algorithm is tested using the “test set”. The test set must not be used during the training step. After testing, the algorithm must not be modified until a new test step is created. Otherwise, the test set becomes a validation set and loses its purpose.

There is no rule for selecting the ratio of training/validation/testing set. The proportion generally used in practical is 50:25:25. However if there are only a few data instances, then 60:20:20 is used.

In this thesis, we refer to the number of data instances in the training set as ratio of training-test instance. For example, if a training set contains 100 data instances, the ratio is 100:1. In case the ratio is too small, the trained model is considered as insignificant.

2.3.3. Performance evaluation

After a ML model is trained, we need to evaluate how well that model will solve our problem. We differentiate between the performance metrics for a classification problem and metrics for a regression problem. In a classification task, the predicted result can be directly evaluated as correct or incorrect. Meanwhile, the predicted result in a regression task can only be evaluated based on correctness measures. [44][41] In the following we

will describe the metrics in details.

Metrics for classification tasks

In ML systems for classification problems, accuracy is the starting point for any evaluation [43]. Accuracy is defined as the percentage of correct outputs from all outputs of the system. Let us take as an example an anti-spam system, which uses machine learning to mark an email as spam or not. If out of 100 emails, the system marks 90 emails correctly, the accuracy is 90%.

Another instrument to evaluate the performance of a classification system is the confusion matrix (also known as error table, matching matrix or contingency table). It is a matrix with a specific table layout. Each column of the table represents the number of instances in an actual class (the actual result). The predicted results for each class is reproduced in the rows of the table. For example, we examine a classification system that has been trained to distinguish between normal and anomalous signals. Assuming a sample of 1000 signals – 900 normal and 100 anomalous, the resulting confusion matrix is presented in Figure 2.3.

		Actual class	
		Normal	Anomalous
Predicted class	Normal	815	60
	Anomalous	85	40
Sum		900	100

Figure 2.3.: Confusion matrix from a classification problem with two classes

Examining the performance of a classification system usually focuses on one class at a time. The focused class is chosen based on the purpose of the system. For instance, if the classification system mentioned above, is built to correctly classify as many anomalous signals as possible, the focus is on the class of anomalous signals. On the other hand, if the system is built to correctly classify as many normal signals as possible, the focus is on the class of normal signals. The following measures are defined to evaluate the system for a certain class:

- Condition positive (P) is the actual number of instances in the focused class.
- Condition negative (N) is the actual number of instances from all other classes.
- True positive (TP or hit) is defined as the number of objects that are correctly assigned to that class by the system.

- True negative (TN or correct rejection) is the number of objects that are correctly assigned to other classes.
- False positive (FP or false alarm, type I error) is the number of objects that are predicted to be in the chosen class but in fact are not.
- False negative (FN or miss, type II error) is the number of objects that are predicted not to be in the chosen class but in fact are.
- True positive rate (TPR – also called sensitivity, hit rate, recall) is the percentage of TP from P. The higher the TPR is, the better the system predicts the focused class.

$$TPR = \frac{TP}{P} \quad (2.1)$$

- True negative rate (TNR – also called specificity) is calculated as the percentage of TN from N. The higher TNR is, the better the system predicts other classes.

$$TNR = \frac{TN}{N} \quad (2.2)$$

- False positive rate (FPR – also called fall-out or probability of false alarm) shows the percentage of false alarm (FP) from the total number of negatives.

$$FPR = \frac{FP}{N} = 1 - TNR \quad (2.3)$$

- False negative rate (FNR – also called miss rate) is the percentage of miss(FN) from the total number of positives (P).

$$FNR = \frac{FN}{P} = 1 - TPR \quad (2.4)$$

Depending on the class that is in focus, the performance of the classification system can differ considerably. In the example shown in Figure 2.3, out of the 900 normal signals, the system predicted 85 signals to be in the wrong category, meaning they are anomalous signals, and out of the 100 anomalous signals, it predicted 40 correctly and 60 incorrectly. If we choose to evaluate the system focusing on the anomalous signals, the TPR is 40% while the TNR is 90.5%. With TPR of 40%, the system performs poorly. On the other hand, if we choose our focus on the normal signals, the TPR is 90.5% and TNR is 40%. In this case, the system performs acceptably good.

Another technique, the Receiver Operating Characteristic Curve (ROC curve) is also used for evaluating the performance [14]. A ROC curve is a two dimensional graph. It plots the FPR in the x-axis against the TPR in the y-axis. The FPRs and TPRs are retrieved from the same classification algorithm with different parameters (cutting points). The result is a curve from point (0,0) of the graph to point (1,1). Generally, a classification algorithm with greater area under the curve (AUC) is better than one with less AUC on the same test data.

Metrics for regression task

In regression task, the predicted outcomes are numbers and may vary in an acceptable zone around the real values. Hence, it is more important to evaluate how close the predicted values are to the actual values than merely assessing the equality between predicted and real values as in the case of classification task. Some of the mainly used metrics are:

Mean absolute error (MAE) is the most used metric in statistic for its simplicity. The MAE is the sum of the absolute values of the differences between the predicted values and real values. The value of MAE can range from 0 to ∞ . The lower MAE is, the better the system performs.

$$MAE = \frac{1}{n} \sum_{i=1}^n |\hat{Y}_i - Y_i| \quad (2.5)$$

where n is the number of predictions made
 \hat{Y}_i is the predictive value of i^{th} instance
 Y_i is the real value of i^{th} instance

Another metric is the *mean squared error (MSE)* – also called mean squared deviation (MSD). It evaluates the squared difference between real values and predicted values (the errors). Compares to MAE, the MSE gives a higher weight to cases where the error is large. MSE is useful when a large error is undesirable.

$$MSE = \frac{1}{n} \sum_{i=1}^n (\hat{Y}_i - Y_i)^2 \quad (2.6)$$

Root-mean-square error (RMSE) or root-mean-square deviation is the square root of MSE. Since it has the same unit as the real value, it is easier to evaluate on a graph than MSE.

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{n} \sum_{i=1}^n (\hat{Y}_i - Y_i)^2} \quad (2.7)$$

Normalized root-mean-square error (NRMSE) is used to compare the results between different data sets. The normalization can be made either through the range of the observed data or through their average value.

$$NRMSD = \frac{RMSD}{Y_{\max} - Y_{\min}} \quad (2.8)$$

or

$$NRMSD = RMSD \times \frac{\sum_{i=1}^n Y_i}{n} \quad (2.9)$$

All of the metrics above disregard the direction of the error (predicting over or under the real value). *Mean signed difference (MSD)* emphasizes the direction by taking into account the error without absolute or squared value.

$$MSD = \frac{1}{n} \sum_{i=1}^n (\hat{Y}_i - Y_i) \quad (2.10)$$

3. Assessment of Anomaly Detection Techniques

In section 3.1, the results of the literature research about anomaly detection techniques are shown. In section 3.2, we describe the result of our benchmark on some of the techniques.

3.1. A survey of the anomaly detection techniques

Many techniques for detecting anomalies were proposed in the past [18][57][20]. We classified them into seven categories: probabilistic-based, distance-based, clustering-based, reconstruction-based, domain-based, isolation-based and information-theory-based. The principle of each category will be described in the following subsections.

3.1.1. Probabilistic-based

This class of techniques is also known as statistical anomaly detection techniques or statistical approach [18][57]. The techniques are based on the underlying principle:

“An anomaly is an observation which is suspected of being partially or wholly irrelevant because it is not generated by the stochastic model assumed” [8]

In other words, the methods are based on the stochastic model of all data from the training sample. We assume that in the model, normal data points occur in regions that have high density while anomalous data points occur in the regions with low density. By applying a statistical inference test to the test data, anomalies can be detected.

The stochastic model can be built using two techniques, parametric or non-parametric.

Parametric techniques assume the knowledge of the underlying distribution from the training data. A probability density function or a statistical hypothesis test, used with the underlying distribution can declare a data instance to be anomalous or normal.

The simplest technique, is to use the distribution mean μ and standard deviation δ of the training data. A threshold value α is defined, so that most of the training data lie inside the range $[\mu - \alpha * \delta, \mu + \alpha * \delta]$. The percentage of training data not lying inside that range is the measurement of noise inside the training data. A data point that lies outside of the defined range will be declared as anomalous.

Another technique is to use the interquartile range (IQR). A system can simulate the use

of IQR in a box-plot to determine anomalies. A box-plot is a graphical way to describe a data set using statistical attributes such as minimum, maximum value, first quartile (Q_1), median, third quartile (Q_3). The interquartile range is defined as $Q_3 - Q_1$. Using IQR and a threshold value α , a boundary between $Q_1 - \alpha * IQR$ and $Q_3 + \alpha * IQR$ is created to detect anomalies. Any data point that lies outside of that boundary will be treated as an anomaly. Figure 3.1 depicts how a box-plot can be used to detect anomaly in a univariate data set.

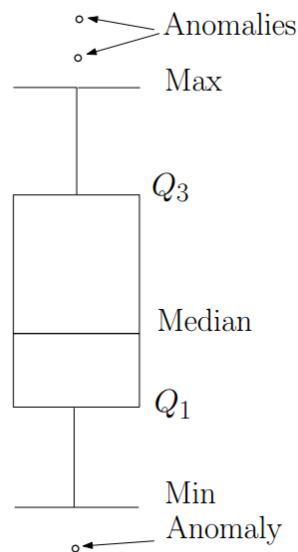


Figure 3.1.: Boxplot for a univariate data set [18]

The parametric techniques are simple but do not always work. They assume that the training data is normally distributed, or in other words, follows a Gaussian distribution. Hence, the training process is very susceptible to extreme data sets. For example, if the test data set has two groups of data instances with high density and the distance between the the two groups is large, parametric techniques will not detect anomalies lying in the space between the two groups.

Parametric techniques can be easily implemented as a rule-based function. However, because of the problem stated previously, they are only implemented in our experiment as a data cleaning operation.

Non-parametric techniques do not require any previous knowledge of the underlying distribution. The structure of the training as well as the test data is analyzed and tested instead of using a statistical measure.

Histogram-based technique is a non-parametric technique that is applied mainly in intrusion detection and fraud detection domain [24][27][29][31]. It consists of two steps. First, based on all the values of the training data, a histogram is built with a fixed bin size. Second, for every test data point, the technique checks if its value falls in any of the bins. The data point is classified as normal if it does and anomalous otherwise. For this technique, the bin size is critical for the accuracy. If it is too small, many normal test values will fall in the empty or rare bins, resulting in a high false positive rate. On the other hand, if the bin size is large, many anomalous test values will fall in frequent bins, resulting in false negatives. [57]

The non-parametric techniques also have the same drawback as the parametric one. Hence, they are excluded from the experiment.

3.1.2. Distance-based

Distance-based methods compute the similarity measure between new instances and training instances to determine anomalies. If the instances have numeric attributes - or in other words, if the attributes are measurable, the similarity measure is simply the distance between data points. For categorical attributes, the similarity measure can be calculated either using a simple matching rule (1 for a total match, 0 for different) or other more complex coefficients [13][17].

The techniques in this class are built upon the key assumption: if a data point is located far from its closest neighbors, it is anomalous. If a data point occurs close to its neighbors, it is normal. There are two approaches to use this assumption.

In the first approach, an anomaly score is defined as the shortest distance from one data point to its k^{th} neighbors [7][28][49]. Using the training data, a threshold is acquired as the maximum distance from any data point to its k^{th} neighbors. Every test data point is combined with the training data to calculate the anomaly score. If the anomaly score is greater than the threshold, the data point is classified as anomalous.

This approach has the following issue: it uses a single distance threshold to test for anomalies. The threshold is predefined and will not work on data sets with different characteristics. This can be improved by defining the threshold as the maximum distance between the k neighbors of the test data point. Nevertheless, this improvement takes a trade-off between accuracy and processing speed as the threshold needs to be determined for every test data point.

The second approach is initialized by predefining a maximum distance to the neighbors that a normal data point can have [47][48][50]. The anomaly score for each test data point is then defined as the number of its neighbors within the predefined distance. Finding of a suitable distance however poses a key challenge in this subclass of algorithm.

Different distances can be used for distance-based techniques. They are chosen

depending on the characteristic of the data distribution. For instance, if the distribution has the elliptical form, it is better to choose the mahalanobis distance. The most typical distances are Euclidean distance and mahalanobis distance [56]. A list of the distances can be obtained from [69], in the scope of this thesis, no further discussion on this topic is provided.

Several authors applied distance-based approaches for anomaly detections. For example, Byers and Raftery built a system to detect land mines from aerial images [15]. In another example, Guttormsson et al. implemented a prototype to detect shorted turns in the rotors of turbine-generators [36]. However, the systems/prototypes they implemented are closed system. So far, we couldn't find any open-sourced implementation, that we can directly use in our system. Since the idea of this technique class looks very promising, we implemented our own prototype for the technique using the improved first approach (kNeighbors). The implementation will be discussed in section 3.2.

3.1.3. Clustering-based

This class of techniques uses clustering techniques as the first step of their algorithms and then processes the data based on the assigned cluster to each data instance.

Cluster techniques (clustering algorithms, clustering analysis occasionally also known as ball analysis) are methods for the discovery of similarity structures in (large) data sets. The groups of "similar" objects found with this technique are referred to as clusters, the group assignment as clustering. The reader can refer to [42][37][72] for more detail about techniques for clustering a data set.

The anomaly detection techniques in this class can be divided in three sub-classes based on their definitions of anomaly [18]:

First sub-class assumes that anomalous data instances do not belong to any of the clusters. It requires the clustering algorithm to assign clusters without forcing all instances to go into at least one cluster. Several clustering algorithms can do that, such as DBSCAN [30], WaveCluster [65], ROCK [35], FindOut [73] or SNN clustering [26]. The problem with this sub-class is that the clustering algorithms are not specialized in finding anomalies but clusters. Hence, the results depend heavily on the training data. If the training data set contains noise or anomalies, the clustering is adapted to them. Therefore, similar anomalies will not be recognized in such cases.

Second sub-class defines anomalies as data instances that lie further from the cluster centroid than normal data instances [66][70][10]. The accuracy of these techniques is based heavily on finding the right distance from the cluster centroid. However, the distance is very specific to every single data set. Hence, it is impossible to systematically determine the distance without having overfitting in the training algorithm. Furthermore, if the anomalous data points are from a small cluster by themselves, the techniques will not be able to detect them.

The **third sub-class** uses the exposed problem of previous sub-classes as an idea to detect anomalies. It defines anomalies as instances that belong to smaller and sparser clusters than normal data instances [28][60][16]. To differentiate between normal clusters and cluster with anomalies, the size and/or density of the clusters is calculated. If a cluster is extremely small in size or has a high density compared to other clusters, it is marked as an anomalous cluster. Using that idea, He et al. [38] proposed FindCBLOF (Find Cluster-Based Outlier Factor) where each data instance is assigned a CBLOF score. The CBLOF score is derived from a function of the size of the cluster that the data point belongs to and the distance of that point to the corresponding cluster centroid.

Most of the clustering-based techniques use some sort of distance measure, like in the case of distance-based techniques. However, the two classes are still different. The key difference between them is that clustering-based techniques measure the distance of each instance to the cluster that it is assigned to, while distance-based techniques measure the distance to its closest neighbors.

3.1.4. Reconstruction-based

The key concept of reconstruction-based techniques is to build a model of the underlying training data. The model is a function that reassembles all data points in a training set. In a time series, the time stamp is one input of the function. A predictive value for each time stamp of the data set is calculated using the trained model. The difference between the predictive value and actual value - also called reconstruction error - are used as novelty score for each data point. The bigger the difference is, the more likely a data point at that time stamp is an anomaly.

The prediction model can be built in two ways: regression or neural network.

In *regression analysis*, the form of the modelling function is defined prior to the training step. The main task is to adjust the parameters of the function so that the curve will fit the training data. The techniques used to achieve this, involve minimizing the reconstruction error of all data points in the training set (ordinary least squares).

The simplest regression technique is linear regression [51], where the function has the form:

$$Y = f(X) = \beta X + \alpha \quad (3.1)$$

where X is the input vector and Y is the output vector. Using values of X and Y taken from training data, the regression function can be easily solved as

$$\beta = \frac{\sum_{i=1}^n (X_i - \bar{X})(Y_i - \bar{Y})}{\sum_{i=1}^n (X_i - \bar{X})^2} \quad (3.2)$$

$$\alpha = \bar{Y} - \beta \bar{X} \quad (3.3)$$

with \bar{X}, \bar{Y} the mean value of input and output vector respectively
 n the number of instances in training data

X_i, Y_i the input and output vector for each data instance.

Figure 3.2 shows an example of using linear regression for anomaly detection. First, the algorithm reconstructs a line through training data. After that, a threshold of allowable reconstruction error is built around the line. All data points lying outside of the allowable region are marked as anomalies.

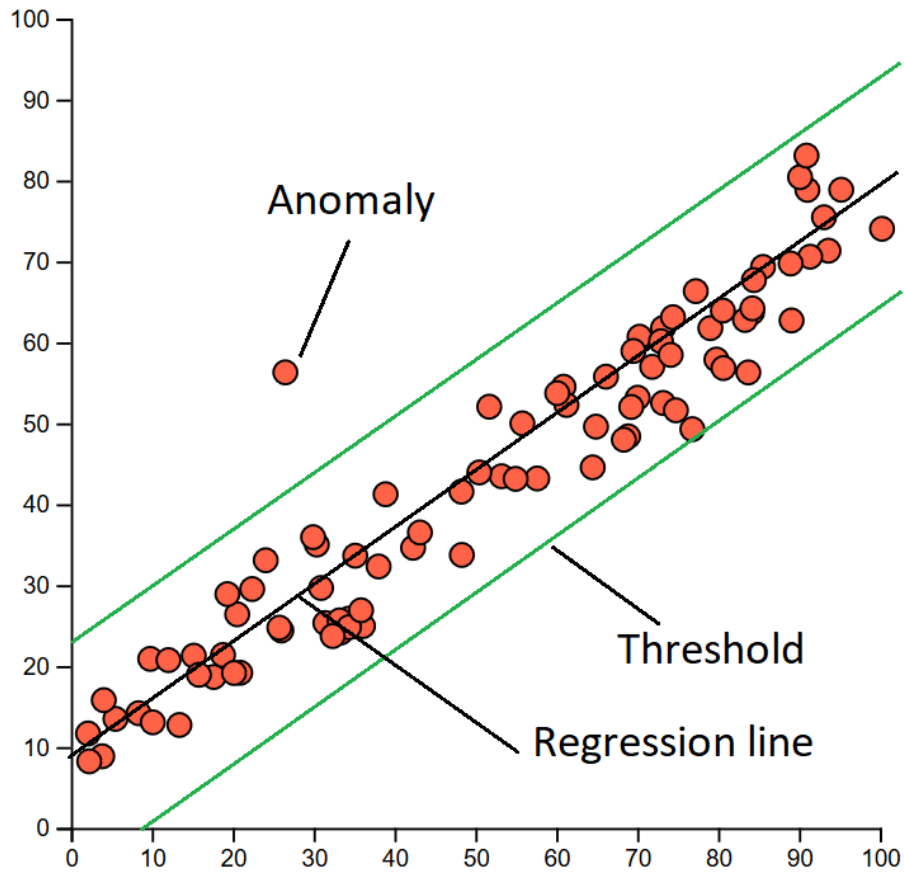


Figure 3.2.: Anomaly detection with linear regression

Other types of functions were also used as the modelling function. David Cox introduced the logistic regression to fit the model to data sets where the outcome values only take two values, "0" and "1" [23]. Gergonne proposed the Polynomial Regression for fitting a n^{th} degree polynomial through the training data [33]. If the modelling function is a nonlinear combination of the input parameters, a nonlinear regression can be used to train the model [59].

Neural network is a technique that mimics the structure of animal brains. It builds a collection of computational units (the neurons), which are interlinked by a system of

connections (the network) [21]. The neurons are divided in different layers. Each neuron in a layer takes as input the output from all neurons in the previous layer. A propagation function or activation function inside each neuron computes the output from the input. In the last layer, the propagation function is usually a linear combination of all input. Meanwhile, the propagation functions of neurons in other layers than the last layer are realized with sigmoid functions. Even when the functions used in the neurons are similar, the parameters from each function are still different. They are usually generated randomly during the initialization phase. After each iteration of feeding the network with training data, the parameters are tuned to adapt to the input values. The most used technique for parameter tuning is backpropagation. It propagates the input forward through the network and then uses the difference in output to propagate back through the network while updating the weight during the propagation [61]. Figure 3.3 illustrates a neuron network with two layers.

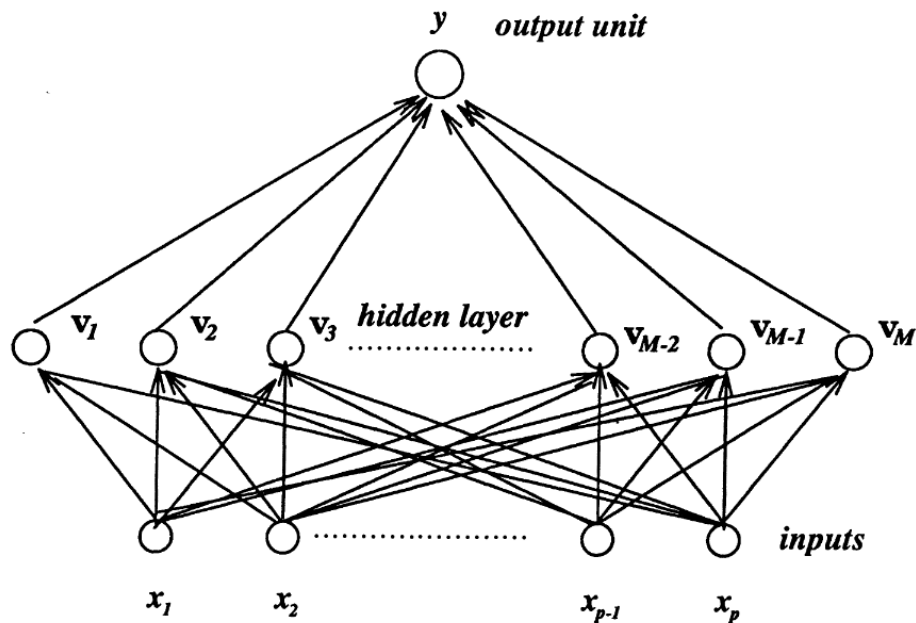


Figure 3.3.: A neural network with two layers [21]

Neural network has been a popular technique in recent years. The main reason for this is, that it does not require a lot of pre-analysis like regression, where the form of the function has to be defined first. On the other hand, neural network can approximate any function, regardless of its form. Hence, it can be used for any problem set.

However, there are some issues with neural network. Due to the high number of tuned parameters (hyper-parameters), neural network tends to over-fit the training data. Consequently, the technique will not work well if the training data is small. Further, the

training of the neurons is very expensive. Therefore, neural network is slower compared to other techniques. Another issue is that the trained model is a black box without any possibility to analyze what is inside.

3.1.5. Domain-based

In this class of technique, the task is to learn and build a boundary around the training data. Based on the boundary, one data point is then classified as normal if it lies inside the normal zone and as anomalous otherwise.

Support Vector Machine (SVM) is usually adapted to solve this task. SVM is a popular technique for separating classes using decision boundary. In SVM, each data instance with n attribute is represented in an n -dimensional space. One or more hyperplanes – a $(n-1)$ -dimensional subspace of the original space – is built to separate the data points. The hyperplane maximizes the separating margin between classes. This set of hyperplane constructs the required boundary for separating data classes. For each new data instance/data point, the dot product is used to evaluate its position to each hyperplane. Combining those evaluations, the position of that data instance/data point with respect to the boundary is calculated. Based on that, the algorithm can then suggest the class membership of an unknown data instance.

The traditional SVM technique requires at least two classes of training data. However, in the anomaly detection scenario, only positive-classified examples are known or the ratio between negative- and positive-classified examples are extremely unbalanced. Schölkopf et al. proposed the “one-class” classification to handle this problem [63].

The idea of Schölkopf’s algorithm is to form a region in the input space that contains most of the normal data points. The region is virtually built by a function f . The objective of f is to assess whether a data point belongs to that region or not. Function f is defined as

$$f(X) = \text{sgn}((\omega \cdot \Phi(X) - \rho)) \quad (3.4)$$

where $\Phi(x)$ is a function that transform the data instance into another feature space, sgn is the sign function. Variable ω and ρ are obtained from the following quadratic program:

$$\min_{\omega \in F, \xi \in \mathbb{R}^t, \rho \in \mathbb{R}} \frac{1}{2} \|\omega\|^2 + \frac{1}{v \cdot l} \sum_i \xi_i - \rho \quad (3.5)$$

$$\text{subject to} \quad (\omega \cdot \Phi(x_i)) \geq \rho - \xi_i, \quad \xi_i \geq 0 \quad (3.6)$$

$l \in \mathbb{N}$ is the number of observations. Indice i is understood to range over $1, \dots, l$. $v \in (0, 1)$ is a tuning parameter. ξ_i are nonzero slack variables that get penalized in the objective function. The minimum function assures that if there is a solution for ω and ρ , the decision function f will be positive for most of the data point x_i .

Figure 3.4 illustrates a contour graph that is built using Schölkopf’s algorithm. The

contour graph is constructed based on the training data. The green region with contour value greater than 0 are the normal classified region. All data points outside of that region will be classified as anomalous.

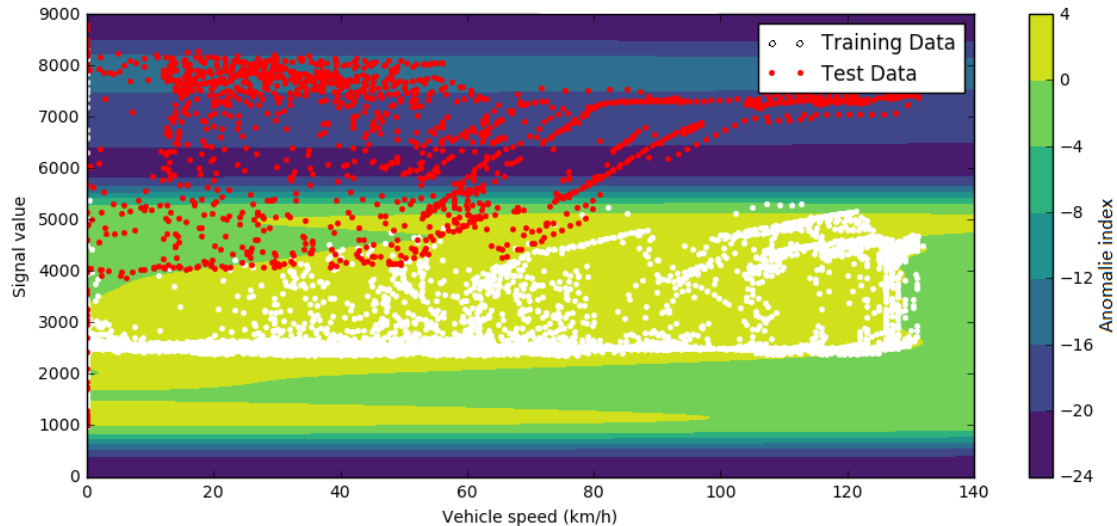


Figure 3.4.: Contour graph created by Schölkopf's algorithm (OneClassSVM). The color bar represents the anomaly index of the contour graph. For anomaly index ≥ 0 , the region represents normal data, and for anomaly index < 0 , the region represents anomalous data. The lower the index value, the more likely the region contains anomalous data.

3.1.6. Isolation-based

This class of technique is developed based on the assumption that anomalies are rare compared to normal instances. Because of that property, the anomalies are prone to the so-called isolation mechanism [55]. The isolation mechanism is the process of separating an instance from the rest of the instances. Generally, the algorithms in this class measure the sensitivity of each individual data point to isolation. Anomalies are data points with the highest sensitivity character.

Liu, Ting et al. proposed the Isolation Forest algorithm that measures how many random steps are required to isolate a data point from other points [5]. The authors proved in their paper that in general, anomalies require less steps to be isolated than normal data points. To count the steps required for each data point, a forest of random splits is built, giving the algorithm the name Isolation Forest. The forest is a collection of binary trees, which are generated using the training data. The root of each tree in the forest is the original data set. The algorithm will choose randomly in each tree a feature of the data set and split it at a random point, dividing the data set in two new data sets. The algorithm

will continue to split the data sets until only one single data point is left in each sub data set. By doing this, in each tree, there is one certain path from the root to one data point. That path represents the splits needed to isolate one data point from all others (also called isolation path).

After a forest of the binary tree is built, for each data point, the lengths of the isolation paths in all trees are calculated. The anomaly score of each data point is calculated by taking the average of the lengths of its corresponding paths. A threshold for the the anomaly score must be defined. Data points with lower score than that threshold are the anomalies.

As an example, Figure 3.5 illustrates the number of steps required to isolate one data point. In the left Figure (a), 11 random splits are required to isolate a normal point x_i . Meanwhile, the right Figure (b) shows that it only takes 4 splits to isolate an anomaly point x_o .

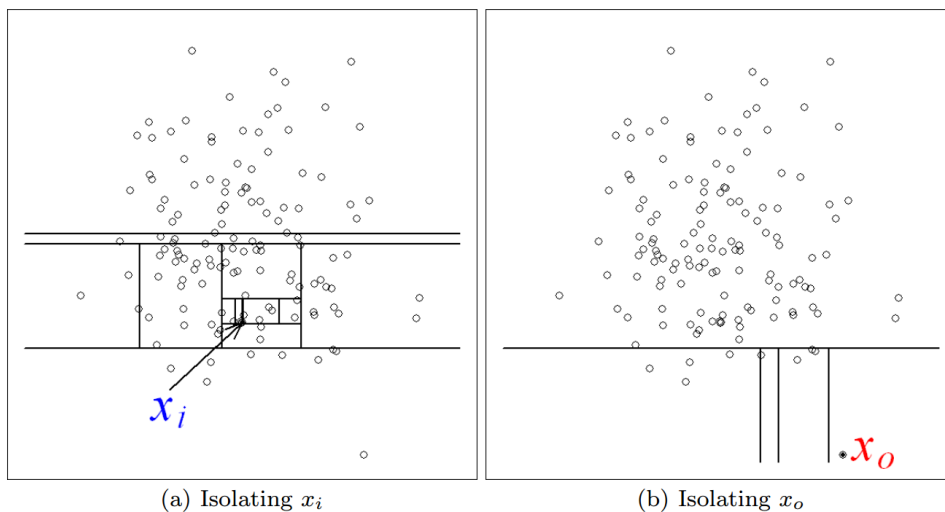


Figure 3.5.: Example of random isolation from [55]

The authors claimed that Isolation Forests could achieve a low linear time-complexity, a small memory-requirement, and deal with the effects of swamping (when the distance between anomalies and normal points reduces) and masking (when the data set has an increasing number of dense anomalies) effectively.

3.1.7. Information-theory-based

Information theory was originally proposed by Claude E. Shannon in 1948 to deal with problems in communication and signal processing techniques [64]. Since then it has been

adapted in many other fields like statistics, pattern recognitions, cryptography, anomaly detection etc. The information theory uses key measures like entropy, joint entropy, mutual information or Kolmogorov Complexity to describe the information content of a data set [45][58]. Introduction of an anomaly into an existing data set will change those measures while adding data instances that have similar property will maintain those quantities. Following that idea, one of the key measure is first calculated for the training data. After that, each single test data point is added to the training data in a consecutive order. The key measure is calculated again using this new data set. The difference between the key measure from the old and new data set is then used to classify the test data point as anomalous or not.

He et al. used entropy as a key measure to identify anomalies in categorical data [39][40]. Ando proposed a technique using mutual information to identify clusters of outliers [6]. Keogh et al. [46] and Lin et al. [53] applied the Kolmogorov Complexity in their approaches. Filippone and Sanguinetti used the Kullback-Leibler divergence as the information-theoretic measure for the identification of event-based novelty [32].

Information-theory-based techniques do not require any assumptions about the distribution of the training data. The only required computation is for the applied information-theoretic measure, hence, it is easier to build than other techniques. However, the accuracy is based heavily on the selection and calculation of the key measure. In general, the techniques can only detect anomalies if there is a significantly large number of anomalies in the test data set [5]. One of the challenges that remains, is to define a suitable threshold for the difference between old and new key measurements. This is normally done by trial and error.

3.2. Comparison of Anomaly Detection Techniques

Due to the time constraint of the thesis, we analyzed only three techniques from three categories in subsection 3.2.1. The techniques are kNeighbors (Distance-based), IsolationForest (Isolation-based) and OneClassSVM (Domain-based). They are open-source and gave promising results in preliminary testing. We benchmarked them to find the most suitable technique to use in our ranking algorithm. The results of the benchmark are shown in subsection 3.2.2.

3.2.1. The techniques

In the following we will describe how the chosen techniques are implemented.
kNeighbors

We implemented kNeighbors with the help of the Python scikit-learn Package `sklearn.neighbors.kNeighborsClassifier` [3]. The library does not help finding anomalies in a data set. Instead, it is developed for classification of data points with discrete labels. We only used the function to find neighbors of a data point. For each data point of the test data set, we find the nearest k neighbors of that point (In the benchmark, k is set to be 10).

The function `predictKNNOutlier()` is implemented to label the test data point as anomalous or not, based on the k neighbors. Basically, the function consists of three steps. First, it finds the maximum distance between the k neighbors. Second, the closest distance from the test data point to any of the k neighbors is calculated. In the last step, the function compares the two distances from the previous two steps to label the test point as an anomaly or not.

IsolationForest

`IsolationForest` has been implemented in the scikit-learn library `sklearn.neighbors.IsolationForest` [2], which was used in our experiment. The training model is first initialized using the Python code

```
isoform = IsolationForest(n_estimators=100, max_samples='auto',
    contamination=0.01, max_features=1.0, bootstrap=False, n_jobs=1,
    random_state=None, verbose=0)
```

The parameter *n_estimator* sets the number of base estimators in the ensemble (the number of binary trees). Parameter *max_samples* sets the number of samples to draw from the training data set to train each base estimator. If *max_samples*="auto", the number of samples is $\min(256, n_samples)$. With the parameter *contamination*, the estimation of anomalies in the training data set is passed to the algorithm. *contamination* is then used to define a threshold for the prediction. The contamination value was set to 0.01, since we know that the training data contains only a small amount of anomalies. The number of features to draw from training data is set with *max_feature*. If the parameter *bootstrap* is true, individual trees are fit on random subsets of the training data sampled with replacement. If it is false, sampling without replacement is performed. *n_jobs* defines the number of jobs to run in parallel for both fit and predict. *random_state* is used to set the random generator. It is used if reproduction of the result is required. Parameter *verbose* enables debugging output.

A model is trained and applied on the test data set by simply calling the following code:

```
isoform.fit(training_data_set)
res = isoform.predict(test_data_set)
```

res is a vector that contains the result of the prediction, where -1 defines anomaly and 1 defines normal data.

OneClassSVM

Similar to `IsolationForest`, we found an implementation of the `OneClassSVM` in the open source package `sklearn.svm.OneClassSVM` [1]. The training model is initialized with

```
ocm = OneClassSVM(kernel='rbf', gamma='auto', nu=0.01, cache_size=200,
    verbose=False, random_state=None)
```

We specified the kernel type to be used in the algorithm with parameter *kernel*. A kernel is

the type of function that the OneClassSVM will use to assemble the boundary. The implemented kernel types in scikit-learn are 'linear', 'poly', 'rbf', 'sigmoid' or 'precomputed'. The default kernel is 'rbf'. Parameter *gamma* sets the kernel coefficient. If *gamma* is 'auto', the coefficient is $1/\text{number_of_features}$. *nu* constraints the upper bound on the fraction of training errors and a lower bound of the fraction of support vectors. The size of kernel cache is limited by *cache_size*. Parameter *verbose* controls the verbosity of the training process. The limit on iterations within solver is defined by *max_iter*. *random_state* is the seed of the random generator used when shuffling data. It is used if reproduction of the results is required.

OneClassSVM model is trained and applied by calling the following code:

```
oem.fit(training_data_set)
res = oem.predict(test_data_set)
```

Similar to IsolationForest, *res* is a vector that contains the result of the prediction, where -1 defines anomaly and 1 defines normal data.

3.2.2. The benchmark results

The three implemented techniques were applied on our data to measure their speed and accuracy. More information about the data set can be acquired from section 4.1.2.

The result of the speed test is shown in Figure 3.6. The speed test measures the time required to train the model as well as the time required to test all data instances in test data set. As we can see, the kNeighbors technique takes significantly more time to process than the other two techniques. The reason for this is that kNeighbors requires many search operations for the neighbors from the training data. During the training phase, kNeighbors simply saves all training data into memory. During the testing phase, for each test instance, a certain number of neighbors must be found from the training data. The searching process requires many access to memory operations, which are bottlenecking the algorithm. Meanwhile, IsolationForest and OneClassSVM are optimized in their implementation so that only a few linear iterations over the training data are necessary during the training phase. This requires less time than in the case of kNeighbors. During the test phase, no access to the training data is demanded by IsolationForest and OneClassSVM.

Since we have no way to identify for each data point whether it is anomalous or not, the target of the accuracy test is to compare the difference between the output of every algorithm to each other. For this purpose, we implemented a function *getGroundTruth()* that also makes a prediction for each test data point, and then compared that output with the predictions from the anomaly detection algorithms. The prediction from the function could also be wrong, but it is not important since we only want to compare the similarity between the three algorithms. Function *getGroundTruth()* works on the data set that is cleaned and enriched through a process that will be describe in section 4.4.1. It compares the value of the test data with the min/max value of the training data at the same

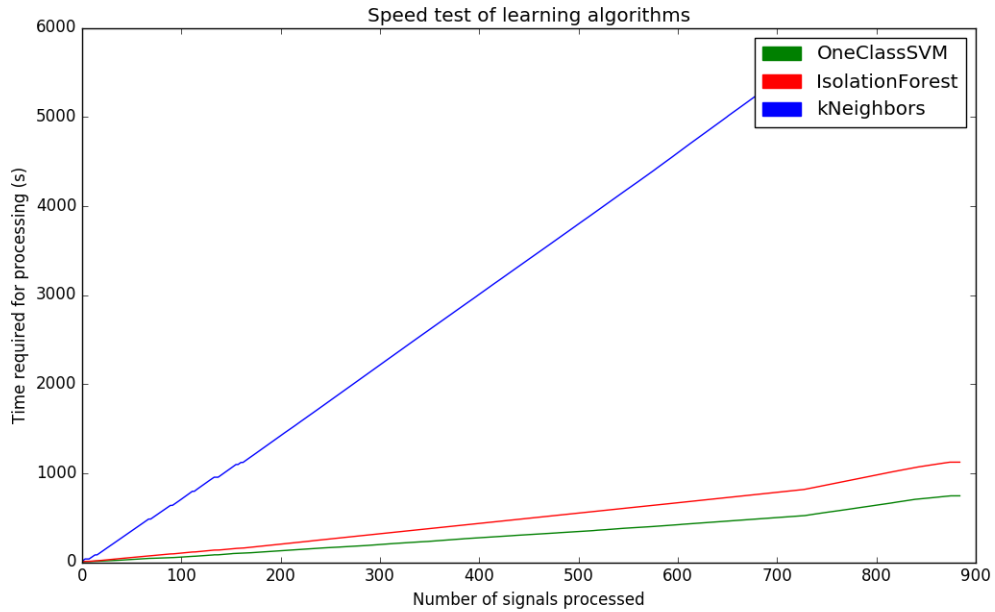


Figure 3.6.: Speed benchmark of the techniques

reference point, which is the time stamp before data preparation and the vehicle speed after data preparation. If the test data lies within the range of the min/max value then we define it as normal, otherwise anomalous. The output from `getGroundTruth()` is then compared with the classification from the three algorithms on the same data point. The accuracy for an algorithm, is the percentage of data points that are correctly classified, compared to `getGroundTruth()`.

Figure 3.7 illustrates the result from the accuracy test for 100 signals. All three algorithms return similar results. However, on average, IsolationForest has better accuracy among the three techniques. This could be explained by the fact that IsolationForest uses random vertical and horizontal cuts on the data set to detect anomalies. That technique works well on data from the automobile domain, since the values of the signals usually vary inside a fixed range. Therefore, with the right cut, IsolationForest can find the anomaly very effectively. Figure 3.8 depicts a typical signal and the result of the classification using the three techniques. In the upper graph, we plotted the data from the training and test data sets. The bottom figure shows the classification from each technique for the test data at each time stamp.

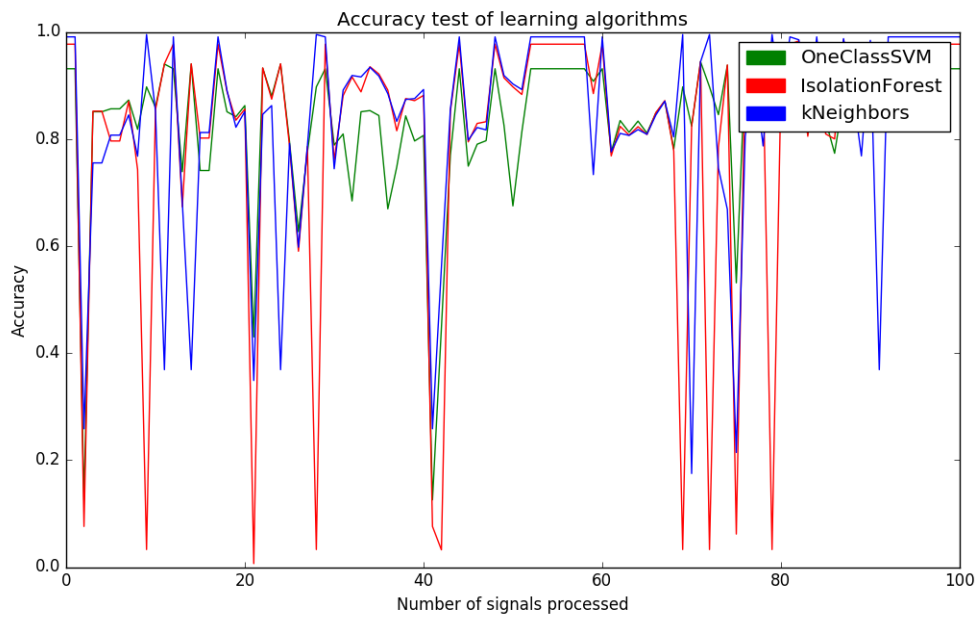


Figure 3.7.: Result of the accuracy benchmark

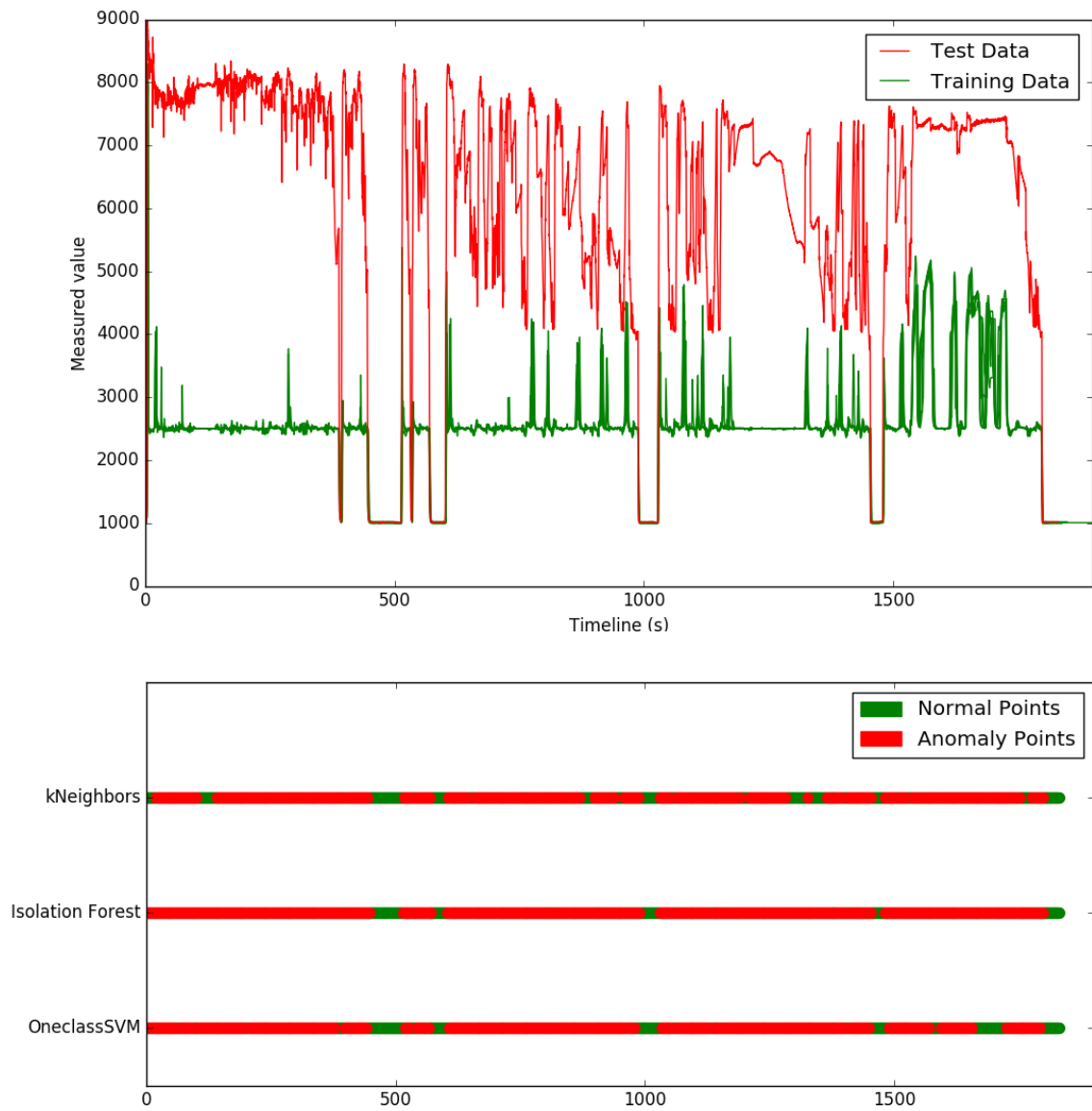


Figure 3.8.: Accuracy test on a typical signal

4. Experimental Environment

4.1. Problem Setting

This section will give the reader a background of the task and the technical aspects of the data used in the experiment.

4.1.1. Review of the task

The company IAV receives from each emission test a data container, which consist of multiple signal measurements in form of time series. The signals are either from emission measurement devices or from sensors inside the car. Based on the measures from emission measurement devices, we can see whether a car meets the required emission limit from the regulator or not.

For the cars that exceed the emission limit, we want to know the root-cause of the high emission measurement. One possible solution is to use the measured signals from the sensors to identify which part is damaged. Until now, that process is manually executed by engineers. Usually it takes a considered amount of time until the root-cause is found.

We want to improve the analysis process by applying machine learning in order to help the engineer. From positive emission tests in the past, measurements were gathered. We will use them as training data for the machine learning algorithm.

The result of the program is a ranking, which brings the most significant signals related to the high emission result to the top. Using the ranking, the engineers will check signals from top to bottom to find the root-cause. A correct ranking system will help the engineers find the problematic signal faster than blindly looking at thousands of signals.

4.1.2. The Dataset

Each dataset is a collection of signals, measured during one emission test with the following characteristics:

Driving cycle:

There are multiple standards for determining the levels of pollutants and emissions. They are called driving cycles or test cycles. The most prominent driving cycles in Europe are WLTC (Worldwide harmonized Light vehicles Test Procedure) and NEDC (New European Driving Cycle). Each cycle defines a strict guidance of an emission test such as temperature of the testing area, tire pressure and type, sort of gasoline used and total car

4. Experimental Environment

weight for each car class. The duration of a WLTC test is always 1800 seconds while a NEDC test lasts for 1200 seconds.

One important factor of each driving cycle is that the speed of the vehicle during the test is well defined. The test driver must drive the car at a defined speed and shift the gear at the correct moment. So when looking at the vehicle speed graph of different emission tests with the same driving cycle, we observe similar graphs as in Figure 4.1 or Figure 4.2.

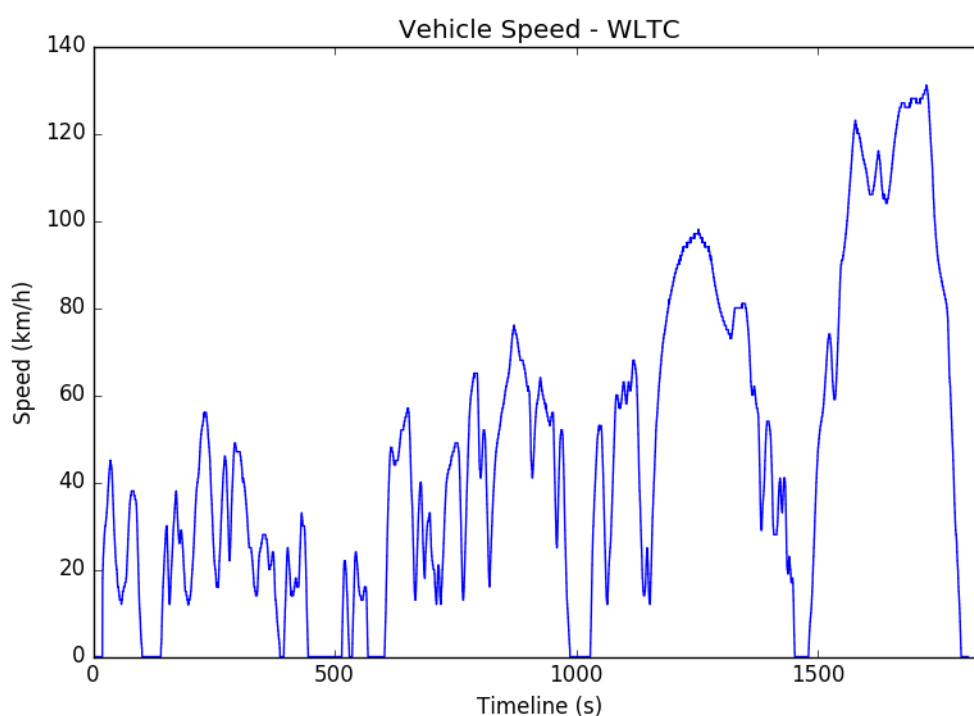


Figure 4.1.: Vehicle speed from a WLTC driving cycle

Signals:

Signals are either from sensors built in a car or from external measurement devices. Because the sensors and measurement devices are from different manufacturers, their configurations are different. They have different resolutions, ranging from 1 to 40 data points per second.

An external measurement device measures the emission of the car. Depending on the measurements, the emission test is classified as passed or failed.

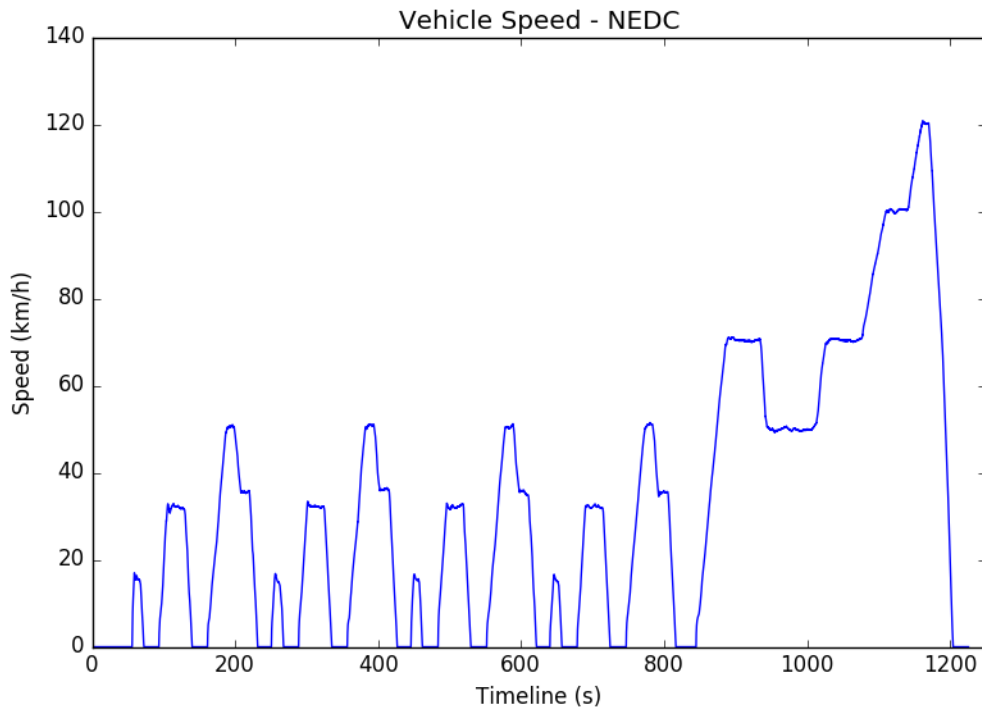


Figure 4.2.: Vehicle speed from a NEDC driving cycle

The data container:

Each emission test is recorded and saved in a MDF file. MDF (Measurement Data Format) is a binary file format that was developed in 1991 by Vector Informatik GmbH in co-operation with Robert Bosch GmbH. Since then, it is widely used in the automotive industry. The format is used for measuring, transferring and post-analysis of the data.

Further details of the MDF file can be found in [71]. The following part will extract the most important information for this thesis from that document.

The MDF file has a hierarchy structure with multiple blocks. Each block has a certain block type and contains many data fields. A block of one type can hold links to other blocks of different types. The link is an absolute pointer to the position of a block in a MDF file. Figure 4.3 depicts the general block structure of a MDF file in version 3.1.3.

The MDF file always begins with the IDBLOCK. It has a fixed size of 64 bytes and contains information related to the source of the file and its format.

From byte 65 the MDF file stores the header block HDBLOCK. It includes information about the contents of the measured data file. Also from the HDBLOCK, we have a pointer to the first data group block DGBLOCK as well as the number of DGBLOCK.

The DGBLOCK contains a link to the channel group block CGBLOCK. Each data group

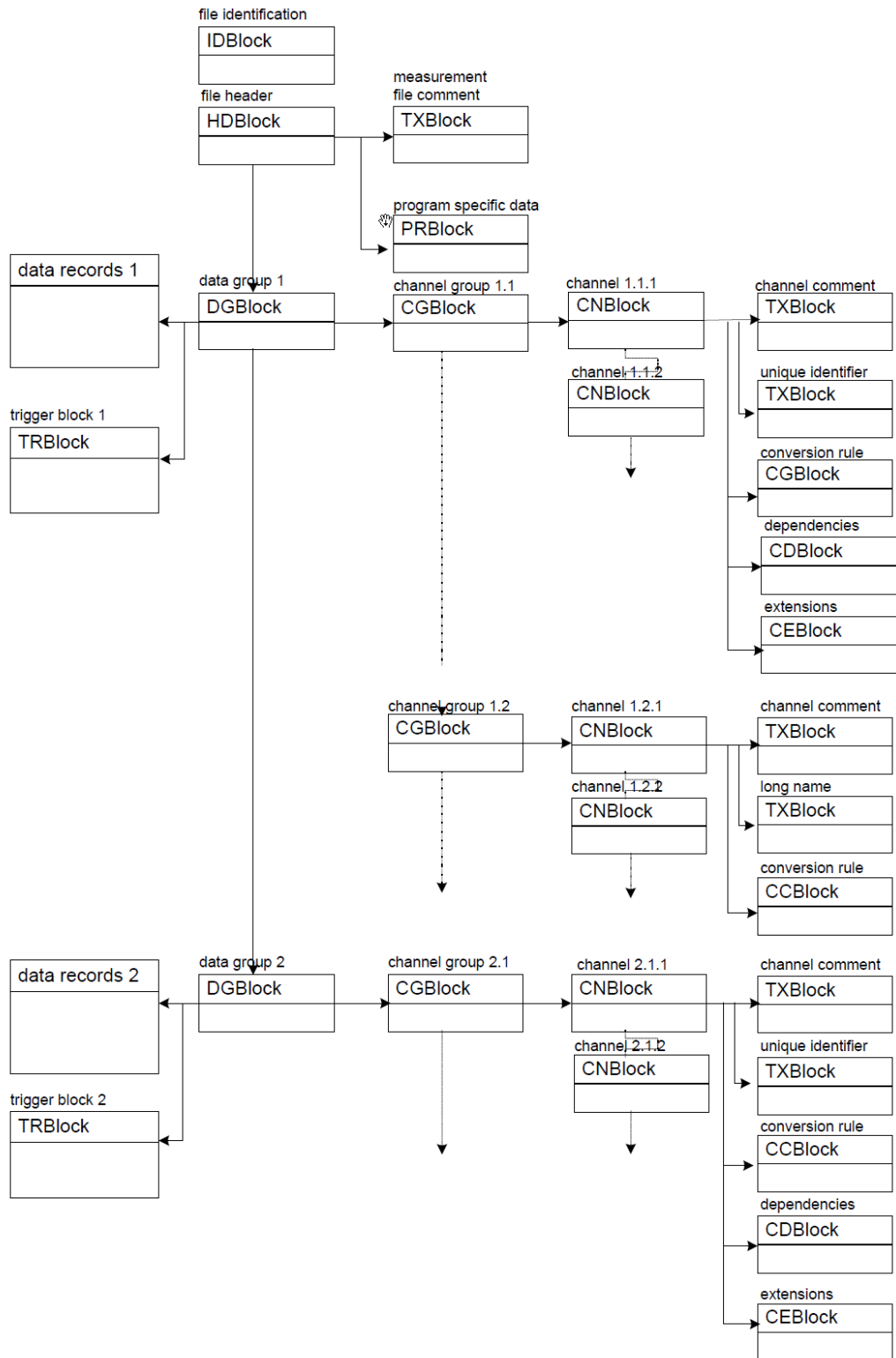


Figure 4.3.: General block structure of an MDF file [71]

block can be understood as a measurement device.

The channel group block is a collection of channels with the same time resolution and length. Each CGBLOCK again contains links to the channel blocks CNBLOCK.

Finally, each channel block CNBLOCK gives detailed information of a time series, which is the actual measured signal that we need.

4.1.3. Definition of our training and test data set

Training datasets are defined as the MDF files that we collected in the past. Those MDFs are from emission tests with a positive result. An emission test has a positive result if the amounts of emission measured from the test vehicle does not exceed the limit.

Test data set is a MDF file from a failed emission test. The government sets for each standard (for example EU4, EU5) a limit of the emission output from a vehicle during a test cycle. The emission could be HC, CO, CO₂, NO_x, Non-Methane Hydrocarbons(NMHC) or Particulate Matter(PM). An example of regulated limits for the regulatory frame EU5 is shown in Table 4.1. The violation of any limit in an emission test results in the failure of the test.

Table 4.1.: Limits for passenger cars from regulatory frame EU5 (NEDC driving cycle)

Emission	Unit	Limit
HC	mg/km	100
CO	mg/km	1000
NO _x	mg/km	60
NMHC	mg/km	68
PM	mg/km	4,5

4.2. Solution proposal

The main task of finding the root-cause for a failed emission test is equal to finding the components that malfunctioned. It is self-evident that if a component of the car malfunctions, the measured signal coming from that component will contain anomalies. Given that assumption, the main task of the solution is in fact to identify signals that have the most anomalous measured values. This task gives us the initial ranking system.

However, just finding signals with anomalies is not enough. In our data set, we found many signals that have high rates of anomaly, but it turned out that they were not the direct reason for a high emission output. In fact, most of them come from a software or hardware adjustment of components that do not affect the emission outcome at all. Therefore, we need to find the signals that are directly related to emission outcomes. For this task, we created a second ranking system using correlation coefficient.

In the following subsections, we will explain how we obtain each single ranking system and combine them to acquire the final ranking. The engineer will work with the final ranking to find the searched root-cause. Section 4.2.1 explains how we used the anomaly detection technique to rank the signals. Section 4.2.2 shows how the correlation coefficient are used to create the second ranking system. Finally, in section 4.2.3, we demonstrate a way to combine the ranking systems to create the final ranking.

4.2.1. Ranking based on anomaly percentage

For each signal, we trained a model using data from positive emission tests in the past. After that, for each signal in the test data set, we use the previously trained model to detect anomalies in the measured values. Every single data point from the measured values is tested using the trained model. They will be classified as normal or anomalous data points.

Given the known number of data points, classified as anomalous or normal, we calculate the anomaly percentage of each single signal as follows:

$$\text{Anomaly percentage} = \frac{nadp}{ndp + nadp} \quad (4.1)$$

with

$$\begin{aligned} nadp &= \text{Number of anomalous data point} \\ ndp &= \text{Number of normal data point} \end{aligned}$$

Using the anomaly percentage, we rank the signals in a descending order. The signals with the highest number of anomalous data points will have the highest rank and vice versa. The result is a first ranking system for the combination of ranking systems as will be described in section 4.2.3.

4.2.2. Correlation between signal and emission

In this subsection, the influence of a signal to the emission outcome is measured. This can be achieved using the following steps:

First, the difference between the emission outcomes of a failed and an accepted emission test is calculated. The result is a time series, which is called change in emission.

Second, for every signal, we again find the difference between data points from the signal in an accepted emission test and a failed emission test. The result for each signal is a second time series, which we call change in signal.

As an example for one signal, Figure 4.4 shows the change in emission and the change in signal.

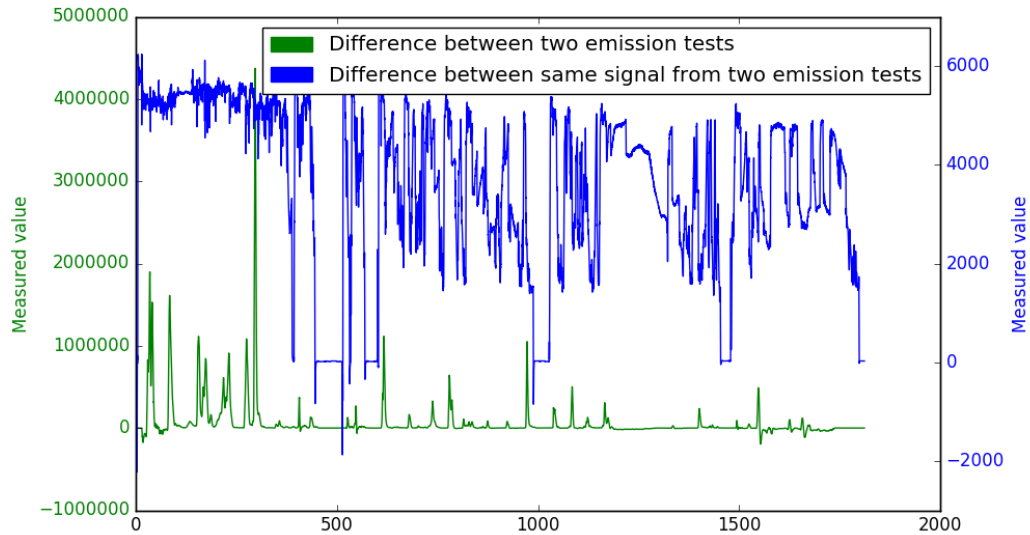


Figure 4.4.: Plot of emission change vs. signal change

In the third step, the correlation between the two time series is calculated. The most general and most used measurement for correlation is the Pearson correlation coefficient (Pearson ρ or Pearson's r) [52]

The formula of Pearson's r is:

$$r = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}} \quad (4.2)$$

where:

4. Experimental Environment

n is the number of samples

x_i, y_i are the single samples indexed with i

$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$ (the sample mean); and analogously for \bar{y}

r has a value between -1 and +1. If it is close to +1, the two time series have a positive linear relationship. This means if the value of one time series increases at one certain point of time, the value of the other time series also increases at the same time. On the other hand, if r is close to -1, the two time series have a negative relationship. If r is approximately 0, the two time series are not correlated to each other. Since both positive and negative linear correlations are relevant, the absolute value of r is taken as the final ranking criterion.

As a last step, after having the Pearson's r calculated for every single signal, all signals are ranked based on their Pearson's r value. The result is a new ranking system for the combination of ranking systems.

4.2.3. Combination of ranking systems

After the ranking systems are built (as explained in the previous steps), we create the final ranking system that will be shown to the engineer.

For each signal, the average of its rankings from different systems is calculated. This average is then used as the criterion to sort all signals. An example of the final ranking is depicted in Figure 4.5

Signal	Ranking_Anomaly	Ranking_Correlation	Average	New ranking
A	1	7	4	3
B	4	6	5	6
C	3	2	2.5	1
D	6	3	4.5	5
E	2	4	3	2
F	5	5	5	6
G	7	1	4	3

Figure 4.5.: Example for combining of ranking systems

By combining the two ranking systems, even when a signal does not have many anomalies in its data, it would be still be located at the top of the final ranking if it

strongly affects the emission outcome (Signal C in the example). On the other hand, a signal that is highly anomalous but is not related to the emission outcome will eventually be pushed down in the final ranking (Signal A in the example).

4.3. Architecture

Figure 4.6 depicts the technology stack used in the system.

First, Hadoop is used to store the time series in text data form. Apache Hadoop is an open-source software created by the Apache Software Foundation. It aims at storing, distributing, and processing of data sets using computer clusters. The clusters can be built with any commodity hardware instead of expensive servers, to minimize the cost.

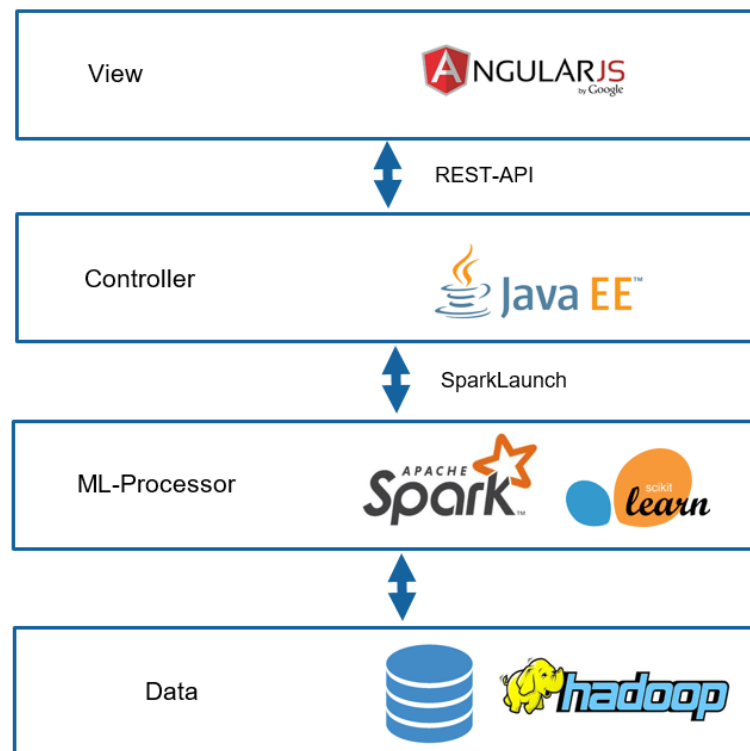


Figure 4.6.: Technology stack

Secondly, Spark is employed for running the machine learning algorithm on a cluster of machines. Spark is another open-source framework built by the Apache Software Foundation. By using Spark, the analysis process of signals can be divided automatically between computers, inside our cluster. Spark automates the process of distribution and aggregation of data and results, as well as balancing of loads between different nodes in

the cluster.

Thirdly, a program written in JavaEE is used to control the Python script on Spark and retrieves the data from Hadoop. It acts as a middleware communicating between the front-end developed in AngularJS and the machine learning layer in Spark. Lastly, AngularJS is used in the front-end to display the result to the engineer.

At the end of the thesis, only the algorithm for the recommender system is fully implemented. The system to manage training data as well as test data is just partly finished. However, a schema for a database that stores the data set is proposed. The schema is presented in Figure 4.7.

After the user uploads the MDF file, the file is saved to Hadoop. The information about the file will be stored in the table `MDF_File`, whereas the information about emission outcome is stored in table `Emission_Measure`. Using the reading procedure, measured time series from the MDF file are extracted into text files. Each text file contains one time series and is saved in Hadoop. The path to each text file as well as the name of the corresponding signal are stored in table `Signal_Measure`. Since the signals could have different names in different tests, we use tables `Signal_Info` and `Synonyms` to note the synonyms. The trivial errors that are marked during the data preparation step are stored in table `Marked_Anomalies`. Table `Model` is used to save the trained model from the training step of the ML algorithm for later application.

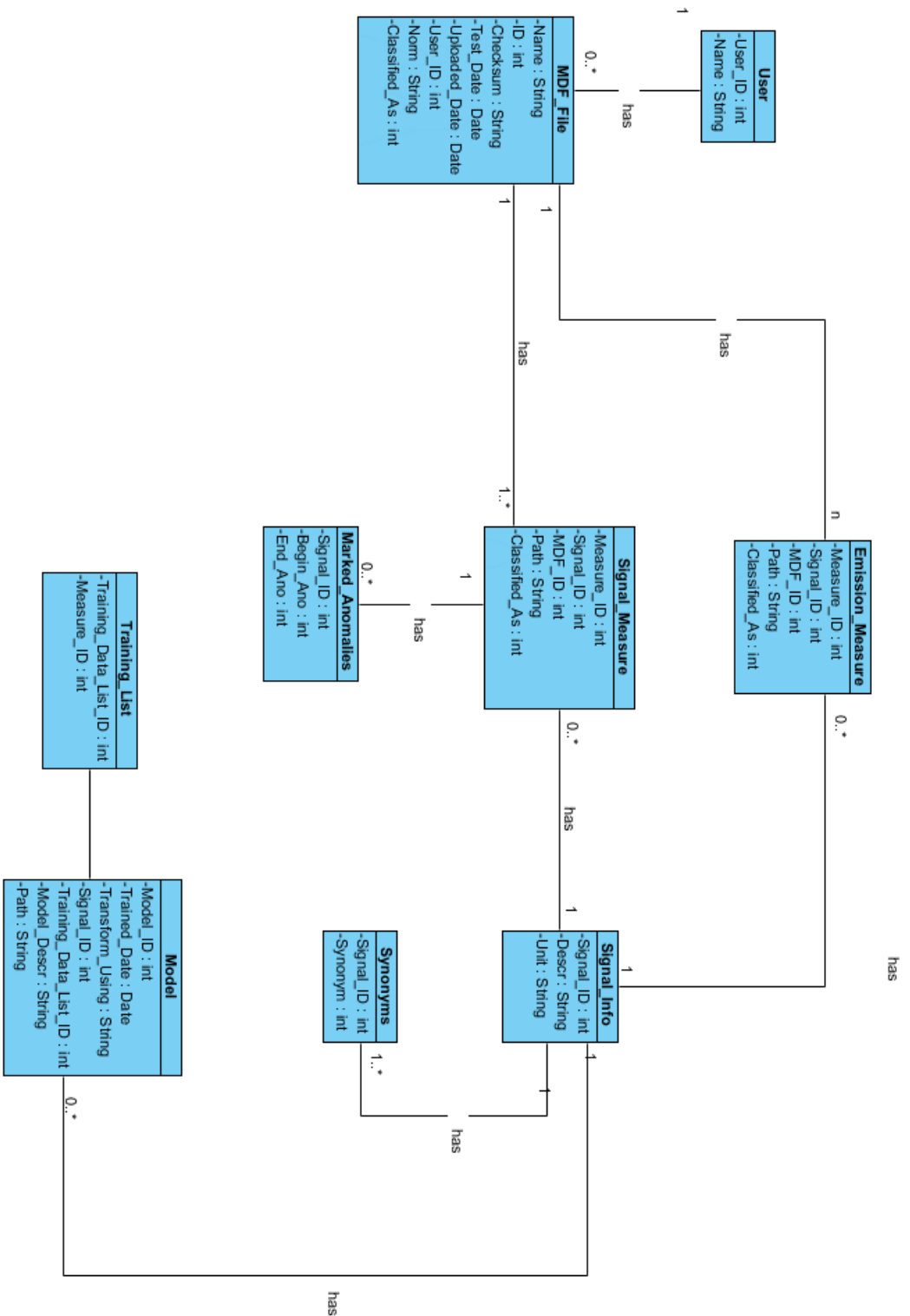


Figure 4.7.: Database schema

4.4. Implementation

In this section, we will talk more about details for the implementation of the analysis process in the back-end as well as the interface of the front-end. Figure 4.8 depicts the data pipeline of the implemented system. The first three blocks of the data pipeline (Upload MDF, Splitting MDF, Detect outlier and suggest of elimination) are described in section 4.4.1. The generation of different ranking is presented in section 4.4.2. In section 4.4.3, we illustrate how the results are visualized.

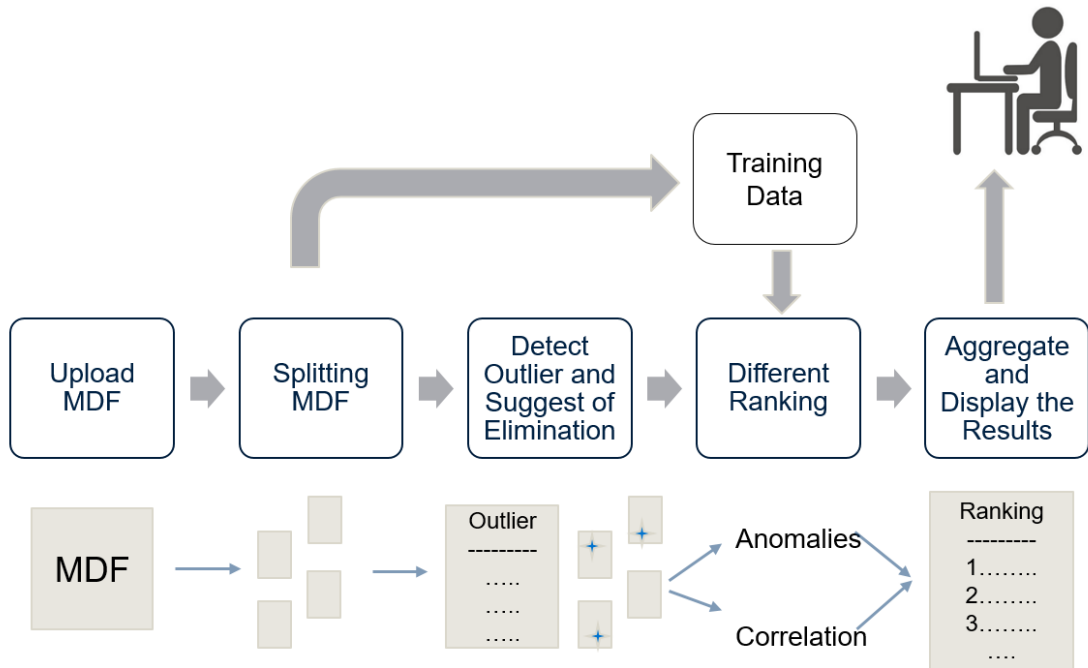


Figure 4.8.: Data pipeline

4.4.1. Data preparation

Several preprocessing steps are required to make the data usable. In the following, we will demonstrate these steps in detail.

First, to further process the time series from the signals and use them for the algorithms, we need to extract each single signal from the CNBLOCK.

Initially, the system was planned to be fully built in JAVA. We found a JAVA library which can extract signal data from a MDF file into the memory. However, the JAVA library is not optimized and hence being very slow and consumes a lot of main memory (RAM). For our first MDF file, which has the size of 80 MB and contains 850 time series, the small program that reads all signals consumed almost 2 GB of RAM. For the second MDF file, which has the size of 200 MB and includes 4000 time series, the program failed to read all

signals into memory.

Another library is then needed to work effectively for machine learning. Unfortunately, no other JAVA library is found that meets our requirement. Looking for other libraries in other languages, we found the MDFReader written in Python from Aymeric Rateau, which is the most promising one [9]. It requires only 2 GB of RAM for our second MDF file and reads the whole file in less than 10 seconds.

However, for both JAVA and Python libraries, simply writing signals into main memory still poses a big problem. Even when we want to read one signal from a MDF file, it is still required to load the whole MDF file into the main memory. One MDF file already consumes 2 GB of RAM. When our training dataset grows, there will be not enough memory for processing.

The proposed solution is to extract all signals into multiple text files and manage the location of them in a database. This way, if we need to detect anomalies in one signal, only the files that contain the time series related to that signal are loaded into memory. The restriction of working memory is then overcome.

In section 4.3, we proposed how the text files and the paths should be stored in the database for easier query and access. In this thesis, for test purposes, the name of the signal and the location of its corresponding text file are stored in `matching.txt`. The following code is used to export all signals into multiple text files and save the matching in another text file:

```
import mdfreader
from io import StringIO
import os

def exportMDF(trainingParentLoc, dat):
    exportFolder = trainingParentLoc + 'Exported\\'
    yop = mdfreader.mdf(dat)

    if not os.path.exists(exportFolder):
        os.mkdir(exportFolder)
    match_str = StringIO()

    i = 1
    for master_channel in yop.masterChannelList:

        signals = yop.masterChannelList[master_channel]
        time_data = yop.getChannelData(master_channel)
        length = range(len(time_data))

        for signal_name in signals:
            if signal_name != master_channel:
```

```
print("fetching " + signal_name)
signal_data = yop.get(signal_name)

signal_str = StringIO()
signal_str.write("%s\n%s\n%s\n" % (signal_name,
    signal_data['unit'], signal_data['description']))
signal_data = signal_data['data']

for j in length:
    signal_str.write("%s %s\n" % (time_data[j],
        signal_data[j]))

f = open(exportFolder + str(i) + ".txt", "w")
f.write(signal_str.getvalue())
f.close()

match_str.write("%s %d\n" % (signal_name, i))
i += 1

matching = open(exportFolder + "Matching.txt", "w")
matching.write(match_str.getvalue())
matching.close()
```

In the second step, we need to eliminate the errors within the data set. These errors are due to malfunctioning sensors or they occurred during the file transfer.

Figure 4.9 shows an example of the vehicle speed measured in an emission test. When all other measured values lie between 0 and 200 km/h, one single data point has a value over 10000 km/h. Since we know the speed of a car in a test can not exceed 200 km/h, that data point is clearly an error. This type of error usually does not affect the measured emission and is not the root-cause of the problem. Therefore, we cleaned the data sets from those trivial errors. For this purpose, some set of rules were then taken into the preprocessing steps. As the system grows bigger, those rules can be extended. In the scope of this thesis, the basic rules that were taken into account are:

- The time marked must be correct according to the test cycle. For example, in a WLTC emission test, the vehicle is only tested for 1800 seconds. So any pair of $(time, value)$ with $time < 0$ or $time > 1800$ will be marked as incorrect and will be eliminated from the training and test data set.
- The measured value must be correct based on the signal. For some signals, the maximum and minimum values are known. Therefore, if the measured value at any certain time does not lie between this range, all values from this signal as well from other signals measured in this time stamp will be deleted. This type of errors happens mostly when an error occurred during the test. For example, a cable is unplugged or loose and then is plugged in again.

In the third step, after the trivial errors are eliminated, we need to process more complicated errors. Those are errors, which could not be found in step two and can not

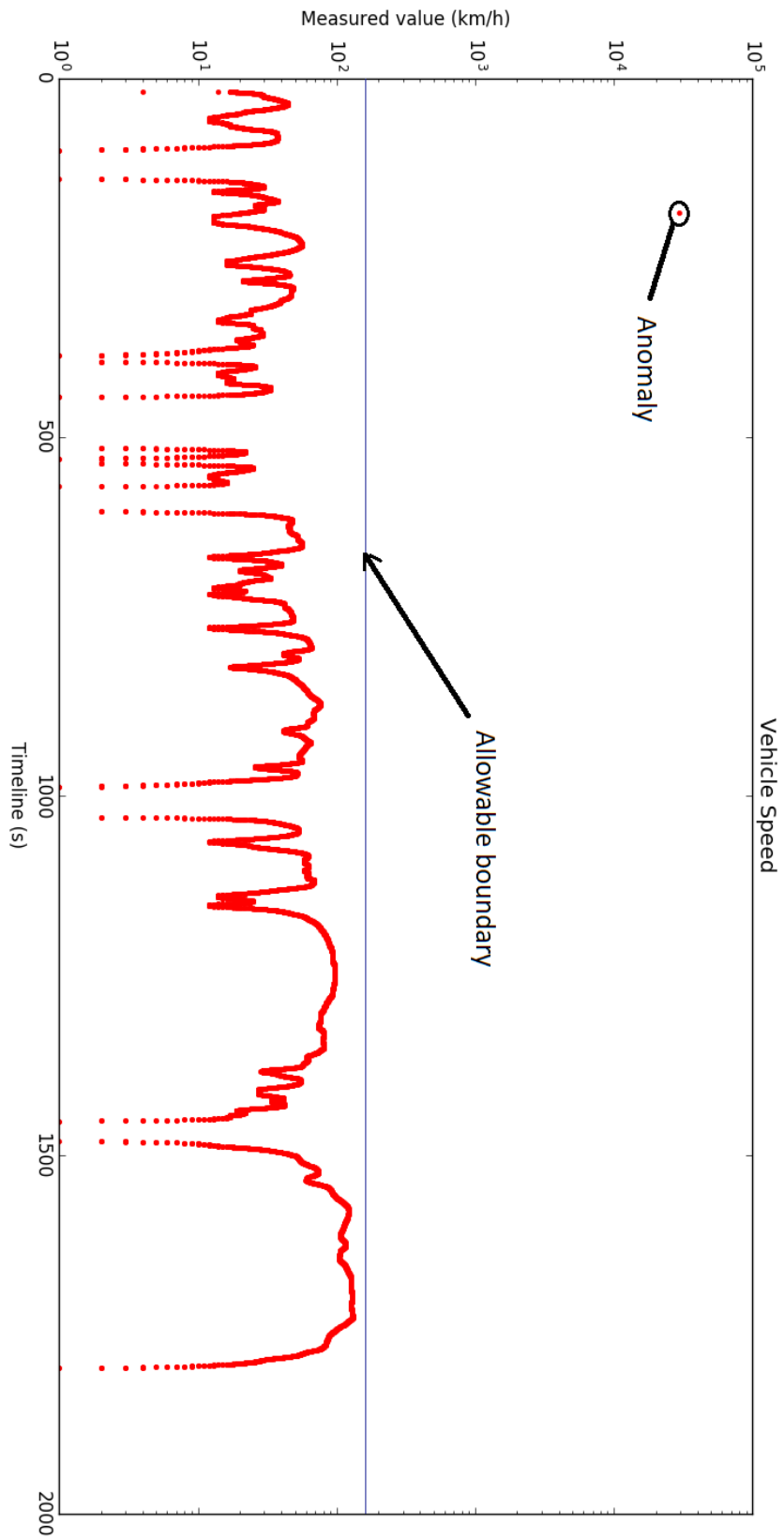


Figure 4.9.: Vehicle speed with error during test

be simply removed without checking further.

For other signals that we do not know the true range of their values yet, we can use a simple anomaly detection technique such as a probabilistic-based one, to detect anomalies inside one data set. The IQR technique is used here. First, we find: 0.1-Quantile (Q_1), median and 0.9-Quantile (Q_2). The IQR is defined as $Q_2 - Q_1$. Using a threshold value of $\alpha = 2$, a boundary between $Q_1 - 2 * IQR$ and $Q_2 + 2 * IQR$ is used to detect anomalies. All values that lie outside of the range $[Q_1 - 2 * IQR, Q_2 + 2 * IQR]$ will be marked as anomalies in the data set. The 0.1- and 0.9-Quantile are chosen so that the range $[Q_1 - 2 * IQR, Q_2 + 2 * IQR]$ is big enough, so only data points that vary extremely from the rest of the data will be detected.

One single anomaly could indeed be the root-cause that we are looking for. Hence, we need to check the correlation between an anomaly and the emission outcome before eliminating the anomaly. We assume that if one anomaly value is responsible for a high emission outcome, the peak of the emission outcome must be found near the time stamp of that value. As a result, we define a time range of $\pm s$ seconds in which the maximum value of the emission outcome (the peak) should occur. If the peak is not found in that time range, the anomaly is declared as trivial and will be eliminated. In case when the peak is actually found in the defined time range, the anomaly will be showed to the engineer.

Figure 4.10 is an example where the detected anomaly is turned out to be a trivial error. Even with a wide defined time range (60 seconds), the peak of emission outcome is still located outside of this range.

In the fourth step, we synchronize the time stamps of all signals.

Since the signals have very different resolutions (from 1 to 40 Hertz) as well as starting time (even just millisecond), we must find a way to synchronize the time stamps so that the data can be used. There are two ways to achieve the synchronization:

The first way is to interpolate all time series with low resolutions to the finest resolution among all signals. By using interpolation, we have more data for training and hence the trained model could be more accurate. On the other hand, interpolation increases the training time vastly. The Python library *scipy* provides the package *interpolate*, which can be easily used as followed:

```
from scipy import interpolate
def interpolate_to_size(target_df, newRes):
    y = target_df['value']
    x = np.arange(0, target_df['value'].size)
    f = interpolate.interpld(x, y)
    xnew = np.linspace(0, target_df['value'].size - 1, newRes,
```

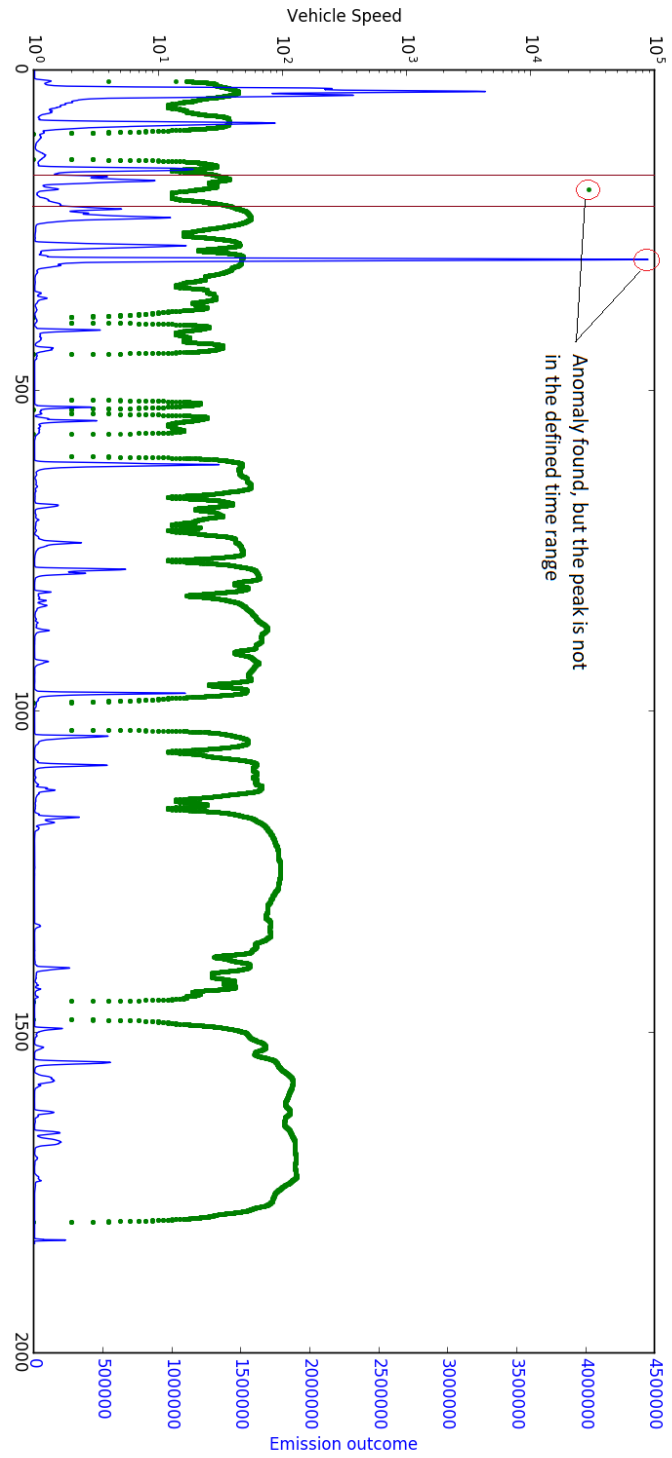



Figure 4.10.: A signal with trivial error

4. Experimental Environment

```
    endpoint=False)
    ynew = f(xnew)
    return ynew
```

The second way is to reduce all time series with a finer resolution to a lower resolution. This can be achieved by taking either the sum or the mean of all measured values between two time stamps in a lower resolution. Reducing data can be carried out more quickly compared to interpolation. The trade-off is less data for the training process during machine learning.

In our implemented system, we combine the two ways to synchronize the time stamps. The final resolution is set as a variable in the system, with default value of one second. A time series with lower resolution will be interpolated. Time series with higher resolution will be run through the reducing process.

The last step is to enrich the data and make it easier to detect anomalies.

Using only the given time series as input, is the traditional way how an engineer finds anomalies. The engineer uses his experience dealing with each signal in the past to create a virtual graph how each signal should be. When a new data set arrives, he can use his experience to notice a change in the graph. This leads to a very time-consuming and error-prone analysis, when the number of signals becomes large. An analog way to automate this limited way is to let the program learn from every time stamp of the training data sets and test on the exact time point of the test data. As an example, if we have seven MDF files as training data sets, at every time stamp of an arbitrary signal X, we let the program learn from the value measured at exactly that time stamp of signal X from seven MDF files. When a new test data set arrives, again for every second we use the learned model from seven training objects to detect anomalies. In short, the ratio of training-test instance is 7:1, which is very low. Another problem of this trivial method is that it is very sensitive to time shifts. Since the method only uses time stamp for marking training and test instance, if any of the test has idle time at the beginning, the result would not be correct any more. Also, as mentioned before, when the test cycle is changed, using the time stamp is not possible any more. To sum up, just using the given time series does not give a good result, we need to enrich the data set.

The first attempt to enrich the data is to use the measured values regardless of the time stamp. In this way, the ratio of training and test instances increases proportionally to the number of time stamps in each training data set. If the time resolution is 0.1 second, in a WLTC test, the ratio is increased by at least 18000 times. However, this method is limited since the value of the signal also depends on other factors such as vehicle speed, acceleration etc. Even when one measured value is normal at one point, it could still be anomalous at another point. We need to find these factors to enrich the data correctly.

One possible influence factor that we use to enrich the data is the vehicle speed. We assume that all signals are influenced by the vehicle speed and it is the most important influence factor. Since the vehicle speed is always measured in emission tests, we can use

this signal for every case. For each signal, we take the measured value at a certain time stamp and combine it with the measured vehicle speed at the exact same time stamp. The result is a pair of vehicle speed and signal value for every single time stamp. This way, we enriched the data successfully.

One problem that is left out in this thesis, is the delayed influence made by vehicle speed.

A change in vehicle speed can lead to a change in measured signal value that happens several seconds later (lag). Sometimes the lag is not constant. The problem could be corrected using dynamic time warping (DTW). Dynamic time warping is an algorithm that measures and maps the similarity between two time series, which may vary in speed. We could use DTW to map the delay between a change in vehicle speed and a change in signal and transform the data accordingly. However, in the scope of this thesis, we did not explore that possibility further. The reader is referred to [12] and [62] for more information.

However, the cited problem above will not affect the accuracy of the algorithm if enough data are collected and the data are enriched as previously described. In that case, the delays are included in the training data.

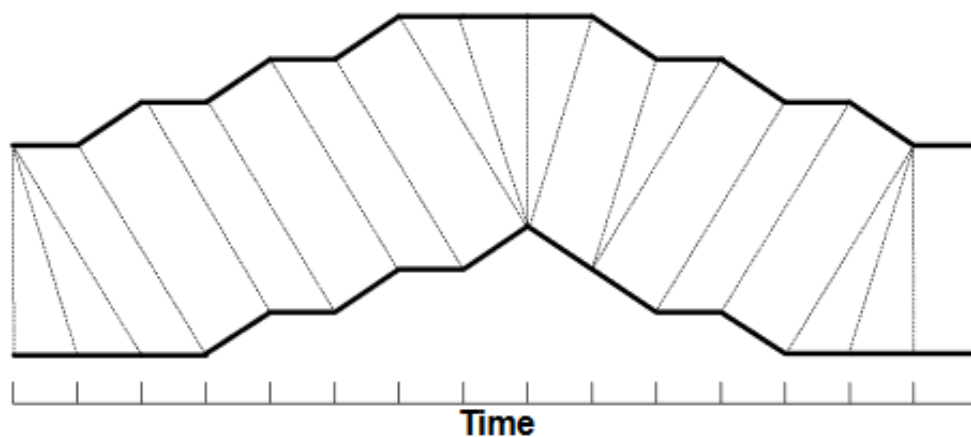


Figure 4.11.: An example of mapping two time series using DTW, taken from [62]

4.4.2. Building the ranking system

We built the ranking system based on the technique proposed in section 4.2.

First, a ranking based on anomaly percentage is created.

For the experiment, we implemented a function *runTraining1()* that trains a model based on training data of a signal and applied it on the test data of that signal right after the training. The model is trained using IsolationForest. The result is a pair of data that contains the name and the anomaly percentage of that signal.

We then used Spark to run the function on all signals simultaneously. This can be done as easily as calling the following code:

```
rank_1 = spark.sparkContext.parallelize(range(1, numberOfSignal),
    partitions).map(runTraining1).collect()
```

Spark automatically distributes the function *runTraining1()* on all machines on the cluster and collects the results at the end. The result *rank_1* is a collection of pairs (signal name, anomaly percentage). Using a sort function on *rank_1* based on anomaly percentage, we obtain the first ranking list.

Second, we built a ranking list based on correlation.

Similar to the first step, we implemented a function *runTraining2()* and used Spark to parallelize the calculation process. *runTraining2()* consists of the function `scipy.stats.pearsonr` from the Python library `scipy`. Again, after sorting the collection received from Spark, we acquired a new ranking list.

In the last step, we combine the two ranking lists created previously by using the combination method proposed in subsection 4.2.3. After this step, we obtain the final ranking list in form of a json file.

4.4.3. Visualizing the results

In the front-end, we implemented a small web-application to visualize the ranking list. Figure 4.12 illustrates the interface of the web-application. The application lets the engineer upload the json file obtained in previous steps and shows the ranking in a table on the left. The engineer can use the search function to search for a certain signal if the name is known. By clicking a row of the table, a REST request is sent to the back-end with the corresponding name of the signal in that row. In the back-end, data from that signal, in all passed and failed emission tests, are collected. The data are then plotted using the Python library `Matplotlib`. Using `Matplotlib`, we generate a figure, send it back to the front-end, and show it to the users on the right side of the web application.

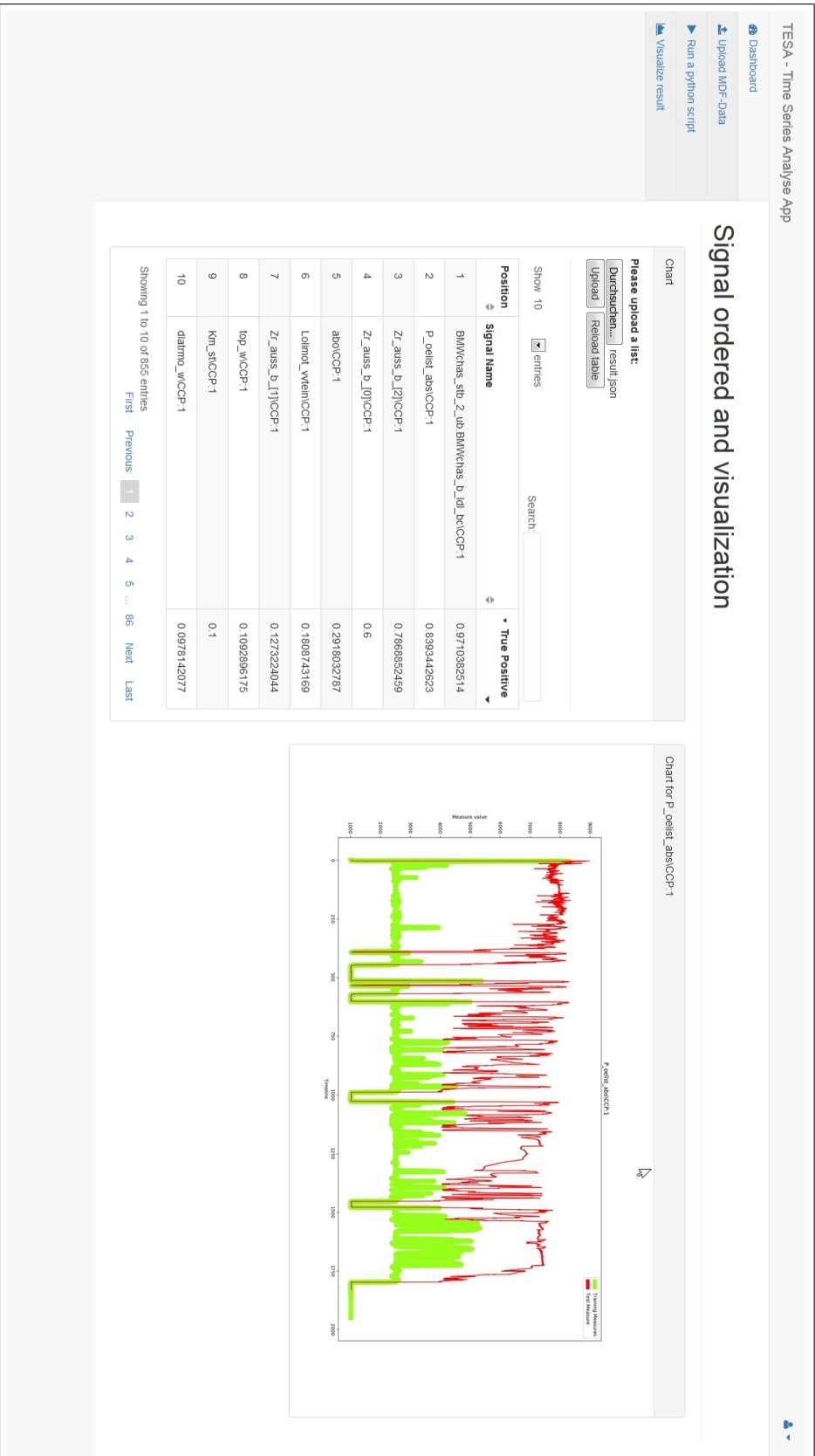


Figure 4.12.: Application interface

5. Evaluation

In the evaluation part, we tested the implemented recommender system on different problem sets and analyzed its accuracy. This chapter describes the evaluation in detail.

The problem set

We received seven problem sets from different departments for the evaluation. Each problem set is composed of one failed emission test as test data and several passed emission tests as training data. The emission tests are delivered in form of a MDF file. For each problem set, we need to find one signal, which is responsible for the failed emission test. The name of the searched signal is unknown to us but is known to the engineer.

The task

Using the prototype implemented, we trained our program on the training data (the passed emission tests) and created a ranking list for the signals in the failed emission test. For each problem set, we created a ranking list of the signals. The ranking lists are then forwarded to the engineer for further evaluation. Our engineers have to confirm whether just looking at the top 20 signals of the ranking list helped him or not.

The result

Out of seven problem sets, we successfully found the root-cause in the top 20 list in one set. That means for that problem set, the engineer saved more than 91% of the time that was required for searching for the root-cause. Instead of blindly analyzing all signals, the engineer just needed to look at 20 signals out of 900 signals.

However, the prototype failed to bring the root-cause signal into the top 20, in the remaining six problem sets. We analyzed these problem sets further and found the reason of the failure. The proposed solution is created with one critical assumption. It is that if a signal is responsible for a failed emission test, a change in that signal should also lead to a change in the emission outcome. However, this was not the case in our six problem sets. Figure 5.1 illustrates the emission outcome and the searched signal from one of the six problem sets. In the figure, the measured values from the failed test are plotted against the measured values from passed ones. We can see that from the start of the test until 900 seconds, the searched signal behaves normally. However, during that time, the emission outcome has many anomalous values. Our proposed solution can simply not find the cause-effect in this type of problem.

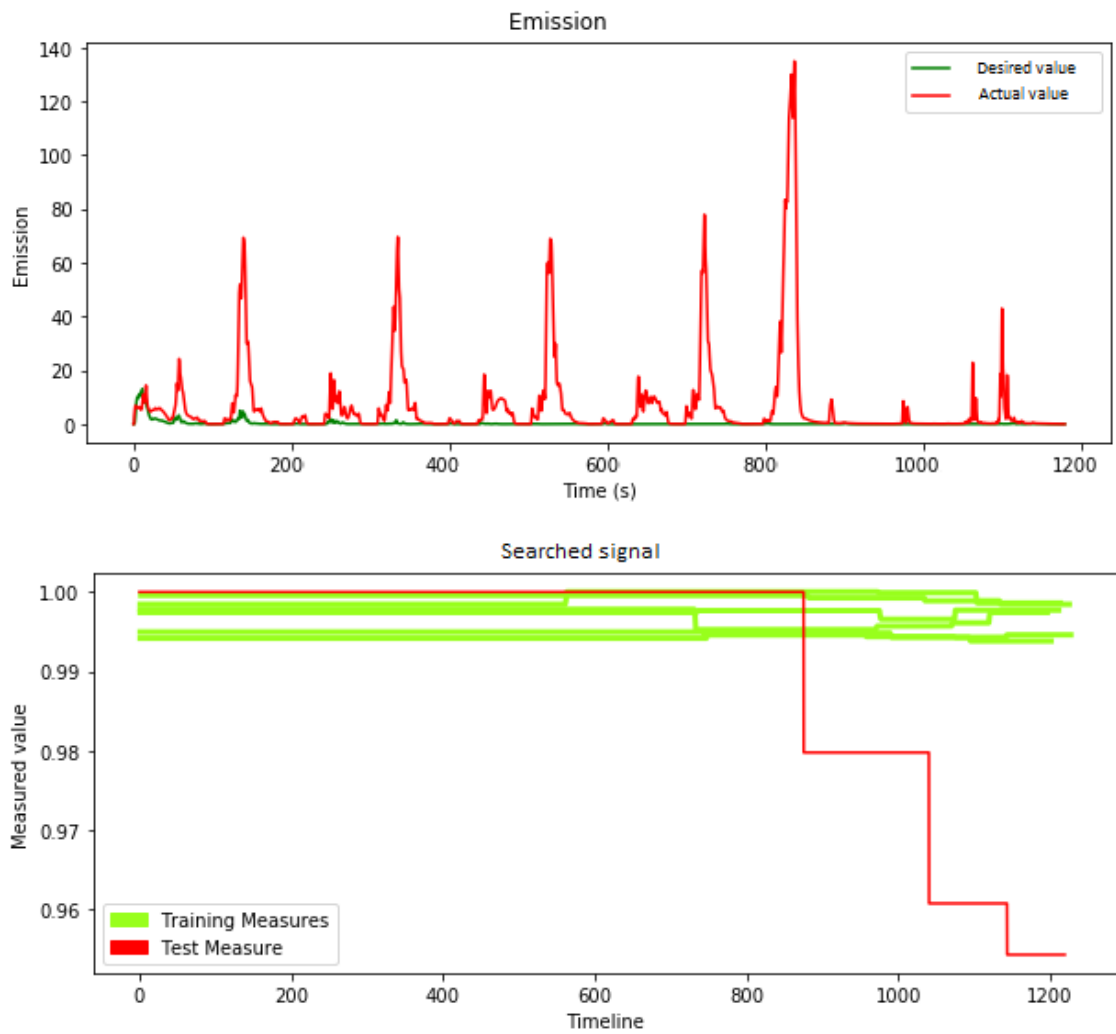


Figure 5.1.: Emission outcome and searched signal from a problem set

There is another possible explanation for why our system didn't detect the root-cause. As we stated before, there could be more than one signal related to one component of the car. The engineer might have identified the defected component first and then pointed out a related signal later. While the one that he pointed out did not meet the required assumption, the other one could have met the assumption. However, we cannot confirm our hypothesis in the scope of this thesis.

The previously mentioned hypothesis shows a drawback of the evaluation. Due to time constraint, the engineer just evaluated the results by confirming whether the searched signal is in top 20 or not. One component could be monitored by multiple sensors, hence it can be traced back by multiple signals. After all, the searched signal is just one possible signal of the defected component. Even when the other signal related to the defected component is in top 20, there is no feedback about it.

6. Conclusion and Future Work

In the scope of this thesis, we reviewed the anomaly detection techniques in the literature and benchmarked three techniques. In our benchmark, we found that IsolationForest is the technique that works best for detecting anomalies in automobile signals. Based on the knowledge gained from the assessment of the techniques, we proposed an algorithm for a recommender system in automobile development. The algorithm ranks the signals based on anomalies detected and their correlation with the emission outcome. At the end of the thesis, we implemented the algorithm to test it, using real data from industrial partners. In one out of seven tests, the algorithm successfully helped the engineer find the cause for the failed emission test. Despite the low rate of success, the recommender system in total still reduced the analysis time of the engineer compared to the regular process where they have to blindly search for the cause.

We pointed out in the evaluation the reason why the system did not work in some cases. We developed the ranking algorithm based on the assumption: If a signal is responsible for a failed emission test, a change in that signal should also lead to a change in the emission outcome. However, in the failed cases, the searched signals pointed out by the engineer did not follow that rule. This is also the weakness of the algorithm. Due to time constraints, we could not investigate further to optimize our algorithm in that regard.

Even though, using the frame of the system that we built, other algorithms can be quickly implemented and added to the recommender system for other cases. This could form another thesis in the future which focuses on finding more algorithms that can resolve our problems, instead of building a framework from scratch as we did in this thesis.

Appendix

A. Raw data

Figure A.1 illustrates example of raw data from two signals with different resolution. The signal on the left has a time step of 0.01 second. The signal on the right has a time step of 0.2 second.

```
625.0446625261782 564
625.0545503744204 564
625.0655933067362 560
625.0745062967642 560
625.0856639184738 560
625.0953223883942 558
625.1047105192981 558
625.1145410228444 558
625.1250924302847 558
625.1352751935453 558
625.1446387484491 562
625.1545184044596 562
625.1651845010608 568
625.1745480557336 568
625.1857056774413 568
625.1951101925777 571
625.2050635775183 571
625.2145500135819 571
625.2246344719158 571
625.2352923762835 571
625.244606778492 567
625.2556333268083 567
625.2654310621226 567
625.2744751255439 563
625.2846169283409 563
625.294750539372 560
```

(a)

```
624.0773656064655 0.0
624.277375370765 0.0
624.4774670557533 0.0
624.6774276776603 0.0
624.8774292746446 0.0
625.0774636398735 0.0
625.2774570448862 0.0
625.4774750261447 0.0
625.6775831203474 0.0
625.8775027967065 0.0
626.0775207780086 0.0
626.2775469513242 0.0
626.4775895088849 0.0
626.677656642695 0.0
626.8775599348926 0.0
627.0775861082645 0.0
627.2816100199095 0.0
627.4776056868202 0.0
627.6776236696467 0.0
627.8776908279403 0.0
628.077635105087 0.0
628.2777514159086 0.0
628.4776793088756 0.0
628.677107484468 0.0
628.875741027631 0.0
629.077700558292 0.0
```

(b)

Figure A.1.: Example of raw data from two signals with different resolution

Bibliography

- [1] sklearn oneclasssvm. URL <http://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html>.
- [2] sklearn isolationforest. URL <http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html>.
- [3] sklearn - kneighborsclassifier. URL <http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html>.
- [4] Training set vs. test set vs. validation set - what's the deal? URL <http://www.machinelearningtutorial.net/2017/04/01/training-set-vs-test-set-vs-validation-set-whats-the-deal/>.
- [5] Marco A.F. Pimentel, David A. Clifton, Lei Clifton, and Lionel Tarassenko. A review of novelty detection, 2014.
- [6] Shin Ando, editor. *Clustering needles in a haystack: An information theoretic analysis of minority and outlier detection*, 2007. IEEE. ISBN 0769530184.
- [7] Fabrizio Angiulli and Clara Pizzuti, editors. *Fast outlier detection in high dimensional spaces*, 2002. Springer.
- [8] F. J. Anscombe. Rejection of outliers. *Technometrics*, 2(2):123–146, 1960. doi: 10.1080/00401706.1960.10489888.
- [9] Aymeric Rateau. Mdfreader. URL <https://github.com/ratal/mdfreader>.
- [10] Daniel Barbará, Yi Li, Julia Couto, Jia-Ling Lin, and Sushil Jajodia, editors. *Bootstrapping a data mining intrusion detection system*, 2003. ACM. ISBN 1581136242.
- [11] Paul Barford, Jeffery Kline, David Plonka, and Amos Ron. A signal analysis of network traffic anomalies. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurement*, pages 71–82. ACM, 2002.
- [12] Donald J. Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *KDD workshop*, volume 10, pages 359–370, 1994.
- [13] Shyam Boriah, Varun Chandola, and Vipin Kumar, editors. *Similarity measures for categorical data: A comparative evaluation*, 2008. SIAM.
- [14] Andrew P Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern recognition*, 30(7):1145–1159, 1997.

- [15] Simon Byers and Adrian E. Raftery. Nearest-neighbor clutter removal for estimating features in spatial point processes. *Journal of the American Statistical Association*, 93 (442):577–584, 1998. ISSN 0162-1459.
- [16] Philip K. Chan, Matthew V. Mahoney, and Muhammad H. Arshad. A machine learning approach to anomaly detection.
- [17] Varun Chandola, Shyam Boriah, and Vipin Kumar. Understanding categorical similarity measures for outlier detection. *Technical report 08–008, University of Minnesota*, 2008.
- [18] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3):15:1–15:58, 2009. ISSN 0360-0300. doi: 10.1145/1541880.1541882. URL <http://doi.acm.org/10.1145/1541880.1541882>.
- [19] Sanjay Chawla, Yu Zheng, and Jiafeng Hu. Inferring the root cause in road traffic anomalies. In *Data Mining (ICDM), 2012 IEEE 12th International Conference on*, pages 141–150. IEEE, 2012.
- [20] Deepthi Cheboli. *Anomaly detection of time series*. PhD thesis.
- [21] Bing Cheng and D. Michael Titterton. Neural networks: A review from a statistical perspective. *Statistical science*, pages 2–30, 1994. ISSN 0883-4237.
- [22] Mooi Choo Chuah and Fen Fu. Ecg anomaly detection via time series analysis. In *International Symposium on Parallel and Distributed Processing and Applications*, pages 123–135. Springer, 2007.
- [23] David R. Cox. The regression analysis of binary sequences. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 215–242, 1958. ISSN 0035-9246.
- [24] Dorothy E. Denning. An intrusion-detection model. *IEEE Transactions on software engineering*, (2):222–232, 1987. ISSN 0098-5589.
- [25] Clifton A Ericson. Fault tree analysis. In *System Safety Conference, Orlando, Florida*, pages 1–9, 1999.
- [26] Levent Ertöz, Michael Steinbach, and Vipin Kumar. Finding topics in collections of documents: A shared nearest neighbor approach. In *Clustering and Information Retrieval*, pages 83–103. Springer US, Boston, MA, 2004. ISBN 978-1-4613-0227-8. doi: 10.1007/978-1-4613-0227-8_{\text{underscore}}3. URL https://doi.org/10.1007/978-1-4613-0227-8_3.
- [27] Eleazar Eskin. Anomaly detection over noisy data using learned probability distributions. In *In Proceedings of the International Conference on Machine Learning*, 2000.
- [28] Eleazar Eskin, Andrew Arnold, Michael Prerau, Leonid Portnoy, and Sal Stolfo. A geometric framework for unsupervised anomaly detection: Detecting intrusions in unlabeled data. *Applications of data mining in computer security*, 6:77–102, 2002.

-
- [29] E. Eskinand and S. Stolfo, editors. *Modeling system call for intrusion detection using dynamic window sizes*, 2001.
- [30] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, KDD'96*, pages 226–231. AAAI Press, 1996. URL <http://dl.acm.org/citation.cfm?id=3001460.3001507>.
- [31] Tom Fawcett and Foster Provost, editors. *Activity monitoring: Noticing interesting changes in behavior*, 1999. ACM. ISBN 1581131437.
- [32] Maurizio Filippone and Guido Sanguinetti. Information theoretic novelty detection. *Pattern Recognition*, 43(3):805–814, 2010. ISSN 0031-3203. doi: 10.1016/j.patcog.2009.07.002. URL <http://www.sciencedirect.com/science/article/pii/S0031320309002726>.
- [33] J. D. Gergonne. The application of the method of least squares to the interpolation of sequences. *Historia Mathematica*, 1(4):439–447, 1974. ISSN 0315-0860.
- [34] J. M. N. Gonzalez, J. A. Jimenez, J. C. D. Lopez, and H. A. P. G. Root cause analysis of network failures using machine learning and summarization techniques. *IEEE Communications Magazine*, 55(9):126–131, 2017. ISSN 0163-6804. doi: 10.1109/MCOM.2017.1700066.
- [35] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. Rock: A robust clustering algorithm for categorical attributes. *Inf. Syst.*, 25(5):345–366, 2000. ISSN 0306-4379. doi: 10.1016/S0306-4379(00)00022-3. URL [http://dx.doi.org/10.1016/S0306-4379\(00\)00022-3](http://dx.doi.org/10.1016/S0306-4379(00)00022-3).
- [36] S. E. Guttormsson, R. J. Marks, M. A. El-Sharkawi, and I. Kerszenbaum. Elliptical novelty grouping for on-line short-turn detection of excited running rotors. *IEEE Transactions on Energy Conversion*, 14(1):16–22, 1999. ISSN 0885-8969. doi: 10.1109/60.749142.
- [37] Jiawei Han, Jian Pei, and Micheline Kamber. *Data mining: Concepts and techniques*. Elsevier, 2011. ISBN 0123814804.
- [38] Zengyou He, Xiaofei Xu, and Shengchun Deng. Discovering cluster-based local outliers. *Pattern Recogn. Lett.*, 24(9-10):1641–1650, 2003. ISSN 0167-8655. doi: 10.1016/S0167-8655(03)00003-5. URL [http://dx.doi.org/10.1016/S0167-8655\(03\)00003-5](http://dx.doi.org/10.1016/S0167-8655(03)00003-5).
- [39] Zengyou He, Shengchun Deng, and Xiaofei Xu. An optimization model for outlier detection in categorical data. *Advances in Intelligent Computing*, pages 400–409, 2005.
- [40] Zengyou He, Shengchun Deng, Xiaofei Xu, and Joshua Zhexue Huang, editors. *A fast greedy algorithm for outlier mining*, 2006. Springer.
- [41] Rob J Hyndman and Anne B Koehler. Another look at measures of forecast accuracy. *International journal of forecasting*, 22(4):679–688, 2006.

- [42] Anil K. Jain, M. Narasimha Murty, and Patrick J. Flynn. Data clustering: A review. *ACM Comput. Surv.*, 31(3):264–323, 1999. ISSN 0360-0300.
- [43] Nathalie Japkowicz and Mohak Shah. *Evaluating learning algorithms: A classification perspective*. 2011. ISBN 9780521196000.
- [44] Nathalie Japkowicz and Mohak Shah. *Evaluating learning algorithms: a classification perspective*. Cambridge University Press, 2011.
- [45] Edwin T. Jaynes. Information theory and statistical mechanics. *Physical review*, 106(4): 620, 1957.
- [46] Eamonn Keogh, Stefano Lonardi, and Chotirat Ann Ratanamahatana, editors. *Towards parameter-free data mining*, 2004. ACM. ISBN 1581138881.
- [47] Edwin M. Knorr and Raymond T. Ng, editors. *A unified approach for mining outliers*, 1997. IBM Press.
- [48] Edwin M. Knorr and Raymond T. Ng, editors. *Finding intensional knowledge of distance-based outliers*, volume 99, 1999.
- [49] Edwin M. Knorr, Raymond T. Ng, and Vladimir Tucakov. Distance-based outliers: Algorithms and applications. *The VLDB Journal—The International Journal on Very Large Data Bases*, 8(3-4):237–253, 2000. ISSN 1066-8888.
- [50] Edwin M. Knox and Raymond T. Ng, editors. *Algorithms for mining distancebased outliers in large datasets*, 1998. Citeseer.
- [51] Max Kuhn and Kjell Johnson. *Applied predictive modeling*, volume 810. Springer, 2013.
- [52] Joseph Lee Rodgers and W. Alan Nicewander. Thirteen ways to look at the correlation coefficient. *The American Statistician*, 42(1):59–66, 1988.
- [53] Jessica Lin, Eamonn Keogh, Ada Fu, and Helga van Herle, editors. *Approximations to magic: Finding unusual medical time series*, 2005. IEEE. ISBN 0769523552.
- [54] Chao Liu, Kin Gwn Lore, and Soumik Sarkar. Root-cause analysis for time-series anomalies via spatiotemporal causal graphical modeling. 05 2016.
- [55] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation-based anomaly detection. *ACM Trans. Knowl. Discov. Data*, 6(1):3:1–3:39, 2012. ISSN 1556-4681. doi: 10.1145/2133360.2133363. URL <http://doi.acm.org/10.1145/2133360.2133363>.
- [56] Prasanta Chandra Mahalanobis. On the generalized distance in statistics. *Proceedings of the National Institute of Sciences (Calcutta)*, 2:49–55, 1936.
- [57] Markos Markou and Sameer Singh. Novelty detection: a review - part 1 : statistical approaches.
- [58] L. I. Ming and Paul M. B. Vitányi. Kolmogorov complexity and its applications. *Algorithms and Complexity*, 1:187, 2014. ISSN 00809339.

-
- [59] Harvey Motulsky and Arthur Christopoulos. *Fitting models to biological data using linear and nonlinear regression: A practical guide to curve fitting*. Oxford University Press, 2004. ISBN 0198038348.
- [60] M. Otey, Srinivasan Parthasarathy, Amol Ghoting, G. Li, Sundeep Narravula, and D. Panda, editors. *Towards nic-based intrusion detection*, 2003. ACM. ISBN 1581137370.
- [61] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation.
- [62] Stan Salvador and Philip Chan. Toward accurate dynamic time warping in linear time and space. *Intelligent Data Analysis*, 11(5):561–580, 2007.
- [63] Bernhard Schölkopf, Robert Williamson, Alex Smola, John Shawe-Taylor, and John Platt. Support vector method for novelty detection.
- [64] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948. ISSN 0005-8580. doi: 10.1002/j.1538-7305.1948.tb01338.x.
- [65] Gholamhosein Sheikholeslami, Surojit Chatterjee, and Aidong Zhang. Wavecluster: A multi-resolution clustering approach for very large spatial databases. In *Proceedings of the 24rd International Conference on Very Large Data Bases, VLDB '98*, pages 428–439, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc. ISBN 1-55860-566-5. URL <http://dl.acm.org/citation.cfm?id=645924.671342>.
- [66] Rasheda Smith, Alan Bivens, Mark Embrechts, Chandrika Palagiri, and Boleslaw Szymanski. Clustering approaches for anomaly based intrusion detection. *Proceedings of intelligent engineering systems through artificial neural networks*, pages 579–584, 2002.
- [67] Xiuyao Song, Mingxi Wu, Christopher Jermaine, and Sanjay Ranka. Conditional anomaly detection.
- [68] Florian Stoffel, Fabian Fischer, and Daniel A. Keim. Finding anomalies in time-series using visual correlation for interactive root cause analysis. In *Proceedings of the Tenth Workshop on Visualization for Cyber Security, VizSec '13*, pages 65–72, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2173-0. doi: 10.1145/2517957.2517966. URL <http://doi.acm.org/10.1145/2517957.2517966>.
- [69] The Scipy community. List of distance in scipy. URL <https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.cdist.html#scipy.spatial.distance.cdist>.
- [70] Marc M. van Hulle. Self-organizing maps. In *Handbook of Natural Computing*, pages 585–622. Springer, 2012. ISBN 3540929096.
- [71] Vector Informatik GmbH. Efficient storage of measurement data with mdf format. URL https://vector.com/downloads/mdf_specification.pdf.
- [72] Rui Xu and Donald Wunsch. Survey of clustering algorithms. *IEEE Transactions on neural networks*, 16(3):645–678, 2005. ISSN 1045-9227.

- [73] Dantong Yu, Gholamhosein Sheikholeslami, and Aidong Zhang. Findout: Finding outliers in very large datasets. *Knowl. Inf. Syst.*, 4(4):387–412, 2002. ISSN 0219-1377. doi: 10.1007/s101150200013. URL <http://dx.doi.org/10.1007/s101150200013>.
- [74] Sheng Zhang, Amit Chakrabarti, James Ford, and Fillia Makedon. Attack detection in time series for recommender systems. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 809–814. ACM, 2006.