Master Thesis Business Analytics

Anomaly detection for predicting failures in waterpumping stations

Written By

Vincent Geschiere (2578811)

Supervisers:

René Bekker

Paul Brincker

Ger Koole

Kitting Lee

Tycho van der Scheer

October 13, 2020

Preface

This thesis is written for the final part of the Master's program in Business Analytics at the Vrije Universiteit Amsterdam. This Master's program is a two year multidisciplinary program aimed at improving business processes by using and combining multiple methods based on mathematics, computer science and economics with highly developed quantitative and communication skills. The Master's degree is concluded with a six to eight-month individual internship at a company. My internship took place at the Commerce and Innovation cluster of the Infra Asset Management department of BAM. This cluster aims to find innovative and cost-efficient strategies for infrastructure asset management.

Over the past seven months I have had a great time at BAM and I would like to thank everyone in the Commerce and Innovation team for letting me feel at home. I want to thank Tycho van der Scheer and Kitting Lee, my supervisors at BAM, for giving me advice on how to tackle certain problems and bringing me into contact with my colleagues. I would also like to thank Paul Brincker for providing me with the data used for my the practical study and answering all my questions regarding the data.

From the VU, I would like to thank Ger Koole, my first supervisor, for thinking along with my research by giving me advice on how to continue and what techniques to use. I would also like to thank both Ger and my second reader René Bekker for giving feedback on my thesis.

Executive summary

This thesis provides a comparative study of multiple unsupervised learning methods aimed at detecting outliers in order to predict and explain upcoming malfunctions. We first perform a literature study on different types of unsupervised outlier detection algorithms and how to explain their results using Explainable AI techniques. Explaining machine learning results can be difficult since machine learning models are usually so called 'black-box' models which can make accurate predictions, but are not able to explain their predictions. Afterwards, these unsupervised learning methods are tested and compared on sensor-data collected from one of the water pumps stationed at the new water pumping station at Eefde.

The main research question this thesis will answer is therefore as follows: *"Which unsupervised learning methods should be used in order to find outliers in sensor-data which indicate whether pro-active maintenance is required for a pumping station and is able to explain why the outliers are classified as such?"*

The algorithms applied to this dataset are nearest neighbor, 3-nearest neighbors with mean values of the distances, 3-nearest neighbors with median values of the distances, Local Outlier Factor, Isolation Forest and autoencoder. Isolation Forest turned out to bring the best results in predicting upcoming failures. Its results could also be explained more clearly than the results of the other algorithms by using the algorithm called Tree SHAP.

The total downtime caused by the malfunctions the algorithms tried to predict was 16.75 hours. Isolation Forest was able to predict two malfunctions which caused 0 and 1 hour of downtime respectively. This means that 6.07% of the downtime could have been prevented by implementing Isolation Forest. This might not sound like much, but considering the fact that BAM can receive fines of up to 10,000€ per 15 minutes of downtime, this can still result in a major cut in costs.

The results given by Isolation Forest are explained by using Tree SHAP. This showed that the values for the vibration speeds of the top bearings and the (differences between the) values of the electricity currents of the three phases of the motor are the most important features for predicting upcoming failures.

However, the results can definitely be improved. We therefore give two recommendations. The first is to collect data of more parts of the installation. Especially data of the rinse water pump, since this pump was responsible for 71.64% of the downtime. The most important features to measure are the bearing vibration speeds and the used electricity flow, since SHAP proved that these attributes are the most important outlier indicators.

We also recommend increasing the measuring frequency. Currently, all datapoints are measured once per minute. However, some attributes, especially the vibration speeds of the bearings, can differ significantly within smaller time periods, sometimes even within a few milliseconds. Increasing this measurement frequency could potentially provide a significant increase in the accuracy of the model. Future research could determine whether these extra features and increased measurement frequencies will improve the results or not.

Contents

1 introduction

Big construction companies like BAM manage a significant amount of different assets. Assets like bridges, water pumping stations, docks and locks have to be operational as often as possible. If an asset is malfunctioning, BAM can receive hefty fines. For example, if a water pumping station is not operating when it should, it can result in a fine of 10,000€ per 15 minutes of downtime. These assets therefore have to be maintained so that they will not malfunction when they need to operate. According to Cheng et al. (2020), there are three different asset management strategies.

The first is reactive maintenance, where a component is repaired or replaced after it malfunctions. This can be very expensive due to the production failure and inefficiently planned maintenance. The second strategy is preventive maintenance, where staff inspects or replaces components at predetermined periods of time (Zhao et al. (2010b)). This, however, is only useful if there is a strong age-dependent factor present, otherwise it can easily result into performing maintenance too often or not often enough (Beebe (2004)). The third and final strategy is condition-based maintenance (CbM), or predictive maintenance (PdM), which aims to detect trends of component conditions using historical data in order to predict upcoming failures so actions can be taken before a component starts malfunctioning (Thyago et al. (2019); Mobley (2002); Zhao et al. (2010b)).

However, it is also argued that condition-based and predictive maintenance are two different strategies (Tiddens et al. (2018)). Namely that condition-based maintenance only looks at the current condition of a component, while predictive maintenance also takes prognostic information into account. This means that predictive maintenance is a step further than condition-based maintenance. Since both maintenance strategies are defined differently in different papers and there is not one clear definition for both policies, this thesis will consider both terms to be the same. This is done because, to our knowledge, the majority of the papers have defined PdM and CbM as the same strategy (Yuan et al. (2013); Zhao et al. (2010a); Thyago et al. (2019); Mobley (2002); Zhao et al. (2010b)).

Generally speaking, there are two different classes of machine learning: supervised and unsupervised learning. Supervised learning contains a labeled dataset. These labels can be either categorical (for classification models), or numerical (for regression models) values. An unsupervised learning algorithm, on the other hand, has unknown labels (Moleda et al. (2020)).

Within the field of PdM, supervised learning is successfully used to classify whether an installation is in a healthy condition or not. This is done by training the algorithm on data of an installation in different conditions with labels corresponding to that condition. For example, Biswal and Sabareesh (2015) developed a model used to classify the operational conditions of a wind turbine. They did this by collecting vibration data of the wind turbine in healthy condition and in a deteriorated condition (by replacing a healthy component with a defective component). This paper used an artificial neural network to

train on this data so that it could classify new data into either a healthy or a deteriorated condition. Various other Artificial Intelligence techniques, such as Random Forest and Support Vector Machine have been used for similar research (Thyago et al. (2019)).

Unsupervised learning methods, on the other hand, mostly work as outlier detection algorithms (Moleda et al. (2020)). An outlier, also called an anomaly, is a datapoint which deviates a lot from the other datapoints. An often used definition for outliers is the definition of Hawkins (1980): "Observation which deviates so much from other observations as to arouse suspicion it was generated by a different mechanism". These outliers can then indicate an upcoming failure.

The data provided only shows when a failure of the pump occurred, but not when the component(s) became defective since this is usually unknown. Therefore it is not possible to give labels to the data indicating when the installation is in a healthy or in a deteriorated condition. For this reason we decided to only test unsupervised learning methods for outlier detection.

The purpose of this thesis is to investigate whether PdM can be applied to the water pumping station located at lock Eefde. These water pumps aim to keep the water level of the Twentekanaal in between 9.85 mNAP and 10.1 mNAP by pumping water from the IJssel river to the Twentekanaal. BAM collects data from these water pumps using multiple sensors including temperature, vibrations and pressure. Another important aspect for the BAM is the ability to explain why maintenance is necessary. This can help with finding the cause of the failure, increasing trustworthiness from the user and providing clarity on what data is important for predicting failures. It is therefore necessary to be able to both predict and explain upcoming failures.

The main research question of this master thesis is therefore formulated as follows: *"Which unsupervised learning methods should be used in order to find outliers in sensor-data which indicate whether pro-active maintenance is required for a pumping station and is able to explain why the outliers are classified as such?"* Our hypothesis is that Isolation Forest will give the best results since tree-based algorithms often give better prediction than Neural Network based algorithms with tabular-style data (Chen and Guestrin (2016)) and the data used for this research has a tabular-style. Tabular-style data is structured data which can be stored in a table. A matrix containing numbers, for example, is tabular-style data, while a dataset containing images is not. On top of that, there is a very useful explainable AI technique for tree-based algorithms which is able to explain which features cause a datapoint to be classified as an outlier or not: Tree SHAP (Lundberg et al. (2020)). For most other algorithms based on distance, however, there are no useful Explainable AI algorithms yet. So, since tree-based algorithms usually give good results with tabular-style data and there exists a good algorithm to explain the given results, we expect Isolation Forest to give the best results.

According to Aggarwal (2016), there are three different types of approaches for anomaly detection: statistical, proximity based, and deviation based. Statistical anomaly detection assumes that the data follows a certain distribution and then calculates the probability of anomalousness for each datapoint. Prox-

imity based outlier detection assumes that anomalous datapoints are far away from non-anomalous datapoints. This modeling can be done in three ways: distance based, density based and clustering based. The anomaly score is then based on either the distance to other points, the density w.r.t. other points or whether the point belongs to a (big enough) cluster respectively. The final and third method, deviation based anomaly detection, works by reconstructing data based on previously trained-on data and calculates the difference between the actual and reconstructed data. The higher the difference, the more anomalous the datapoint.

Aggarwal (2016) also mentions that statistical anomaly detection methods were primarily used when computers were not yet available. Many modern-day used techniques are based on the foundations of statistical based techniques, but the techniques themselves are mostly outdated. Therefore, this thesis will only test proximity based and deviation based methods.

The structure of this thesis is as follows: Chapter 2 will provide information on the given dataset, the clean-up process, which and why features were added and removed, the analysis of the dataset and finally which failures/malfunctions of the pump the models tried to predict.

Chapter 3 will explain the concepts behind Explainable AI. It will explain how one of the currently most used Explainable AI techniques, SHAP, works and how it is used in practice.

The next chapter, chapter 4, will go into more details on proximity based anomaly detection. It will provide information on a common problem within proximity based anomaly detection, the curse of dimensionality, and different distance measurements. Afterwards, it will explain how multiple algorithms work and in the final section the results of these algorithms are shown and analyzed.

Afterwards, in chapter 5, one of the most commonly used deviation based anomaly detection algorithms is explained: the autoencoder. Afterwards, the results calculated by the autoencoder are presented.

Chapter 6, the last chapter, will give a final conclusion and discussion in which the main research question will be answered and suggestions for further research will be provided.

2 Dataset

This chapter will provide more details about the dataset. Pumping station Eefde consists of 4 pumps: two old and two new pumps. This dataset is produced from one of the two new pumps.

The dataset contained a total of 28 attributes. Table 9 in Appendix B gives an overview of all attributes, what they mean and the measurement unit. Every attribute contains a value for each minute from 1/10/2018 00:00 up until 29/9/2019 23:59. Roughly speaking, the attributes consist of the time when it was measured, the water heights, statuses of different pumps, frequency, rpm, temperatures, vibration speed, (total) energy usage, total running hours, electricity current, current water flow rate and whether the vacuum pump is active or not.

In the following section, section 2.1, the cleaning process of the data is explained. In the next section, section 2.2, motivations for adding and removing features are given. Afterwards, section 2.3 will provide information about the data analysis aimed at finding correlation within the data. The final section, section 2.4, will explain which failures of the water pumping station the model will try to predict and why.

2.1 Quality

The data is overall of a relatively high quality. There were, however, quite a few missing values in the dataset and sometimes there were values which did not make sense. Both the missing and false values had to be imputed before the anomaly detection approaches could be used. First, section 2.1.1 will show how the missing values were imputed. Afterwards, section 2.1.2 will do the same, but then for the false values.

2.1.1 Imputing missing values

As shown in table 10 in Appendix B, the amount of missing values can be substantial. The percentage of the amount of missing values is sometimes even 17% (the total amount of observations is 507030). Luckily, imputing the missing values was relatively easy.

Missing values usually occurred randomly. Because of this, it rarely happened that missing values occurred for two or more consecutive minutes. It was usually the case that when a value for one point in time was missing, it was not missing in multiple consecutive points in time (e.g. multiple days) before and after that missing value. Since each value is produced every minute, these values could be interpolated, which means that the missing value is imputed by the average of the value of 1 minute before and after that missing value. For example, at 2-10-2018 10:01, the value for 'temp_topbearing' was missing. The values for this attribute at both one minute before and after were 55.13. The missing value at 2-10-2018 10:01 will then be imputed with the average of these values, which is again 55.13.

It might be possible that the temperature was a little bit higher or lower than estimated by the interpolation, but because the datapoints are given per minute, the difference should be so small that it is considered insignificant. This technique was applicable for all these random cases. Sometimes, however, values were missing for two or three consecutive minutes. In these cases we still decided to interpolate these missing values, since the difference between the actual and the interpolated value should then still be insignificantly small.

It could also occur that values would be missing for one or more consecutive hours. In this case, interpolating could give unreliable results. However, this was only the case when a pump was inactive. This could be seen because the total running time of the pump stayed the same while the other values were missing for multiple consecutive hours. These missing values were in these cases again easy to guess. Namely, their value would always be 0, or equal to the previous value if it was a total value. In the first case, for example, the flow rate would occasionally have long periods of missing values, but if the pump is inactive then the flow rate is automatically 0. The second case was applicable for the total energy usage and total amount of water pumped, since energy usage and flow rate are both 0 when the pump is inactive and thus the total amount stays the same. The temperature and vibration speed values were never missing for long consecutive time periods and thus could be interpolated.

2.1.2 Imputing false values

Imputing false values, however, could be a bit harder and should be done with caution. This is because it can be hard to see whether a value is false, or whether it is an outlier. Since the purpose of this research is detecting outliers, we should be really careful not to impute these outliers, since if they are imputed with regular values it is impossible to detect them.

The first way of detecting whether an attribute contains false and or outlier values is by plotting a boxplot. The boxplots which immediately stood out were the boxplots containing the 'flow_l1', 'flow_l2', 'flow_l3', 'flow_3f' and 'energy' attributes (in Dutch these attributes are named 'stroom_l1', 'stroom_l2', 'stroom_l3', 'stroom_3f' and 'energie'). These boxplots are shown in figures 1 and 2.

As can be seen, all attributes in figure 1 suddenly have values higher than 5000. An Ampere value of 5000 or higher is impossible, since the motor is not capable of delivering such a high electricity current. From this we can conclude that these values are definitely wrong. The values shown in figure 2 are also clearly wrong. The total energy values should be counted up each minute, while at some time-periods the value increased by more than 10,000 in one minute (often while the pump was inactive) and then decreases again to the value it was before the sudden spike. This implies that the sudden spike must be a false value. According to multiple domain experts, these high values are probably so-called 'dummy' values, meaning they are extremely high on purpose when the system does not receive any data. The reason for making these high values is so that it can easily be seen that these values are incorrect.

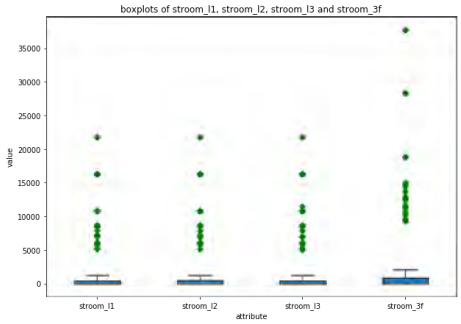


Figure 1: 'Flow' attributes before cleaning.

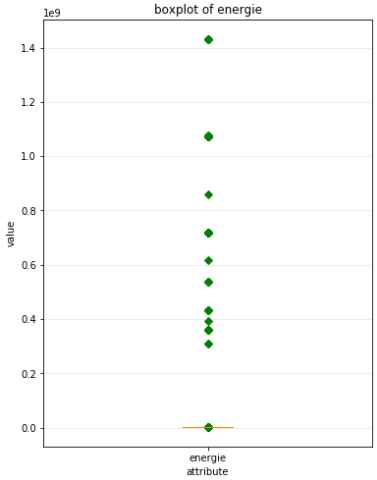


Figure 2: Energy attribute before cleaning.

When these high values occurred for the 'flow_l1', 'flow_l2' and 'flow_l3' attributes while the pump was inactive, these values would be imputed by 0, since the electricity usage is 0 when the pump is inactive. Otherwise, the values were interpolated. The value for 'flow_3f' is then recalculated by summing up the values of 'flow_l1', 'flow_l2' and 'flow_l3' and dividing this sum by $\sqrt{3}$. This formula gives the total electricity current in a three-phase electric power system.

For the total energy attribute the false values would also be imputed by interpolating. However, when the pump was inactive it would not be replaced by 0, but by the previous normal value, since it is a value which does not change when no electricity is used. The boxplots for the previously mentioned attributes after this cleaning process are shown in figures 3 and 4 respectively. As can be seen in figure 4, there are still multiple outliers present in the 'energy' attribute. However, those outliers make sense since it is a cumulative attribute and so the last few values are considered very high compared to the values at the beginning.

There were also attributes with strange outliers which at first also seemed like false values but were, in fact, actually correct values. An example of this is shown in figure 5. As can be seen, there is one extreme outlier at the top. At first, it can be assumed that this is a false value, since the other values do not lead up toward this peak. However, this is still a correct value. This value is, namely, caused by an external factor. There were constructions happening outside of the building at the time. During these constructions the constructions workers had to dig into the ground which caused these vibrations.

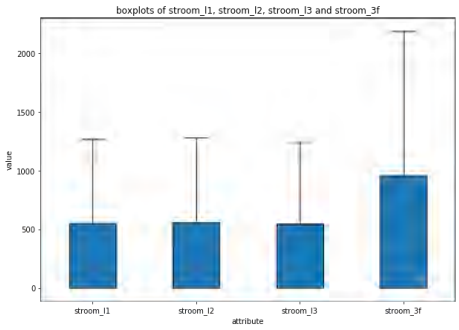


Figure 3: 'Flow' attributes after cleaning.

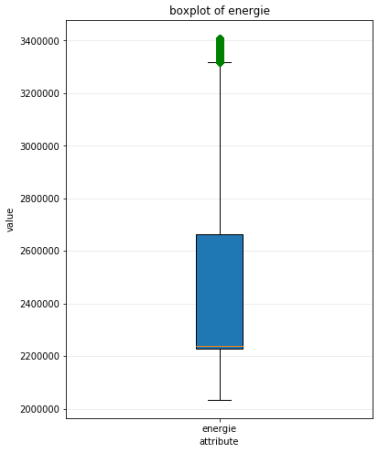


Figure 4: 'Energy' attribute after cleaning.

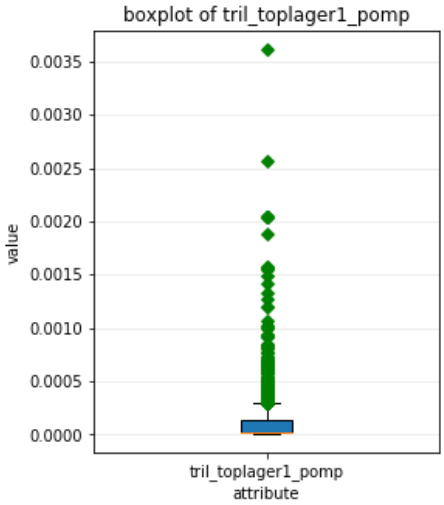


Figure 5: Boxplot of vib_topbearing1_pump with an extreme outlier.

2.2 Removing and adding features

In order to improve the accuracy of predictive machine learning algorithms, multiple features were added to and removed from the data. Removing features can sometimes actually improve the accuracy, since too much unnecessary attributes can cause overfitting. Overfitting is "the production of an analysis that corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably" (Leinweber (2007)).

Having too many unnecessary features can cause your data to find correlations which are in your data by coincidence but do not apply to other datasets. This will cause the model to make incorrect predictions on the datasets it has not trained on.

At first, the attributes 'stat_vacvalve_vp', 'stat_vacvalve_pump', 'stat_pres', 'stat_pump' and 'stat_vcpump' were dropped. There were two reasons for doing this. The first reason is the relatively high amount of missing values. There were multiple instances where the values were missing for one or more consecutive days, making it a lot harder to estimate what the values should have been. The second reason is that none of the domain experts know exactly what every code means. These codes were created by external programmers and therefore none of the domain experts are 100% familiar with these codes. Summarizing, it is impossible to precisely know what these codes mean and how to fill in the missing data and therefore it is decided to drop these attributes.

Another attribute which was dropped is 'vacuum_active'. This is because the value is always 'False'. The vacuum pump is rarely active and has never been active in the time period of the provided dataset. Thus, it is an unnecessary attribute for this research and will only result in an increase of the computational effort.

We also decided to drop all the cumulative attributes, which were the total hours of running time, total energy usage and total amount of water pumped. The malfunctions we tried to predict as shown in table 1 consisted of different causes and different malfunctioning parts of the pumping station. We also did not know how long these parts had ran since the last repair or replacement. Therefore, some parts could be a lot older than others, meaning that those cumulative values are not trustworthy enough to predict upcoming failures for all these different parts. For example: If the pump has a total running time of 6000 hours and the top bearing of the pump starts to malfunction, we cannot know whether this is caused because of the total running time or not. We do not know when the latest top bearing was placed or repaired and how long the pump has been active since its placement. Therefore, the total running time of the top bearing of the pump might actually 'only' be 100 hours and thus these features were considered to be untrustworthy indicators for upcoming malfunctions.

There are also features added to the dataset. The first added feature was a boolean feature indicating whether the pump was active or not. The activity of the pump was derived from whether or not the pump was using a lot of electricity (more than 50 Ampere). The benchmark was set at this value since an active pump requires hundreds of Amperes to run. This feature was added because a high activity can cause more malfunctions.

The differences between the 'flow_l1', 'flow_l2' and 'flow_l3' attribute values were also added as features. This is done because, according to domain experts, a high difference between the electricity current of the three phases of a three phase power can also indicate a malfunction since they should, in theory, be equal to each other. The total difference of these attributes, so the sum of these three differences, is also added as a feature. The difference with flow_3f is not included, because this value is calculated by dividing the sum of the electricity

currents of the three phases and dividing this sum by $\sqrt{3}$.

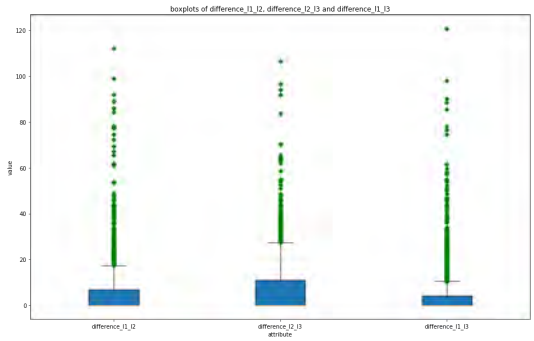


Figure 6: Boxplots of the differences between the 'flow' values.

The boxplots of the three differences are shown in figure 6. As can be seen, the differences can turn out to be really high. This is usually the case when the pump is either just activated or when it is in the process of shutting down since both the activation and the shutting down of the pump take approximately two minutes. In the other cases it might be an indication of a failure. We later also decided to remove the datapoints where the pump was either just activated or in the process of shutting down. This was done because these points were classified as outliers by multiple algorithms. This is because the electricity current for each phase is then lower than when to pump is fully active, but other values, for example the current water flow, are already given their regular values.

For example, if the pump required an electricity current of 600 Amperes per phase in order to pump 5 cubic meters of water per second, a datapoint which was measured just after the pump was activated could give a value of 200 Amperes for 5 cubic meters per second. Since the electricity current is relatively low it could be considered an outlier, while this is actually not the case, since the pump is in the process of starting up. These values should therefore not indicate an upcoming failure and would then be classified as False Positives (as explained in more detail in section 2.4).

Another feature we added was the water pumping height, which is the difference between the water levels of the Twentekanaal and the IJssel. The pumping height can have a significant impact on the electricity usage of a pump. Section 2.3 shows this and explains why this is important for failure prediction since the electricity usage is an important predictor for failures.

Afterwards, we removed the water height features from the dataset. The reason for doing this is that the pump is not allowed to pump water when the water level of the IJssel is below 2.5 mNAP. This is because of an increased risk of cavitation. Cavitation is a phenomenon where in a low pressure environment (in this case the entrance to the pump), air bubbles are created due to a lower evaporation temperature. When these air bubbles move to an environment with a higher pressure (in this case when the bubbles get pumped upwards), they

implode which can cause significant damage to the inside of the pump. Since the pump is not active when the water level of the IJssel is low and a high IJssel water level is not able to cause damage, this feature was omitted.

The height of the water level of the Twentekanaal is also not an indication of a malfunction. The only way it could indicate a malfunction has occurred is when the water level is too low, meaning that no water or not enough water is being pumped into the canal due to a shortage of pumps. However, this will only happen when the malfunction has already occurred and it is therefore already too late for the prediction. A high water level will also not indicate a failure since the only way this can be caused by the water pumping station is when the pumps have been active for too long (meaning they are running and not malfunctioning) and need to be shut down.

The final two features that were added were not based on the data provided by BAM, but from external sources. According to a domain expert, both the water temperature and the outside temperature could influence the state of the water pump. The data of the water temperature was collected from Rijkswaterstaat, the Dutch ministry of infrastructure and water management. The dataset provided the water temperature in the river IJssel for every 10 minutes and is publicly available. Since the water temperature cannot fluctuate significantly within 10 minutes, it was decided to interpolate in order to impute the missing values in between these 10 minutes.

The data of the outside temperature was collected from the KNMI, the Royal Netherlands Meteorological Institute. The KNMI has multiple weather stations throughout The Netherlands which record data every hour and is also publicly available. The weather station we used was located at Deelen. This weather station is the closest weather station to the water pumping stations at Eefde with a distance of approximately 28 km. It is assumed that the temperature difference at this distance is small enough in order for the data from the weather station at Deelen to be trustworthy. Since the dataset provides datapoints for every hour there are again multiple missing values. However, it is assumed that the temperature will not fluctuate significantly within one hour and therefore the missing values were also imputed by interpolation.

We also decided to add the cosine and sine functions of the month, day, hour and minute. Time values are cyclic values and therefore it is useful to transform these values using equations (1) and (2), where x is the value of the month, day, hour or minute. Equations (1) and (2) always output a value $y \in [-1, 1]$ on the x-plane and y-plane respectively. This way, the algorithm 'knows', for example, that the month December with value 12 is actually very close to the month January with value 1. An illustration of this for the months is shown in figure 7. This process is called a Fourier Transform (Shatkay (1995)).

$$\sin(2 * \pi * x) \tag{1}$$

$$\cos(2 * \pi * x) \tag{2}$$

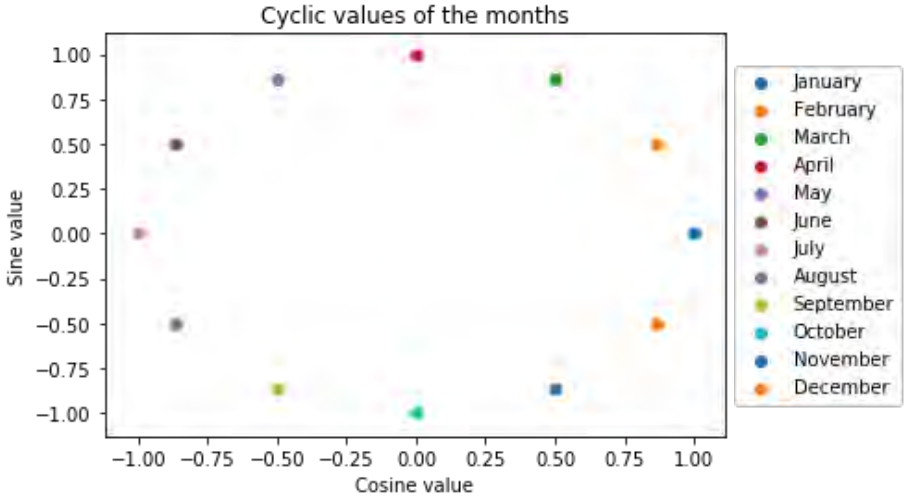


Figure 7: Illustration of the months plotted as cyclic values.

This was done in order to find out whether certain times of the year have more malfunctions than other times of the year. This was indeed the case. However, that is actually very straightforward since the period of May the first up until the 30th of October is called the 'pumping season'. During this period of time the water level of the Twentekanaal is often low due to less rain and therefore the pumps have to be more active during this season than outside of this season. Since the pumps are more active, there exist more malfunctions during this 'pumping season'. It is therefore not trustworthy to add these features to the model and so they were later removed from the dataset and not used for the final models.

The final dataset consisted of a total of 24 attributes and 505479 rows. Except for the 'minute' attribute, all these attributes and rows were used for the models.

2.3 Data Analysis

According to the domain experts of water pumping station Eefde, the electricity flow is a very important indicator for upcoming failures. This is because if a pump requires more power for the same task than usual, it might indicate something is wrong. There could, for example, be wear and tear on the pump or there might be a lot of dirt inside the pump. For this reason, we tried to find features which can be used as good predictors for the electricity flow. This section will go into more details on this.

The first feature we looked at was the flow rate of the water. We plotted the water flow rate against the 'flow_3f' attribute. This plot can be seen in figure 8. This plot only contains data of the periods where the pump was active, since

both values are 0 when the pump is inactive. This plot indicates that there is a somewhat linear relationship between the two attributes. There are also many datapoints where the flow rate is 0, but the value for 'flow_3f' is not. This is mostly the case when the pump is either starting up or shutting down. During these processes electricity starts flowing, but the pumps are not yet pumping (a lot of) water. There is, however, still a considerable amount of variation so we concluded that there should be more features which (combined) could help predict the electricity flow more accurately.

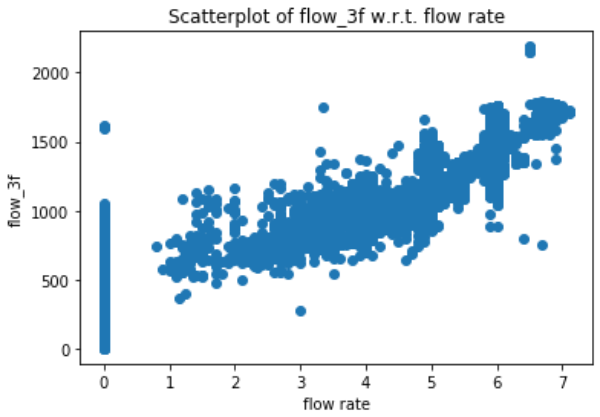


Figure 8: Plot of 'flow_3f' against the water flow rate.

We therefore also looked at the rotational frequency. We used the same kind of plot as shown in figure 8. This plot is shown in figure 9. This graph shows, with the exception of a few outliers, an almost linear relation between the two attributes. This makes sense, since a higher rotational frequency of the motor will produce a higher electricity flow. One might consider not using this feature since a high electricity flow is caused by a high rotational frequency and not vice versa. However, if there is an instance where the rotational frequency is low and the electricity flow is high, then it might still indicate that something is wrong and thus we decided to include it into the models.

As explained in the previous section, we also included the height at which the water had to be pumped, so the difference between 'level_in' and 'level_out'. When plotting 'flow_3f' against this difference we get the graph shown in figure 10. The top seems to indicate that there might be a small correlation, but there is still much randomness and variation. Therefore, we decided to filter out the datapoints where the pump is starting up and shutting down. This gives the plot shown in figure 11. This does remove a lot of the randomness, but it is still not completely clear whether there exists a good correlation between the two features or not.

We therefore took the average values of 'flow_3f' and the water pumping height for each time the pump was active without the beginning and ending sequences. We plotted these values (in total 246 datapoints) against each other

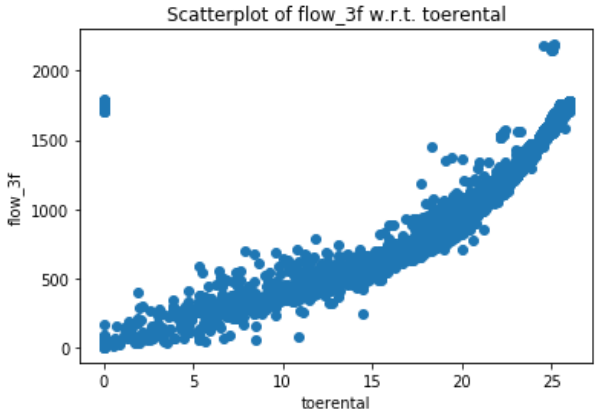


Figure 9: Plot of 'flow_3f' against the rotational frequency.

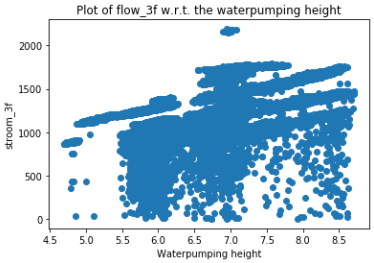


Figure 10: Plot of 'flow_3f' against water pumping height with beginning and ending sequence.

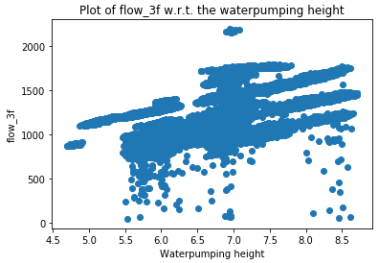


Figure 11: Plot of 'flow_3f' against water pumping height without beginning and ending sequence.

as shown in figure 12. This figure still has some similarities with the top parts of figures 10 and 11. However, a lot of noise has been reduced. It can also be seen that there is at least a small correlation between the two features since it appears to show 'lines' in the data.

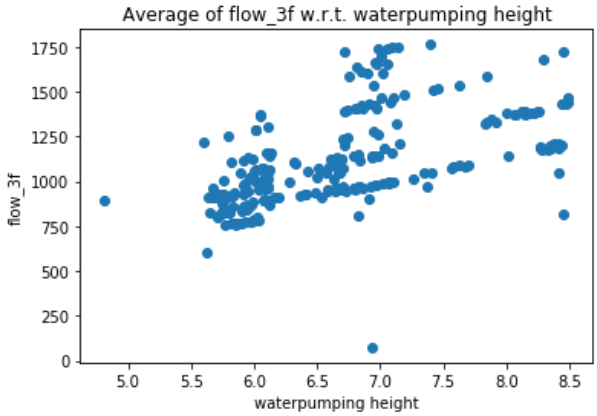


Figure 12: Plot of the average values of 'flow_3f' against the average values of the water pumping height for each time the pump was active.

We theorized that these lines were caused by the water flow rate since it would make sense that a high water flow rate and water pumping height combined will require a lot of power and vice versa. We therefore plotted the average values of the water flow rate and water pumping height for each time the pump was active without the beginning and ending sequences. This plot is shown in figure 13. This plot also shows these 'lines'.

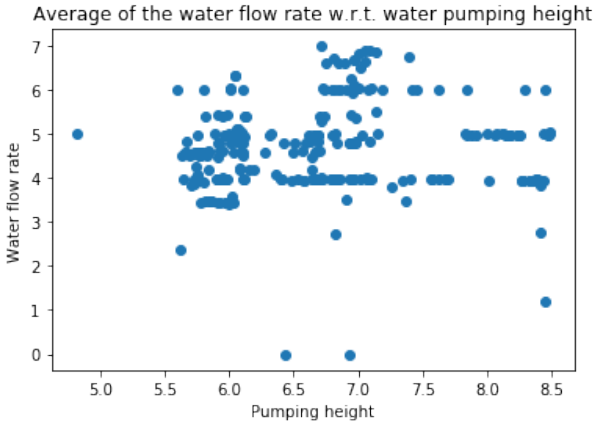


Figure 13: Plot of the average values of the water flow rate against the average values of the water pumping height for each time the pump was active.

We therefore plotted the same averages of the values for all three variables mentioned above in one 3D plot which can be seen in figure 14. The 'lines' are again visible as relatively straight diagonal lines. We also changed the angle of the plot as shown in figures 15 and 16. Figure 15 shows that the lines correspond

to the water flow rate and figure 16 shows how the energy usage increases on each line with an increasing water pumping height. These plots confirmed our theory that the lines in figure 12 are caused by the water flow rate. The figures also show that combining high values for the water pumping height and flow rate results in a high energy usage.

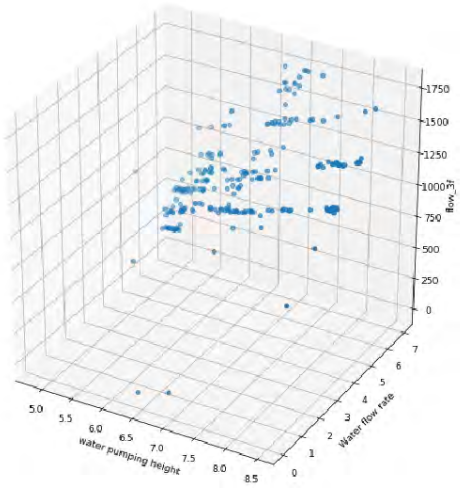


Figure 14: 3D plot of the average values of the water flow rate against the average values of the water pumping height and 'flow_3f' for each time the pump was active.

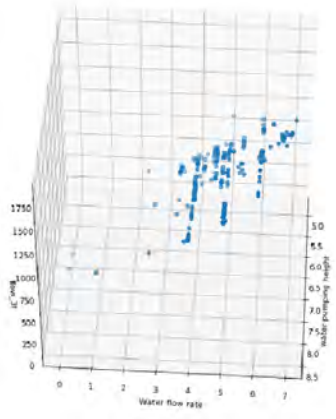


Figure 15: 3D plot angle showing the 'lines' on the water flow rate.

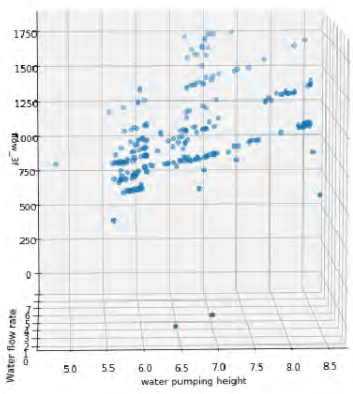


Figure 16: 3D plot angle showing the increase in energy usage.

2.4 Malfunctions and target variable

Next to the dataset described in the previous sections, BAM also provided us with a dataset containing all malfunctions that happened at water pumping station Eefde in the period of 01/10/2018 up until 30/09/2019. Many malfunctions occurred at the water pumping station during that time period, but some of them were more important than others. There were also malfunctions which were impossible to predict up front. We therefore decided to make a selection of malfunctions to predict based on their importance and predictability.

The importance is based on the amount of downtime occurring as a consequence of this malfunction. Some malfunctions did not cause any downtime. An example of these malfunctions are malfunctions to the scumboard cleaner. The scumboard is a board which blocks the waste in the water from entering the pump. The waste could otherwise cause significant damage to the pump from the inside. If the scumboard cleaner is not operational, the pump is still able to pump water from the IJssel to the Twentekanaal. Therefore, a malfunctioning scumboard cleaner will not immediately cause downtime. The cleaner does have to be repaired after a while, however, since the pumps can become inoperative if the scumboard does not get cleaned in time. This is because at one point the amount of waste will block the water and it can therefore not reach the pumps. It therefore does have to be repaired in time, but since this can take multiple days it is not considered to be an urgent malfunction.

The predictability of a malfunction was based on the description of that malfunction. An example of a malfunction which was unpredictable occurred at 15/08/2019, where a communication error occurred. The cause of this communication error was a loose UTP cable. This model is not capable of predicting loose cables and therefore this malfunction was not included. Another example of an unpredictable malfunction occurred at 11/09/2019, where the value for vibr_topbearing2_motor was incredibly high with a value of 0.016 mm/s while the pump was inactive. This value is then usually 0. This high value occurred due to construction happening outside of the water pumping station. This is the same external factor which was mentioned previously in section 2.1.2. Since this malfunction occurred due to an external factor this cannot be predicted by the machine learning models. The malfunctions which are included into the model are shown in table 1 with the corresponding date of occurrence and caused downtime.

The calculated downtime is based on the information given in the dataset and the time the pump was inactive. As can be seen, the first malfunction has a downtime of 0. This is because it is a malfunctioning scumboard cleaner. The reason for still including this malfunction is that the best algorithm was actually able to predict this malfunction without having data of the scumboard and scumboard cleaner. We were therefore interested in finding out how the model was still able to predict this malfunction and thus we decided to include it. There are also many malfunctions with an unknown cause and we were interested in seeing whether the algorithm could predict these malfunctions and maybe explain their cause. Therefore these failures were also included.

Date	Cause	Downtime in hours
15/10/2018 10:36	Malfunctioning scumboard cleaner	0
03/01/2018 12:14	UPS failure	1
16/01/2019 16:02	Rinse water pump[1]melted	10[2]
01/06/2019 22:23	Unknown	0.5
02/07/2019 15:07	Unknown	1
03/07/2019 12:09	Unknown	0.5
22/07/2019 22:03	Unknown	1.5
02/08/2019 15:39	No pressure in the Rinse water pump due to extreme pollution	2[2]
18/09/2019 13:00	Unknown	0.25

1 A Rinse water pump is a pump with cold water used to cool down specific parts of the water pumps.

2 The pump did not have to be active when this malfunction occurred. However, if it did have to be active the downtime would be the given value.

Table 1: Table of all included malfunctions.

The goal of the model is then to predict an upcoming failure as early as possible through outlier detection. We decided to test if the model could predict these failures three days up front since three days provides enough time to prepare for and prevent the malfunction and more than three days seemed untrustworthy. We did this by trying to predict outliers in the data three days before a failure occurred. Therefore, all datapoints occurring 3 days before one of the failures mentioned above are considered to be outliers. If the algorithm classifies a datapoint in this period as an outlier, it is called a True Positive (TP). If this datapoint is not classified as an outlier by the model, it is called a False Negative (FN). A point correctly not classified as an outlier - meaning it was not present in the time period of three days before a malfunction and is classified as an inlier - is called a True Negative (TN). Finally, points that are classified as outliers while not occurring three days before a malfunction are called False Positives (FPs).

3 Explainable AI

The downside of many of the techniques used in this thesis is that the techniques contain a 'black box', meaning their results cannot be explained. The models can predict whether a point is an outlier or not, but they cannot explain why this point is an outlier. This is a common problem within the AI field. An explainable model can ensure a users trust, provide insight into how a model may be debugged and help with understanding how the process is being modeled. Multiple models have been developed to tackle this problem (Lundberg and Lee (2016); Ribeiro et al. (2016); Shrikumar et al. (2016)). This thesis will make use of the SHAP (SHapley Additive exPlanations) method, first proposed by Lundberg and Lee (2017). This method is chosen since it has its own useful Python package and there have been multiple publications using SHAP where it is considered to be one of the most prominent Explainable AI techniques (Antwarg et al. (2019); Rathi (2019); Mokhtari et al. (2019)). The sections 3.1, 3.2 and 3.3 will explain how SHAP works. All the information given in these section originates from the original paper introducing SHAP (Lundberg and Lee (2017)). Afterwards, section 3.4 explains how difficult it still is to explain distance based and density based techniques and why no explaining models are used for these techniques in this thesis.

3.1 Additive feature attribution methods

SHAP assigns an importance value to each feature for a single prediction. It is therefore called an additive feature attribution method. In order to understand how SHAP works, we first have to define additive feature attribution methods. An additive feature attribution method is a method which has an explanation model that is a linear function of binary variables as shown in equation (3), where $z' \in \{0,1\}^M$, M equals the number of simplified input features and $\phi_i \in \mathbb{R}$.

A simplified input feature is a feature in which the value is simplified. A simplified value x' always has mapping function $h_x(\cdot)$ such that $h_x(x') = x$, where x is the original and not simplified input. Local methods always try to ensure that $g(z') \approx f(h_x(z'))$ if $z' \approx x'$, where f is the prediction model and g is the explanation model.

This means that for every simplified input x', there is a mapping function which can derive the original values before they were simplified. Local methods then try to find a single explanation model g which gives approximately the same output as the explanation model f if all inputs are similar to each other and use the same mapping function. Equation (3) shows how an explanation model g calculates its output. Equation (3) focuses on a local explanation, meaning it is designed to explain a prediction $f(x)$ based on a single input x.

$$g(z') = \phi_0 + \sum_{i=1}^{M} \phi_i z'_i \tag{3}$$

ϕ_0 is the base value, meaning that ϕ_0 is the value the model g would expect

to be predicted by f if the input contained zero features. ϕ_i is the effect of adding feature i to the prediction. The sum of the effects of all the added features to the base value ϕ_0 is then the value the model g would expect to be predicted by f when all features are known.

3.1.1 Properties of additive feature attributions

Additive feature attributions have three desirable properties. The first property is known as local accuracy. The second is known as missingness and the final property is known as consistency.

Local accuracy implies that when the original model f is approximated for an input x, it requires the explanation model to match the output of f for a simplified input x'. Equation (4) shows the local accuracy property. This formula means that when $\phi_0 = f(h_x(\mathbf{0}))$ represents the model output where all simplified inputs are missing, the explanation model, $g(x')$, is equal to the original model $f(x)$ if $x = h_x(x')$.

$$f(x) = g(x') = \phi_0 + \sum_{i=1}^{M} \phi_i x_i^{'} \tag{4}$$

The missingness property is shown in equation (5). The missingness property requires that when the simplified inputs represent feature presence, the missing features should not have any impact on the result.

$$x_i^{'} = 0 \implies \phi_i = 0 \tag{5}$$

The third and final property, consistency, requires that if a model changes in such a way that a simplified input's contribution either stays equal or increases regardless of other inputs (meaning that the predicted value of the new model f' increases or stays the same), the attribution of that input (ϕ_i) should not decrease. Put mathematically, if we let $f_x(z') = f(h_x(z'))$ and $z'\backslash i$ denote setting $z_i^{'} = 0$ while equation (6) holds for any two models f and f', then for all inputs $z' \in \{0,1\}^M$ it holds that $\phi_i(f', x) \geq \phi_i(f, x)$

$$f_x^{'}(z') - f_x^{'}(z'\backslash i) \geq f_x(z') - f_x(z'\backslash i) \tag{6}$$

There is only one possible explanation model which follows our previous definition of additive feature attribution methods and satisfies all three above mentioned properties. This model is shown in equation (7). This equation "follows from combined cooperative game theory results, where the values ϕ_i are known as Shapley values" (Lundberg and Lee (2017)). Here, $|z'|$ equals the number of non-zero entries in z' and $z' \subseteq x'$ is the set of all z' vectors where the non-zero entries are a subset of the non-zero entries in x'.

$$\phi_i(f, x) = \sum_{z' \subseteq x'} \frac{|z'|!\,(M - |z'| - 1)!}{M!} \left(f_x(z') - f_x(z'\backslash i)\right) \tag{7}$$

3.2 Shapley

SHAP calculates the effect of every feature by using Shapley values from Game Theory (Shapley (1952)). A Shapley value is the contribution of a player x_i to the game p. When applying Shapley values to machine learning, the 'player' x_i is a feature and the 'game' is a single prediction p.

The procedure works as follows. The first step is to create subsets containing all possible permutations of features including the empty set. For example, if a dataset contains the 3 features *feature1*, *feature2* and *feature3*, SHAP creates the following 8 subsets: {ø} (an empty set), {*feature1*}, {*feature2*}, {*feature3*}, {*feature1, feature2*}, {*feature1, feature3*}, {*feature2, feature3*} and {*feature1, feature2, feature3*}

Then, for a prediction p, the Shapley values of every feature i is calculated by using equation (8), where F is the set of all features, S is a subset of features and $f_S(\cdot)$ is the function which predicts a value based on the subset of features S. The right hand side of the equation calculates the effect to a prediction after adding feature i to a subset which does not contain feature i. This is then multiplied by the left side of the calculation. The sum of all these calculations gives the Shapley value, which is a weighted average because of the left side of the equation.

$$\phi_i = \sum_{S \subseteq F\{i\}} \frac{|S|!\,(|F| - |S| - 1)!}{|F|!} \left(f_{S\cup\{i\}}\left(x_{S\cup\{i\}}\right) - f_S\left(x_S\right)\right) \tag{8}$$

SHAP uses this calculated ϕ_i value for the formula given in equation (7). z_i will then be a binary value where a value of 1 indicates that the input is included in the model and 0 indicates exclusion from the model. After setting $\phi_o = f(\text{ø})$, all values for equation (7) are known.

A big downside of this is that there are $2^{|F|}$ possible permutations for the subset of features. This means that $2^{|F|}$ computations are needed to calculate the SHAP values of one single prediction. So, with 'only' 23 features in our dataset, there are already 8,388,608 possible permutations for every point in the data. In order to tackle this problem, two steps are taken. The first is to apply sampling approximations to equation (8). The following section will go into more detail on one such sampling method: Kernel Shap. There are, however, more approximation methods, such as Linear SHAP, Low Order Shap and Deep SHAP, all explained in Lundberg and Lee (2017). These algorithms are not explained in this thesis since they will not be used for our research. The second step is to approximate the effect of removing a variable from the model by integrating over samples from the training dataset.

3.3 Kernel SHAP

Kernel SHAP is a combination of another additive feature attribution method, LIME (Ribeiro et al. (2016)), with Shapley values. LIME uses Linear LIME in order to approximate *f*. In order to find ϕ, LIME minimizes the objective function shown in equation (9), where $\pi_{x'}$ is a local kernel which gives a weight

to the simplified inputs and ω penalizes the complexity of g. Equation (9) can be solved using penalized linear regression (Mao (2014)) because g follows equation (3) and L is a squared loss (Lundberg and Lee (2017)).

$$\epsilon = \arg\min_{g \in G} L(f, g, \pi_{x'}) + \Omega(g) \tag{9}$$

If the loss function l, weighting kernel $\pi_{x'}$ and regularization term Ω are chosen in such a way, Shapley values can also be deduced from equation (9). The values are calculated as shown below, where $|z'|$ is the number of non-zero elements in z'. The proof for this is shown in Lundberg and Lee (2017).

- $\Omega(g) = 0$
- $\pi_{x'}(z') = \frac{(M-1)}{(M choose\ |Z'|)|z'|(M-|z'|)}$
- $L(f, g, \pi_{x'}) = \sum_{z' \in \mathbb{Z}} \left(f\left(h_x\left(z'\right)\right) - g\left(z'\right)\right)^2 \pi_{x'}(z')$

Penalized linear regression can still be used and will now derive the Shapley values. Retrieving the Shapley values by regression instead of by using the classic Shapley equations will cause a significant decrease in computational costs. However, a big disadvantage is that the computed SHAP values are approximations and might therefore not be 100% accurate.

3.4 EXPLAIN-IT

A big downside of the before mentioned techniques like SHAP is that they cannot be used for all unsupervised Machine Learning techniques in order to explain results. This is especially the case for unsupervised Machine Learning algorithms based on distance and density. This is because the currently proposed Explainable AI algorithms for distance and density based unsupervised machine learning are mostly focused on whether a point is closer to a certain center than other points, which "obscures the impact of individual features" (Frost et al. (2020)). Morichetta et al. (2019) has therefore proposed a different technique for Explainable AI specifically made for unsupervised machine learning based on distance and density. Techniques like LOF and K-NN could potentially be explained globally by using this technique.

EXPLAIN-IT works relatively straightforward. It starts by running the desired algorithm, for example K-NN, on the dataset which divides the dataset into k clusters. Each datapoint then receives a label which tells to which cluster this datapoint belongs. Afterwards, a supervised Machine Learning algorithm is used (the original paper uses Support Vector Machine (SVM) as an example) which will again predict to which cluster each datapoint belongs. This supervised machine learning algorithm can then be explained by using for example SHAP (the original paper uses LIME because it is less time-consuming, but it is less accurate).

The downside of this technique, however, is that the unsupervised algorithm makes a prediction which is usually not 100% accurate. Then afterwards, the

supervised algorithm makes a prediction based on the previous prediction, making it even less accurate. This problem can get even worse for outlier-detection since the percentage of outliers is supposed to be really low. This makes the dataset very imbalanced, making a supervised algorithm even less accurate. Morichetta et al. (2019) also mentions that EXPLAIN-IT is "not a final system" and that the paper "sets the initial steps into the overall ambitious goal" of explaining unsupervised machine learning algorithms. So, this technique is not yet fully trustworthy and there are, to our knowledge, no better techniques yet. Therefore, we have decided not to use Explainable AI for the different k-NN algorithms and LOF, explained in sections 4.3 and 4.4.1 respectively.

4 Proximity based anomaly detection

Proximity based anomaly detection works by selecting anomalies based on the distance between datapoints and the rest of the data (Mehrotra et al. (2017)). Before going into more details about this approach, the notation will be described beforehand:

- Υ indicates the entire dataset.
- An uppercase symbol, e.g. P, indicates a set of points, so $P \subset \Upsilon$.
- A lowercase symbol, e.g. p, indicates a single point, so $p \in \Upsilon$.
- $d(p, q)$ indicates the distance between points $p,\ q \in \Upsilon$.

Three primary questions should be addressed according to Mehrotra et al. (2017):

1. **Measurement:** How anomalous is a given point? In order to answer this question, a function α should be defined which measures the anomalousness $\alpha(p) \in \mathbb{R}$ of $p \in \Upsilon$.
2. **Absolute:** Is a point anomalous? This is determined by setting a threshold θ with the property that when $\alpha(p) > \theta$, a point is considered anomalous.
3. **Relative:** Is one point more anomalous than another? This is determined by determining whether $\alpha(p) > \alpha(q)$ or not.

From the questions given above, two more questions can be answered which are often used in practice:

1. **What are the m most anomalous points in the dataset?** This can be answered by ordering the datapoints by the output given by function α.
2. **What are the m most anomalous points in the dataset which are also absolutely anomalous?** This can be answered by including the threshold mentioned earlier.

First, section 4.1 will explain a common problem occurring within machine learning algorithms based on proximity, called the curse of dimensionality, and how to deal with this problem. Afterwards, section 4.2 will explain how different distance measurements work and why we decided to include or exclude certain distance measurements for our research. Sections 4.3, 4.4 and 4.5 will explain multiple proximity based machine learning algorithms used for anomaly detection based on distance, density and clustering respectively. In the final section, section 4.6, the results brought by the chosen algorithms are analyzed and compared.

4.1 Curse of dimensionality

Before diving into various anomaly detection techniques based on proximity, we would first like to introduce a common problem occurring within machine learning, especially regarding proximity based techniques. This problem is called the curse of dimensionality, which was first introduced by Bellman (1957).

The curse of dimensionality covers multiple problems occurring with high dimensionality. The first problem already occurs after 3 dimensions, namely that data cannot be visualized. If data cannot be visualized it can be much harder to interpret and analyze. On top of that, there are also many computational problems with increasing dimensions.

Adding dimensions to the data increases the sparsity of the data, meaning that points are further away from each other. This results in that the variation in distance can decrease rapidly by increasing the amount of dimensions, making it more and more difficult to distinguish between close and closer points. With certain algorithms, such as k-nearest neighbours (explained in section 4.3), it can also increase the required time exponentially (Venkat (2018)).

The only way of dealing with the curse of dimensionality is by reducing the amount of dimensions. Some distance based algorithms are more vulnerably to the curse than others, but eventually all algorithms will be influenced by it. There are multiple dimensionality reduction techniques in order to prevent these problems from occurring. One of the most used dimensionality reduction techniques is PCA, which is explained in Appendix D. One can also remove dimensions with a lot of noise or which seem to be (completely) uncorrelated with the other dimensions.

4.2 Distance measurements

Before applying distance approaches, the right distance measurements should be chosen. This section will explain different distance measurements and why we decided to use the Euclidean distance metric.

4.2.1 Mahalanobis distance

An often used and well-known distance measurement is the Euclidean distance. However, the Euclidean distance is actually a reduced form of the Mahalanobis distance. Therefore, this section will first describe this distance metric and afterwards how it can be reduced to the Euclidean distance.

The Mahalanobis distance is calculated by using equation (10), where S is the covariance matrix which measures the mutual correlations between dimensions for all points in Υ. If the covariance matrix is the identity matrix, where $S_{ij} = 1 \forall i = j$ and $S_{ij} = 1 \forall i \neq j$, it reduces to the Euclidean distance, which is shown in equation (11).

$$d(p, q) = \sqrt{(p - q)^T S^{-1} (p - q)} \tag{10}$$

$$d(p,q) = \sqrt{\sum_{i=1}^{d} (p_i - q_i)^2} \tag{11}$$

Since the Mahalanobis distance uses the inverse covariance matrix, it does have two major drawbacks (Maesschalck et al. (2000)). The first drawback is that when there is a large number of variables, there might be a lot of redundant or correlated information. This is called multicollinearity. This results into a (nearly) singular covariance matrix which cannot be inverted. The second drawback is that the number of objects in Υ should be larger than the number of variables. If this is not the case, the number of variables will have to be reduced by using the methods previously explained.

4.2.2 Minkowski distance

The Minkowski distance is the general form of the Euclidian, Manhatten and Chebyshev distance. The Minkowski distance is shown in equation (12). For the Euclidian, Manhatten and Chebyshev distance, the values for l are 2, 1 and ∞ respectively (Berry et al. (2016)).

$$d(p,q) = \left(\sum_{i=1}^{n} |p_i - q_i|^l \right)^{\frac{1}{l}} \tag{12}$$

4.2.3 Chosen distance metric

The calculated distances in the algorithms shown in the following sections are calculated by using the Euclidean distance. The Manhatten distance and cosine similarity (measured by $\frac{\sum_{i=1}^{n} x_i * y_i}{\sqrt{\sum_{i=1}^{n} x_i^2} * \sqrt{\sum_{i=1}^{n} y_i^2}}$) were also tried as distance measurements. However, these distance measurements proved to cost too much computational effort for the hardware provided to perform the research since it required too much RAM.

The biggest drawback of the Euclidean distance is that it is very susceptible to the curse of dimensionality. However, this should not be a big problem for this research. When analyzing the results of the research done by Venkat (2018), the Euclidean distance should not be influenced too much by the curse when the amount of dimensions is 'only' 23. We therefore concluded that the Euclidean distance is a sufficient metric for this research. However, should there be more data available in the future with more features it might be worthwhile to test the algorithms using the other distance metric mentioned previously with hardware possessing more RAM.

4.3 Anomaly detection Algorithms based on distance

This section will explain multiple algorithms which define the anomalousness based only on distance.

4.3.1 Distance to all points

This approach calculates the distance (by using a distance measurement such as the ones discussed above) of each point $p \in \Upsilon$ against all other points in Υ. Afterwards, the sum of all these distances is calculated. This sum defines the anomalousness of the point p. The formula for this algorithm is shown in equation (13).

$$\alpha(p) = \sum_{q \in \Upsilon} d(p, q). \tag{13}$$

This is a very simple approach, but it does contain two major disadvantages. The first drawback is that the anomalousness of a non-anomaly also increases heavily because of other anomalies. This can cause points to easily be classified as an anomaly if the absolute threshold is set too low. The second major disadvantage is the high computational effort of calculating the distances to all other points for every point.

This computational effort turned out to be too high for the computer this algorithm was tested on. The computer had 8 gigabytes of RAM, but this turned out not to be enough. For this reason, this algorithm has not been tested on the data. The methods explained in section 4.3.2, 4.3.3 and 4.3.4 were, however, tested. As shown in section 4.6, these algorithm did not deliver good results. Since this algorithm is fairly similar to these algorithms it is assumed that this algorithm would also not bring good results.

4.3.2 Distance to nearest neighbor

This approach is very similar to the algorithm mentioned above. However, there is one key difference. Instead of taking the sum of all distances, the minimum distance is taken. Meaning that the anomalousness of a point is measured by the distance to the nearest neighbour. The formula is shown in equation (14). This requires significantly less computation power since not all distances have to be stored.

$$\alpha(p) = \min_{q \in \Upsilon, q \neq p} d(p, q) \tag{14}$$

The advantage of this approach is that it is more robust to other extreme outliers. This is due to the fact that the anomalousness of non-outliers is not increased because it is only dependent on the nearest point. The disadvantage, however, is that when an outlier is close to another outlier, the point can be considered as not anomalous, since the distance to its nearest neighbour is low. This means that if multiple outliers are grouped together they could be classified as inliers instead of outliers.

4.3.3 Average distance to k-nearest neighbors

Instead of only calculating the distance to the nearest neighbor, this approach calculates the averages of the k nearest neighbors. The formula for this algo-

rithm is shown in equation (15), where $Near(p, j)$ indicates the nearest points j. The value k will have to be predefined. The algorithm mentioned in the previous section is actually a special case of this algorithm, since it is the same algorithm with a value of 1 for k. This approach deals with the disadvantage that goes along with only taking the distance to the nearest neighbor into account. If an anomaly lies close to $k-1$ different anomalies, this algorithm can still classify that point as an anomaly since the difference to at least 1 other datapoint in the dataset Υ is relatively big. A group of anomalies is therefore more likely to be classified as anomalies than with the previously mentioned algorithm.

$$\alpha(p) = \frac{\sum_{j=1}^{k} d\left(p, Near\left(p, j\right)\right)}{k} \tag{15}$$

4.3.4 Median distance to k-nearest neighbors

This approach works very similar as above. However, in this case the median of distances to the k nearest neighbors is calculated instead of the average. The advantage of this is that it is less robust to extreme outliers. The disadvantage of this however, is that it requires a greater computational effort (Mehrotra et al. (2017)).

4.4 Anomaly detection Algorithms based on density

Density based algorithms are based on the density of points. Typically, the less dense a point is, the more likely it is to be an outlier. This technique makes it easier to look at local outliers, rather than global outliers. Section 4.4.1 will explain a commonly used outlier detection algorithm called Local Outlier Factor (LOF) and shows the potential benefit of finding local outliers as opposed to finding global outliers.

4.4.1 Local Outlier Factor

LOF is an algorithm first proposed by Breunig et al. (2000). It works by measuring the local deviation of every point in Υ in order to find anomalous points. It does this by using the concept of local density. The locality of a point is given by its k nearest neighbors. The distance to these k nearest neighbors is then used to estimate the anomalousness.

Before showing the potential benefit of local outliers to global outliers, we first have to define a DB(*pct, dmin*)-Outlier (Distance Based(*pct, dmin*)-Outlier). A point p is a DB(*pct, dmin*)-Outlier in Υ if there is a minimum percentage *pct* of all points in Υ which have a distance greater than *dmin* (*dmin* stands for distance minimum). In other words, the cardinality (size) of the set $\{q \in \Upsilon | d(p, q) \leq dmin\}$ is less than or equal to (100 - *pct*)% of the size of Υ.

These outliers are defined based on the distance between all points, meaning it looks for outliers based on a global scale. However, in many cases it might be favorable to look at distances in between points close to each other, meaning

that it might be favorable to look for outliers on a local scale. An example of this is shown in figure 17. Here it can easily be seen that there are 2 clusters, $\mathbf{C}_1$ and $\mathbf{C}_2$, and, by Hawkin's definition, two outliers: $\mathbf{o}_1$ and $\mathbf{o}_2$.

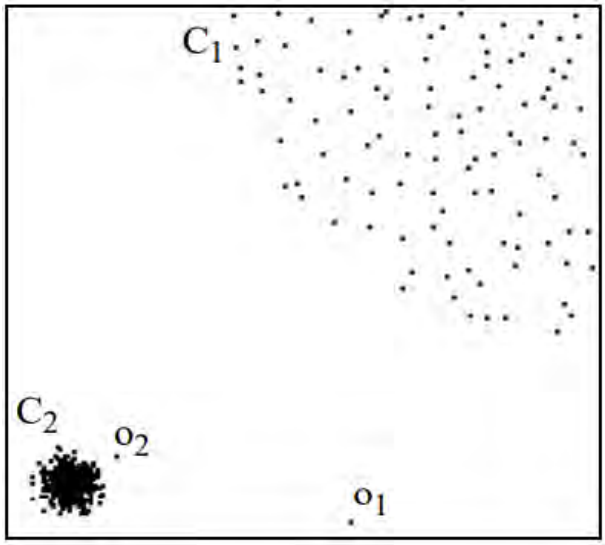


Figure 17: Example of clustered outliers. Image taken from Breunig et al. (2000).

There are two main important differences between the clusters. The first main difference is that cluster $\mathbf{C}_1$ is a lot less dense than cluster $\mathbf{C}_2$. The second difference is the amount of points of each cluster. Clusters $\mathbf{C}_1$ and $\mathbf{C}_2$ contain, respectively, 400 and 100 datapoints. If for every point q in $\mathbf{C}_1$ the distance between its nearest neighbor is bigger than the distance between $\mathbf{o}_2$ and its nearest neighbor in $\mathbf{C}_2$ (denoted as $d(o_2, C_2)$), it can be shown that it is impossible to give values to *pct* and *dmin* whereby $\mathbf{o}_2$ is an outlier and all points in $\mathbf{C}_1$ are not. This will be shown below.

If the above mentioned condition is valid and $dmin < d(o_2, C_2)$, then all 500 points in $\mathbf{C}_1$ and point $\mathbf{o}_1$ are further away from $\mathbf{o}_2$ than *dmin*. This will also be the case for all points within $\mathbf{C}_1$. This is because the condition $d(p, q) > d(o_2, C_2)$ holds for all points $(p,\ q) \in \mathbf{C}_1$. Therefore, all 500 points in $\mathbf{C}_1$ and $\mathbf{o}_2$ will be DB(*pct, dmin*)-Outliers.

The other case would be that $dmin > d(o_2, C_2)$. In this case, the cardinality of the set $\{p \in \Upsilon | d(p, o_2) \leq dmin\}$ is always bigger than the cardinality of $\{p \in \Upsilon | d(p, q) \leq dmin\}$. When $\mathbf{o}_2$ is a DB(*pct, dmin*)-Outlier, multiple points in $\mathbf{C}_1$ will also be DB(*pct, dmin*)-Outliers. It can also be the case that $\mathbf{o}_2$ is not a DB(*pct, dmin*)-Outlier, but that would also be incorrect since it should be a DB(*pct, dmin*)-Outlier.

Now that the importance of local outliers is understood, we will explain how LOF finds these local outliers. In order to do this, we first have to make

3 definitions. The first definition is the k-distance of p. Afterwards we will explain the k-distance nearest neighborhood of p. The final definition is the reachability distance of p with respect to q.

- The k-distance is the distance $d(p, q)$ for which the following two conditions hold:
  1. for at least k objects (k is a positive integer) $q' \in \Upsilon \backslash \{p\}$ it holds that $d(p, q') \leq d(p, q)$
  2. for at most $k - 1$ objects $q' \in \Upsilon \backslash \{p\}$ it holds that $d(p, q') < d(p, q)$
- the k-distance neighborhood of p, denoted as $N_k(p)$, is the set of all points whose distance from p is smaller or equal than the k-distance of p. In mathematical terms, the k-distance neighborhood of p is $N_{k-distance(p)}(p) = \{q \in \Upsilon \backslash \{p\} \mid d(p, q) \leq k - \text{distance}(p)\}$
- The reachability distance of p w.r.t. q is defined as $\textit{reach-dist}_k(p, q) = \max \{ \text{k-distance}(q), \textit{d(p, q)} \}$.

Usually, density algorithms work by using 2 parameters. The first parameter is *MinPts*, which is a minimum amount of points. The second parameter is a volume. A certain threshold is then set, meaning that if *MinPts* are located within the volume of point p, point p is dense (Breunig et al. (2000)). In order to define outliers based on the density, a density based algorithm needs to compare the density of different sets of points. This means that the density of these sets have to be determined dynamically instead of using a threshold. Therefore, the parameter for volume is omitted in LOF. Instead, the values of the reachability distances are used to determine the density of point p. These values are namely used to calculate the local reachability density of p. The formula for this is given in equation (16). This is actually the inverse of the average reachability distance based on the *MinPts*-nearest neighbors of p.

$$lrd_{MinPts}(p) = 1/ \left(\frac{\sum_{o \in N_{MinPts(p)}} \textit{reach-dist}_{MinPts}(p, o)}{|N_{MinPts}(p)|} \right) \tag{16}$$

Finally we can define the final formula which calculates the Local Outlier Factor. This formula is shown in equation (17). It is the average of the ratio of the local reachability density of p and those of p's *MinPts*-nearest neighbors. This means that if p has a low reachability density while p's *MinPts* nearest neighbors have a high reachability density, p has a high LOF and thus the point is more anomalous. These LOF values are then sorted from high to low to see which points are most anomalous. It can then be said that a certain top percentage of the datapoints is an anomaly.

$$\text{LOF}_{MinPts}(p) = \frac{\sum_{o \in N_{Minpts(p)}} \frac{lrd_{MinPts}(o)}{lrd_{MinPts}(p)}}{|N_{MinPts}(p)|} \tag{17}$$

4.5 Clustering Based

Clustering algorithms work by dividing the datapoints into different clusters. This can be used for unsupervised classification problems, but also for anomaly detection. According to Duan et al. (2009), most papers identify an anomaly as a single point which deviates from the formed clusters. However, according to the definition of an outlier given by Hawkins (1980), it can also be possible that outliers form a (small) cluster. This could especially be the case with PdM since a single outlier value could be a dummy value as described in section 2.1, while a malfunctioning machine can produce abnormal values for a longer period of time.

An example of such clustered outliers is shown in figure 18. According to many local distance measurements, such as the distance to nearest neighbor as explained in section 4.3.2, the points in clusters $\mathbf{C}_1$ and $\mathbf{C}_3$ will not be identified as outliers. According to Hawkin's definition however, these points are still outliers. Therefore, this section will explain an algorithm which can identify clustered anomalies, Isolation Forest. We also tried to test the algorithm DBSCAN, which is explained in Appendix C. However, the RAM shortage proved to be a problem again. We tried to overcome this problem by subsampling the data and lowering the values for *Eps* and *MinPts*, but this resulted in very unreliable results. However, DBSCAN contains many similarities with LOF and - as seen later in section 4.6.4 - the results of LOF are insufficient for predicting upcoming failures. We therefore also expect DBSCAN to deliver insufficient results. Nevertheless, it might be worthwhile to test this algorithm in the future with hardware possessing more RAM and therefore we added the explanation of this algorithm in Appendix C.

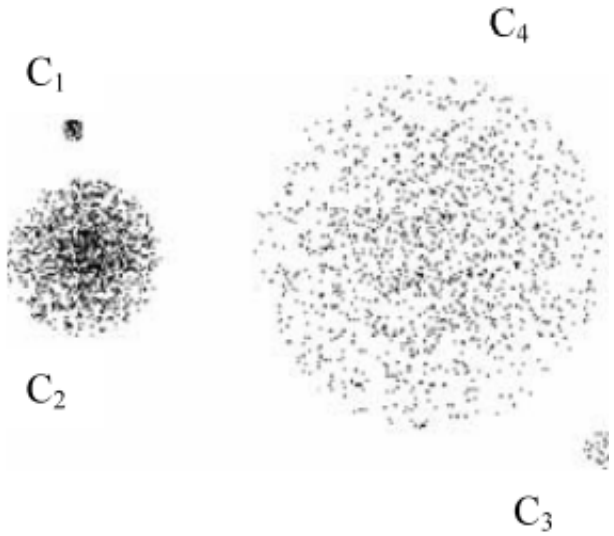


Figure 18: Example of clustered outliers. Image taken from Duan et al. (2009).

4.5.1 Isolation Forest

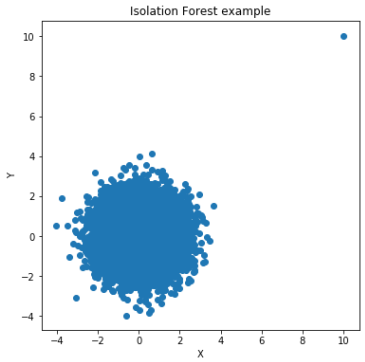
Isolation forest is an anomaly detection algorithm initially proposed by Liu et al. (2008). It works by clustering hierarchically, which creates 'trees'. The algorithm then distinguishes anomalies based on that anomalies are more easily isolated from the rest of the data than non-anomalies, since they are further away from other points.

In short, during the training part it creates random cuts in the dataset until either all points are separated from each other, or a maximum number of cuts has been made. For every cut it creates a new 'leaf' in the 'tree'. It repeats this process a predefined amount of times, meaning that a predefined amount of times, e.g. 100, trees are made. Hence the algorithm is called Isolation Forest, since it is based on isolating outliers and it makes multiple 'trees' which together form a 'forest'.

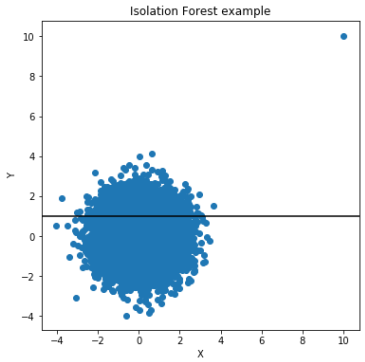
During the evaluation part, a scoring function for each point is derived from the path lengths. The path lengths are the lengths through which that point traverses through every tree until it is either isolated from the rest of the data or reaches the maximum length of the tree. The points are then ordered by descending order of their anomaly scores and a predefined top percentage of points is then called an anomaly.

An example of one such iteration with random cuts is shown in figure 19. The cluster in the lower-left corner is created by sampling 10000 random variables for x and y from the normal distribution with a mean of 0 and standard deviation of 1. It is easy to see that the value at coordinate (10,10) is an outlier and this example will show how the algorithm will find this outlier. The algorithm starts by making a random cut, this time at y = 1. Afterwards, it makes a new random cut at x = 5. This cut has isolated the point in the top-right corner, which is therefore now marked as a red point. Afterwards, it will continue making new random cuts (the first one also visualized with y = -3.5) and will continue up until either a maximum amount of cuts are made or when all points are isolated. In this case it will probably not isolate all points since the points in the center of the cluster are very dense and will therefore require a lot of random cuts before they are isolated. The random cuts are visualised in figure 19 and the tree created by the cuts is shown in figure 20. The tree does not show the last cut, since the tree would become too big and would no longer fit on the paper.

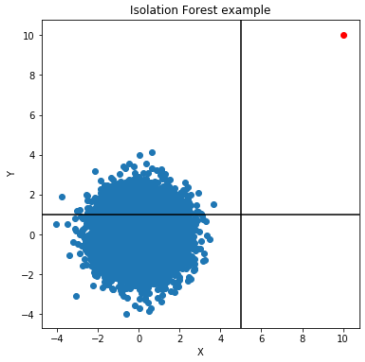
Every layer is created after a split, so every edge indicates a split. Thus, the first time the data is separated into two dataset is after the split of $y = 1$. So, "Dataset 1" is the part of the data where $y \leq 1$ and "Dataset 2" is the part of the data where $y > 1$. Afterwards, the data is split at $x = 5$, meaning that "Dataset 1.1" is the part of the data where $y \leq 1$ and $x \leq 5$ and "Dataset 1.2" would be the part of the data where $y \leq 1$ and $x > 5$, but this is an empty set and therefore it is not added to the tree. "Dataset 2.1" is the part of the data where $y > 1$ and $x \leq 5$. The part of the data which satisfies $y > 1$ and $x > 5$ is only one point, namely the point at (10,10). This means that the tree cannot be split further from this node. This is therefore an isolated point and the node is called an external-node. The tree can still be split at the other nodes. All these



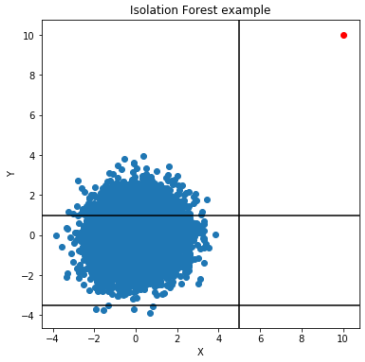
(a) Data without cuts.



(b) Data with 1 random cut.



(c) Data with 2 random cuts.



(d) Data with 3 random cuts.

Figure 19: Example of random cuts in random dataset.

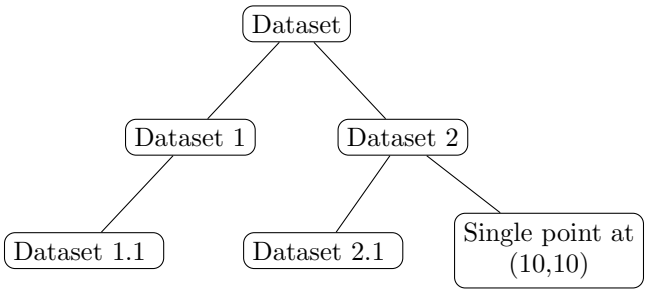


Figure 20: Tree diagram up until picture **C** from figure 19.

nodes will then also get two daughters (provided the daughters do not consist of empty sets) and are called internal-nodes.

The algorithms used to train Isolation Forest are summarized in algorithms 1

and 2. The tree height limit l in algorithm 1 is set by $l = ceiling(\log_2 \phi)$, which is approximately the average tree height (Knuth (1998)). The reason for setting this tree height is that the algorithm is designed to find anomalies and it is assumed that anomalies have path lengths (amount of cuts) shorter than the average tree height (Liu et al. (2008)). ϕ is the sampling size which, when increased, will increase the performance. However, the original paper finds that setting it higher than 256 will barely improve the results while the required time and computational power increases linearly. Therefore, setting the value for ϕ too high will result in an increased required time which will not outweigh the results. Also, according to the original paper, a value of 100 for t makes the path lengths usually converge well, while increasing it will only result in a longer running time. Therefore, this value should also not be set too high since the results will not outweigh the extra running time.

Algorithm 1: Isolation Forest (database Υ, Number of trees t, subsamplingsize ϕ)

Result: Forest (Set of trees)
Forest = Empty Set
height limit $l = ceiling(\log_2 \phi)$
for $i = 1$ *to* t **do**
  $\Upsilon' = sample(\Upsilon, \phi)$
  Forest.append(iTree($\Upsilon', 0, l$))
end
RETURN *Forest*

Algorithm 2: iTree(Database Υ, current tree height e, height limit l

Result: iTree
if $e \geq l$ *or* $|\Upsilon| \leq 1$ **then**
  RETURN ExternalNode{Size = $|\Upsilon|$}
else
  Select random attribute q from Υ
  X_{min} = minimum value of attribute q
  X_{max} = maximum value of attribute q
  Select random split point $p \in [X_{min}, X_{max}]$
  Υ_{left} = all points in Υ where $p < q$
  Υ_{right} = all points in Υ where $p \geq q$
  RETURN InternalNode{Left = iTree($\Upsilon_{left}, e + 1, l$),
  right = iTree($\Upsilon_{right}, e + 1, l$),
  splitAtt = q,
  splitValue = p}
end

After this training process the model has to evaluate all points and check which ones are anomalies and which ones are not. Therefore, an anomaly score

is derived based on the expected path length $E(h(x))$, which is calculated by passing the datapoint through all the iTrees. It does this by using the algorithm shown in algorithm 3. If an external node is reached, the algorithm returns the path length plus an adjustment. This adjustment is added to the path length to compensate for the possibility that a tree has stopped splitting because it has reached the maximum amount of splits, and not because it isolated all points (if $T.size = 0$ the adjustment function outputs 0). The formula for the correction function is given in equation (18), where n is the amount of datapoints and $H(x)$ is the harmonic number, which can be estimated by the function $\ln(i) + 0.5772156649$. The second part of this sum is Eulers constant.

Algorithm 3: PathLength(datapoint p, iTree T, current path length e (initially zero))

Result: Path length of p
if *T is external node* **then**
  RETURN $e + c(T.size)$
end
a = T.*splitAtt*
if $x_a < T.splitValue$ **then**
  RETURN *PathLength(x, T.left, e+1)*
else
  RETURN *Pathlength(x, T.right, e+1)*
end

$$c(n) = 2H(n-1) - (2(n-1)/n) \tag{18}$$

After deriving all the path lengths of all trees an anomaly score is produced by using equation (19), where x is the tested datapoint, n is the total amount of datapoints, $E(h(x))$ is the average of all path lengths and $c(\cdot)$ is the equation shown in equation (18). All points are then sorted by descending order based on their anomaly score.

$$s(x, n) = 2^{-\frac{E(h(x))}{c(n)}} \tag{19}$$

4.6 Results of Proximity based anomaly detection

This section will analyse the results of all algorithms mentioned in this section. In order to obtain these results, the Python package PyOD was used (Zhao et al. (2019)) on Python 3.6. PyOD stands for Python Outlier Detection and is a package which uses multiple widely used Python Machine Learning packages, such as SciKit-Learn (Pedregosa et al. (2011)) and Keras (Chollet et al. (2015)), and uses the Machine Learning algorithms provided by these packages specifically for outlier detection purposes. This package makes it easier to interpret and obtain the results required for this thesis.

The SHAP values calculated from the results of Isolation Forest will also be discussed and analyzed afterwards. In order to obtain these values and graphs the SHAP Python package was used (Lundberg and Lee (2017)). Lundberg et al. (2020) provides an algorithm which is able to calculate the exact SHAP values for tree-based algorithms in polynomial time: Tree SHAP. This algorithm has also been used for this thesis since it is more accurate and requires less computational effort than other heuristics for SHAP, such as Kernel SHAP.

4.6.1 Distance to nearest neighbor results

The results from using the distance to the nearest neighbor algorithms (which is equivalent to using the k-NN algorithm and setting k equal to 1) is shown in figure 21. The red horizontal line indicates the threshold of the anomaly score, meaning that all datapoints with an anomaly score higher than that threshold are classified as outliers. Before running an algorithm through PyOD, a value for the contamination has to be set (percentage of outliers). After trial and error, we decided to set the contamination value equal to 0.6% (which in this case means there are 304 outliers). Therefore, the 0.6% of points with the highest anomaly score are considered outliers. The threshold is then equal to the lowest anomaly score of all the outliers. In this case, this value is equal to approximately 0.1118. The value for the contamination for all other algorithms was also set at 0.6% in order to make a trustworthy comparison of the results.

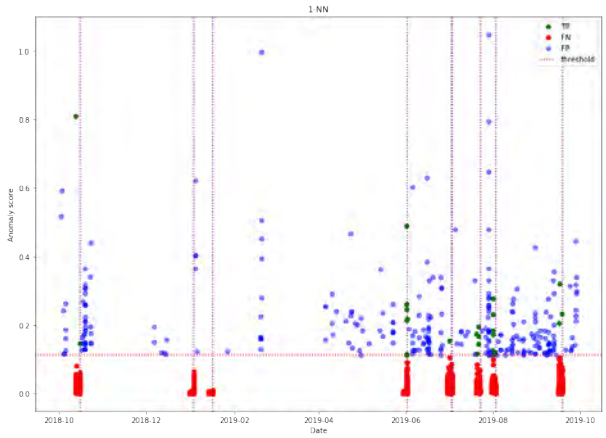


Figure 21: Results of Nearest Neighbor algorithm.

The vertical purple colored dotted lines are placed at the points in time where the malfunctions shown in table 1 occurred. So, for example, the first purple colored dot line is at the time point of 15/10/2018 10:36, where the scumboard cleaner malfunctioned.

The red colored points are the False Negatives, the blue colored points are the False Positives and the green colored points are the True Positives. The definitions of these points were explained previously in section 2.4. The True

Negatives are not plotted since the number of True Negatives is very high compared to the other values (in this case 469556). Plotting all these points will make it a lot harder to get a good oversight of the FPs. To illustrate this, a plot including the True Negatives for this algorithm is shown in figure 22. The amount of yellow points makes it a lot harder to spot the red points which are more interesting for this research. The True Negatives will, for this same reason, also not be shown in the resulting plots of the other algorithms.

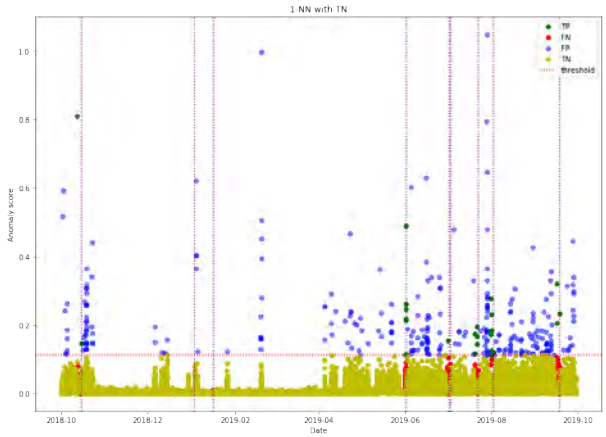


Figure 22: Results of Nearest Neighbor algorithm including TN's.

The confusion matrix is shown in table 2. A confusion matrix gives an oversight of the amount of TP's, FP's, FN's and TN's. These values are shown in the upper-left, upper-right, lower-left and lower-right respectively. As can be seen, the amount of TP's is only 25, which is very low, while the amount of FP's is very high.

	Point in time 3 days before malfunction	**Point in time not 3 days before malfunction**
Outlier	25	279
No outlier	35619	469556

Table 2: Confusion matrix of Nearest Neighbor algorithm.

When combining these results with figure 21, we can see that the TP's and FP's appear to be placed quite randomly. This will result into a lot of unnecessary inspections which will cost a lot of time and money. Another downside of this is that after a while the pump operators can lose trust in the algorithm if there are too many unnecessary inspections and might therefore, after a while, ignore the results completely. The True Positives will therefore always be assumed to be False Positives, meaning that they will be ignored and therefore the malfunctions will not be prevented. This problem is made even worse since it is

still almost impossible to explain the results by using Explainable AI techniques (as explained in section 3.4).

When looking only at the TP's, we decided that they are also not trustworthy enough to predict an upcoming malfunctions. For example, for the first malfunction, which occurred at 15/10/2018 10:36, the algorithm only classified two datapoints as outliers: the datapoints at 12/10/2018 11:34 (almost three days before the malfunction occurred) and 15/10/2018 08:27. Since the detected outliers are more than 48 hours apart from each other, it seems more like the classified outliers occurred in the three days before the malfunction by coincidence, and not because the model actually predicted the upcoming malfunction. This is also the case for the other malfunctions.

The next 'group' of TP's which are detected close to each other occurs at 01/06/2019. It is a group consisting of 7 outliers within a more than 2 hour time period (from 09:57 AM up until 12:10 PM). The malfunction, however, occurred at 22:23, meaning that in the 10 hours before this malfunction the model did not detect any outliers. After combining this with the fact that there are so many points in time with falsely detected outliers, we decided that we cannot conclude that this malfunction was actually predicted by the model.

The only other relatively big group of close outliers is a group consisting of 6 outliers in the time period from 31/07/2019 20:54 PM up until 31/07/2019 22:27. The corresponding malfunction, however, occurred 2 days later, at 02/08/2019 15:39. We decided that this group is also not trustworthy enough to have predicted this malfunction for the same reasons as for the previous group of outliers.

All other TP's occur quite randomly and not in 'groups' as the ones mentioned above. We therefore concluded that this model has not been able to predict any malfunctions and thus has not been able to reduce downtime. It will also cause a lot of unnecessary inspection rounds due to the amount of randomly placed FP's which will cost a lot of money and will eventually result in a loss of trustworthiness.

4.6.2 Average distance to 3-nearest neighbors results

The algorithm for k-NN was also tried with a value of 3 for k. This section will show and analyze the results for taking the mean value of the distances to the 3 nearest neighbors. The next section will do the same, but then for taking the median value of the distance of the 3 nearest neighbors, instead of the mean. The resulting plot of this algorithm is shown in figure 23, where the threshold is now placed at approximately 0.1317. The corresponding confusion matrix is shown in table 3.

The results are very similar to the results of 1-NN, implying that for this research, the value for k does not change the results significantly. The amount of TP's, FP's, etc. are almost equal and the points which are classified as such, again, seem to be completely random. This gives the same problem of untrustworthy results mentioned in the previous section.

The two 'groups' of TP's mentioned in section 4.6.1 also appear here at the

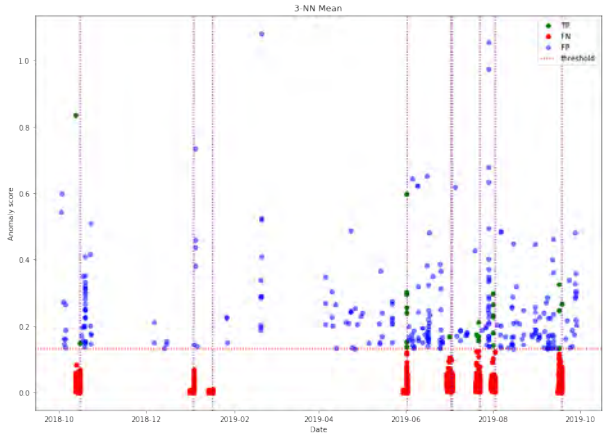


Figure 23: Results of 3-NN algorithm with mean of distance values.

same dates of 01/06/2019 and 31/07/2019. They are also of the same size and as mentioned before, the classification appears to occur very randomly resulting in a loss of trustworthiness. Therefore we have concluded the same as before, namely that this algorithm is unable to predict upcoming malfunctions.

	Point in time 3 days before malfunction	**Point in time not 3 days before malfunction**
Outlier	24	280
No outlier	35620	469555

Table 3: Confusion matrix of 3-NN algorithm with mean value.

4.6.3 Median distance to 3-nearest neighbors results

Since we concluded that the value of k is in this case insignificant for the results of k-NN, we decided to check if taking the median of the distance values instead of the mean value would bring different results. The results given by using this algorithm are shown in figure 24 with a threshold of approximately 0.1343. The corresponding confusing matrix is shown in table 4.

Again, the results are very similar to the previously given results. The values shown in the confusion matrix are even exactly the same as those shown in the confusion matrix of 1-NN. The 'groups' of TP's also occur at the same dates with the same sizes. The randomness of all points will also result in a loss of trustworthiness. Thus, we can conclude that using either the mean or median value for the distances calculated by the k-NN algorithm does not result in a significant change in the results. We also decided not to raise the value for k to much higher values since this requires too much computational effort.

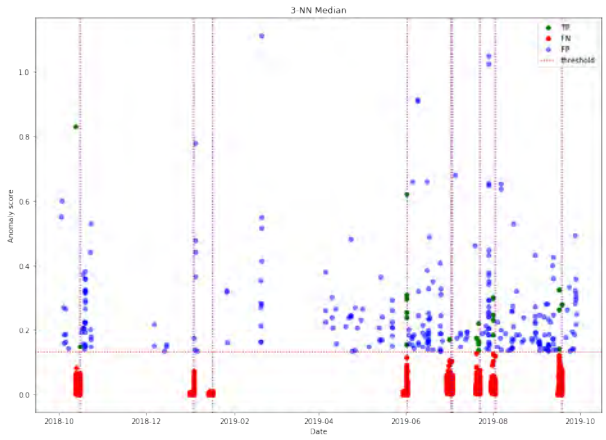


Figure 24: Results of 3-NN algorithm with median of distance values.

	Point in time 3 days before malfunction	**Point in time not 3 days before malfunction**
Outlier	25	279
No outlier	35619	469556

Table 4: Confusion matrix of 3-NN algorithm with median value.

4.6.4 LOF results

This section will show the results given by LOF. Next to the contamination of 0.6%, there was also a parameter for the amount of k-nearest neighbors the algorithm should use in order to calculate the anomaly score. We have experimented with multiple values for k. First we started with a low value for k, namely 5, but this did not bring good results. We hypothesized that in order to find local outliers based on density in a dataset with more than half a million datapoints we had to increase the value for k. Therefore we ran the algorithm again with increasing values for k, namely 10, 20 and 30. However, the changes to the results were negligible, even when using a value of 100 for k. The only thing which did increase significantly was the time required to run the algorithms. Since all results were almost identical we will only show the results for when k is equal to 30. The results are shown in figure 25 and the corresponding confusion matrix is shown in table 5.

The results shown in figure 25 can be interpreted in the same way as the figures with the results for the previous algorithms. The only difference is that for LOF, the points below the threshold (which is equal to approximately -4.6024) are considered outliers, instead of above. This is due to the implementation of the PyOD package. Just like with the nearest neighbor algorithms, the points which are classified as outliers are spread out very randomly, causing the same loss in trustworthiness.

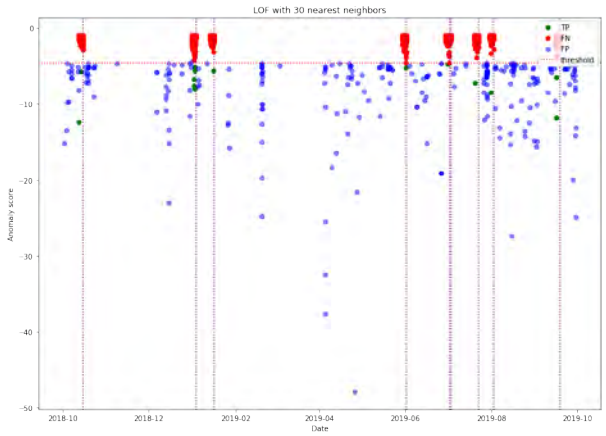


Figure 25: Results LOF with 30 nearest neighbors.

It can be seen that there is one 'group' of multiple detected outliers right before a malfunction. This group is a group of 5 outliers starting from 02/01/2019 07:17 up until 02/01/2019 16:41. However, the malfunction occurred the day after. Combining the fact that there are no outliers detected on the day the malfunction actually happened and the algorithm losses trustworthiness due to randomly spread out outliers, we again decided that the algorithm was not able to predict this malfunction. All other TP's are also spread out randomly with at least one day in between and therefore unable to predict a malfunction.

	Point in time 3 days before malfunction	**Point in time not 3 days before malfunction**
Outlier	14	290
No outlier	35630	469545

Table 5: Confusion matrix of LOF algorithm with 30 nearest neighbors.

As seen in table 5, the amount of TP's is also significantly lower than with the nearest neighbor algorithms. The amount of TP's of LOF is 14, while the amount of TP's of the k-NN algorithms were either 24 or 25, meaning that LOF gives worse results. We therefore concluded that local outliers are not good indicators for predicting upcoming malfunctions for this research.

4.6.5 Isolation Forest Results

Isolation Forest was ran with 200 trees and each tree was trained by drawing 256 random samples. This amount of training samples for each tree is the default amount and was not increased since this was recommended by the original paper as explained in section 4.5.1. The amount of trees, on the other hand, was increased from the default value of 100 to 200 due to the data size. The

resulting plot of the anomaly scores and corresponding confusion matrix are shown in figure 26 and table 6 respectively.

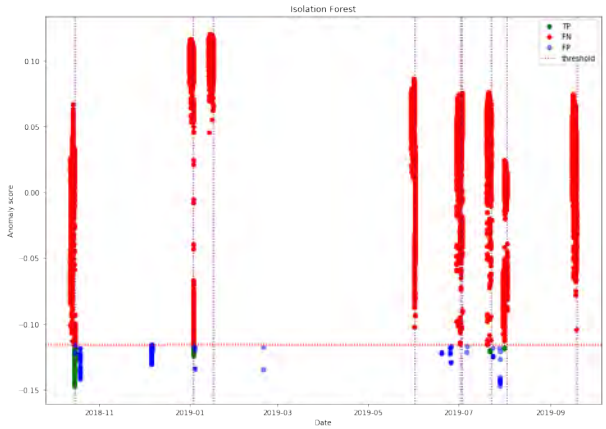


Figure 26: Results of Isolation Forest.

Just like with the plot of the anomaly scores calculated by LOF shown in figure 25, the points below the threshold (which is placed at approximately -0.1160) are classified as outliers. A clear distinction between this plot and the plots resulting from the previously mentioned algorithms is that almost all identified outliers are grouped together. This immediately makes the results a lot more trustworthy since the False Positives will cause a lot less unnecessary inspection rounds. We think these groups are created because if a group of outliers is close to each other, they are all almost equally easy to isolate from the rest of the data, resulting in an almost equal anomaly score.

	Point in time 3 days before malfunction	**Point in time not 3 days before malfunction**
Outlier	105	199
No outlier	35539	469636

Table 6: Confusion matrix of Isolation Forest.

The confusion matrix resulting from IF also shows significantly better results than the other confusion matrices. The amount of TP's in this confusion matrix is 105, while it was 15 after applying LOF and the nearest neighbor algorithms resulted in 24 and 25 TPs. The amount of FP's also decreased to 199 compared to 290 with LOF.

This number is, however, still relatively high. This high number of FP's can have multiple causes. It might be the fact that IF truly classified these points incorrectly as outliers. However, it might also be the case that something did happen to the pump causing damage. This damage might not have been significant enough for the pump to stop functioning and therefore no malfunction

has been reported even though damage still has been done. The fact that many FP's are consecutive points in the data, as can be seen by the 'groups' of blue points in figure 26, might suggest that this is the case. However, this is pure speculation and it is impossible to discover whether this truly was the case or not.

It is also much clearer to see whether a TP actually predicted an upcoming failure or if it is purely coincidence. The first two reported failures, occurring at 15/10/2018 10:36 and 03/01/2019 12:14, are predicted up front. Starting from 15/10/2018 8:27 (so approximately two hours before the reported failure), the model classifies 90 points in the data as outliers before the failure occurs. Meaning that almost every point from two hours before the failure at 15/10/2018 is classified as an outlier. This gives us enough evidence to conclude that this failure truly would be identified up front.

The first TP before the second failure, which occurred at 03/01/2019 12:14, is at the time point of 03/01/2019 9:11, so approximately 3 hours before the reported failure. Up until 10:09 AM of the same day 9 other time points are classified as outliers as well. Meaning that the algorithm detects a total of 10 outliers within 3 hours before the failure occurs. We therefore conclude that if these warnings would have been given to the pump operator, the pump operator would have sent an inspection team to find out why these warnings were given and this failure could therefore have been prevented.

The other 5 TPs are considered to be too much of a coincidence in order to truly predict a malfunction. There are three at approximately 14 hours before the malfunction at 22/07/2019 22:03 PM. They occur exactly at 08:23, 08:24 and 08:25 of the same day, but since only these three datapoints are considered as outliers and closer points are not, this is not considered to be trustworthy. The other two TPs are found on 31/07/2019 at 20:38 and 20:54. The corresponding malfunction occurred at 02/08/2019. Since only two outliers were detected two days before and 0 outliers were detected closer to the malfunction, we concluded that this malfunction would also not be prevented.

Given the results above, IF would have prevented the first two malfunctions, which caused 0 and 1 hour of downtime respectively. This means that 1 hour out of the in total 16.75 hours of downtime would have been prevented by using this model. Summarizing, IF would have prevented 6.07% of the total downtime. This might not sound as much, but considering BAM can receive fines of up to 10,000€ per 15 minutes of downtime for water pumping stations, a cost reduction of 6.07% is still significant.

As mentioned before, Tree SHAP has been applied in order to determine why a datapoint has been identified as an outlier. Tree SHAP does this both on a global and on a local level. Figure 27 shows the SHAP values for the 20 most important attributes for all TPs. The features are ordered by importance from top to bottom. This means that the top feature, the difference in value of the electric current of phases 1 and 3, is the most important feature. The second most important feature is the vibration speed of the first top bearing of the motor, etc.

The further away a value is placed from the vertical line at 0.0, the more

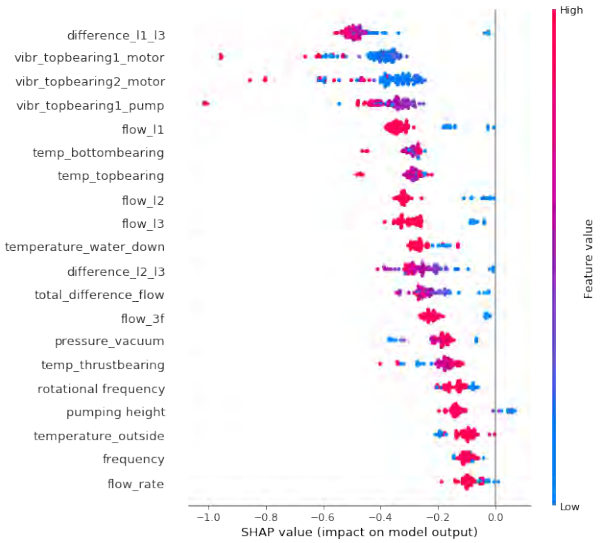


Figure 27: SHAP values for all TPs.

impact it has on being classified as on outlier or not. This means that for the difference between phase 1 and 3, a high value is a stronger indication for outliers than a low value, as was initially expected by the domain experts. The same can be said for vibration speeds and the electric current values for the different phases. Globally speaking, it can be seen that vibration speeds of the top bearings and the values for the electricity current are important indicators for outliers. The water temperature of the IJssel is also listed in the top 10 features, so it does have some impact, but it is not the strongest indicator.

Figure 28 shows the same type of plot, but then for all points which are considered outliers (so both TPs and FPs). From this plot it can also be concluded that the vibration speeds of the top bearings are important for both TPs and FPs. However, the electricity flow is deemed a lot less important. This means that when an outlier is detected right before an upcoming failure, the value for the electricity flows is deemed one of the most important features, but this is not the case when an outlier is detected when it is not close to a malfunction. This implies that when a pump operator receives an error message from this model and he/she has to decide whether or not an inspection crew should be send, he/she can check whether the failure was hugely depended on the electricity current or not.

Figures 29, 30 and 31 (called force plots (Lundberg et al. (2018)) enable the analysis of the classification process on a local scale. The base value of 12.03 is the value IF would predict if no features were available. What this value means is actually a black box as well, but the further away the predicted value is after adding all features, the more likely that the point is an outlier. The first

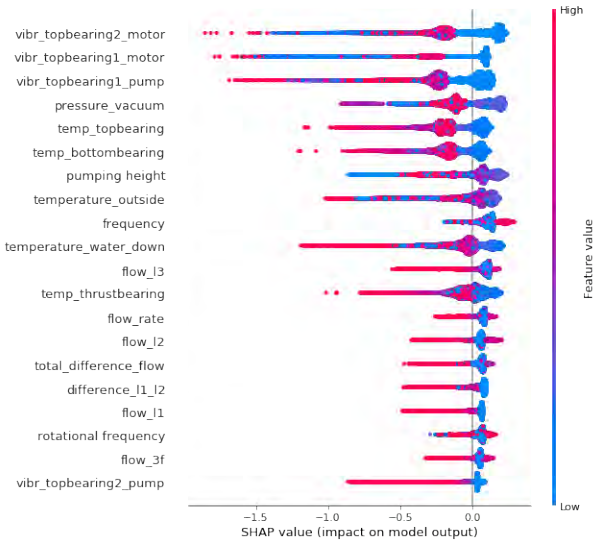


Figure 28: SHAP values for all outliers.

feature is again the most important feature, the second feature the second most important, etc.

Figure 29 shows the result of applying Tree SHAP to the datapoint at 15/10/2019 09:17, which is one of the outliers which predicted the scumboard cleaner failure. When comparing this plot to the global results shown in figure 27, it can be seen that for this instance the results are consistent. On both global and local scale, the difference in electricity current of phases 1 and 3 is the most important feature, then the vibration speeds of the top bearings and then the electricity current of phase 1. When this model is implemented, these results could be shown to the pump operator so that the pump operator can decide whether or not it will be necessary to send an inspection team.

Figure 29: Important features of outlier before malfunctioning scumboard cleaner.

Figure 30 shows the result of applying Tree SHAP to the datapoint at 03/01/2019 09:22, which is one of the outliers that predicted the UPS failure. Again, these results are also consistent with the results shown in figure 27 by having the same features in the top 4 most important features.

On the other hand, figure 31 shows an FP outlier at 06/12/2018 01:55. This

Figure 30: Important features of outlier before malfunctioning UPS.

time, the results are more consistent with figure 28 than with figure 27. This can be seen by the fact that the electricity current is not considered to be an important factor, but instead the pressure of the vacuum pump is considered to be important. This is the case with many FPs. This implies that when the explanation for an outlier gives a high importance to the pressure of the vacuum pump and not to the electricity flow, the outlier could be ignored and seen as an FP.

Figure 31: Important features of an FP at 06/12/2018 01:55.

5 Deviation based anomaly detection

In this chapter deviation based anomaly detection is explained. Deviation based anomaly detection algorithms work by predicting what value should occur given historical data. The more the predicted datapoint deviates from the actual datapoint, the more anomalous this datapoint is. Section 5.1 will explain how the autoencoder works, which is currently one of the more widely-used deviation based algorithms. Afterwards, section 5.2 will analyse the results given by using the autoencoder and the calculated SHAP values.

Another commonly used deviation based anomaly detection algorithm is Principal Component Analysis. At first we wanted to implement this algorithm as well, but in the end we decided not to implement PCA. The calculation of the covariance matrix required too much RAM for the hardware which was provided to us. On top of that, autoencoders are nowadays considered to be a better alternative to PCA. This is because they are both deviations techniques, but autoencoders can find both linear and non-linear correlations within the data, while PCA can only find linear correlations. PCA could be really helpful for reducing the amount of dimension before using a clustering algorithm if the curse of dimensionality occurs. However, as mentioned before, the curse of dimensionality is not a big issue with 23 features. If more attributes are provided in the future, however, this problem could occur and PCA could be a good way of dealing with it. Therefore, we decided to include the explanation on how PCA works in Appendix D.

5.1 Autoencoder

The autoencoder is an often used deviation based approach for anomaly detection (Lutz et al. (2020); Yang et al. (2018); Park et al. (2019)). The reason an autoencoder works well for anomaly detection is that it can accurately find patterns in historical data and use this data to recreate new data. An autoencoder is a special form of an artificial neural network (ANN) (Wani et al. (2020)). So in order to understand how an autoencoder works, it is necessary to first understand how an ANN works (Anzai (1992)).

5.1.1 Forward propagation

An ANN is based on the neurons inside the human brain. An example of what an ANN looks like is given in figure 32. An ANN exists of at least 2 layers: the input layer, which inputs all the given datapoints, and the output layer, which outputs the calculated result. There can be more layers in between the input and output layer (as seen in figure 32). These layers are called the hidden layers. Hidden layers make it possible to identify more complex patterns. Multi-layer neural networks can be used for multiple problems such as function approximation, pattern classification, system identification, process control, optimization and robotics (da Silva et al. (2017)).

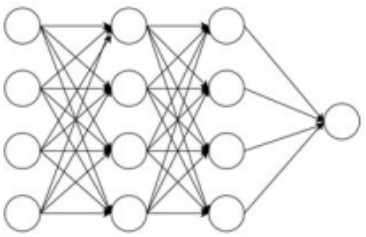


Figure 32: Example of an artificial neural network. Image taken from Wani et al. (2020).

All nodes in the set $U = \{\mu_1, \ldots, \mu_n\}$ are connected to the nodes in the next and previous layers through links (except for the input layer which has no previous layer). All nodes, except for the nodes in the input layer, contain a bias and all links contain a weight. The set of links is written as shown in equation (20).

$$L = \{(i,j) \,|\mu_i \in U, u_j \in U\} \subseteq U \text{ x } U \tag{20}$$

After a node receives information from the previous layer, it calculates an output for the next layer and updates its states based on an activation function. This can be mathematically written for the input layer with D variables as shown in equation (21) (Bishop (2006)):

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \tag{21}$$

The variables $w_{ji}^{(1)}$ and $w_{j0}^{(1)}$ represent the weights of the edges going from node j to i from the input layer to the next layer and biases of each node of the first layer after the input layer respectively. The values a_j are known as activations. These activations will then be transformed using a non-linear activation function $z_j = h(a_j)$. This activation function is usually a sigmoidal function such as the logistic sigmoid, $\frac{1}{1+e^{-x}}$, which always outputs a value between 0 and 1, or the tanh, $\frac{2}{1+e^{-2x}} - 1$, which always outputs a value between -1 and 1, as can be seen in figures 33 and 34. Both functions are plotted with values of x ranging from -10 to 10.

The same process as above is then performed for the next layer. The activation value of each node in the next node is the sum of all incoming weights times the value of the node from the previous layer + the bias of the node of the next layer. These activation values are then also activated through an activation function. The formula for this is shown in equation (22), where z_i is the output of a node from the previous layer that is connected to node j, w_{ji} is the weight of that link from node i to node j and w_j is the bias of node j. Afterwards, a_j is then transformed to z_j by an activation function $h(\cdot)$. This step is repeated up until the output layer, where the final output is given.

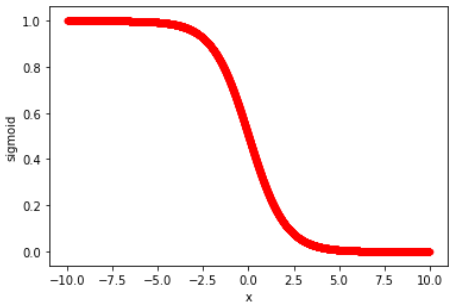


Figure 33: Plot of Sigmoid function.

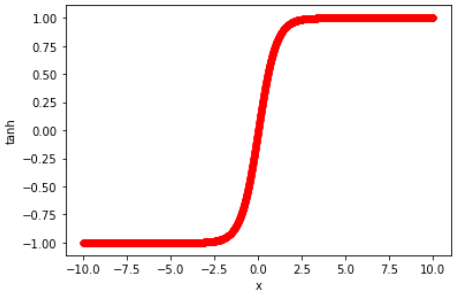


Figure 34: Plot of Tanh function.

$$a_j = \sum_i w_{ji} z_i + w_j \tag{22}$$

In order for a neural network to train itself on the data, the accuracy of each prediction is measured through a loss function. One such loss function is the sum-of-squares error function, which has to be minimized. This function is shown in equation (23), where $n = 1, \ldots, N$, $E(\mathbf{w})$ is the sum of squared errors given a vector of weights $\mathbf{w}$, $\mathbf{y}(\mathbf{x}_n, \mathbf{w})$ is the output vector of the neural network given input vector $\mathbf{x_n}$ and vector of weights $\mathbf{w}$. And finally, $\mathbf{t}_n$ is the vector of the true values.

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} ||(\mathbf{y}(\mathbf{x}_n, \mathbf{w}) - \mathbf{t}_n||^2 \tag{23}$$

The weights, which are initially randomized, will have to be adjusted in order for a neural network to learn. This is done by backpropagation and a weight adjustment method. One such commonly used weight adjustment method is gradient descent (Rumelhart et al. (1986b)).

5.1.2 Gradient Descent

As mentioned above, the weights have to be adjusted in such a way that the loss function is minimized. Put mathematically, a vector $\mathbf{w}$ has to be found which minimizes the loss function $E(\mathbf{w})$. Gradient descent does this by finding the steepest steps towards a local minimum. The process is visualized in figure 35.

This process starts by calculating the gradient of loss function $E(\mathbf{w})$: $\bigtriangledown E(\mathbf{w})$, which is the vector containing all partial derivatives of the loss function. Since the loss function is a continuous curve, the negative value of the gradient will show how the weights will have to be adjusted in order to decrease the loss function the most. The weights will then be adjusted as shown in equation (24), where τ is the iteration step and η is the learning rate > 0.

$$\mathbf{w}^{\tau+1} = \mathbf{w}^{\tau} - \eta \bigtriangledown \mathbf{w}^{\tau} \tag{24}$$

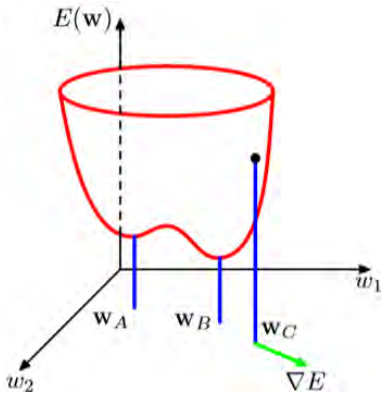


Figure 35: Example of Gradient Descent. Image taken from Bishop (2006).

This is then repeated until $\bigtriangledown E(\mathbf{w}) = 0$. This is because a gradient equal to zero indicates that a local minimum has been reached. The algorithm can also stop when the gradient does not decrease significantly after a pre-defined amount of steps. This indicates that the algorithm is very close to a local minimum and the decrease of the loss function is so small that it does not outweigh the extra computational costs.

The risk of this algorithm is that a solution can get stuck in a local minimum and there is no guarantee whether the local minimum found is also the global minimum. One way to solve this problem is to repeat this process multiple times with different starting position. This increases the chance of finding multiple local minima after which the best local minimum can be selected.

5.1.3 Backpropagation

Before gradient descent can be applied in order for the neural network to learn, it is necessary to calculate all the derivatives. Backpropagation is a useful method for doing this and can be applied to networks with different kinds of activation and loss functions.

First, the derivative of E_n w.r.t. to weight w_{ji} is considered. E_n depends on weight w_{ji} via the summed input a_j to node j. Therefore, it is possible to apply the chain rule for the partial derivatives. This gives the following equation:

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}} \tag{25}$$

It is now useful to use the notation shown in equation (26).

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} \tag{26}$$

When taking the partial derivative of equation (22) w.r.t. z_i it is seen that:

$$\frac{\partial a_j}{\partial w_{ji}} = z_i \tag{27}$$

Combining equation (25) with (26) and (27) gives the equation shown below.

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i \tag{28}$$

This equation shows that the required derivative is obtained by multiplying the value of δ of the receiving node by the value of z of the sending note. For the output nodes, δ_k is simply calculated by using equation (29).

$$\delta_k = y_k - t_k \tag{29}$$

For the nodes in the hidden layer, the partial derivative is again used, as is shown in equation (30). When combining this equation with equations (22), (26) and $z_j = h(a_j)$ (where $h(\cdot)$ is the activation function), we get the final equation, equation (31).

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j} \tag{30}$$

$$\delta_j = h^{`}(a_j) \sum_k w_k \delta_k \tag{31}$$

So, through backpropagation the derivatives are easily calculated and can then be used in order to improve the weights through e.g. gradient descent.

5.1.4 Selecting the amount of hidden neurons

Selecting the right amount of hidden neurons can be very important for the optimal functionality of a neural network. However, there is no straightforward way of finding the right amount of hidden layers and amount of neurons per layer. According to Heaton (2015) there are three rules of thumbs as listed below. However, these rules of thumb will never automatically give the right combination and in some cases they will not be applicable at all. Therefore, it will always still come down to trial and error in order to find the optimal combination of hidden layers and amount of neurons per layer.

1. The number of hidden neurons should be between the size of the input layer and the size of the output layer.
2. The number of hidden neurons should be 2/3 the size of the input layer, plus the size of the output layer.
3. The number of hidden neurons should be less than twice the size of the input layer.

5.1.5 Difference between autoencoders and neural networks

Autoencoders, as mentioned before, are a special type of neural network. They also contain an input, an output and (multiple) hidden layer(s). The output values are also calculated by forward propagation and the weights are adjusted by backpropagation and learning methods like gradient descent. There is, however, one mayor difference: The goal of the autoencoder is to reduce the input into smaller dimensions by using hidden layers with less nodes than the input layer and then tries to recreate the same data as the input (Rumelhart et al. (1986a)). 'Regular' ANNs, on the other hand, are used for either classification or regression problems. The loss function of the autoencoder is therefore the difference between the original and recreated values. This difference can be calculated through, for example, the MSE.

An illustration of the structure of an autoencoder is given in figure 36. The encoder is the part where the data is reduced. The decoder is the part where the network tries to recreate the input data based on the reduced data produced by the encoder. The middle part is called the bottleneck. The advantage of this approach is that the network discovers patterns in the data. This can be used to fill in missing data (e.g. fill in missing parts of images) (Fan et al. (2019); Yu et al. (2018)), image colorization (Schmitt et al. (2018)) and anomaly detection (Sakurada and Yairi (2014)).

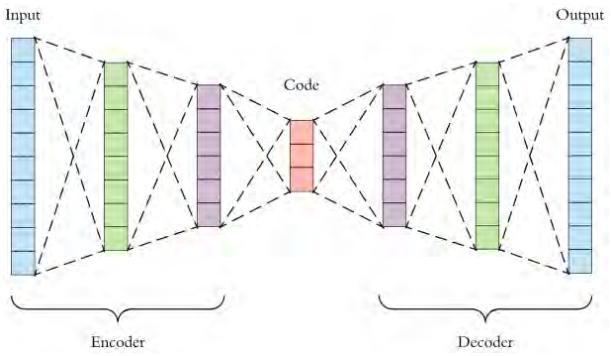


Figure 36: Illustration of an autoencoder structure. Image taken from Wolpe and de Waal (2019).

Choosing the right amount of hidden neurons and layers will, just like described in section 5.1.4, require trial and error. AE's are also usually designed in a symmetric way, just as figure 36 shows. However, this does not necessarily have to be the case.

5.1.6 SHAP with autoencoders

In order to explain why datapoints are classified as outliers or not, Antwarg et al. (2019) proposed a method on how to combine Kernel SHAP with outlier classification by AE's. To the best of our knowledge, this is the only paper

which proposes a method to explain anomalies found by an autoencoder. This section will explain how this method works.

The method tries to explain the reconstruction error calculated by the MSE. Before explaining how the method works, we first have to explain the notation in this section. $x_1, x_2, \ldots, x_n$ are the values for the n features of a specific instance in the dataset. $x'_1, x'_2, \ldots, x'_n$ are the values of the reconstructed features, calculated by the AE model f.

After calculating the difference for each feature of a specific instance, the method sorts the features in descending order of reconstruction error. This ordered list is written as $x_{(1)}, x_{(2)}, \ldots, x_{(n)}$. It therefore orders the features as such: $|x_{(1)} - x'_{(1)}| \geq |x_{(2)} - x'_{(2)}| \geq \cdots \geq |x_{(n)} - x'_{(n)}|$. The top M features in this list are stored in *topMfeatures*. Then, it uses Kernel SHAP in order to obtain the SHAP values for every feature i. The input for Kernel SHAP is the model f and a set of j background instances. This is summarized in algorithm 4, where *explainer.shapvalues(x,i)* returns the predicted value of feature i of instance X.

Algorithm 4: SHAP values for *topMfeatures*(Instance X, background-set $x_{1..j}$, Ordered list of error per feature *ErrorList*, Autoencoder model f)

ErrorList = ErrorList.order(descending)
topMfeatures = pick top M values from ErrorList
SHAP_of_top_M_features = Empty List
for *each* $i \in$ *topMFeatures* **do**
  explainer = shap.KernelExplainer(f, $X_{1..j}$)
  SHAP_of_top_M_features.append(explainer.shapvalues(X,i))
end

Now that all SHAP values are derived, they have to be divided into values contributing to the anomaly (meaning they push the predicted value away from the actual value) and offsetting the anomaly (meaning they push the predicted value towards the actual value). This process is shown in algorithm 5. In words, for every feature we first determine whether the actual value is higher than the predicted value or not. If this is the case, the contributing features contain negative SHAP values and the offsetting features contain positive SHAP values. If this is not the case, it is the other way around. Meaning that if the actual value is lower than the predicted value, the contributing features contain positive SHAP values and the offsetting features contain negative SHAP values.

The final step is to select the features with the highest SHAP values. The higher the SHAP value, the more important the feature is to the prediction. So, in order to know which features contributed the most to an outlier, the instances with the highest reconstruction error are selected and for all these instances the features with highest SHAP values are selected. The entire process is summarized in figure 37.

Algorithm 5: Divide SHAP into contributing and offsetting (list of SHAP values for *topMfeatures* *SHAP_of_top_M_features*, instance X, predicted value for instance X X'

```
ShapContribute = []
ShapOffsetting = []
for i in SHAP_of_top_M_features do
    for all SHAP_values of i do
        if x_i > x'_i then
            if SHAP_value > 0 then
                shapOffsetting.append(SHAP_value)
            else
                ShapContribute.append(SHAP_value)
            end
        else
            if SHAP_value > 0 then
                ShapContribute.append(SHAP_value)
            else
                shapOffsetting.append(SHAP_value)
            end
        end
    end
end
RETURN ShapContribute, ShapOffsetting
```

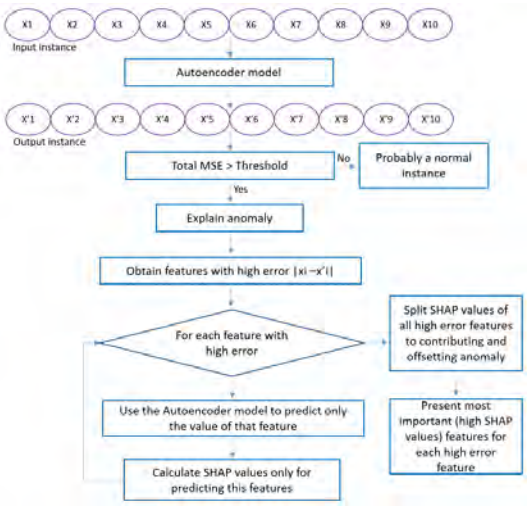


Figure 37: Flow chart of method for combining SHAP with AE anomaly detection.

5.2 Results of autoencoder

The autoencoder we used contained three hidden layers containing 25, 15 and 25 hidden neurons respectively. Different amounts of hidden neurons and layers were tested, but the differences of the results were negligible. The results of the anomaly scores are shown in figure 38 and the corresponding confusion matrix is shown in table 7. In figure 38, the points above the threshold of approximately 9.0571 are classified as outliers.

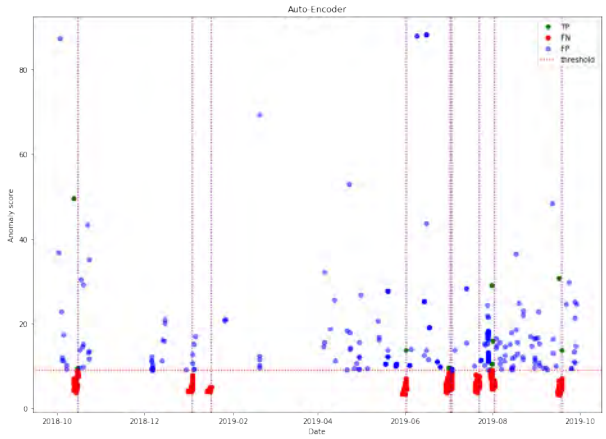


Figure 38: Results of autoencoder.

The results are very comparable to those of the k-NN algorithms and LOF. The amount of TP's is again very low (20) and all the outliers are placed seemingly random. Therefore, we concluded the same for the results of AE as for the previously mentioned algorithms, namely that the AE is not able to predict any malfunctions.

	Point in time 3 days before malfunction	**Point in time not 3 days before malfunction**
Outlier	20	284
No outlier	35624	469551

Table 7: Confusion matrix of autoencoder.

The usage of SHAP with AE is also a lot less functional compared to the usage of SHAP with IF. Because SHAP has to be used in the way mentioned in the previous section, it is impossible to plot the global and local result plots like figures 28 and 29. It is therefore a lot harder to conclude which features are the most important and whether a high or low value indicates an outlier or not. It is also a lot slower and less trustworthy since it uses Kernel SHAP instead of Tree SHAP.

Nevertheless, we still used the method mentioned in the previous section and analyzed the resulting SHAP values. It turned out that for the TP's, the top 3 most important features were the total electricity flow ('flow_3f'), the electricity flow of phase 2 and the electricity flow of phase 1. These features were the top 3 most important features for 17 out of 20 TP's (85%). For the other 3 TP's the outside temperature, the temperature of the bottom bearing and the frequency were considered to be the most important features.

When comparing these results to IF they do have in common that electricity flow is important. However, the SHAP results from AE indicate a low importance of vibration speed. Since the TP's of IF are considered more trustworthy in predicting upcoming failures than those of AE and Tree SHAP gives more precise results than Kernel SHAP, we trust the SHAP results of IF more than the SHAP results of the AE.

6 Conclusion and discussion

The main challenges for predictive maintenance are predicting upcoming failures and the ability to explain the results. Isolation forest turned out to be the best solution to these two problems. Based on the historical data, Isolation Forest is able to decrease the downtime of one of the water pumps of the new pumping station at Eefde by 6.07%.

Because of Tree SHAP, the Isolation Forest results are also relatively easy to explain. Tree SHAP is able to easily explain which features have caused a datapoint to be classified as an outlier so that a pump operator can decide whether or not an inspection crew should be send to find potential damage to the installation. This can reduce the amount of unnecessary inspections, which reduces costs. It also makes the model a lot more trustworthy for the user. The most important features were the difference between the values of the electricity currents of phases 1 and 3, the vibration speeds of the top bearings and the electricity current of the phases.

So, the answer to the research question *"Which unsupervised learning methods should be used in order to find outliers in sensor-data which indicate whether pro-active maintenance is required for a pumping station and is able to explain why the outliers are classified as such?"* is Isolation Forest, meaning that the hypothesis was also correct.

However, there is still much room for improvement. When looking at the malfunctions causing downtime, it is clear that the rinse water pump failures have caused the most amount of downtime in the period of 01/10/2018 up until 29/09/2019. 12 out of the 16.75 hours of downtime were caused by the rinse water pump (71.64%). We therefore recommend to start collecting data on the rinse water pump. Our recommendation is to collect temperature, pressure and bearing vibration speed data of the rinse water pump.

The recommendation for collecting temperature and pressure data is based on the malfunctions from last year. Temperature and pressure sensors could have easily prevented the downtime caused by the malfunctions. On top of that, in the case of the temperature, the melting of the pump could have been prevented which could have saved extra equipment costs. It would have also increased the safety of the employees present. Collecting bearing vibration speed data is recommended because of the SHAP results of Isolation Forest. These results showed that the bearing vibration speeds were important features for classifying outliers. Therefore, these values should also be taken into account when collecting data on the rinse water pump.

Collecting data on other parts of the installation will also make it easier to predict where the malfunction is happening. This can make the algorithm even more trustworthy and allows more precise searches for the inspection team, which saves both time and money.

Another recommendation we would like to give is to increase the measuring frequency. Some features, especially the vibration speeds of the bearings, can have significantly different values within a small time period. Sometimes even a few milliseconds. Receiving measurement for every second or maybe even a

few milliseconds instead of minutes could significantly improve the accuracy of the algorithm and explain the cause of a potential failure.

To conclude, Isolation Forest is a good algoritm for predicting upcoming failures. In order to make more precise predictions, however, adjustments to the data collection have to be done. First of all, more sensors have to be placed on specific parts of the installation, especially the rinse water pump. This will make the algorithm more precise, gain more trust and it makes it easier to point out the cause of a potential failure. The most important data features are the vibration speeds of the bearings and the (differences between the) electricity currents of the three phases. It is therefore also recommended to increase the measurement frequency, since the values of the vibration speeds of the bearings can differ a lot within much smaller periods of time than only one minute. Future research could determine whether these extra features and increased measurement frequencies will improve the results or not.

A List of abbreviations

AE	Autoencoder
AI	Artificial Intelligence
ANN	Artificial Neural Network
CbM	Condition-Based Maintenance
DBSCAN	Density-based spatial clustering of applications with noise
FN	False Negative
FP	False Positive
LOF	Local Outlier Factor
MSE	Mean Squared Error
NN	Nearest Neighbor
PdM	Predictive Maintenance
PC	Principal Component
PCA	Principal Component Analysis
RAM	Random-access memory
ROC	Receiver Operator Characteristic
rpm	rounds per minute
SHAP	SHapley Additive exPlanations
SVM	Support Vector Machine
TN	True Negative
TP	True Positive
UPS	Uninterruptible Power Supply

Table 8: Table of all abbreviations.

B Attributes

Attribute	Description	Unit
minute	Minute the measurements were made	time period
level_in	Height of the water of the IJsselmeer	mNAP (meters above average Northsea level)
level_waste	Height of dirt in water of the IJsselmeer	mNAP
level_out	Height of water in the Twenthekanaal	mNAP
stat_vacvalve_vp	Status of the vacuum valve of the vacuum pump	Integer code
stat_vacvalve_pump	Status of the vacuum valve of the regular pump	Integer code
stat_pres	Status of the presser	Integer code
stat_pump	Status of the pomp	Integer code
hours	Total amount of running hours (starting from 6062)	Amount of hours
frequency	Frequency of the motor	Hz
rotational frequency	Frequency at which the impeller inside the pump rotates	radial per second
stat_vcpump	Status of vacuum pump	Integer code
pressure_vacuum	Pressure produced by vacuum pump	Bar
temp_topbearing	Temperature of the top bearing	Degrees Celsius
temp_thrustbearing	Temperature of the thrust bearing	Degrees Celsius
temp_bottombearing	Temperature of the bottom bearing	Degrees Celsius
vibr_topbearing1_motor	Vibration speed of the first top bearing of the motor	mm/s
vibr_topbearing2_motor	Vibration speed of the second top bearing of the motor	mm/s
vibr_topbearing1_pump	Vibration speed of the first top bearing of the pump	mm/s
vibr_topbearing2_pump	Vibration speed of the second top bearing of the pump	mm/s
energy	Total amount of energy used (staring from 2033531)	KwH
flow_l1	Electricity current of first phase in 3 phase power	Ampere
flow_l2	Electricity current of second phase in 3 phase power	Ampere
flow_l3	Electricity current of third phase in 3 phase power	Ampere
flow_3f	Sum of the electricity currents of the three phases divided by $\sqrt{3}$	Ampere
flow_rate	Amount of water pumped per second	kubic meters
total_flow	Total amount of water pumped (starting from 10344703)	kubic meters
vacuum_active	Whether the vacuum pump is active or not	Boolean

Table 9: Table of all attributes and their description.

Attribute	Missing values amount	Percentage
minute	0	0.00%
level_in	323	0.06%
level_waist	289	0.06%
level_out	72	0.01%
stat_vacvalve_vp	88251	17.41%
stat_vacvalve_pump	88251	17.41%
stat_pres	88195	17.39%
stat_pump	88220	17.40%
hours	75726	14.94%
frequency	19279	3.80%
rotational frequency	6482	1.28%
stat_vcpump	88251	17.41%
pressure_vacuum	7998	1.58%
temp_topbearing	53049	10.46%
temp_thrustbearing	41992	8.28%
temp_bottombearing	50877	10.03%
vibr_topbearing1_motor	9632	1.90%
vibr_topbearing2_motor	107	0.02%
vibr_topbearing1_pump	170	0.03%
vibr_topbearing2_pump	39543	7.80%
energy	19344	3.82%
flow_l1	19316	3.81%
flow_l2	19324	3.81%
flow_l3	19321	3.81%
flow_3f	2	0.00%
flow_rate	19558	3.86%
total_flow	54926	10.83%
vacuum_active	76704	15.13%

Table 10: Table with total amount and percentage of missing values for each attribute.

C DBSCAN

DBSCAN, Density-based spatial clustering of applications with noise, is a clustering algorithm first proposed by Ester et al. (1996). This algorithm works by clustering datapoints based on their density. Clustering based on the density instead of on the distance can in certain cases have much better clustering results. This can clearly be seen when comparing figures 39 and 40. It is clearly seen that clustering based on density by DBSCAN works a lot better than clustering based on distance in this example. The distance clustering algorithm used in this example is CLARANS (Ng and Han (1994)). DBSCAN also does not cluster the outliers (as can be seen when comparing database 3), since it can automatically detect outliers. This makes it a very capable outlier detection clustering algorithm.

However, it also has a few disadvantages. First of all, it can be hard to choose the right parameters, since it requires two parameters - Minpts and Eps - which heavily affect the results. DBSCAN can also not cluster datasets with clusters which have high differences in the density. This is because there cannot be an optimal combination for Minpts and Eps for all clusters (Kriegel et al. (2011)).

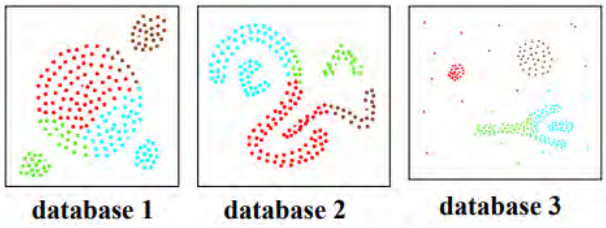


Figure 39: Three databases clustered by distance. Image taken from Ester et al. (1996).

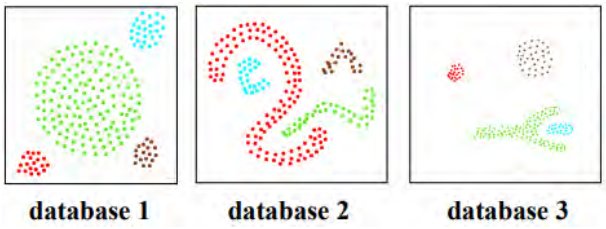


Figure 40: Three databases clustered by density. Image taken from Ester et al. (1996).

In order to explain how DBSCAN works, we first have to make multiple definitions.

- The Eps-neighborhood of a point p, denoted as $N_{Eps}(p)$, is defined as

$N_{Eps}(p) = \{q \in \Upsilon | \text{dist}(p,q) \leq \text{Eps}\}$. Intuitively, the EPS-neighborhood of a point p is the set of points of which the distance to these points is smaller than Eps.

- A point p is directly density-reachable from a point q w.r.t. Eps and MinPts (Minimum amount of points) if $p \in N_{Eps}(q)$ and $|N_{Eps}(q)| \geq$ MinPts (this is called the core point condition). It can be the case that a point p is directly density-reachable from q, while q is not directly density-reachable from p. An illustration of this is shown in figure 41.

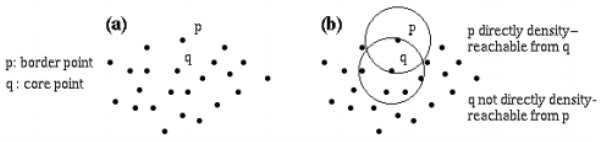


Figure 41: Illustration of how a point p can be directly density-reachable from q while q is not directly density-reachable from p. Image taken from Ester et al. (1996).

- A point p is density-reachable from a point q w.r.t. Eps and MinPts if there is a chain of points $p_1, \ldots, p_n$, where $p_1 = q$ and $p_n = p$ such that p_{i+1} is directly density-reachable from p_i. So, for example, if we look at figure 41 and set MinPts at 1 and Eps at such a value that every point p has a nearest neighbor q for which *dist(p,q)* $\leq$ Eps, every point would be density-reachable to all other points.

  However, just as with directly density-reachability, it is possible for a point p to be density-reachable from q, while q is not density-reachable from p. This is also due to the core point condition.

- A point p is density-connected to a point q w.r.t. Eps and MinPts if there is a point o such that both p and q are density-reachable from o w.r.t. Eps and MinPts.

With these definitions DBSCAN can finally define a cluster C and noise (outliers). DBSCAN defines a cluster as a non-empty subset of Υ where the following two conditions hold:

1. $\forall p, q$: if point $p \in C$ and point q is density-reachable from p w.r.t. Eps and MinPts, then $q \in C$.

2. $\forall p, q$: p is density-connected to q w.r.t. Eps and MinPts. This is called the connectivity condition.

All the datapoints in dataset Υ which are not located in any of the clusters C_i, where $i = 1, \ldots, k$, according to the conditions given above are then classified as outliers.

Now that we have all the definitions mentioned above we can explain how the algorithm works. The pseudo-code of DBSCAN is shown in algorithm 6. In words, clusterID is first set to NOISE. Then, the algorithm checks for every point in a set of points, which is either the whole database or a cluster discovered in a previous run, whether it is known that it is a cluster or noise. If this is not yet known, it checks whether a cluster can be extended. If this is the case, the clusterID is updated to a new cluster.

Algorithm 6: DBSCAN (database Υ , Eps, Minpts)

```
ClusterId = nextId(NOISE)
for all points p in SetOfPoints do
    if p.ClusterId = UNCLASSIFIED then
        if ExpandCluster(SetOfPoints, p, ClusterId, Eps, Minpts) then
            ClusterId = nextId(ClusterId)
        end
    end
end
```

The algorithm to check whether a cluster should be extended is shown in algorithm 7. The function regionQuery returns the eps-Neighborhood of the point p as a list. The clusterID which has been marked as noise may be changed to a cluster if the points are density-reachable from other points.

Algorithm 7: ExpandCluster(setOfPoints, p, ClusterId, Eps, Minpts)

```
Result: boolean
seeds = regionQuery(SetOfPoints, p, Eps)
if seeds.size smaller than MinPts then
    SetOfPoints.changeClusterId(p, NOISE)
    RETURN False
else
    SetOfPoints.changeClusterId(seeds, ClusterId)
    seeds.delete(p)
    while seeds is not empty do
        currentP = seeds.first()
        result = regionQuery(SetOfPoints, currentP, Eps)
        if result.size ≥ MinPts then
            for all points q in result do
                if q.ClusterId ∈ {UNCLASSIFIED, NOISE} then
                    if q.ClusterId = UNCLASSIFIED then
                        seeds.append(q)
                    end
                    SetOfPoints.changeId(q, ClusterId)
                end
            end
        end
        seeds.delete(q)
    end
    RETURN True
end
```

D Principal Component Analysis

Principal Component Analysis (PCA) is a dimensionality-reduction technique first proposed by Karl Pearson (1901). Meaning it can, for example, reduce 3 dimensions in a dataset into 2 dimensions. This can be extremely useful for multiple purposes. First of all, when reducing the dimensions, the most important patterns in the data are kept and the noise is removed. Autoencoder based anomaly detection is also based on this principal (as shown in section 5.1). Second of all, it can help with clustering since less dimensions can improve the accuracy of clustering algorithms due to the curse of dimensionality, which is shown in section 4.1. And last but not least, 3+ dimensions can be reduced to two or three dimension so they can be easily visualized and analyzed by humans.

The first step before applying PCA is to standardize the data. This means that all features have to be scaled to the same size. If this is not the done, features with higher scales can incorrectly dominate features with smaller scales (this will become clear later). To do this, one can use the formula shown in equation (32), where $x_i \in X$, $i = 1, \ldots, n$ is the 'normal' value of the n values of attribute X, $\bar{x}$ and σ_x are the average and standard deviation of all n values of X and z_i is the calculated standardized value.

$$z_i = \frac{x_i - \bar{x}}{\sigma_x} \tag{32}$$

The next step is to calculate the covariance matrix. The covariance matrix is a squared symmetric matrix, meaning that its amount of columns equals the amount of rows (squared) and that it is equal to its transpose (symmetric). The amount of columns and rows is equal to the amount of dimensions (p). The entry of the covariance matrix $x_{i,j}$ where $i \neq j$ is the covariance between attributes i and j. The other entries, where $i = j$, are the variances of attribute i (the covariance between sets X and X results in the variance of set X). The equation for calculating the covariance is shown in equation (33), where Z_i and Z_j, $i, j = 1, \ldots, p$, are the regularized attributes of the dataset, N is the amount of samples and Z_{ik} is the kth value of attribute Z_i.

$$Cov(Z_i, Z_j) = \frac{\Sigma_{k=1}^{N} \left(Z_{ik} - \bar{Z}_i\right) \left(Z_{jk} - \bar{Z}_j\right)}{N} \tag{33}$$

After calculating the covariance matrix, the eigenvectors and eigenvalues of this matrix have to be computed. These are used to determine the Principal Components (PC). PC's are new variables that consist of linear combinations or mixtures of the original data. The amount of PC's is equal to the amount of attributes of the original data. These PC's have two properties. The first is that they are fully uncorrelated. The second is that the most information is placed in the first PC, the second-most information in the second PC, etc. This way dimension can be reduced by removing the last PC's.

The first PC is a vector which maximizes the variance, the second PC also maximizes the variance, but with an additional condition, namely that it is independent (i.e. perpendicular) to the first PC. The third PC is then the PC

which maximizes the variance with the conditions that it is independent from both PC1 and PC2. This goes on until p PC's are made.

That is why the eigenvectors and eigenvalues are needed. The eigenvector of the covariance matrix is namely the vector containing the direction of the axis with the highest variance, so the eigenvector is the PC. The eigenvalue is then the coefficient which gives the amount of variance contained in the PC.

These eigenvectors and eigenvalues are derived from the formula shown in equation (34), where $\mathbf{A}$ is the covariance matrix, $\mathbf{I}$ is the identity matrix and λ is the coefficient to be derived, which will eventually give the values for all eigenvalues. The identity matrix has a value of 1 as entries for the main diagonal (meaning $I_{ij} = 1$ for $i = j$) and 0 for all other entries.

$$det\left(\mathbf{A} - \lambda\mathbf{I}\right) = 0 \tag{34}$$

After calculating the eigenvalues, the eigenvectors can be derived. This is done by using the following formula: $\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$, where $\mathbf{A}$ is again the covariance matrix, $\mathbf{x}$ is the eigenvector and λ is the eigenvalue. This formula is characteristic for eigenvectors and eigenvalues. It can then be rewritten to $\left(\mathbf{A} - \lambda\mathbf{I}\right)\mathbf{x} = 0$, where 0 is a vector containing only zeros. This can be solved by solving the equation $\mathbf{A} - \lambda\mathbf{I} : \mathbf{0}$ for every previously derived eigenvalue and converting it to row echelon form. After reducing the matrix to row echelon form the entries for the eigenvector $\mathbf{x}$ can be derived.

After deriving all eigenvectors from their corresponding eigenvalues they have to be sorted in descending order based on the eigenvalue. This is done because the eigenvalue is the value for the amount of variance contained in the eigenvector (PC) and the first PC should contain the most information. The reason for regularizing the data first (as mentioned earlier) is that without regularization, the entries in the covariance matrix can vary a lot. This will cause extreme differences in the eigenvalues and therefore also eigenvectors. This will make it look like the first PC's contain almost all information, while this is only because the range of the values is a lot higher, which does not necessarily mean that they contain more information. If all values are in the same ranges this will not be a problem.

In order to derive an anomaly score by using PCA, the projection of the new input on the previously calculated eigenvectors is calculated. The difference between the original value and the projection is called the deviation score. The bigger the deviation score, the more anomalous the point.

References

C. C. Aggarwal. *Outlier Analysis*. Springer Publishing Company, Incorporated, 2nd edition, 2016. ISBN 3319475770.

L. Antwarg, R. M. Miller, B. Shapira, and L. Rokach. Explaining anomalies detected by autoencoders using shap, 2019.

Y. Anzai. 10 - learning by neural networks. In Y. Anzai, editor, *Pattern Recognition & Machine Learning*, pages 297 – 335. Morgan Kaufmann, San Francisco, 1992. ISBN 978-0-12-058830-5. doi: https://doi.org/10.1016/B978-0-08-051363-8.50014-4.

R. Beebe. Predictive maintenance of pumps using condition monitoring. *Predictive Maintenance of Pumps Using Condition Monitoring*, pages 1–181, 04 2004.

R. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition, 1957.

K. J. Berry, P. W. Mielke, and J. E. Johnston. *Completely Randomized Data*, pages 29–55. Springer International Publishing, Cham, 2016. ISBN 978-3-319-28770-6. doi: 10.1007/978-3-319-28770-6_2.

C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 0387310738.

S. Biswal and G. R. Sabareesh. Design and development of a wind turbine test rig for condition monitoring studies. In *2015 International Conference on Industrial Instrumentation and Control (ICIC)*, pages 891–896, 2015.

M. Breunig, H.-P. Kriegel, R. Ng, and J. Sander. Lof: Identifying density-based local outliers. volume 29, pages 93–104, 06 2000. doi: 10.1145/342009.335388.

T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

J. Cheng, W. Chen, and Q. Wang. Data-driven predictive maintenance planning framework for mep components based on bim and iot using machine learning algorithms. *Automation in Construction*, 112, 02 2020. doi: 10.1016/j.autcon.2020.103087.

F. Chollet et al. Keras. `https://keras.io`, 2015.

I. N. da Silva, D. H. Spatti, R. A. Flauzino, L. H. B. Liboni, and S. F. dos Reis Alves. *Artificial Neural Networks*, pages 21–28. Springer, Cham, Cham, 2017. ISBN 978-3-319-43162-8. doi: 10.1007/978-3-319-43162-8.

L. Duan, L. Xu, Y. Liu, and J. Lee. Cluster-based outlier detection. *Annals OR*, 168:151–168, 04 2009. doi: 10.1007/s10479-008-0371-9.

M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. pages 226–231. AAAI Press, 1996.

G. Fan, J. Li, and H. Hao. Lost data recovery for structural health monitoring based on convolutional neural networks. *Structural Control and Health Monitoring*, 26(10):e2433, 2019. doi: 10.1002/stc.2433. e2433 STC-19-0088.R1.

N. Frost, M. Moshkovitz, and C. Rashtchian. Exkmc: Expanding explainable k-means clustering, 2020.

D. M. Hawkins. *Identification of outliers / D.M. Hawkins*. Chapman and Hall London ; New York, 1980. ISBN 041221900.

J. Heaton. *Artificial Intelligence for Humans, Volume 3: Deep Learning and Neural Networks*. Artificial Intelligence for Humans. Createspace Independent Publishing Platform, 2015. ISBN 9781505714340. URL `https://books.google.nl/books?id=q9mijgEACAAJ`.

F. Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901. doi: 10.1080/14786440109462720.

D. E. Knuth. *The Art of Computer Programming, Volume 3: (2nd Ed.) Sorting and Searching*. Addison Wesley Longman Publishing Co., Inc., USA, 1998. ISBN 0201896850.

H.-P. Kriegel, P. Kröger, J. Sander, and A. Zimek. Density-based clustering. *WIREs Data Mining and Knowledge Discovery*, 1(3):231–240, 2011. doi: 10.1002/widm.30.

D. J. Leinweber. Stupid data miner tricks. *The Journal of Investing*, 16 (1):15–22, 2007. ISSN 1068-0896. doi: 10.3905/joi.2007.681820. URL `https://joi.pm-research.com/content/16/1/15`.

F. T. Liu, K. M. Ting, and Z. Zhou. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422, 2008.

S. Lundberg and S.-I. Lee. An unexpected unity among methods for interpreting model predictions, 2016.

S. M. Lundberg and S.-I. Lee. A unified approach to interpreting model predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4765–4774. Curran Associates, Inc., 2017.

S. M. Lundberg, B. Nair, M. S. Vavilala, M. Horibe, M. J. Eisses, T. Adams, D. E. Liston, D. K.-W. Low, S.-F. Newman, J. Kim, et al. Explainable machine-learning predictions for the prevention of hypoxaemia during surgery. *Nature Biomedical Engineering*, 2(10):749, 2018.

S. M. Lundberg, G. Erion, H. Chen, A. DeGrave, J. M. Prutkin, B. Nair, R. Katz, J. Himmelfarb, N. Bansal, and S.-I. Lee. From local explanations to global understanding with explainable ai for trees. *Nature Machine Intelligence*, 2(1): 2522–5839, 2020.

M.-A. Lutz, S. Vogt, V. Berkhout, S. Faulstich, S. Dienst, U. Steinmetz, C. Gück, and A. Ortega. Evaluation of anomaly detection of an autoencoder based on maintenace information and scada-data. *Energies*, 13(5), 2020. ISSN 1996-1073. doi: 10.3390/en13051063.

R. D. Maesschalck, D. Jouan-Rimbaud, and D. Massart. The mahalanobis distance. *Chemometrics and Intelligent Laboratory Systems*, 50(1):1 – 18, 2000. ISSN 0169-7439. doi: https://doi.org/10.1016/S0169-7439(99)00047-7.

G. Mao. Efficient penalized estimation for linear regression model. *Communications in Statistics - Theory and Methods*, 44:141217112804005, 12 2014. doi: 10.1080/03610926.2012.763094.

K. G. Mehrotra, C. K. Mohan, and H. Huang. *Anomaly Detection Principles and Algorithms*. Springer Publishing Company, Incorporated, 1st edition, 2017. ISBN 3319675249.

R. K. Mobley. An introduction to predictive maintenance. Elsevier, 2002.

K. E. Mokhtari, B. P. Higdon, and A. Başar. Interpreting financial time series with shap values. In *Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering*, CASCON '19, page 166–172, USA, 2019. IBM Corp.

M. Moleda, A. Momot, and D. Mrozek. Predictive maintenance of boiler feed water pumps using scada data. *Sensors*, 20:571, 01 2020. doi: 10.3390/s20020571.

A. Morichetta, P. Casas, and M. Mellia. Explain-it. *Proceedings of the 3rd ACM CoNEXT Workshop on Big DAta, Machine Learning and Artificial Intelligence for Data Communication Networks - Big-DAMA '19*, 2019. doi: 10.1145/3359992.3366639. URL `http://dx.doi.org/10.1145/3359992.3366639`.

R. T. Ng and J. Han. Efficient and effective clustering methods for spatial data mining. Technical report, CAN, 1994.

P. Park, P. Di Marco, H. Shin, and J. Bang. Fault detection and diagnosis using combined autoencoder and long short-term memory network. *Sensors*, 19:4612, 10 2019. doi: 10.3390/s19214612.

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

S. Rathi. Generating counterfactual and contrastive explanations using SHAP. *CoRR*, abs/1906.09293, 2019. URL `http://arxiv.org/abs/1906.09293`.

M. T. Ribeiro, S. Singh, and C. Guestrin. "why should i trust you?": Explaining the predictions of any classifier, 2016.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. Mcclelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, pages 318–362. MIT Press, Cambridge, MA, 1986a.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–336, 1986b. URL `https://doi.org/10.1038/323533a0`.

M. Sakurada and T. Yairi. Anomaly detection using autoencoders with nonlinear dimensionality reduction. In *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*, MLSDA'14, page 4–11, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450331593. doi: 10.1145/2689746.2689747.

M. Schmitt, L. Hughes, M. Körner, and X. Zhu. Colorizing sentinel-1 sar images using a variational autoencoder conditioned on sentinel-2 imagery. 06 2018.

L. S. Shapley. A value for n-person games. In *Contributions to the Theory of Games 2.28*, page 307–317, 1952.

H. Shatkay. The fourier transform-a primer. 1995.

A. Shrikumar, P. Greenside, A. Shcherbina, and A. Kundaje. Not just a black box: Learning important features through propagating activation differences, 2016.

P. C. Thyago, A. A. M. N. S. Fabrízzio, V. Roberto, R. da P. F., P. B. João, and G. S. A. Symone. A systematic literature review of machine learning methods applied to predictive maintenance. *Computers Industrial Engineering*, 137: 106024, 2019. ISSN 0360-8352. doi: https://doi.org/10.1016/j.cie.2019.106024.

W. Tiddens, A. Braaksma, and T. Tinga. Selecting suitable candidates for predictive maintenance. *International Journal of Prognostics and Health Management*, 9(1), 5 2018. ISSN 2153-2648.

N. Venkat. The curse of dimensionality: Inside out. 09 2018. doi: 10.13140/RG.2.2.29631.36006.

M. A. Wani, F. A. Bhat, S. Afzal, and A. I. Khan. *Unsupervised Deep Learning Architectures*, pages 77–94. Springer Singapore, Singapore, 2020. ISBN 978-981-13-6794-6. doi: 10.1007/978-981-13-6794-6_5.

Z. Wolpe and A. de Waal. Autoencoding variational bayes for latent dirichlet allocation. 12 2019.

B. Yang, J. Cao, R. Ni, and L. Zou. Anomaly detection in moving crowds through spatiotemporal autoencoding and additional attention. *Advances in Multimedia*, 2018:1–8, 09 2018. doi: 10.1155/2018/2087574.

J. Yu, C. Hong, Y. Rui, and D. Tao. Multitask autoencoder model for recovering human poses. *IEEE Transactions on Industrial Electronics*, 65(6):5060–5068, 2018.

Y. Yuan, X. Jiang, and X. Liu. Predictive maintenance of shield tunnels. *Tunnelling and Underground Space Technology*, 38:69 – 86, 2013. ISSN 0886-7798. doi: https://doi.org/10.1016/j.tust.2013.05.004.

Y. Zhao, Z. Nasrullah, and Z. Li. Pyod: A python toolbox for scalable outlier detection. *Journal of Machine Learning Research*, 20(96):1–7, 2019. URL `http://jmlr.org/papers/v20/19-011.html`.

Z. Zhao, F. li Wang, M. xing Jia, and S. Wang. Predictive maintenance policy based on process data. *Chemometrics and Intelligent Laboratory Systems*, 103(2):137 – 143, 2010a. ISSN 0169-7439. doi: https://doi.org/10.1016/j.chemolab.2010.06.009.

Z. Zhao, F.-l. Wang, M.-x. Jia, and S. Wang. Predictive maintenance policy based on process data. *Chemometrics and Intelligent Laboratory Systems - CHEMOMETR INTELL LAB SYST*, 103:137–143, 10 2010b. doi: 10.1016/j.chemolab.2010.06.009.