Cornelius Grieg Dahling

Anomaly Detection of Sensors in Unsupervised Long Time-Series Data

master project, spring 2019

Department of Computer and Information Science
Faculty of Information Technology, Mathematics and Electrical Engineering

Abstract

The maintenance conducted on large-scale machinery, amount to one of the machinery's main costs, which is why the machines are fitted with a large number of sensors. With the rise of Machine Learning, there have been a lot of implementations trying to detect anomalies in time-series data of sensors. However, there has been a lack of unsupervised methods used in time-series sensor anomaly detection. In this thesis, a Long Short-Term Memory network is used to predict the values of the sensors at the current timestep in an unsupervised manner.

The data used for this thesis is long time-series data from one of Equinor's turbines. As input for the network, feature selection is used with both Random Forest Regression and Hierarchical Agglomerative Clustering (HAC), with Random Forest Regression consistently resulting in better prediction.

In most cases, the network is able to learn correlations between the feature selected sensors and the one we want to predict. The experiments conducted show promising results, with an average Mean Absolute Percentage Error of 0.558 when selecting sensors with Random Forest Regression.

Sammendrag

Å vedlikeholde stort maskineri, er noen av de største kostnadene som fører med til drift av store maskiner. Med den stadig økende populæriteten til Maskinlæring, har vi de siste årene sett stadig nye metoder for å finne feil i sensor data. Likevel, har det vært mangel på ikke-veildet algoritmer som kan brukes for å finne feil i tidsserier av sensordata. I denne avhandlingen bruker vi et Long Short-Term Memory nettverk for å forutsi verdiene til sensorene i det nåværende tidstrinnet.

Dataen som er brukt i denne avhandlingen er lange tidsserier fra en av Equinor sine turbiner. Hvilke sensorer som skal brukes som input i nettverket er testet med både Tilfeldige Skoger for Regresjon (Random Forest Regression) og Hierarisk Klyngeanalyse (Hierarchical Agglomerative Clustering), hvor Tilfeldige Skoger for Regresjon konsekvent gir best forutsetninger.

Nettverket er i stand til å lære sammenhenger mellom sensorverdiene som er valgt og den som skal spås. De gjennomførte eksperimentene viser lovende resultater, med en gjennomsnittlig Mean Absolute Percentage Error på 0.558 for sensorene valgt med Tilfeldige Skoger for Regresjon.

Preface

This thesis was finalized during spring of 2019, as part of the completion of a 5 year Master of Science (MSc) in Software engineering at the Norwegian University of Science and Technology, Faculty of Information Technology and Electrical Engineering, Department of Computer Science.

I would like to thank my supervisors, Jingyue Li and Zhe Li for their guidance and expertise in conducting this project. I would also like to thank Equinor for entrusting me with their turbine dataset, allowing me to explore the possiblity of anomaly detection in Long Time-Series Data.

Cornelius Grieg Dahling
Trondheim, June 15, 2019

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

This chapter introduces the thesis. Here the reader will find background information and the motivation for conducting the project. This includes concise information of Equinor's previous work. The reader will also be informed of the structure of the remainder of the thesis in section 1.5.

1.1 Motivation

Maintenance is an integral part of large-scale machinery. Most, if not all companies working with large scale machinery will tell you that it is cheaper to conduct maintenance on a regular, scheduled basis, as opposed to a piece of machinery breaking down, shutting down the entire production. In the case of total failure, your maintenance costs skyrockets. You will have delays in productions, perhaps customers who are not getting the product or service as agreed, and workers who are unable to complete their tasks. It is a situation you want to avoid, and this is why we conduct preventive maintenance. It is also why large-scale machinery is usually fitted with a large number of sensors, to monitor various aspects such as temperature and pressure. These sensors can be vital in warning the operators if something is not as it should, and can be essential in determining when to conduct maintenance. However, performing maintenance can be expensive and one also has to consider that there are times when sensors are malfunctioning. A sensor might be indicating a fault, when, in fact, the system is in perfect health. Rather than conducting maintenance, every time a sensor gave off a warning, it would be useful to verify that the sensor is functioning correctly, before addressing the warning.

Equinor is an international energy company. They recently re-branded from Statoil (direct translation is state oil), as they are focusing on more than just oil

nowadays. They have more than 20 000 employees, operating in more than 30 countries [Equinor, 2019]. Working in energy production, Equinor maintains a lot of large-scale machinery with complex networks of sensors, which are pivotal in determining the machinery's health.

Equinor are hoping to create a system which can verify the sensor values in these machines. They operate multiple, large turbines, which generate energy. These turbines have > 100 sensors that continuously monitor the health of the turbine. Currently, Equinor is doing well with condition-based maintenance. However, as described in the first meeting with Equinor (Appendix A), they have noticed that they are conducting unnecessary maintenance. Equinor is, therefore, looking to find ways to reduce maintenance costs. One of the ways to reduce these costs is to verify that sensor warnings are correct.

Equinor, therefore, wants a system that can be used to verify or disprove the results of sensor warnings. This system will keep Equinor from conducting unnecessary maintenance, which will save cost. They also specify that the system created has to be generalized because the various turbines do not necessarily have the same sensors, and the sensors are often not mapped together.

1.2 Goals and Research Questions

This section showcases the research questions for this thesis. Here the goal of the project is stated, which will be the success criteria for evaluating whether the project has achieved the desired outcome.

Goal Can sensor values in turbines be verified?

This project aims to use time-series data of sensor values from Equinor's turbine to be able to determine the correct value of a sensor at a given time.

Rearch question 1 Can multivariate time-series data with missing values be pre-processed to remove missing values while maintaining enough data time-stamps to train and test the model(s) used in this project.

It is important that after preprocessing the data, there are still enough rows to train a neural network.

Research question 2 Is it possible to correctly determine which sensor values are useful for predicting each other?

These large-scale machinery have an abundance of sensors located over a relatively large area. It, therefore, stands to reason that not all sensors will be useful in predicting the values of a sensor x (where x is any arbitrary sensor value). Therefore to be able to verify a sensor value x, which sensor values *a, b, c* are needed?

Research question 3 Is it possible to build a system that uses deep learning algorithms to predict a sensor value based on n other sensor values?

The project goal of verifying the values of sensors requires building a system that can predict the sensors' value.

Research question 4 Is it possible to verify that the model's prediction are precise?

To be able to evaluate the success of the project, it is crucial that methods are used to verify or disprove the findings.

1.3 Research Method

This thesis follows the design science research, which is a research technique, in which the researcher better understands the system by creating new theory by the creation of information systems or analyzing existing systems and their utility. [Vaishnavi, V., Kuechler, W., and Petter, S., 2017].

1.4 Contributions

This thesis contributes evidence to showing that state-of-the-art Long Short-Term Memory models are able to be used for anomaly detection of unsupervised data. It also evaluates Random Forest Regression and Hierarchical Agglomerative Clustering, respectively as methods for feature selection.

1.5 Thesis Structure

Next is chapter 2, where we put forward the fundamental theory and techniques required to follow the rest of this thesis.

In chapter 3, we discuss and evaluate related work that has inspired the system built for this project.

In chapter 4, we look at the structure of our implementation and its components.

In chapter 5 we explain how our experiments were set up and conducted. In section 5.4, we then showcase the results that stemmed from these experiments.

Finally in chapter 6 we evaluate our results, discuss the limitations of the created system, discuss this thesis' contributions, before looking at possible improvements and future work.

Chapter 2

Background Theory

In this chapter, the reader is informed on the theory required to follow this paper. In section 2.1, we explain what time-series analysis is. In section 2.2 and 2.3 we explain clustering and feature importance respectively, as methods for feature selection. The background theory concludes with an in-depth explanation of Recurrent Neural Networks and specifically Long Short-Term Memory networks, in section 2.4

2.1 Time-series analysis

Time-series data is a sequence of continuous, real-valued elements. It is a type of dynamic data because the data values change as a function of time. Observations at different timestamps are correlated [Wei, 2013]. This makes a lot of the statistical analysis tools ineffective, or even irrelevant. The difficulty of time-series analysis stems from not being able to isolate data points, but to look at the overall structure of the data.

2.2 Clustering

"Clustering is a data mining technique where similar data are placed into related or homogeneous groups without advanced knowledge of the group's definitions" [Aghabozorgi et al., 2015]. What this means is that when clustering, one is trying to group items which are most similar in the same group. Clustering is useful for exploring data, as it can be difficult to determine patterns in unlabeled data, and organizing it into groups can then give it more structure. Clustering is also often used as a pre-processing stage for more advanced data analysis methods.

2.2.1 Similarity measure

Similarity measure, or distance measure (as it is also known as), is the distance between data points. In order to compare time-series with different intervals, it is very important to figure out the correct similarity measure. Research suggests that the most effective similarity measures are the ones utilizing dynamic programming, such as Dynamic Time Warping. Dynamic Time Warping is a non-linear *elastic* function that allows similar time-series shapes to match even if the distance between the data points is large [Müller, 2007]. However these dynamically programmed clustering algorithms are also the most computationally expensive, having a cost of $O(m^2)$, where m is the length of the time-series. The most common methods for clustering time-series is still therefore *Euclidean distance*. The Euclidean distance between two data points i and j is simply the absolute value of the difference of the two data points' value, and can be expressed as $x(i, j) = |X_i - X_j|$.

2.2.2 Hierarchical Agglomerative Clustering

Hierarchical Agglomerative Clustering (HAC) uses a bottom-up approach, meaning each data point is considered an individual cluster, and for each iteration the closest pairs of clusters are merged (based on the similarity measure). The Clustering process stops when all clusters have been merged into a single cluster. The clustering process can then be visualized as a dendrogram, as can be seen in figure 2.2. The fact that the clustering process is visualized, makes it useful in using in situations where the user has little to no domain knowledge.

Initially, when all clusters are singular data points, finding the closest pairs, will purely be done by the similarity measure. However, as the clusters add more and more items, several different methods can be used to calculate the pairings. Some examples are *Nearest Point algorithm*, *Farthest Point algorithm*, *Centroid* and *Ward*. Nearest Point and Farthest Point algorithm are simply finding the pairing based on closest data point in the cluster and the one furthest away, respectively. The most common methods can be found in table 2.1.

Cophentic Correlation Coefficient

Which method works best, is highly dependent on the problem you are trying to solve. Therefore one can evaluate how well the clustering works using the Cophenetic Correlation Coefficient (CCC). The CCC *"...is a measure of how faithfully a dendrogram preserves the pairwise distances between the original unmodeled data points."*[Saraçli et al., 2013]. What this means is that we are determining the distance between the matrix of similarity measure and the dendrogrammatic distance, where the dendrogrammatic distance is the distance between two data items, when they were clustered. Assume you have 2 clusters, *1* and *2*. Before

Name	Distance update formula FORMULA for $d(I \cup J, K)$	Cluster dissimilarity between clusters A and B
single	$\min(d(I,K), d(J,K)$	$\min_{a \in A, b \in B} d[a,b]$
complete	$\max(d(I,K), d(J,K)$	$\max_{a \in A, b \in B} d[a,b]$
average	$\frac{n_I d(I,K) + n_j d(J,K)}{n_I + n_j}$	$\frac{1}{\lvert A \rvert \lvert B \rvert} \sum_{a \in A} \sum b \in B d[a,b]$
weighted	$\frac{d(I,K) + n_J d(IJ,K)}{n_I + n_J}$	
ward	$\sqrt{\frac{(n_I + n_K) d(I,K) + (n_J + n_K) d(J,K) - n_K d(I,J}{n_I + n_J + n_K}}$	$\sqrt{\frac{2[A\|\|B]}{[A\|+\|B]} \cdot \lvert\lvert \overrightarrow{c}_A - \overrightarrow{c}_B \rvert\rvert_2}$
centroid	$\sqrt{\frac{n_I d(I,K) + n_J d(J,K)}{n_I + n_J} - \frac{n_I n_J d(I,J)}{(n_I + n_J)^2}}$	$\lvert\lvert \overrightarrow{c}_A - \overrightarrow{c}_B \rvert\rvert_2$
media	$\sqrt{\frac{d(I,K)}{2} + \frac{d(J,K)}{2} - \frac{d(I,J)}{4}}$	$\lvert\lvert \overrightarrow{w}_A - \overrightarrow{w}_B \rvert\rvert_2$

Table 2.1: Formulas for clustering methods. *"Let I, J be two clusters joined into a new cluster, and let K be any other cluster. Denote by n_I , n_J and n_K the sizes of (i.e. number of elements in) clusters I, J, K, respectively."* reprinted from [Müllner, 2011].

these 2 clusters are combined (clustered), cluster 1 contains item $A \rightarrow 1 : \{A\}$, while cluster 2 contains item B, C and $D \rightarrow 2 : \{B, C, D\}$. The Euclidean distance (for example) between pair (A and B) and pair (A and C) will not be the same (unless the data points have the exact same value), but since they are in the same cluster, their dendrogrammatic distance will be the same. Therefore, it also means that for either one or both of B and C, the dendrogrammatic distance will not be equal to the Euclidean distance, which means we will not get a perfect CCC of 1. Continuing with the Euclidean distance as our choice of similarity measure, as $x(i,j)$ and defining the dendrogrammatic distance as $t(i,j)$. The cophenetic correlation coefficient is then as decribed in figure 2.1.

$$c = \frac{\sum_{i<j}(x(i,j) - x)(t(i,j) - t)}{\sqrt{[\sum_{i<j}(x(i,j) - x)^2][\sum_{i<j}(t(i,j) - t)^2]}}$$

Figure 2.1: Formula for Cophenetic Correlation Coefficient reprinted from [Saraçli et al., 2013]

Stopping criterion

Once the correct method is determined, the dendrogram can be used visually

to evaluate a criterion to stop clustering. Since the HAC algorithm will keep clustering until a single cluster remains, a stopping criterion needs to be determined. Using the dendrogram, one can determine an adequate stopping criterion, even without having any domain knowledge. HAC always clusters by choosing the closest clusters first, therefore one can set a max distance for the clustering. Once the max distance is set, this can be used to form flat clusters.

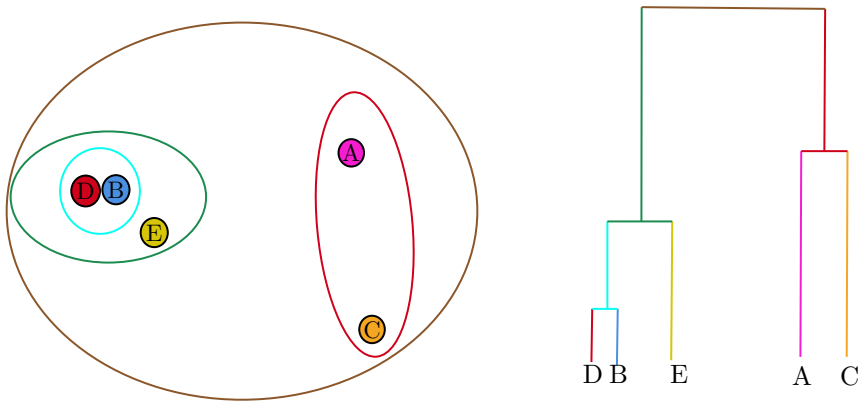


Figure 2.2: Example of how Hierarchical Agglomerative Clustering works. A, B, C, D and E are data points. This example highlights the Hierarchical Agglomerative Clustering process, by showing that all the data points ends up in a single cluster (left) and the clustering process is visualized in the dendrogram (right). Created using [Nha and Nhat, 2019]. Notice that both (D and B) will form a cluster and then merge with E to form cluster (B, D, E) before A and C are merged. This is because A and C are the data points that are furthest away from each other. This is highlighted in the dendrogram, with A and C having longer *lines*, which represent the (clustering method) distance measure from table 2.1.

2.3 Feature Importance

The feature Importance of a feature is its *"discriminative power in distinguishing a target of interest from other individuals."*[Liu et al., 2012]. Feature importance, as the name implies is the importance a feature y is in predicting feature x. If you change the order of the data for feature y, the prediction error will increase for feature x if it is an important feature, meaning the feature importance is high.

2.3.1 Random Forest

One of the most popular Feature importance methods, is Random Forest Regression. The Random Forest method can be used for both classification and regression. It is an ensemble learning method [Breiman, 2001]. What this means is that the Random Forest method uses multiple learning algorithms. This achieves better performance, in terms of increasing the accuracy and preventing overfitting. The name Random Forest comes from the algorithm using a *forest* of decision trees. Random Forest uses a modified version of Bootstrap Aggregation, also known as Bagging. Bagging is used to reduce the variance, in cases where the variance is high, which is the case with decision trees [Bühlmann and Yu, 2002]. Bagging, as the term suggests, is the process of placing sub-samples of the data into *bags* with replacement. A certain amount of *bags* of data are used to create sub-models from the original dataset, before the results are combined [Breiman, 1996]. Random Forest, improves on this technique. In order to avoid overfitting, Random Forest reduces the correlation between the sub-models as much as possible. Rather than allowing the bagging to happen from the entire dataset, Random Forest's improved technique only allows the bagging to happen from a randomly chosen subset of the data, that is selected at each time-step. As explained by the creator of Random Forest, Breiman *"In my experiments with random forests, bagging is used in tandem with random feature selection."*[Breiman, 2001].

An example of a Random Forest Classifier predicting which season we are in, can be seen in figure 2.3.

2.3.1.1 Random Forest Regression for feature selection

Unlike with Random Forest Classification, Random Forest Regression has to have real values rather than a label. Random Forest Regression works by taking the average real-valued output from the decision trees, using the formula $\{h(x, \theta_k)\}$, where θ represents a random vector and k is the number of trees, and x is a feature from the training set [Breiman, 2001].

As mentioned in [Hapfelmeier and Ulm, 2013] *"Random Forests are also used as a means to distinguish relevant from irrelevant variables in variable selection approaches"*, which means that by calculating the feature importance, one is able to determine which features are important in predicting each other.

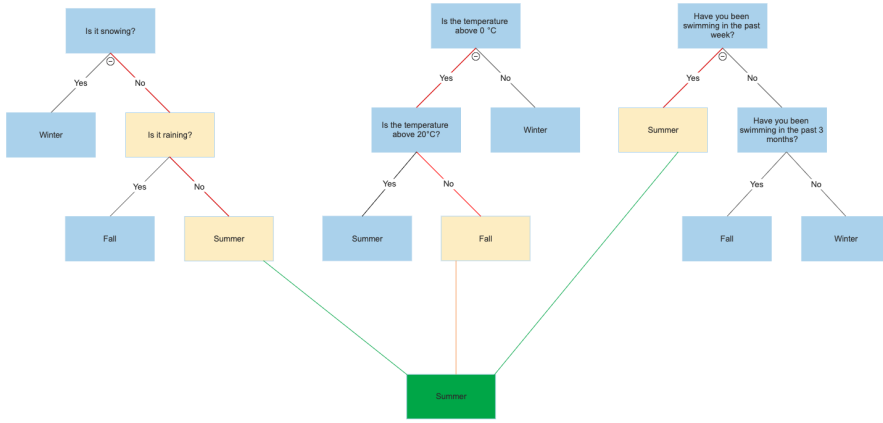


Figure 2.3: A simplified example of a Random Forest Classification with 3 decision trees, working to determine if the season is summer, fall or winter. In this example, we are experiencing a very cold **summer**, which means that for the second decision tree, the outcome is fall, when in fact it should be summer. However because the majority of decision trees decide the classification, the season is correctly classified as summer. The diagram was created using SmartDraw.

2.4 Recurrent Neural Networks

Recurrent Neural Networks (RNN) are Artificial Neural Networks (ANN) that incorporate a memory component. *"Unlike standard feedforward neural networks, recurrent networks retain a state that can represent information from an arbitrarily long context window."*[Lipton, 2015].

In many situations, it is not enough for the ANN to learn a series of patterns, it may also need to know what has happened before. Predicting hand-written digits, is a standard starting point for learning about ANN. All you need is a simple network and to train it on x number of training data and you are able to predict the hand-written digits. Every single new instance is isolated. What digit you are prediciting now, is irrelevant to what digit you will be predicting afterwards. On the other hand, you may have a situation in which you are trying to understand the context of a sentence. For example, take the sentence, *The car is blue.* It is simple enough for a person to understand the context of what is blue. However it is not so simple for a simple feedforward neural network to understand context because it only generates its output based on the current input.

In order to be able to understand the context of a word, we need to be able to *remember* what came before it. RNNs generates its output for a sequence of data, not just by its current input, but also its previous inputs and outputs. By having a network with loops, information can persist, which makes it possible to understand the context of an item in the sequence.

The general idea of an RNN can be seen in figure 2.4. We can see that information is passed to itself. It becomes clear what this exactly means when the network is unrolled, rather than just a network with a loop. Once unrolled, we are left with what looks more similar to a series of simple ANNs, and it becomes much clearer what is actually going on. We see that each node is connected to the next node with direction. We have an input layer and activation layer and an output layer, however from the activation layer, we are passing on information to the next state.

RNNs work by using the previous hidden state to calculate the current hidden state. As opposed to using the previous output, this allows the network to remember the *entire* history of the sequence.

Now rather than just passing on the output of the previous state, we are able to get the *entire* history of the sequence by passing on the hidden layer state, as shown by the function below.

$$h^{(t)} = \sigma(W_{hx}x + W_{hh}h^{(t-1)} + b_h)$$

Where *"W_{hx} is the matrix of weights between the input and hidden layers and W_{hh} is the matrix of recurrent weights between the hidden layers at adjacent*

time steps. The vectors b_h and b_y are biases which allow each node to learn an offset."[Lipton, 2015]

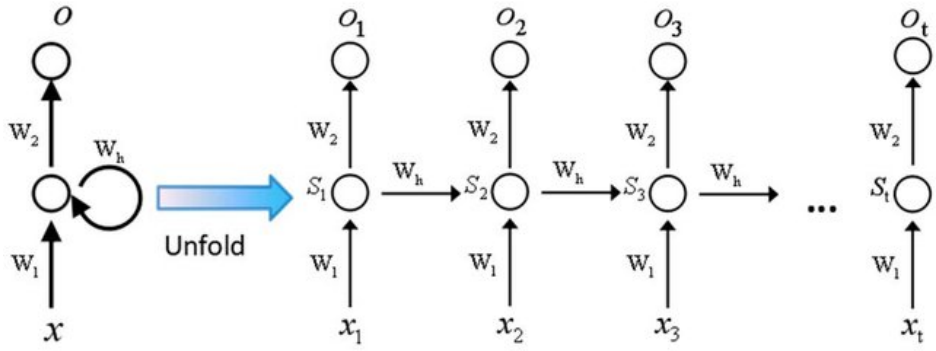


Figure 2.4: An unrolled recurrent neural network reprinted from [Zhang et al., 2018]

improve

2.4.1 The vanishing gradient problem

We explained the need for RNNs by using the simple sentence, *The car is blue*, with the objective being to find out what *blue* was referring to. A standard RNN would be able to do this with ease. However most sentences are not this simple, and in many cases, understanding the context is substantially more difficult. Take for example the two sentences *Cheetahs are the fastest animal on land, and can reach speeds of up to 120 km/h. They can also accelerate up to 100km/h in just 3 seconds.* In order to be able to understand the context of *cheetah*, the network needs to *remember* many time-steps back. If we think of each word as a time-step, and *They* is x_t, then we would need to recall the word at time-step x_{t-16}, *Cheetahs* to understand what *They* is referring to at time-step x_t. However this can be difficult to achieve, due to the vanishing gradient problem.

The vanishing gradient problem is the result of conducting BPTT using gradient descent [Bengio and Pascanu, 1986]. The aim of the gradient descent is to find the minimum of the cost function as by incrementally moving towards the smallest derivative of the (cost) function, as demonstrated in the figure 2.5.

BPTT is used to propagate back through the network to update the weights. Now in a shallow non-recurrent neural network, this is not a problem, because you are just updating the weights of (maximum a couple of layers) on this one state. However in an RNN, you are back propagating through time, because (as

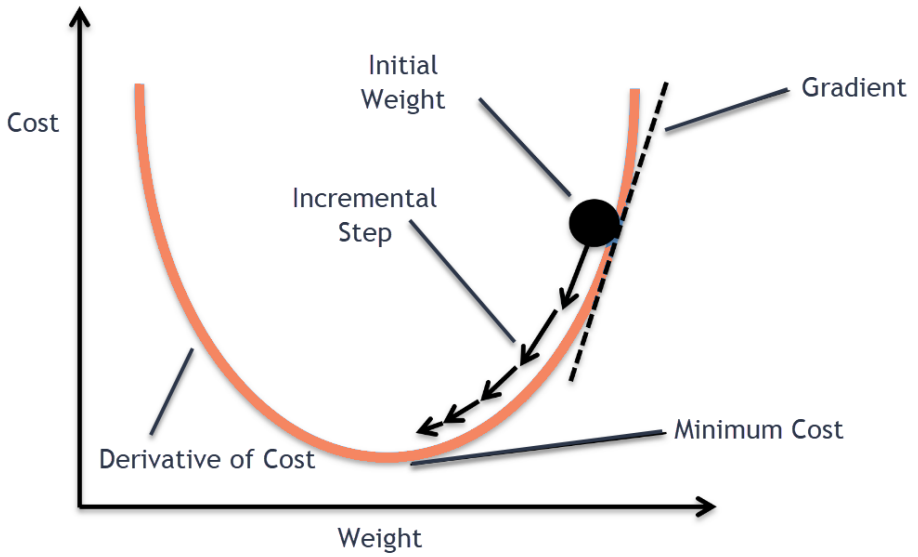


Figure 2.5: Gradient descent reprinted from [Lanham, 2018]

discussed earlier), previous time-steps are used as inputs to the current time-step. Since you are able to calculate the cost at each time-step, this means that the weights are updated for each time step, when propagating backwards. This is where the problem arises. To get from time-step x_t to x_{t-1}, we would have to multiply x_t with the recurrent weight. When initializing the weights at the beginning of training, we initialize them to a small number close to 0. This means that when updating the weight through back propagation, for each time-step, the gradient is going to get smaller and smaller. This is because we are multiplying by a weight < 1, which means we have a value decreasing rapidly for each time-step moving backwards. Since we use the gradient with respect to a weight w, to update the weight w, it means that the update will be negligible (or vanishing). If the weight updates are very small, the network is unable to *learn*. It is important to note that this problem can also apply to weights > 1, and is known as the exploding gradient problem, where the updates are too large to learn.

2.4.2 Long Short-Term Memory networks

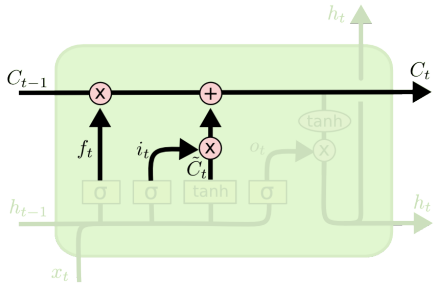
The Long Short-Term Memory network (LSTM) thrives in remember long-term dependencies. The network solves the vanishing gradient problem by introducing three gates, *forget gate*, *input gate* and *output gate*[Gers, 1999].

Forget gate

The first gate is the forget gate. This is the gate that decides what the network will remember. It utilizes the current input x_t and the previous hidden state h_{t-1}, and runs it through a sigmoid function. This sigmoid function will output a result between 0 (keep nothing) and 1 (keep everything). This can be done using the function $f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f$ where W_f and b_f refer to the weights and biases respectively. The forget gate was not in the first iteration of the LSTM, but has recently become standard practice. The reason for its later implementation is, without it the network would have to remember too much information, which can lead to the network crashing. The forget gate resets the memory of the network, once the information is no longer useful and therefore prevents this overload of the memory cells in the LSTM [Gers, 1999].

Input gate

The input gate is what decides what new information we are going to store. It is seperated into 2 parts, the *input gate layer* and the *tanh* layer. The input gate layer uses a sigmoid function to determine which values to update, while the tanh layer creates the *candidate (memory) values* by running the input and previous hidden layer through a tanh function rather than the sigmoid function. The functions to calculate these 2, are $i_t = \sigma(W_i \cdot [h_{t-1}, x_t + b_i)$ and $\widetilde{C_t} = tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$ respectively. These two layers (functions) are then combined to create the new memory but with the sigmoid scaling how much we should update each memory value. This is added to the function that decided which memory to keep. This decision is done by multiplying the old memory C_{t-1} with the forget gate f_t. A diagram a long with the formula for the whole equation is shown below in figure 2.6.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure 2.6: How new memory is combined with old memory reprinted from [Olah, 2015]

Output gate

Finally we have the output layer which, well, is used for generating our output.

Like all the other gates, in the output gate, the previous hidden state and current input is put through a sigmoid function. This will determine what parts of the hidden state, we are going to output. In order to calculate this current hidden state, the output is combined with tanh squishing function of the candidate values $h_t = o_t \times tanh(C_t)$. The final full version of the LSTM can be shown in figure 2.7.

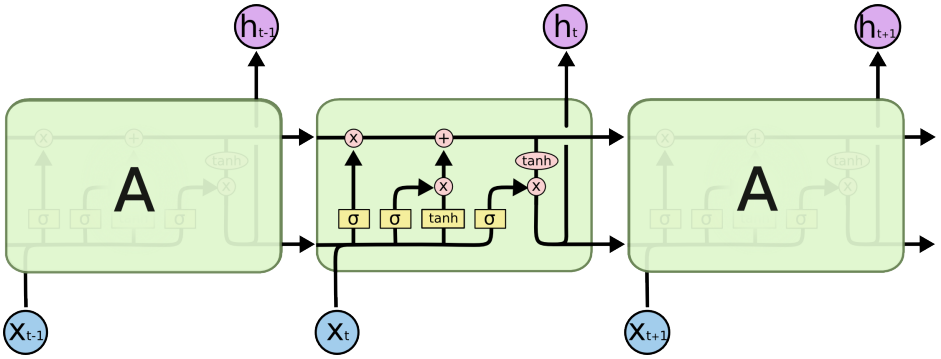


Figure 2.7: Complete LSTM network reprinted from [Olah, 2015]

Chapter 3

Related work

In this chapter, related work is explored to discover and evaluate the state-of-the-art implementations for anomaly detection in sensor values for time-series data. Although the study of anomaly detection is decades old, it is in recent years and with the introduction of machine learning algorithms, that we have generated the best results. Therefore all the related work will focus on machine learning approaches.

3.1 LSTM-based Encoder-Decoder for Multi-sensor Anomaly Detection

The paper [Malhotra et al., 2016], uses an LSTM-based Encoder-Decoder to determine anomalies in time-series with multiple sensors. The system uses an encoder-decoder LSTM to re-create the time-series. The time-series recreation error is then used to detect anomalies in the data. Malhotra et al. only uses non-anomalous time-series data for training because having anomalies in the training data for the encoding-decoding model would make the reconstruction unreliable in determining anomalies.

Malhotra et al.'s approach tackles the same problem as this thesis, determining anomalies in multivariate time-series data of sensor values. However in the paper, the data used to train the encoder-decoder reconstruction is purely non-anomalous. Only using non-anomalous data for training is not achievable in this project, because Equinor does not specify which data points in the dataset (used in this project), are anomalous. It is also unnecessary to create a model that can recreate an entire time-series when, for this thesis, we only need to predict a single sensor value at the current time-step.

3.2 Learning Representations from Healthcare Time Series Data for Unsupervised Anomaly Detection

[Pereira and Silveira, 2019]'s implementation is similar to the one discussed in section 3.1. An LSTM network with an autencoder is used for time-series recreation and the recreation error is used for determining anomalies. The main difference between the two papers, is that Pereira and Silveira works on unsupervised data. Instead of using a standard autoencoder, they make use of a variational autoencoder, which rather than recreating the data exactly as before, creates data that is similar to the original. This allows for using the recreation error, while training the encoder-decoder on a mixture of (anomalous and normal) unlabeled data.

Pereira and Silveira's implementation would be suitable for this project, and in fact may have yielded good results. However in their case, they had a labeled version of the test dataset, so they were able to verify their results. This unsupervised model show promising results, with accuracy and F-score just 3% lower than their supervised counterparts at $\sim 95\%$.

3.3 Variable Selection in Time Series Forecasting Using Random Forests

In the paper [Tyralis and Papacharalampous, 2017], Random Forest Regression is evaluated as a method in selecting variables for time-series prediction. Specifically this paper aims to evaluate Random Forest Regression's performance on one-step prediction of time-series. The study finds that Random Forest does well in variable selection and found that using many variables for prediction, gave worse results, because the more variables are used, the lower the importance value of the last selected variables.

This study shows that Random Forest Regression is useful for feature selection. However, the study specifies that their tests were only conducted on short time-series data, whereas Equinor's data is long time-series data. However they use both simulated and real-world data, to show that it is an effective approach. This makes it a viable tool to test for long time-series prediction.

3.4 A Feature Selection Method Using Hierarchical Clustering

The paper [Park, 2013], evaluates Hierarchical clustering as a method of unsupervised feature selection. In this study, the final cluster represents variables that are important to each other, in other words, are correlated.

The paper by Park shows that hierarchical clustering is a viable choice, for unsupervised feature selection. However its experimentation is not very thorough only showing the average accuracies using cross validation. Therefore more experimentation needs to be done in this thesis to evaluate the approach.

3.5 Structured Literature Review Protocol

In the 90s, there were several studies in the medical field, including [Antman, 1992] that concluded that the research in systematic reviews exceeds that of experts. For years, now they have successfully used EBM, Evidence-Based Medicine. However, it is in more recent years that similar methodology has been adopted for software engineering. As stated in [Dybået al., 2005], Evidence-Based Software Engineering (EBSE) tries to improve on the software development process by incorporating practical experiences in software development with evidence from research. One of the main components for finding evidence in the ESBE is the Structured Literature Review (SLR). A structure (or systematic) literature review is as defined by Antman, *"a methodologically rigorous review of research results"*. This means that it is structured, there are steps one has to follow to conduct SLR. The steps are as follows:

1 Research questions

The first step after identifying a goal, is to ask answerable literature research questions. It is important to have the questions related to the goal, but still be broad enough to not exclude any important evidence. This process resulted in the following questions.

LRQ1 What are the state-of-the-art deep learning algorithms for anomaly detection in time-series data?

LRQ2 What research exists for unsupervised anomaly detection in time-series data?

LRQ3 What are the state-of-the-art algorithms for feature selection for time-series prediction?

2 Search process

It is important to uncover all the literature that is relevant to the Literature Research Questions. The process began by identifying relevant search engines:

- Google scholar [Google, 2019a]
- Springer Link [Springer, 2019]
- ACM Digital Library [ACM, 2019]
- IEEE Xplore Digital Library [IEEE, 2019]

These search engines were then utilized with various combinations of the following search terms:

- Time-series / Time series
- Anomaly detection
- Sensor (value) prediction
- Deep learning
- Feature/Variable selection
- Clustering
- Unsupervised

A long with the results gathered from the search engines, the snowballing technique was used to discover more research. Snowballing is the practice of finding literature from the citations of a paper the reader has read [Wohlin, 2014]. Using a specific piece of literature, new literature was also discovered by reversing the snowball technique and looking at which other studies cited the specific piece of literature.

3 Exclusion criteria

After an initial set of literature was identified, individual research was eliminated if it was clearly irrelevant, outdated, a duplicate or the paper was without any cleraly defined research questions.

4 Quality assessment

After excluding papers which are unuseful, the research remaining needs to be evaluated in terms of certain critera, mainly inclusion criteria and quality criteria. This step is known as the quality assessment, and the criterias which were used can be found in table 3.1. The final list of literature is found in table 3.2

ID	Criteria
IC1	The literature adresses one or more of the literature research questions.
IC2	The literature includes at least two of the described search terms.
QC1	The findings of the literature is reproducible.
QC2	The algorithms in the literature are explained comprehensively.
QC3	Results are analyzed critically and with authenticity.
QC4	The evaluation metrics used are reasonable for the experiment types.
QC5	The conclusions aligns with the results.

Table 3.1: The criterias for quality assesment in the SLR. IC = Inclusion criteria and QC = Quality criteria.

ID	Title	Citation
1	LSTM-based Encoder-Decoder for Multi-sensor Anomaly Detection	[Malhotra et al., 2016]
2	Learning Representations from Healthcare Time Series Data for Unsupervised Anomaly Detection	[Pereira and Silveira, 2019]
3	Variable Selection in Time Series Forecasting Using Random Forests	[Tyralis and Papacharalampous, 2017]
4	A Feature Selection Method Using Hierarchical Clustering	[Park, 2013]
5	Fast Hierarchical, Agglomerative Clustering Routines for R and Python	[Müllner, 2013]
6	Comparison of hierarchical cluster analysis methods by cophenetic correlation	[Saraçli et al., 2013]

Table 3.2: Final selection of literature chosen

Chapter 4

Implementation

In this chapter we will look at how the system was implemented. We start by looking at the tools used in this project, in section 4.1. We then describe the pre-processing required to get the data ready for machine learning, in section 4.2. We will then look at the implementations of Random Forest Regression and Hierarchical Agglomerative Cluster, respectively in section 4.3 and 4.4. We then describe how the data was prepared to be used as input to the model, in section 4.5. Before, finally we look at the implementation of our Long Short-Term Memory (LSTM) network in section 4.6.

4.1 Tools used in this project

There are a variety of tools that were used in this project, that helped complete the project. As a disclaimer, it should be noted that most, if not all the tools explained below, will have alternatives that could be explored. However for the reproducibility of this thesis, below are the tools that were used in this project.

4.1.1 Pandas

Pandas is used for data analysis and to maninpulate data structures [Pandas, 2018]. In this project it is used to interact with the comma-separated values (csv) files, that equinor's data was formatted in. Using Pandas DataFrame, one can store the data in a tabular structure, that has labelled axis and can be modified. Equinor's data was manipulated with the use of the Pandas Dataframe, and because you could easily get all the values in a 2D array, by calling the method *values* on the DataFrame, it was simple to use it for the machine learning aspect as well.

Pandas also has a host of other useful tools, such as *sampling*, which can be used to generate the training and test set based on (for example) what percentage of the data you would like to be train and test.

4.1.2 Scikit-learn

Scikit-learn is a vast library of computational tools, which was used for tasks such as normalizing data, and finding the importances using the Random Forest Regression.

4.1.3 SciPy library

The SciPy library is a host of mathematical and statistical tools. It was used to conduct Hierarchical Agglomerative Clustering. The reason this was chosen, as opposed to [Müllner, 2013], is that SciPy's version allows the user to use a pre-computed similarity measure.

4.1.4 Matplotlib

Matplotlib is a graphing tool and was used to create all the graphs that are shown in chapter 5 and 6.

4.1.5 Tensorflow

Tensorflow is a deep learning framework that uses *tensors*, which are multi-dimensional arrays to create data flow graphs[Tensorflow, 2019]. In the graph, nodes are mathematical operations, and the tensors are edges that connect them. Tensorflow is created to be easy to deploy for computation on multiple GPUs.

4.1.6 Keras

Keras is a high level API built (in this case) on top of Tensorflow [Keras, 2019], which makes it easy to design machine learning models. It was therefore the tool of choice for designing, training, validating, and testing the Long Short-Term Memory (LSTM) network models. Like Tensorflow, Keras is easy to deploy to be used with multiple GPUs.

4.1.7 Google Colaboratory

Colaboratory is a free environment for writing and executing python code in the cloud, without any setup [Google, 2019b]. During the period which this project was conducted, Google gave its users access to T4 GPUs, with 15GB

video memory. This made Colaboratory extremely useful for training the LSTM models for this project, as this would have been a lot slower using the CPU on consumer-grade hardware. This is also why it is important that Tensorflow and Keras, as stated above, allows the computation to be deployed to multiple GPUs with ease.

4.2 Pre-processing

The data received for Equinor was spread out over many csv files, and had a lot of missing values, which meant it needed substantial pre-processing before utilzing machine learning algorithms. The goal was to end up with a single DataFrame that contained all the sensors (as columns) along with their values, without any missing values.

4.2.1 Equinor's data

Equinor delivered 34.1GB of sensor data from a Compressor Turbine in the Grane oil field on the West coast of Norway. The data contains anomalies, but these are not labeled and the amount of anomalies in the dataset is unknown.

The data was received in 107 folders, one for each sensor. In each sensor folder, there were a certain amount (varying, depending on when the recordings started, and frequency of value recordings) of csv files which all were in the format:

Sensor ***Value of sensor*** ***Timestamp*** ***Unknown value***

The * represents the fact that these csv files did not have columns names, they only had values. The values were of the following data types:

String **Float** **Timestamp** **Integer**

All the csv files had 1000s of rows (data entries). The **Unknown value** was constant for all sensors, and all rows, so it could be an id for the turbine, as it was an integer. Most sensors had their values recorded every 30 seconds, however sensors in the dataset were also recorded at 60 and 270 seconds.

4.2.2 Merging the data

The first pre-processing step was to merge all the data into a single Pandas DataFrame.

1. First, give all the columns, names - *Name*, *Value*, *Timestamp* and *Unknown*, respectively.

2. Merge all the files for a sensor.
3. Drop the columns *Name* and *Unknown*, as they are not needed.
4. Rename column *Value* to *Sensor name*, where Sensor name, refers to the actual name of the sensor.
5. Timestamps are sorted by newest, to oldest, reverse this order. As this prepares the next pre-processing step.
6. Repeat for all 106 remaining sensors.

After, we are left with 107 DataFrames, but a single DataFrame is needed. The next step is therefore to merge all these DataFrames into one. A problem that arises here is that the sensor values are not recorded exactly 30 seconds apart every time, this means that it is very rare that a sensor value recording has the exact same timestamp as another. Therefore to merge this data, we create windows. Datas within the same window are merged on the same timestamp. A new DataFrame is created which only has a *Timstamp* column with the chosen windows. In this case, windows of 30 seconds were chosen for a period larger than the first and last timestamp in all the dataset. 2011-01-01 and 2018-12-31 was chosen as the starting and end point, respectively. The window size of 30 seconds was chosen because this is the minimum amount of time between sensor value recordings, and it applies to most of the sensors. After the merge process finished, rows with all Not a Number (NaN) values were dropped, as this meant that none of the sensor value recording in the dataset fit within that window.

4.2.3 Interpolating rows

After merging the data, our single DataFrame contained a lot of NaN values, which we needed to remove before continuing. However, because there were a lot of missing values in the dataset, removing rows that contained any missing values, left us with < 10000 rows, which is far too few. Therefore, the data was linearly interpolated. This was done in 3 iterations because the sensors were recorded at 3 different intervals. For each iteration, all the columns recorded at a specific interval (30, 60 or 270 seconds), were interpolated twice its interval, both backwards and forwards. That means that a sensor recording its value every 30 seconds, with a value at 15:00:00, would fill in values at 14:59:00, 14:59:30, 15:00:30 and 15:01:00.

4.2.4 Removing irrelevant rows from merge window

Interpolation helped filling in missing data, however there were still missing values in the DataFrame, which makes it difficult to use the data for machine learning.

Therefore, the final step of the pre-processing was removing all rows that had any amount of missing values (NaN). This reduces the number of timestamps with more than 90%, leaving $\approx$ 700000 rows of usable data.

4.3 Random Forest Regression

For Random Forest Regression, the most important parameter that had to be decided, was how many decision trees would be used in the forest. 100 decision trees were used in the forest, as this allowed for accuracy, while not being too computationally complex. Another potential important parameter for calculating the importances using the random forest, is the criterion for evaluating the quality of a split. Due to the unsupervised nature of the dataset, it is not possible to evaluate which criterion would be best, but according to [Cutler et al., 2011] Mean Squared Error is most commonly used, so this was selected. When conducting this experiment, it is advised that the user always chooses as high, a number of decision trees as possible, and as time will allow for. Once the importance values are calculated, the sensors are ranked, and the 10 features with the highest importance are selected. 10 featutes were selected, as this gave good results while still resulting in a high dimensionality reduction.

Parameters	**Values**
Measuring quality of split	Mean Squared Error
Number of decision trees	100
Number of features chosen	10 most important

Table 4.1: Parameters used for Random Forest Regression.

4.4 Hierarchical Agglomerative Clustering

Lacking processing power, it was decided that the clustering process would be split up, first calculating the similarity measure. As explained in 2.2, one of the most common similarity measures for HAC, is the Euclidean distance, which was the choice for this project. To calculate the Euclidean distance for time-series data, we have to calculate the Euclidean distance between every set of data points (timestamps), and combine those values into a similarity matrix of 107×107. This means that we need to make $700000 \times 107 \times 107 \approx 8$ billion calculations.

The way the similarity matrix is calculated is by looping through all the columns and then looping through each timestep, and for each timestep, calculating the Euclidean distance between the 2 sensor values. Then for each iteration

that Euclidean distance, is added to the total sum of Euclidean distances between the two columns, giving us a total distance between the data columns. The algorithm is shown in Algorithm 1. Note that in order to have the values be on the same scale, the data is normalized before calculating the Euclidean distance between the sensors. All data normalized in this thesis, is normalized according to the MinMaxScaler from sklearn. Where X is the entire multidimensional array of values in the dataset, this is the formula for scaling (normalizing) the data

$$\frac{X_i - min(x)}{max(x) - min(x)}$$

This makes sure that the shape of the data is maintained, but that they are on the same scale.

input : A DataFrame with normalized values of size $S \times R$
output: A Euclidean distance similarity matrix, M of size $S \times S$

$M \leftarrow []$
for $i \leftarrow s_0$ **to** s_S **do**
  $T \leftarrow []$
  for $j \leftarrow s_0$ **to** s_S **do**
    for $k \leftarrow r_0$ **to** r_R **do**
      $E \leftarrow EuclideanDistance(s_i r_k, s_j r_k)$
    end
    Append $E \rightarrow T$
  end
  Append $T \rightarrow M$
end

Algorithm 1: Algorithm for calculating Euclidean distance similiarity matrix. S is the number of columns and R is the number of rows, in the DataFrame, respectively.

After the similarity matrix had been generated, the clustering process could begin. Centroid was chosen as the clustering method, as it seemed to yield best results, based on the cophenetic distances after clustering the data. This aligns with the findings in [Saraçli et al., 2013], which suggest that the best clustering method is centroid or average. As explained in section 2.2, we can use the dendrogram generated from the clustering process to determine the maximum distance a data point can be from the centroid of a cluster and still be added to that cluster. Looking at the dendrogram, seen in figure 4.1, it was found that 5×10^5 was a reasonable max distance. Once this distance was determined visually, we created the flat clusters, which are the final clusters to be used as part of the feature selection process.

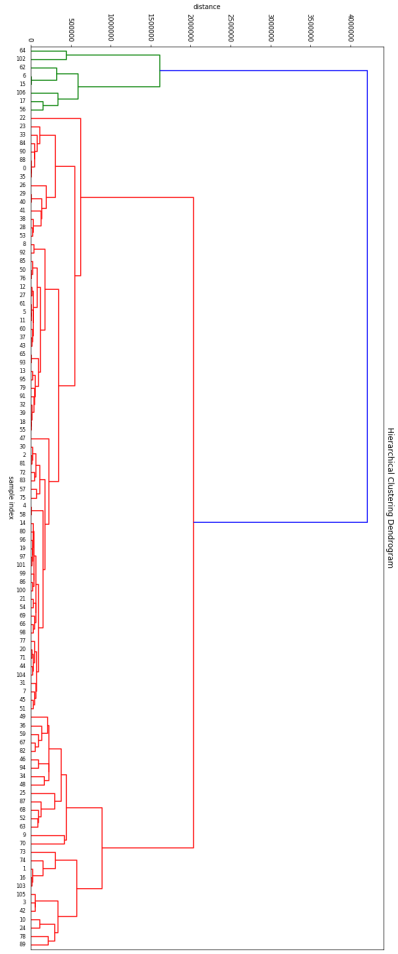


Figure 4.1: The dendrogram created in the Hierarchical Agglomerative Clustering implementation.

4.5 Dataset preparation

Using either Hierarchical Agglomerative Clustering or Random Forest Regression, a subset of sensors are selected. Their values are then normalized. The data is then formatted, so it has 3 lag time steps, leaving us with something similar to table 4.2, where S_1 represents the sensor we wish to predict (verify).

$\mathbf{S_1(t\text{-}2)}$	$\mathbf{S_2(t\text{-}2)}$	$\mathbf{S_3(t\text{-}2)}$	$\mathbf{S_1(t\text{-}1)}$	$\mathbf{S_2(t\text{-}1)}$	$\mathbf{S_3(t\text{-}1)}$	$\mathbf{S_1(t)}$	$\mathbf{S_2(t)}$	$\mathbf{S_3(t)}$
0.15	0.49	0.2	0.2	0.5	0.2	0.21	0.52	0.2
0.2	0.5	0.2	0.21	0.52	0.2	0.21	0.48	0.19
0.21	0.52	0.2	0.21	0.48	0.19	0.2	0.5	0.19

Table 4.2: Table demonstrating data with lagtime.

Now our y column here (the one we want to predict), is $S_1(t)$. Therefore we can remove $S_1(t-1)$ and $S_1(t-2)$ since we do not want the model to learn from those values. We also move $S_1(t)$ to be the last column, to make it easier to split into y and X for training and testing, leaving us with table 4.3

$\mathbf{S_2(t\text{-}2)}$	$\mathbf{S_3(t\text{-}2)}$	$\mathbf{S_2(t\text{-}1)}$	$\mathbf{S_3(t\text{-}1)}$	$\mathbf{S_2(t)}$	$\mathbf{S_3(t)}$	$\mathbf{S_1(t)}$
0.49	0.2	0.5	0.2	0.52	0.2	0.21
0.5	0.2	0.52	0.2	0.48	0.19	0.21
0.52	0.2	0.48	0.19	0.5	0.19	0.2

Table 4.3: Lag time preparation.

4.5.1 Training and testing data

The more data you use for training, the better trained the model will be, but you will have less data items to evaluate (test) it on. Since there are only approximately 700 000 rows, a split of 90% (630 000 rows) for training was decided. This of course left 10%, or roughly 70 000 rows for testing how well our model could predict the data. A random seed of 178 was also used, to allow for reproducibilty. Using the DataFrame method *sample*, 90% of the data was chosen randomly to be used for training, and the remainder was selected for testing.

The dataset split selects which rows will be used for training and testing. We still however have to format the data to work with the LSTM, which requires a 3-dimesional input. The 3 dimensions are, the batch size, the number of lagtime steps and the number of features. This means selecting and reshaping *train_X*, *test_X*, *train_y*, *test_y* and reshaping it to work with the LSTM. Selecting *train_X*,

test_X, *train_y*, *test_y* is easy with the data formatted, as in table 4.3. Simply for X values for train and test, choose the train rows with all the columns except the last one, and for y do the same, but only selecting the last column.

4.6 Long Short-Term Memory network

The LSTM has 50 neurons (hidden nodes), the input shape is the tuple **(train$_X$.shape[1], train$_X$.shape[2])**, that is the lag time steps and number of features used for prediction, respectively. We add a Dense (fully connected) layer, that connects every input to every output by a weight. We then compile the model using the Mean Absolute Error as the loss function and it is optimized using the Adam Optimizer. The Adam Optimizer is chosen because the hyperparameters do not require much tuning and it is suited for large datasets [Kingma and Ba, 2014]. The Adam Optimizer, has the following parameters:

- Learning rate = 0.001
- $\beta_1 = 0.9$
- $\beta_2 = 0.999$
- $\epsilon = 1 \times 10^{-8}$

The compiled LSTM model can be seen in figure 4.2.

4.6.1 Generating predictions

After the keras LSTM model is *fit*, it returns a history variable, which contains the loss function and other metrics, which the model was compiled with. These are then plotted, and the plots are saved.

The last step, is to predict the values based on *test_X*. This is done, of course using the model. Once the prediction is calculated, both the prediction and actual values are inversed, according to the MinMaxScaler, to return their actual values.

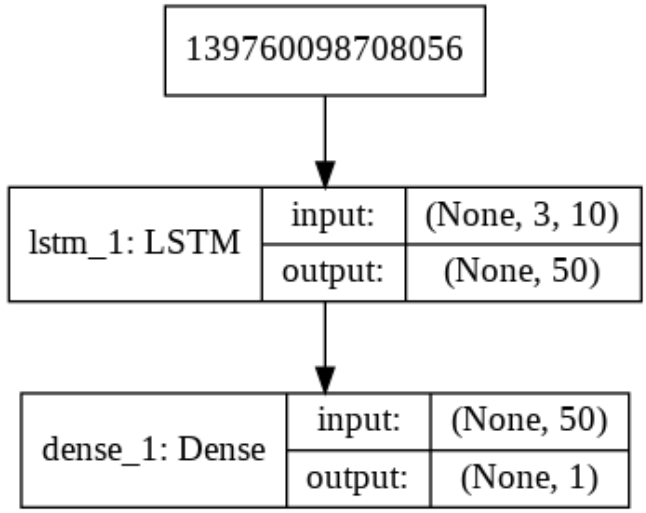


Figure 4.2: LSTM model architecture. *None* is referring to the batch size, which is not defined when the model is compiled.

Chapter 5

Experiments and Results

This chapter explains how the experiments for this project were conducted and what results were obtained from it.

The main aim of this experiment was to investigate whether certain sensor values can be used to determine if another sensor value is giving off faulty readings. We wanted to find, what (if any) set of sensor values *2, 3, ...n*, could be used to predict the value of sensor *1*.

5.1 Experimental Plan

- Select features using either Random Forest Regression or Hierarchical Agglomerative Clustering (HAC), to find subset of sensors from the dataset.
- Format subset into 3 lag time steps, $t-2$, $t-1$ and t.
- Split into training and test data, where 90% is selected for training and 10% for test.
- Train the Long Short-Term Memory network on the training data, leaving 10% of the training data for validation.
- Save the model, if the validation loss has decreased from one epoch, to the next.
- Use the model to predict values from the test data.
- Evaluate prediction using the evaluation metrics, described in 5.3.
- Repeat the entire process for all 107 sensors.

5.2 Experimental Setup

5.2.1 Specifications

All the experiments are run on Google Colaboratory. The Hardware and Software specifications can be seen in table 5.1

Software	**Hardware**
Ubuntu 18.04 Bionic	Intel Xeon CPU 2.30GHz
Python 3.6.7	Tesla T4 2560 CUDA cores 15GB GDDR6 VRAM
	12GB RAM

Table 5.1: Harware and Software setup

5.2.2 Training the model

The training is done for 200 epochs and a batch size of 72 is used.

5.3 Evaluation

In order to determine, which setup works best we need to define some metrics, that determines how well our model fairs on our test data.

For evaluating the training phase, the Mean Absolute Error (MAE) would be used to calculate the loss. Its formula a long with all the other mathematical formulas used for evaluation can be found in table 5.2. The MAE is useful, because it simply sums up the total error and divides by the number of data point. Using the absolute value, ensures that positive error and negative error do not lead to lower total error. It is therefore a useful measurement for the training process. The Mean Absolute Percentage Error (MAPE) was also used on the validation data, as this is useful in comparing data of different scales. For evaluating the model on the test set, the Root Mean Squared Error (RMSE) formula was used.

Mean absolute error	$\text{MAE} = \frac{1}{n}\sum_{t=1}^{n} \mid \hat{y}_t - y_t \mid$
Root mean squared error	$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{t=1}^{n}(\hat{y}_t - y_t)^2}$
Mean absolute percentage error	$\text{MAPE} = \frac{1}{n}\sum_{t=1}^{n}\left\lvert\frac{\hat{y}_t - y_t}{y_t}\right\rvert$

Table 5.2: Mathematical formulas for computing validation metrics. n refers to the number of data items evaluated, $\hat{y_t}$ is the prediction and y_t is the actual value

5.4 Results

The validation and training loss for the 2 models (one using feature importance and the other using clustering as method for feature selection) predicting the same sensor, s_1, can be seen in figures 5.1 and 5.2. Both models are trained for the same 200 epochs, but they differ in having used different feature selection methods. Figure 5.1 uses the Random Forest Regressor as its feature selection method, while figure 5.2 uses Hierarchical Aggolmerative Clustering to determine which sensors are correlated. From the loss graphs, it seems that both models are able to learn to predict the value of sensor s_1 for the 10% of validation data. It is worth noting that the figure 5.1 has more rapidly decreasing validation and training loss and in fact it reaches both a lower loss at 0.000573 than figure 5.2, which only drops to 0.000865.

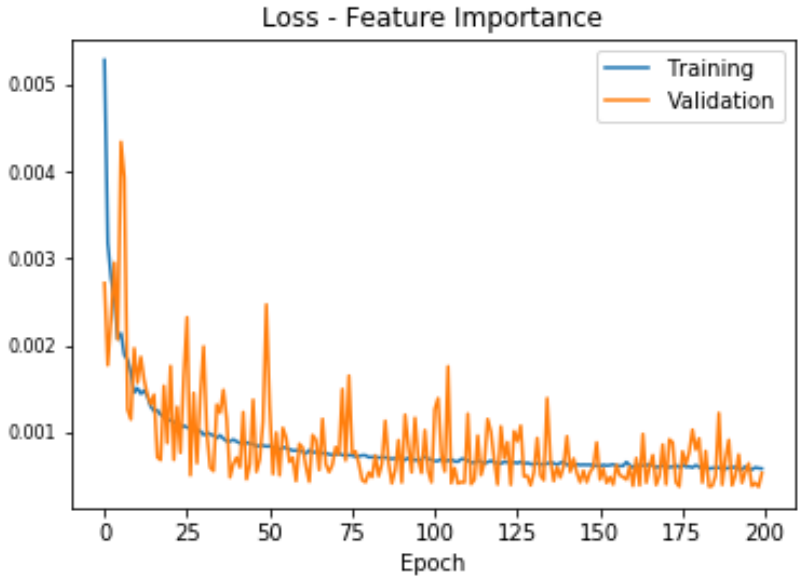


Figure 5.1: Training and validation loss for sensor s_1, using feature importance as the feature selection method using 10 features.

We also see that the model that uses the feature importance method has a lower MAPE (figure 5.3) reaching a minimum MAPE of 0.269 compared to the model using Hierarchical Agglomerative Clustering (HAC) (figure 5.4), which only reaches 0.703.

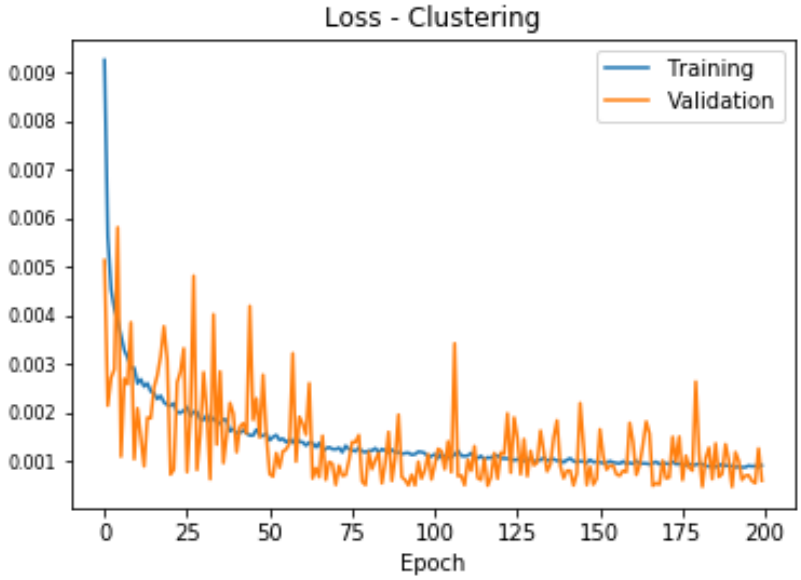


Figure 5.2: Training and validation loss for sensor s_1, using Clustering as the feature selection method.

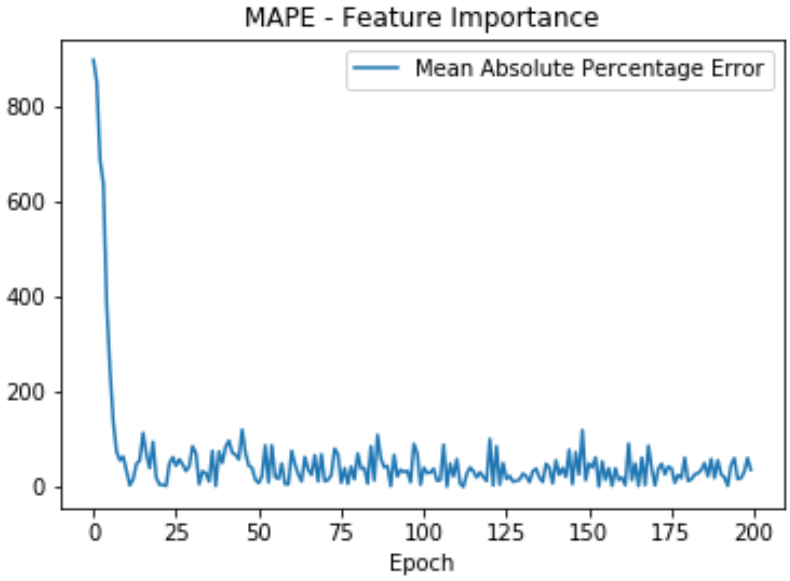


Figure 5.3: Mean Absolute Percentage Error for sensor s_1, using Feature Importance as the feature selection method.

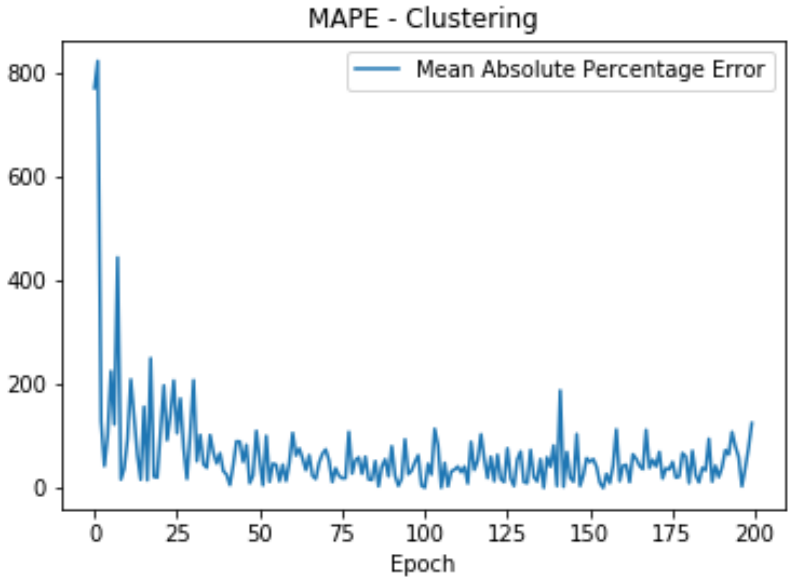


Figure 5.4: Mean Absolute Percentage Error for sensor s_1, using Clustering as the feature selection method.

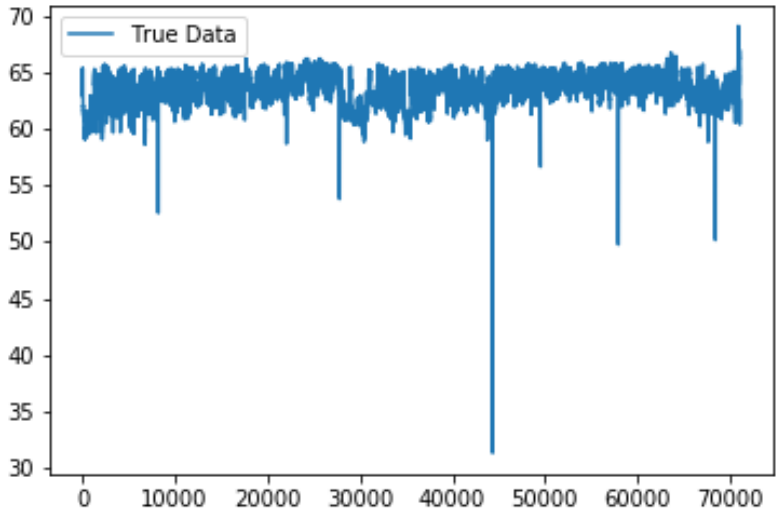


Figure 5.5: Actual data for sensor s_1

This is all highlighted by the fact that the model for sensor s_1 has a lower Root Mean Squared Error for the model using Feature Importance compared to the model using HAC. The actual test data, prediction for feature importance and prediction for Clustering can be seen in figures 5.5, 5.6, 5.7, respectively.

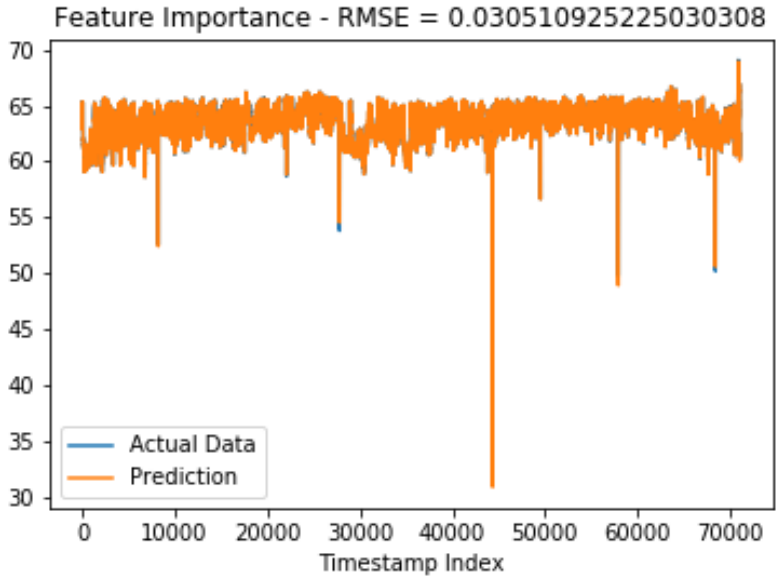


Figure 5.6: Predicted values compared to actual data for sensor s_1, using Feature Importance as the feature selection method.

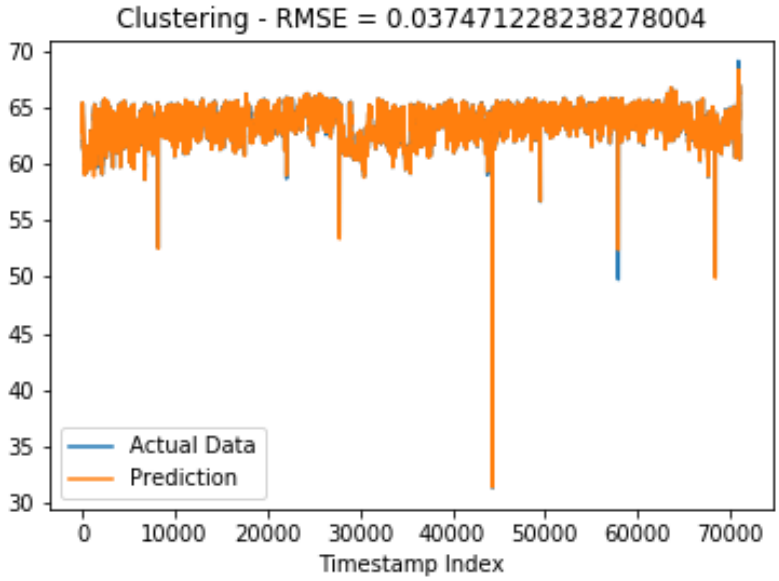


Figure 5.7: Predicted values compared to actual data for sensor s_1, using Clustering as the feature selection method.

Although both models have a very low RMSE, and are even able to predict abrupt changes reasonably well, the model using Random Forest Regression for feature seelction achieves better results for this sensor, by having a lower loss for training, lower MAPE for training and lower RMSE for the test data. Even visually from the prediction graphs (figure 5.6 and 5.7), it is possible to see that the model trained using features selected using Random Forest Regression follows the actual data more closely, meaning it is a better representation of the actual data.

It is important to remember that the dataset has 107 sensors and therefore the result of one model, is not adequate in determining how well the network predicts, or which feature selection works best. In fact, from table 5.3, we see that the average RMSE is quite a lot higher for both Random Forest Regression and HAC, compared to the RMSE in figure 5.6 and 5.7, respectively. Moreover we see that across the board, Random Forest Regression beats HAC, especially for max and average values. This indicates that Hierarchical Agglomerative Clustering is able to predict well on some sensors, but unable on certain others. The table shows very low minimum RMSE but high maximum RMSE and even quite high average RMSE. However, it is important to note RMSE is affected by scale, and their is a huge variation in scale in the dataset. The scale ranges all the way from 10^{-1} to 10^{4}. Therefore the MAPE has also been included, which is unaffected by scale. The MAPE for Random Forest Regression shows promising results, with an average < 0.56 and a maximum of 2.3. The average MAPE for HAC, although much higher than Random Forest, indicates it could be useful in feature selection for prediction, however it has a very high maximum, which means it is less stable.

	RMSE			**MAPE**		
	Min	***Max***	***Avg***	***Min***	***Max***	***Avg***
Random Forest Regression	0.000878	43.1	4.08	0.00508	2.30	0.558
Hierachical Agglomerative Clustering	0.00144	1090	59.8	0.00509	25.6	3.20

Table 5.3: Min, Max and Average prediction RMSE and MAPE for Random Forest Regression and Hierarchical Agglomerative Clustering, to 3 significant figures.

Chapter 6

Evaluation and Conclusion

This chapter will conclude the thesis. First, the results are evaluated in section 6.1. The merits and shortcomings of the thesis are then discussed in section 6.2. In section 6.3, the contributions of this thesis are presented and explained. Finally in section 6.4, we look at the future work that can be done to improve the system presented in this thesis.

6.1 Evaluation

In this thesis, a deep learning network that aims to use a subset of sensors in a dataset to determine if another sensor in the dataset is displaying the correct value at a time point t has been researched and implemented. Throughout this thesis, we have tried to answer the following research questions:

1. *Can multivariate time-series data with missing values be pre-processed to remove missing values while maintaining enough data time-stamps to train and test the model(s) used in this project.*

2. *Is it possible to correctly determine which sensor values are useful for predicting each other?*

3. *Is it possible to build a system that uses deep learning algorithms to predict a sensor value based on n other sensor values?*

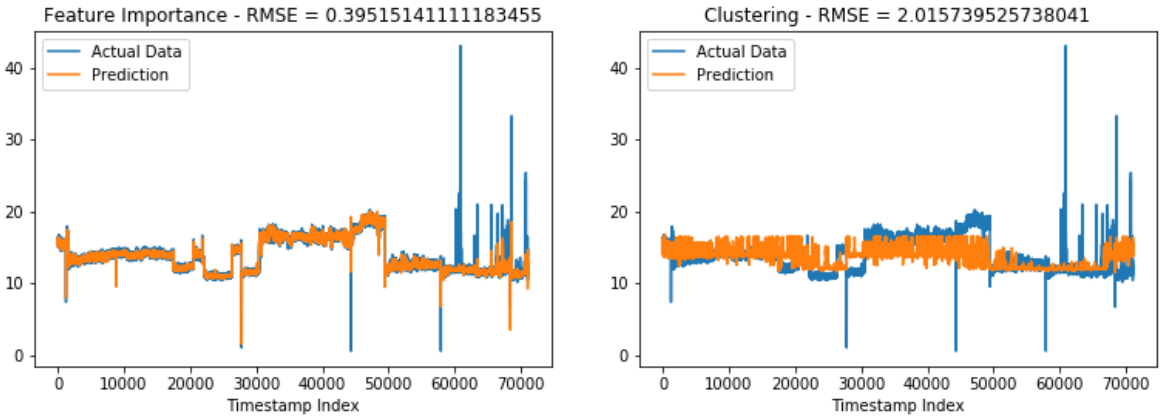
4. *Is it possible to verify that the model's prediction are precise?*

The dataset used in this thesis had a lot of missing values, and simply removing the rows which had missing values, left $\approx$ 2000 rows, which is far too

few to conduct any data analysis. In order to keep the dataset adequately large after, preprocessing, linear interpolation was used. Even with interpolation, after removing rows with NaN values, there was still more than a 90% reduction in the dataset, leaving only $\approx$ 700000 rows of data, compared to the original $\approx$ 7.5 million rows in the data that contained missing values. However, as the results show in section 5.4, the prediction error calculated using the Root Mean Squared Error (RMSE), are very low, and in fact, it most cases < 0.1 for the data normalized in the range 0 to 1.

6.1.1 Feature selection

In order to choose the subset of sensors used as input to the network, two different methods were explored. Finding the Feature Importance using Random Forest Regression (and selecting the 10 most important features), and clustering the sensors using Hierarchical Agglomerative Clustering (HAC). In almost all instances it was found that using the Feature Importance yielded better prediction, than HAC. That being said, both methods achieved promising results in selecting appropriate sensors for prediction. However, it has been specified in this thesis that the goal of this project is to be able to use a subset of the dataset to predict a sensor value not in said subset. If we look at table 6.1, we can see that our 9^{th} and final cluster only has a single value in its cluster. This means that it will not be able to use a subset of the dataset (not including itself) to generate a prediction. In fact, when looking at the table, we find that $\approx$ 50% of the sensors are placed in cluster 1. This means that there is only half dimensionality reduction, when creating half the models, as opposed to the constant $\approx$ 90% found when using the 10 most important features in Feature Importance. Having half the sensors in the first cluster, also means that the remaining clusters will be more sparse. This of course is highlighted with the 9^{th} cluster only having a single item, but we also have cluster 8 only containing 2 sensors. This means that a single sensor value is used to predict the other sensor in the cluster. Only being able to use a single sensor column to predict the other, makes it much harder to predict the value of the sensor, as demonstrated in figure 6.1, where we see that the sensor using Random Forest Regression for features selection has a substantially lower Root Mean Squared Error compared to the one using HAC, with them being 0.395 and 2.016, respectively.



(a) Random Forest Regression

(b) Hierarchical Agglomerative Clustering

Figure 6.1: Evaluating predicted values, where HAC only has two columns used for prediction cluster index 8 in table 6.1.

Cluster index	Number of sensors in cluster
1	56
2	7
3	3
4	5
5	14
6	16
7	3
8	2
9	1

Table 6.1: The number of sensors in each cluster that was used for feature selection.

6.2 Discussion

It was found that Random Forest Regression consistently outperformed HAC. However a possibile reason for the lower performance of HAC is the choice of similarity measure. The Euclidean Distance, does not learn the time-series data *shape*, and instead looks at the linear relationship. Therefore trying a different similarity measure might result in more appropriate clusters.

Although promising results were presented in chapter 5, with regards to predicting sensor values, it is not possible to verify a sensor warning as correct or report it as anomalous without having access to any of Equinor's anomaly data. However, as shown, the errors from predictions are relatively low, and the model seems capable of learning the representation quite well, which indicates that this model could be used for anomaly detection, if anomaly in the data were to be labeled.

As discussed in 3.2, unsupervised anomaly detection can also be conducted using an autoencoder, where the outliers in the recreation indicates anomalous data. This, however may not be required for Equinor, as they only need to predict the current time-step, when a sensor is giving off a warning.

6.3 Contributions

This thesis has showcased the state-of-the art in anomaly detection of time-series data.

We have presented results for unsupervised feature selection of time-series data, using both Random Forest Regression and Hierarchical Agglomerative Clustering, and evaluated them with respect to their ability as input for a predictive model.

Using Long Short-Term Memory (LSTM) networks, a deep learning approach has been presented for predicting sensor values, using a subset of sensors, and without requiring any domain knowledge. The prediction accuracy, suggests that the models have good potential in detecting anomalies.

By incorporating labeled anomalies, these anomalies could for example be averaged for each sensor and be used as thresholds in conjunction with current models to determine anomalies when predicting the sensor values. These models can then be used for anomaly detection by Equinor.

6.4 Future Work

6.4.1 Testing different similarity measures for HAC

As mentioned in section 6.2, the choice of similarity measure could be the reason for the low performance of the predictive model using HAC. Choosing a different similarity measure, may therefore have created more appropriate clusters, which would have reduced the prediction error. This is backuped by [Aghabozorgi et al., 2015], which states that research suggests that dynamic programming similarity measures, which learn the shape of the data, are the most applicaple for clustering time-series data. This however is computationally expensive, and would require upgraded hardware, compared to that used in this project.

6.4.2 Classification using anomaly data

Obtaining data of the labeled anomalies is essential for the importance of this research. Having obtained anomaly data, one would be able to change the results of this thesis from regression to classification, as it is no longer of interest (in theory) what the actual values the network predicts are, but rather whether or not the predicted value is anomalous.

6.4.3 Automatically selecting number of features to use for Random Forest Regression

It was found that Random Forest Regression worked well in feature selection, and choosing the 10 most important sensors for preciting, yielded good results, while still obtaining $> 90\%$ dimensionality reduction. Although choosing exactly 10 features (sensors) worked well in this case, Equinor specified that the system need to be able to generalize, and that there is not necessarily a lot of correlation between the internals of their various turbines. Therefore it might be neccesary to automatically determine how many sensors are needed, as explained in [Genuer et al., 2010].

6.4.4 Testing on different datasets

Due to the fact that Equinor has a lot of different internals in their turbines, the models created in this thesis need to be thoroughly tested on datasets from Equinor's other turbines, as only the dataset from a single turbine was acquired for this thesis.

Bibliography

ACM (2019). Acm digital library. `https://dl.acm.org`. (Accessed on 06/12/2019).

Aghabozorgi, S., Shirkhoshidi, A. S., and Wah, T. Y. (2015). Time-series clustering - a decade review. *Information Systems*, 53:16–38.

Antman, E. M. (1992). A comparison of results of meta-analyses of randomized control trials and recommendations of clinical experts. *Jama*, 268(2):240.

Bengio, Y. and Pascanu, R. (1986). Learning to deal with long-term dependencies. *Universite de Montreal*.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, 24(2):123–140.

Breiman, L. (2001). Random forests. *Mach. Learn.*, 45(1):5–32.

Bühlmann, P. and Yu, B. (2002). Analyzing bagging. *The Annals of Statistics*, 30(4):927–961.

Cutler, A., Cutler, D., and Stevens, J. (2011). *Random Forests*, volume 45, pages 157–176. Spring US.

Dybå, T., A. Kitchenham, B., and Jørgensen, M. (2005). Evidence-based software engineering for practitioners. *Software, IEEE*, 22:58 – 65.

Equinor (2019). About us. `https://www.equinor.com/en/about-us.html`.

Genuer, R., Poggi, J.-M., and Tuleau-Malot, C. (2010). Variable selection using random forests. *Pattern Recognition Letters*, 31:2225–2236.

Gers, F. (1999). Learning to forget: continual prediction with lstm. *9th International Conference on Artificial Neural Networks: ICANN 99*.

Google (2019a). Google scholar. `https://scholar.google.no/`. (Accessed on 06/12/2019).

Google (2019b). Welcome to colaboratory. https://colab.research.google.com/notebooks/welcome.ipynb#scrollTo=5fCEDCU_qrC0.

Hapfelmeier, A. and Ulm, K. (2013). A new variable selection approach using random forests. *Computational Statistics & Data Analysis*, 60:50–69.

IEEE (2019). Ieee xplore search results. https://ieeexplore.ieee.org. (Accessed on 06/12/2019).

Keras (2019). Keras: The python deep learning library. https://keras.io/.

Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *International Conference on Learning Representations*.

Lanham, M. (2018). Learn arcore - fundamentals of google arcore. https://www.oreilly.com/library/view/learn-arcore-/9781788830409/e24a657a-a5c6-4ff2-b9ea-9418a7a5d24c.xhtml.

Lipton, Z. (2015). A critical review of recurrent neural networks for sequence learning. *arXiv*.

Liu, C., Gong, S., Loy, C. C., and Lin, X. (2012). Person re-identification: What features are important? *Computer Vision - ECCV 2012. Workshops and Demonstrations Lecture Notes in Computer Science*, pages 391–401.

Malhotra, P., Ramakrishnan, A., Anand, G., Vig, L., Agarwal, P., and Shroff, G. M. (2016). Lstm-based encoder-decoder for multi-sensor anomaly detection. *CoRR*, abs/1607.00148.

Müller, M. (2007). *Dynamic Time Warping*, pages 69–84. Springer Berlin Heidelberg, Berlin, Heidelberg.

Müllner, D. (2011). Modern hierarchical, agglomerative clustering algorithms. *arXiv*.

Müllner, D. (2013). fastcluster: Fast hierarchical, agglomerative clustering routines for r and python. *Journal of Statistical Software, Articles*, 53(9):1–18.

Nha, B. D. and Nhat, P. T. M. (2019). Mathcha. https://www.mathcha.io/. (Accessed on 06/03/2019).

Olah, C. (2015). Understanding lstm networks. http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

Pandas (2018). Python data analysis library. https://pandas.pydata.org/.

Park, C. H. (2013). A feature selection method using hierarchical clustering. *Mining Intelligence and Knowledge Exploration Lecture Notes in Computer Science*, page 166.

Pereira, J. and Silveira, M. (2019). Learning representations from healthcare time series data for unsupervised anomaly detection. *2019 IEEE International Conference on Big Data and Smart Computing (BigComp)*.

Saraçli, S., Doğan, N., and Doğan, İ. (2013). Comparison of hierarchical cluster analysis methods by cophenetic correlation. *Journal of Inequalities and Applications*, 2013(1):203.

SmartDraw (2019). Documents - smartdraw. `https://cloud.smartdraw.com/`. (Accessed on 05/31/2019).

Springer (2019). Home - springer. `https://link.springer.com/`. (Accessed on 06/12/2019).

Tensorflow (2019). Tensorflow. `https://github.com/tensorflow/tensorflow`.

Tyralis, H. and Papacharalampous, G. (2017). Variable selection in time series forecasting using random forests. *Algorithms*, 10(4):114.

Vaishnavi, V., Kuechler, W., and Petter, S. (2017). Design science research in information systems. `http://www.desrist.org/design-research-in-information-systems/`.

Wei, W. W. (2013). *Time Series Analysis*, volume 2 of *Statistical Analysis*, pages 458–459. Oxford Library of Psychology.

Wohlin, C. (2014). Guidelines for snowballing in systematic literature studies and a replication in software engineering. *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering - EASE 14*.

Zhang, W., Zhang, Q., Xie, Y., and Zhang, J. (2018). Lstm-based pitch range estimation from spectral information of brief speech input. *2018 11th International Symposium on Chinese Spoken Language Processing (ISCSLP)*.

Appendix A

Equinor’s practice

Equinor have a lot of public data available on their open platform Volve Data Village (https://data.equinor.com/dataset/Volve)

A problem they are experiencing is that their turbine’s do not necessarily have the same sensors and are not mapped together, which makes it difficult to conduct predictive maintenance large scale, because of the need for generalization.

Currently, Equinor are doing well with condition based maintenance. Condition based referring to, if a certain threshold is crossed for a sensor value, a warning is issued. This is done for very critical machines, such as big pumps. The warning is issued based on a single sensor value. In these cases they use complex mathematical models to assess whether the threshold should be adjusted for the sensor. This means that threshold adjustments are done fairly frequently, which is not inherently bad. However this mathematical models require a lot of domain knowledge, so are therefore not very generalized, and therefore time consuming.

Equinor’s maturity levels

Equinor have 5 maturity levels, regarding maintenance.

The maturity levels are:

1. **Condition monitoring** - This is something Equinor feel they have achieved well.
2. **Anomaly detection of critical equipment** - Equinor are working on this, but are not as of completely satisfied that they have reached this level.
3. **Probability of failure or machine degradation** - This is the level they want us to help them reach (minimum), by using machine learning.
4. **Predict time failure** - If we achieve this we have done very well.
5. **Prescriptive maintenance** - Future goal

Equinor's current challenges

1. Some key sensors may have failures themselves. When it happened, it is difficult to identify whether the anomaly is caused by sensors or the turbine. Therefore, it is important to understand which sensors are correlated to the key indicator sensors for cross-checking and eliminate the false positives.

2. Equinor is working on some research about applying autoencoder to identify anomaly condition caused by sensors. However, it is difficult to determine the threshold for the difference between predicted and actual values.

3. Equinor wants provide explanation for model output to build up user confidence. But it is difficult to interpret all the hidden connections inside the models.

4. Equinor has collected many sensor data, how to fully understand and utilize all these industry big and raw data would be a challenge.

5. Equinor has made some research for single or several turbines. How to transform the research outputs and knowledge to other equipments with totally different environment or in a fleet level instead of doing new research, which requires corresponding expertise again, would be problematic.

Equinor's expectations of the CIRCit WP4 team

Equinor wants a system that allows for creating scalable solutions, which across machine types and operation patterns. As such, Equinor prefer scalability (generalizable) over domain knowledge (specialized). They want to achieve as much as possible balanced recall and precision, but with slight emphasis on recall.

It is **vital** for Equinor that the CIRCit team deliver an end-user focused user experience. What this means is that for all the models we create, the system needs to clearly explain the reasons for the model output. It needs to have actionable recommendations for root cause analysis.

The potential user of outputs of the WP4 team will be the operator in the central control center. There are currently only a few operators who monitor many machines. If some sensors data show that a certain threshold is reached, the operators will ask somebody to check the machine to see if anything goes wrong with the physical machine. After this project, Equinor hopes that our system can predict the possible outreach of the key indicator sensor from other closely

related sensors early, i.e. before the key sensors report error, and could also help the operators to understand why the errors happen.

Equinor’s data

The data that has been supplied by Equinor is from a Turbine. The data is time series data from 107 sensors stored in 107 CSV files. Equinor govern approximate 30 platforms with 2-3 turbines for each. The data ranges from 2011 to 2018, but Equinor emphasizes that data may be missing, although all data from 2013 onwards should be present.

A diagram of the turbine was included with the data. It uses a standard naming convention. The naming convention is as follows. All conventions that begin with: