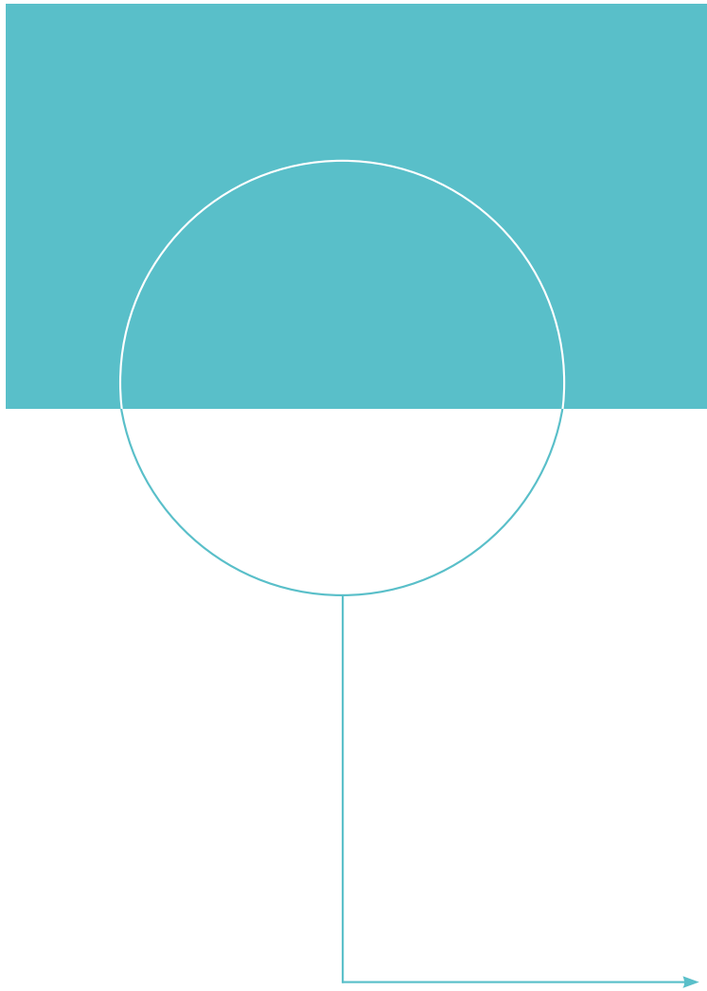

Master's thesis

2020

Kolbjørn Trøstheim Flaarønning, Mari Elida Tuhus

Master's thesis

NTNU
Norwegian University of
Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering

Kolbjørn Trøstheim Flaarønning
Mari Elida Tuhus

Initial Integration of Data-Driven Health-Indicators in the Petroleum Industry

June 2020

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

Initial Integration of Data-Driven Health-Indicators in the Petroleum Industry

Kolbjørn Trøstheim Flaarønning
Mari Elida Tuhus

Master's Thesis in Engineering and ICT
Submission date: June 2020
Supervisor: Jørn Vatn

Norwegian University of Science and Technology
Department of Mechanical and Industrial Engineering

Preface

This thesis marks the end of the five-year study program Engineering and ICT, at the Norwegian University of Science and Technology. The master thesis was written in spring 2020 as a final work of research. The master thesis was conducted in close collaboration with Teekay Offshore Production AS, as the company wanted to explore the possibilities for implementing data-driven health indicators. It is assumed that the reader has fundamental knowledge within the field of maintenance, and machine learning knowledge is preferred. Then again, the theoretical framework required for understanding the fundamentals of the machine learning techniques used in this thesis is found in Chapter 3.

Trondheim, June 9, 2020

Kolbjørn Trøstheim Flaarønning

Mari Elida Tuhus

Acknowledgment

This master thesis was written in collaboration with Teekay Offshore Production, and we would like to express our deepest gratitude for the close collaboration, supervision, relevant data and great discussions. In particular, we would like to thank Roar Bye, Erlend Meland, Kristian Holm Jensen, Jostein Vada, Bjørn Tore Tvestad and Jan Nesset. Besides, we

would like to thank our supervisor Jørn Vatn from the RAMS group, for his valuable insight and supervision during the conduction of this master thesis, as well as making the collaboration with Teekay Offshore Production possible.

Abstract

Condition monitoring has become a vital maintenance strategy across many industries. Obtaining information regarding health-condition of the system's components can profoundly reduce the operational costs as well as reduce the risk of catastrophic events. With today's emerging technology, it is possible to install sensors on every imaginable system component. Sensor technology generates a vast amount of data which, if interpreted correctly, leads to more insightful information. However, this does require sophisticated strategies regarding data acquisition, processing and advanced predictive techniques, as real-life data tends to be inconsistent, noisy and incomplete. This thesis proposes the initial step for implementing data-driven models to strengthen the predictive abilities in Teekay Offshore Production's maintenance strategy. A health-indicator for systems including compressors, turbines and diesel engines require a comprehensive data collection and might be too intricate to solve with the laws of physics. Thus, this thesis aims to research the possibility of using machine learning models to strengthen the predictive abilities of companies in the petroleum industry. Similar to most real-life datasets, the obtained data was of high dimension, complex structure and low quality. This thesis specifically focused on ensuring high data quality by performing an extensive grid-search through the domain of preprocessing techniques. The techniques were validated after their ability to improve the prediction accuracy, as the main objective of this thesis is to create an accurate health-indicator for a compression train.

Three types of recurrent neural networks, alongside two baseline models, one classical statistical model and one multilayer perceptron, were created. The baseline models created the possibility of assessing the potential value of increasing the complexity of the models. The three recurrent neural networks outperformed both baseline models. The highest performing recurrent neural network was the Long-Short Term Memory with an increased performance of 17.18% in terms of prediction ac-

curacy. Finally, a hybrid model combining the strengths of the highest performing recurrent neural network and the classical autoregressive integrated moving average model was implemented. The hybrid model leveraged the strength of both individual models and obtained an increased performance of 27.51% compared to the baseline models. A hybrid model achieved the best result in this research, as it captured both the linear and non-linear relationships in the real-life dataset.

Sammendrag

Tilstandsovervåkning har blitt en essensiell vedlikeholdsstrategi i flere bransjer. Muligheten til å samle systemdata om helsetilstanden til komponenter gjør det mulig å redusere driftskostnader betraktelig, samt redusere risikoen for katastrofale hendelser. Med dagens teknologiske framskritt er det mulig å installere sensorer på og overvåke alle tenkelige gjenstander. Sensorteknologi generer en enorm mengde data, noe som gir en verdifull innsikt hvis den tolkes korrekt. Ettersom data fra industrien ofte er ufullstendig, inkonsistent og inneholder støy, kreves det sofistikerte strategier ved datainnsamling og dataprosessering, samt avanserte prediktive fremgangsmåter. Denne masteroppgaven foreslår implementering av datadrevne modeller for å øke den prediktive evnen i vedlikeholdsstrategien til Teekay Offshore Production. En helseindikator som overvåker kompressorer, turbiner og dieselmotorer krever en omfattende innsamling av data. Derfor er det lagt et stort fokus på å sikre høy datakvalitet ved å utføre omfattende tester av ulike databehandlingsmetoder.

Tre *recurrent neural networks*, samt to baseline-modeller, henholdvis en klassisk statistisk modell og et flerlags perceptron, er implementert. Baseline-modellene sikrer muligheten til å vurdere verdien av å øke kompleksiteten på modellene. *Recurrent neural networks* modellene presterte bedre enn samtlige baseline-modeller, hvor *long-short term* modellen skilte seg spesielt ut med en relativ forbedring på 17.18%. Til slutt ble en hybridmodell implementert for å kombinere styrkene fra den klassiske *autoregressive integrating moving average* modellen og fra *recurrent neural networks*. Hybridmodellen oppnådde en forbedring på 27.51% sammenlignet med baseline-modellene og er dermed det beste resultatet i dette arbeidet.

Abbreviations

AI	Artificial Intelligence
ANN	Artificial Neural Network
ARIMA	Autoregressive Integrated Moving Average
BPD	Barrels Per Day
CM	Corrective Maintenance
DM	Data Mining
DNN	Deep Neural Network
FMEA	Failure Mode and Effects Analysis
FPSO	Floating Production Storage and Offloading
FTD	Fault Tree Diagram
GRU	Gated Recurrent Unit
ICT	Information and Communication Technology
IoT	Internet of Things
KNN	K Nearest Neighbors
KPCA	Kernel Principal Component Analysis
LR	Linear Regression
LSTM	Long-Short Term Memory
LTU	Linear Threshold Unit
MAE	Mean Absolute Error
ML	Machine Learning
MLP	Multilayer Perceptron
MRSE	Mean Root Square Error
MSE	Mean Square Error
NAN	Not A Number
PCA	Principial Component Analysis
PM	Preventive Maintenance
PSD	Process Shut Down
ReLU	Rectified Linear Unit
RFL	Reinforcement Learning
RMSE	Root Mean Square Error
RNN	Recurrent Neural Networks
SCADA	Supervisory Control And Data Acquisition
SCE	Safety Critical Equipment
SL	Supervised Learning
STD	Standard Deviation
SVD	Singular Value Decomposition
SVM	Support Vector Machines

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

The exponential increase of generated data facilitates the opportunity for companies to extract valuable information concerning their business operations. With today's emerging technology and Internet of Things, it is possible to monitor and collect data from every component imaginable. The rise of Artificial Intelligence (AI) and Machine Learning (ML) has proven to be a reliable tool in data mining across many industries. Increased data processing capacity and computationally efficient models result in new approaches that can infer intricate patterns and relationships that were otherwise not obtainable. Increasing the amount and quality of the knowledge regarding relevant systems have always been a desirable action and can be achieved by data-driven machine learning models. However, moving from theory to practice and from simulated to real-life data proposes several challenges.

In order to research the challenges of setting theory to practice, this master thesis is written in collaboration with Teekay Offshore Production AS. Teekay is a leading company in providing FPSO solutions to companies extracting oil and gas in deep water and under harsh weather conditions. The safety and reliability of the plant is the primary concern for

Teekay, and requires continuous monitoring of the system, as well as a carefully scheduled maintenance routine. Being able to move from a preventive and corrective to a predictive maintenance routine is desirable, to stay competitive in an industry currently experiencing harsh weather. Creating a model able to say something about the future health-state of a component would be a first step for the company towards an entirely data-driven maintenance routine. Machine learning has over the last decades, become a contestant to the classical statistical models in terms of forecasting. Artificial neural networks (ANN) have proven to be the state-of-the-art within the ML discipline regarding time-series forecasting. This claim is widely supported by a broad specter of research, such as Adebiyi, Adewumi, and Ayo (2014), Karbasi, Laskukalayeh, and Seiad Mohammad Fahimifard (2009) to name a few. Nevertheless, the classical statistical models, such as ARIMA, are still severe contestants to neural networks. Nevertheless, in most research, these models and their performance are compared using simulated data, which differs significantly from real-world time-series. Han, Pei, and Kamber (2011) define the quality of data by three parameters; consistency, accuracy and completeness. However, real-life data tend to be inconsistent, incomplete and unstructured. Thus, when dealing with real-life data, an extensive preprocessing phase is required. Naduvil-Vadukootu, Angryk, and Riley (2017) highlight in their research how an extensive preprocessing phase resulted in a data-driven model with a simpler structure and a more accurate predictor. Also, Elsworth and Güttel (2020) highlights the importance of preprocessing, but as real-life datasets vary from domain and problem of interest, there still does not exist a pipeline of preprocessing-techniques transforming raw data to high quality data.

The main incentive behind this thesis is to apply the theory of information technology and machine learning in a real-world application. Teekay, among other companies in the petroleum industry, has a strong desire to improve their operations by continuously integrating new tech-

nology. Teekay's available data and domain expertise combined with the ICT knowledge of the authors of this thesis, paved the way for an experiment to integrate machine learning into Teekay's maintenance strategy. The availability of real-life data also allowed for the comparison between the state-of-the-art ML-models and traditional statistical models, from a real-life perspective. Moving from theory to practice naturally results in more uncertainty and unexpected challenges. Hence, the goal of this thesis is to highlight, discuss and overcome the challenges of implementing machine learning on real-life data. The research aims to be the beginning of a digital transition, which potentially could increase Teekay's competitive advantage. This thesis includes the necessary steps in converting the available data from the Piranema vessel into data applicable in a machine learning model with the hypothesis that a machine learning model could enhance the system monitoring on the vessel. Multiple machine learning models, alongside a traditional predictive model, are created and reviewed to determine the benefits and challenges of incorporating machine learning into the maintenance strategy.

1.2 Problem description

Teekay has multiple FPSO vessels where the Piranema Spirit located outside of Brazil is the one regarded in this thesis. Since the production started in 2007, the Piranema unit has experienced a problem with its gas compression system. The FPSO, short for Floating Production Storage and Offloading, extracts and processes the crude oil, which is later stored onboard the vessel. The gas extracted from the wells is later reinjected, in order to extract the remaining crude oil from the wells. In order for the gas injection to be effective, the injected natural gas needs to maintain a certain level of pressure. Increasing the pressure of the gas is done by compressing the gas through a three-stage compression train. Figure 1.1 shows the positioning of the exhaust outlets for the OPRA turbines

and the air inlets for the Solar Turbines. The OPRA turbines function as the primary power source on the FPSO and the Solar turbines power the shafts running the compression trains. Given a certain wind degree and wind speed, the exhaust from the OPRA turbines enters the air inlets of the Solar turbines. Polluted air entering the Solar inlets decreases the turbine function, resulting in reduced compression performance. By collecting historical sensory data from the FPSO, the scope of this thesis is to create a machine learning model that can indicate the future health state of the compression trains. The health indicator aims to predict the future performance for the compression trains. Predicting future performance creates the opportunity to execute preventive actions to avoid critical events such as process shutdowns.

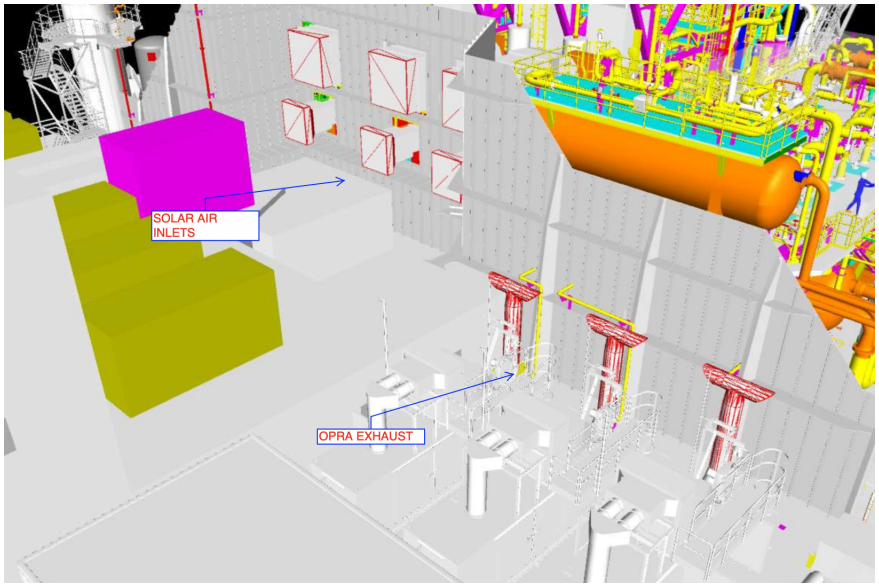

Figure 1.1: Illustrative overview of the position of the OPRA exhaust outlets and the Solar air inlets.

1.3 Objectives

- Understand the fundamentals and specifics of the OPRA turbine systems and gas compressor system, by performing qualitative analysis such as a fault tree diagram and a failure mode and effect analysis.
- Find a suitable target variable that accurately indicates the performance of the compression system.
- Research strategies and techniques regarding data collection, extraction and processing of real-life data.
- Search the literature to identify machine learning models suitable for time-series prediction.
- Implement multiple ML models as well as a statistical model for performance comparison for time-series prediction, in order to create a health-indicator.
- Explore possible benefits for Teekay Offshore Production of implementing predictive models

1.4 Scope & Limitations

The models implemented in this thesis are designed to evaluate the gas compression system on the Piranema Spirit vessel. The acquired data is limited to three OPRA turbines, three Solar turbines, three compression trains and weather data. Although this already is a complex system with many components, the models are capable of extending the even larger system. The data provided by the IP21 database consist of component sensory time-series data which are not designed to perform comprehensive data analysis. Hence, the main work involves preparing the data such that it is usable in machine learning models. The scope of the

ML models is limited to predicting a health indicator of the compression system.

1.5 Contributions

As this thesis functions as a first step of implementing ML into Teekay's maintenance strategy the main contributions are as follows:

- An in-depth review of the state-of-the-art of machine learning on real-life data. A comprehensive literature review was completed in the fall of 2019 where as Chapter 4 summarizes the relevant findings. Particularly highlighted is the use of ML on real-life applications and machine learning in time-series predictions.
- A data preprocessing approach transforming Teekay's available data into data suitable for machine learning analysis.
- An analysis of multiple machine learning based health indicators for the compression train to asses the benefits and challenges of different model configurations.

1.6 Outline

First and foremost, Chapter 1 presents an introduction to the area of research, the main objectives and the contributions from this thesis. As this thesis is written in collaboration with a company, Chapter 2 gives a brief introduction of Teekay Offshore Production AS and their maintenance routine. Chapter 3 covers the theoretical framework for the thesis, while Chapter 4 shortly summarizes the literary findings. Introduced in Chapter 5 is the problem of research. The chapter gives a detailed explanation of the compressor trains, and the components involved. Chapter 6 depicts the preprocessing phase, followed by the modelling phase in

Chapter 7. The results and analysis are summarized in Chapter 8, while a comprehensive discussion is given in Chapter 9. The conclusion of the thesis is drawn in Chapter 10, and last but not least, Chapter 11 suggest areas of research for future work.

1.7 Barriers

As this thesis was conducted during the global pandemic Covid-19, it brought some challenges to the conduction of the research. When Norway closed down the 12^{th} of March 2020, we were still in daily meetings with the engineers in Teekay, outlining the thesis formulation, and looking at potential features of interest. Domain knowledge is a crucial first step of any Machine Learning task, and when the pandemic peaked, the company had to prioritise differently. It was a time-consuming phase getting access to the software required for working from home, involving getting internal Teekay computers shipped to Oslo. We want to send our gratitude once again to Teekay, especially four employees, Erlend Meland, Kristian Holm Jensen, Bjørn Olav Ness and Jostein Vada, for helping us through this challenging phase of the research.

Chapter 2

Teekay Offshore Production AS

2.1 Introduction

Starting as a regional shipping company, Teekay Offshore Production is now a market leader in providing FPSO solutions to oil production companies. FPSO is an acronym for "Floating Production Storage and Offloading", used in the production and processing of hydrocarbons and storage of oil. As the world's leading marine services company, Teekay is an important link in the global energy supply chain. The goal is to contribute to sustainable business and environment by limiting the climatic footprint. Sustainability and safety are among the critical values for the company, and all decisions consider the people, planet and potential profit.

2.2 Teekay's Floating Vessels

2.2.1 FPSO

FPSO is a floating unit used to process, produce and store oil until it is shipped to shore (Gupta and Grossmann 2011). The floating production system receives and process the crude oil from a sub-sea reservoir, sep-

arates the refined oil and stores it onboard until it is offloaded to tanker vessels. The majority of FPSO units are shaped like ships, and due to their shape they are suitable for a broad range of water depth and environmental conditions (Duggal, Heyl, Ryu, et al. 2009). FPSOs are often preferred in frontier offshore regions as they are easy to install. Using such a device instead of oil-rigs results in reduced upfront investments, retained value because the device can be relocated to other fields and a low abandonment costs.

2.2.2 The Piranema Spirit Vessel

This thesis explores a current issue at the Piranema Spirit vessel, one of Teekay's FPSO units. Piranema is a new built FPSO, designed by Teekay for Petrobras, a Brazilian multinational corporation in the petroleum industry. The unit has been operating since 2007 and has the capacity of producing 25.000 barrels per day (BPD). Piranema is also the name of the oil-field, discovered in 2001 and is localized at the continental shelf the Brazilian state of Sergipe, 37 km from the coastline. The production started in 2007, and back then the Piranema unit produced on average 4254 barrels per day of refined oil (OffShore-Energy-Today 2019). Shown in Figure 2.1 is a photo of the unit in operation.

Figure 2.1: Photo of the FPSO vessel Piranema. (Photo:Petrobras)

2.3 Maintenance Strategy at Piranema

The maintenance strategy on the vessel focuses on ensuring the required standards to be met, relating to safety-critical equipment, reduce unplanned shutdowns, and optimization of the total cost for the maintenance routine. The strategy is obtained by implementing sufficient and correct maintenance routines, followed by a precise and correct execution. Planning of forthcoming preventive maintenance routines, and ensuring the availability and quality of spare parts is a crucial step for meeting these requirements. Another crucial step includes having the necessary systems, procedures, routines, personnel and material for testing, inspection and maintenance of all technical equipment clearly defined. Teekay highlights the focus on the improvement of documentation of maintenance and operation. All preventive and corrective maintenance tasks are specified using the Star Information and Planning System described in Section 2.3.4.1.

2.3.1 Constraints

As the FPSO is located in the Brazilian shore, it is the Brazilian Shelf State legislation that sets the requirements and constraints. An example is that the Piranema Vessel needs to follow some essential guidelines, such as the NR-13[1] , which is a regulation establishing the minimum conditions for the installation and operation of boilers and pressure vessels in Brazil. In addition, Teekay has its own class and flag state requirements. The contractors can state individual requirements in the contract. For example, the current contractor fines Teekay if the water injection system is not functioning, even if its usage is not required. The contractor also has requirements in terms of up-time, but not what Teekay does to satisfy the up-time requirement.

[1] `http://www.braziliannr.com/brazilian-regulatory-standards/nr13-boilers-and-pressure-vessels/`

2.3.2 Maintenance at the FPSO

The maintenance onboard the FPSO is carried out as a combination of preventive, corrective and conditional maintenance. An integrated part of the maintenance strategy is to have spare parts and other equipment available onboard for preventive maintenance, and the predefined critical equipment available for corrective maintenance. The items are classified in terms of criticality, where criticality is defined as the potential consequence of a failure based on safety, health, environment, and direct cost to restore its function. The result of such a classification is the guidance for the selection of maintenance strategy and prioritization of maintenance task. A component in need of maintenance is defined as an item with lost intended function, its function significantly reduced or if it is malfunctioning. STAR generates work orders for preventive maintenance, and safety-critical elements are tested regularly to ensure that they meet the performance standards. Work orders for corrective maintenance cover unplanned maintenance activities and breakdowns.

2.3.3 Condition Monitoring

By using condition monitoring, the development of degradation is estimated based on different measurements such as performance, vibration level and temperature. The necessary corrective or preventive actions can be planned according to the level of degradation. For various pumps and electric motors, such monitoring is an adequate measure. However, for more sophisticated types of machinery, like compressor, turbines and diesel engines, comprehensive data collection and calculation is required to produce an adequate evaluation of the performance.

2.3.4 Information Systems

2.3.4.1 STAR Information and Planning System

STAR Information and Planning System is a computer-based maintenance system used for the planning and administration of the maintenance work on the FPSO. The primary purpose of such a system is to provide an effective administration of the maintenance work, an overview of historical data and available spare parts. STAR IPS controls the planning of maintenance including implementation and reporting of all completed preventive and corrective activities. All maintenance activities are historically reported into STAR IPS, where each report contains relevant findings, consumption of spare parts and work description. The reported data allows for continuous improvement of the maintenance activities and the selection of required equipment.

2.3.4.2 Win CC

The Figure in 2.2 shows a snapshot of the system used by the control room operators when monitoring and controlling all the processes on the production plant. A general term for this type of system is a supervisory control and data acquisition system (SCADA), and the specific type used on the Piranema vessel is Win CC, delivered by Siemens. By using such systems, the process engineers can control and monitor the process plant, by gathering and analyzing data from the different sensors and valves.

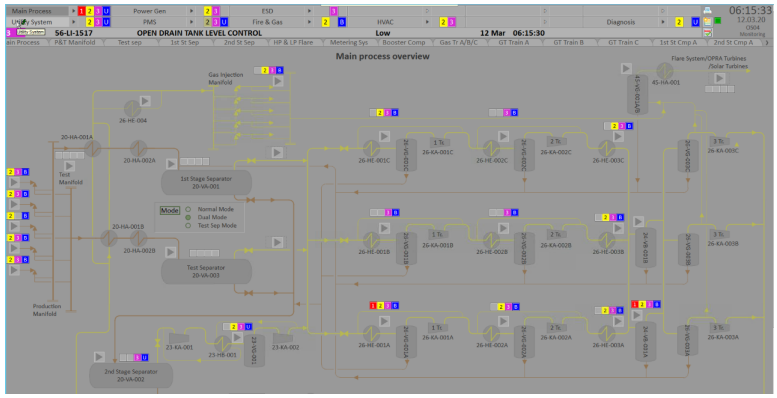

Figure 2.2: Snapshot of the SCADA system on the Piranema Vessel

2.3.4.3 IP21

The Win CC system receives sensor data in real-time in order to control and monitor the process plants. The data is later stored in the IP21 information system. IP21 is known to be a powerful tool used in a variety of fields within petroleum production. The system is used to improve work practices and to reduce downtime by providing historical data for trending, reporting and other analyses.

Chapter 3

Theoretical Framework

3.1 Data Preprocessing

The exponential increase in generated data facilitates the opportunity for companies to extract valuable information concerning their business. However, utilizing real-life data is easier said than done. Real-life data is often inconsistent, incomplete and gathered from multiple sources. In order to utilize data-mining (DM) and machine learning (ML) techniques to extract valuable information, the data requires processing such that it is suitable and optimal. The preprocessing step is necessary to provide high-quality data to the subsequent models. Han, Pei, and Kamber (2011) defines the quality of data by the three parameters; consistency, accuracy and completeness. Raw real-life data is often of low quality; hence there is a great emphasis on data preprocessing in every DM and ML problem. Different preprocessing steps and techniques are reviewed in the following subsections.

3.1.1 Data Preparation

Data preprocessing can be divided into two main steps, data preparation and data reduction (Garcia, Luengo, and Herrera 2015). Roughly,

the data preparation step involves the transformation of raw data into usable and model-suitable data. The data reduction step transforms the data such that model performance increases. The data extracted from relational databases are oftentimes not processed and contain errors, inconsistencies and noise. The following subsections explain the most common techniques in transforming raw data into usable data.

3.1.1.1 Data Cleaning

Data cleaning is an essential step of the preprocessing phase when dealing with real-life data. The data might be dirty, meaning that it can contain missing or wrong values (Garcia, Luengo, and Herrera 2015). Having a dirty dataset will profoundly impact the performance of machine learning models in various degrees. Especially susceptible to dirty data are distance-based models which are highly dependent on the data values (Garcia, Luengo, and Herrera 2015). If these values are dirty, there is a high probability that the model will provide incorrect predictions. According to W. Kim et al. (2003), dirty data can be divided into three forms; missing data, wrong or noisy data and inconsistencies. Identifying and processing dirty data instances is not straight forward. The wrong detection will result in the removal of correct data, and missed detections lead to falsely trained models. There are different techniques to handle missing data and wrong data:

- Noisy data is often due to random errors in the measured variables, which results in outlier instances. There are multiple techniques to identify and remove these outliers, which is further explained in Section 3.1.1.5.

- Similarly, there exist multiple techniques to treat missing values in a dataset. Section 3.1.1.3 presents a further explanation of the topic.

3.1.1.2 Data Normalization

Real-life data is often collected from multiple sources, and the attribute's properties may vary. Most machine learning algorithms do not perform well when the numerical range of the attributes have different scaling. Thus, data normalization or feature scaling, as it is sometimes called, is required. Normalization has also proven to speed up the learning process of artificial neural networks (Garcia, Luengo, and Herrera 2015).

3.1.1.2.1 Min-Max Normalization

Min-Max Normalization is a rather simple scaling technique, used to transform each attributes to a standard desirable range. Given a numerical attribute A with numerical values v, the goal is to scale the values to an optimal range denoted by $[new-min_A, new-max_A]$. The new values of v are then denoted as v' and calculated in Equation 3.1.

$$v' = \frac{v - min_A}{max_A - min_A}(new - max_A - new - min_A) + new - min_A, \quad (3.1)$$

where max_A is the original maximum value of attribute A and max_B is the original value of attribute B.
It is common when normalizing the data to assign the attributes in the range [0,1], which means that $new - min_A = 0$ and $new - max_A = 1$. Figure 3.1 is an example of scaling within the interval [0,1]. The interval [-1,1] is also very often applied.

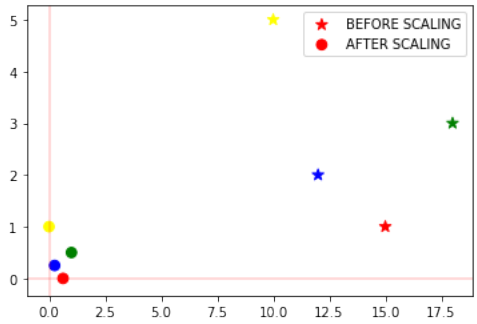

Figure 3.1: Graphical plot illustrating the effect of the Min-Max normalization technique. The original datapoints are scaled to the range of [0,1].

3.1.1.2.2 Z-score Normalization

Another widely used normalization technique is the Z-score normalization, which utilizes the average value and the standard deviation of the attributes. Given an attribute A with values v, mean value $\overline{A}$ and standard deviation σ_A, then the new value, v' is calculated in Equation 3.2.

$$v' = \frac{v - \overline{A}}{\sigma_A}, \tag{3.2}$$

where the mean value of A is calculated in Equation 3.3.

$$\overline{A} = \frac{1}{n}\sum_{i=1}^{n} v_i, \tag{3.3}$$

where n is the total number of values v of attribute A. The standard deviation is calculated in Equation 3.4.

$$\sigma_A = +\sqrt{\frac{1}{n}\sum_{i=1}^{n}(v_i - \overline{A})^2} \tag{3.4}$$

This transformation ensures that the attribute has a mean value of 0 and a standard deviation equal to 1. Figure 3.2 illustrates the Z-score normalization technique.

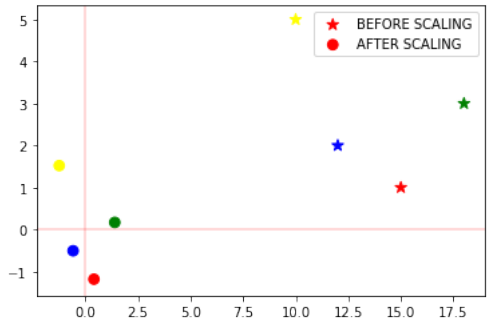

Figure 3.2: Graphical plot illustrating the effect of the Z-score normalization technique. The original datapoints that are transformed such that the attribute has a mean value of zero and standard deviation of one.

3.1.1.2.3 Decimal Scaling Normalization

A third possible normalization technique is the decimal scaling normalization. This method ensures that the values of an attribute are lower than 1 after the transformation. This is done by a simple division, as shown in Equation 3.5.

$$v' = \frac{v}{10^j}, \tag{3.5}$$

where v is the original value, v' is the new value after transformation and j is assigned such that $new - max_A < 1$.

3.1.1.3 Data Imputation

When working with real data, missing data points will occur in the dataset. Missing data points can be a result of component failure, sensor failure, among many other reasons. *Imputation* is the process of substituting these missing data points, often referred to as *NaN* (Not a Number), with substituted values (Jerez et al. 2010). The following subsections summarizes the most commonly used imputation techniques.

3.1.1.3.1 Univariate Imputation

Univariate imputation techniques substitute the missing value by replacing the *NaN* values with a value derived from its own observations. Summarized below are the most commonly used techniques for univariate imputation.

Mean

Using mean as imputation technique is as simple as substituting the missing value with the mean value of the attribute. Such an approach has the benefit of not changing the sample mean for the variable, and at the same time being computationally cheap. Mean is an unbiased estimate for an observation randomly selected from a normal distribution. Nevertheless, the missing values in a real-life dataset are seldom random, and such an approach may lead to inconsistent bias. The simple technique does neither consider the time-aspect if the dataset was to be sequential.

Median

The median refers to the middle value in a sequence of observations, when the data points are arranged from the least to the most significant number. Replacing the missing values with the median of an attribute is suitable when the attribute has a skewed distribution, and the number of missing observations is low. For an attribute having a large number of missing values, such a technique will result in a significant loss of variation.

Observations

Substituting the *NaN* values with other observations is a commonly used technique when dealing with time-series data. The missing data-points can either be replaced using *Forward Filling* which propagates the last observed non-null value forward, or *Backward Fill* which propagates the first observed non-null backwards.

Interpolation
By using interpolation, a mathematical function is fitted to the existing data points, and this function is used to impute the missing data points. The simplest type of interpolation is *Linear Interpolation*, which calculates a mean between the values before and after the missing data point. Whereas *Polynomial interpolation* is a mathematical function fitted to the existing data points of the lowest possible degree, later used to calculate the missing data points. Interpolation is a commonly used technique when dealing with time-series, as it considers the sequential property of the data.

3.1.1.4 Multivariant Imputation

In contrast to the univariate techniques, multivariate imputation techniques consider all the available features and its observations when imputing and substituting for the missing values.

KNN
K-nearest neighbours is a well-known machine learning algorithm, and a commonly used imputation technique. The algorithm imputes the missing values by finding the k nearest observations, referred to as neighbours, and calculate the mean or the weighted mean of the neighbours. Hence, a distance metric is required, and the Euclidean distance, the Manhattan distance and the Minkowski Distance are commonly used, as seen in Equation 3.24, 3.25 and 3.26, respectively. The algorithm is elaborated further in Section 3.3.1.2.

Model Based Imputation
Model-based imputation is an iterative technique, where a data-driven model is created, and the missing values are treated as the target variable, one at the time. By creating different machine learning models, one can iterate through the whole dataset and replace all the missing values

with the output of the ML-model. Typical machine learning model used for this matter are *Logistic Regression*, *Decision Trees* and *Support Vector Machines*.

3.1.1.5 Noise Identification & Outliers

Noisy dataset with many outliers can lead to a falsely trained model. It is important to detect and observe abnormal instances. Some outliers might occur due to measurement errors which then needs to be transformed such that the model trains on valid data. Another reason for outliers is that the system is experiencing abnormal behaviour. In such cases, it is interesting to detect these outliers such that preventive actions can be executed. In both scenarious, detection of outliers are essential and the following subsection presents one of these techniques.

3.1.1.5.1 DBSCAN

There exist several ways to deal with outliers; one of them is the cluster-based algorithm called Density-based Spatial Clustering of Applications with Noise (DBSCAN). As the name implies, the DBSCAN is a density-based clustering algorithm. The goal of a density-based clustering approach is to identify randomly shaped clusters (Ashour and Sunoallah 2011). Clusters are defined as dense groupings of data points which are separated by sparse regions of data points. An example is depicted in Figure 3.3, where two clusters, the red and green data points, have been identified by their dense regions of data points.

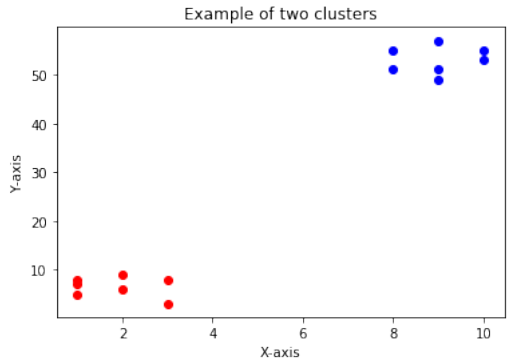

Figure 3.3: Graphical plot illustrating datapoints that form two clusters.

The main benefit of the DBSCAN and density-based algorithms, in general, is that they require less domain knowledge in terms of determining the number of clusters, compared to non-density-based clustering algorithms. For example, the K-means algorithm, which is a non-density-based clustering algorithm, requires a suggestion of the number of clusters in the dataset as an input, whereas the DBSCAN will detect the natural number of clusters automatically (Gan, Ma, and J. Wu 2007). However, it does require some domain knowledge to determine its' two main parameter inputs:

- eps is the minimum distance between two data points. If the distance between two data points is lower than or equal to the predetermined eps, then these two data points are considered as neighbours.

- min_point is the minimum amount of data points to create a cluster.

The algorithm works as follows, given a set of data points K with predetermined values of eps and min_point, then all of the data points can be classified as either an outlier, a core-point or a density-reachable point.

- A point x is a core point if there exist at least min_points points that are within the distance of eps. This includes point x itself.

- A point y is classified as directly reachable from core point x, if the distance between point y and point x is less than eps.

- A point y is classified as reachable to core point x, if there exist a path $x_1, ..., x_n$, where $x_1 = x$ and $x_n = y$ and each x_{j+1} is directly reachable from x_j. In other words, the entire path must consist of core points, with the possible exception of the final point y.

- Points that are not classified as a core point, nor a density-reachable point is classified as an outlier. This means that the outlier point is not reachable from any other point.

The DBSCAN creates clusters and core points by finding data points that have at least min_points data points that are within the distance eps of the core point. Then it iteratively finds directly reachable points from the core points. When all points are processed, the natural number of clusters have been discovered, as well as each point have been assigned to its corresponding cluster or defined as an outlier.

3.1.2 Data Reduction

With today's technology and IoT, the collected datasets are often extensive, containing millions of samples with thousands of attributes. Although many machine learning models require vast amounts of data, the model efficiency can be improved by reducing the size of the dataset, without affecting the accuracy of the model. Data reduction techniques are used to create new representations of the dataset, where most of the integrity is kept, and the volume is reduced (Garcia, Luengo, and Herrera 2015).

3.1.2.1 Feature Selection

Feature selection is the process of creating a subset of relevant features from the original dataset (Guyon and Elisseeff 2003). The end goal is a reduced subset that makes the predictive model nearly as accurate or sometimes even more accurate than a model with the original dataset. There are multiple reasons why feature selection is an essential preprocessing step in machine learning. First, reducing the dataset also reduces the computational complexity of the problem. With today's technology, it is possible to monitor and gather data from every imaginable component. However, not all data are equally relevant and can be disregarded to achieve more efficient models. In addition to computational complexity, feature selection also reduces the amount of redundant information, which makes it easier to understand the behaviour of models. Also, performing analysis on high-dimensional data might be more challenging than on lower-dimensional data due to the *curse of dimensionality* (Friedman 1997). The next subsections explain the automated feature selection process by reviewing the most common selection techniques.

3.1.2.1.1 Amount of Variance

One way to categorise the relevance of the features is to measure how much they vary. A feature that does not vary much often has little predictive power (Guyon and Elisseeff 2003). In other words, the feature does not contain much information. For instance, a feature with a zero variance will always have one distinct value which will not have an impact on the performance of the model. Therefore, it is desirable to identify features with zero or low variance, which is done with the following equation:

$$VAR(x) = \sigma^2 = \frac{1}{n}\sum_{i=n}^{n}(x_i - \mu)^2, \tag{3.6}$$

where x_i is the value of sample i of feature x, μ is the mean value of feature x, $VAR(x)$ is the variance of feature x and σ is the standard deviation of feature x.

3.1.2.1.2 Feature Correlation

Removing redundant information will reduce computational complexity without significantly reduce model performance. Redundant information can be identified by calculating the correlation coefficients between the features (Meng, Rosenthal, and Rubin 1992). If two features are highly correlated, one of them should be removed. The three most common ways to calculate the correlation coefficients between features are:

- Pearson's Coefficient of Correlation
- Spearman's Ranking Method
- Kendall Tau's Coefficient of Correlation

Pearson's Coefficient of Correlation

Pearson's Coefficient of Correlation is often used to determine how correlated two numerical attributes are (Meng, Rosenthal, and Rubin 1992). The formula is given by:

$$r_{A,B} = \frac{\sum_{i=1}^{m}(a_i - \overline{A})(b_i - \overline{B})}{m\sigma_A\sigma_B} = \frac{\sum_{i=1}^{m}(a_i b_i) - m\overline{AB}}{m\sigma_A\sigma_B}, \tag{3.7}$$

where a_i and b_i are the values of attributes A and B, $\overline{A}$ and $\overline{B}$ represent the mean values of A and B, m is the number of instances, and σ_A and σ_B are the standard deviation of A and B, respectively. $r_{A,B}$ ranges between -1 and 1, where a positive value represents a positive correlation between attribute A and B. In other words, if the value of attribute A increases, then the value of attribute B increases. A negative $r_{A,B}$ represents a negative correlation, hence if the value of attribute A increases,

then the value attribute B decreases. The magnitude of $r_{A,B}$ describes how positively or negatively correlated the two attributes are. $r_{A,B} = 0$ means that the two attributes are independent of each other.

Spearman's Ranking Method

Spearman's Ranking Method between two features is the same as the Pearson Coefficient of Correlation with the ranked values of the features. There exist multiple ranking methods Wilcoxon (1992), which are out of the scope of this thesis. The Spearman's rank coefficient is then:

$$\rho_{r_A,r_B} = \frac{Cov(r_A, r_B)}{\sigma_{r_A} \cdot \sigma_{r_B}}, \tag{3.8}$$

where r_A and r_B denote the ranked variables A and B, respectively. Similarly, σ_{r_A} and σ_{r_B} represent the standard deviation of r_A and r_B.

Kendall Tau's Coefficient of Correlation

Kendall Tau's Coefficient of Correlation describes the statistical association between features by their rank (Meng, Rosenthal, and Rubin 1992). A feature's rank is the relationship between the samples, where the samples relationship is defined as either greater than, less than or equal to it's successor. Let $(a_1, b_1), (a_2, b_2), ..., (a_m, b_m)$ be a set of samples for feature A and B. Any pair of samples (a_i, b_i) and (a_j, b_j), where $i < j$ is:

- concordant if both $a_i > a_j$ and $b_i > b_j$, or if both $a_i < a_j$ and $b_i < b_j$,
- discordant if $a_i > a_j$ and $b_i < b_j$ or if $a_i < a_j$ and $b_i < b_j$ and $b_i > b_j$,
- neither concordant nor discordant if $a_i = a_j$ and $b_i = b_j$.

The Kendall coefficient, τ is then defined as:

$$\tau = \frac{(number\,of\,concordant\,pairs) - (number\,of\,discordant\,pairs)}{\binom{m}{2}}, \tag{3.9}$$

where $\binom{m}{2} = \frac{m(m-1)}{2}$ is the *binominal coefficient* which refers to the number of ways to choose two samples from m samples. If the ranking between X and Y is perfect, then the τ coefficient equals 1. If the ranking is perfect negative, then the τ coefficient equals -1. If X and Y are independent of each other, then the τ coefficient is zero.

Usage of Correlation Methods

The difference between the correlation techniques are summarized below:

- Pearson's Coefficient of Correlation is a parametric measure which means that it assumes that the sampled data can be modelled by a probability distribution with defined parameters (Geisser and Johnson 2006). Besides, it tries to evaluate the relationship between two variables with a linear function.
- Spearman's Ranking Method is a non-parametric measure which uses the rank correlation between two features. It attempts to assess the relationship with a monotonic function (Geisser and Johnson 2006).
- Kendall Tau's Coefficient of Correlation is a non-parametric measure. Similar to Spearman, it tries to assess the relationship between two features with a monotonic function. It is often used when the number of samples is small (Meng, Rosenthal, and Rubin 1992).

A correlation matrix can be created by using either of the abovementioned methods which displays the correlation between all features in

a dataset.

3.1.2.1.3 Wrapper methods

The idea behind wrapper methods is to select subsets of features that are separately used to train a model. By comparing the performance of each trained model, the most optimal subset of features can be selected. There exist multiple different wrapper methods, among these are the *Forward Feature Selection* and the *Backward Feature Selection*. The forward feature selection method begins with one feature, and iteratively adds features and measure the model's performances. The arrangement of the way features are added varies. It is ideal to begin with the feature that yields the highest model performance on its own, then add the next best feature and so on. In order to do this, each feature has to be fed into the model and their respective model performances collected. Another arrangement could be to sort the features by their correlation with the target variable. By using these arrangements, it is possible to locate when the model is not significantly improving and remove the remaining features. Backward feature selection works similarly. However, it begins with all the features and iteratively drops one by one. The same arrangements, only reversed, can be applied to this method, and again locate when the model performance is significantly decreasing.

3.1.2.2 Feature Extraction

Feature extraction is the process of reformatting, combining and transforming the original dataset, to create new features from the existing ones (Khalid, Khalil, and Nasreen 2014). By creating new features, the dimensionality of the original dataset is reduced, by filtering out noise and data without significance. A dataset containing more features than necessary can potentially harm the learning of any algorithm, and make the learning phase more computationally expensive. The idea behind feature extraction is to find a lower dimension p, that is a good represen-

tation of the original dataset of dimension n. The downside of applying feature extraction techniques to the dataset is the interpretability of the data, as new features p might be a combination of several features, hence loosing their physical meaning. In the most real-world dataset, the training instances are not spread uniformly across all dimensions. Many features are almost constant, while others are highly correlated. As a result of this, the instances may lie within a much lower-dimensional subspace of the high-dimensional space.

3.1.2.2.1 Principal Component Analysis

Principal Component Analysis (PCA) is the most popular dimensionality reduction algorithm. The algorithm works by first calculating the *principal components* of the dataset, using *Singular Value Decomposition* (SVD). SVD decomposes the matrix $\mathbf{X}$ into the dot product of the tree matrices shown in equation 3.10, and the matrix $\mathbf{V}^T$ contains the principal components. After identifying all principal components, the dimensionality of the dataset can be reduced down to m dimensions defined by the m first principal components, as shown in Equation 3.11. $\mathbf{W}_m$ is the matrix containing the m first principal components (Jolliffe and Cadima 2016).

$$\mathbf{U} \cdot \Sigma \cdot \mathbf{V}^T = \mathbf{X} \tag{3.10}$$

$$\mathbf{X}_{m-proj} = \mathbf{X} \cdot \mathbf{W}_m \tag{3.11}$$

A requirement for the PCA is to chose a number of dimensions that preserves a sufficiently large portion of the variance in the original dataset.

3.1.2.2.2 Kernel Principal Components Analysis

The Kernel Principal Components Analysis (KPCA) is an extension of PCA introducing the *Kernel Trick*. *The Kernel* is a mathematical technique which maps the data instances into a higher dimension space,

known as the feature space. By projecting the data into a higher dimension, non-linear relationship between the data instances can be discovered. The approach is based on the logic that a linear decision boundary in a high dimensional space is equivalent to a non-linear decision boundary in a low dimensional space. By mapping data to a high dimensional space, the non-linear structures in the data can be represented (Géron 2019). *KPCA* applies this logic and can represent the non-linear structures of the data. Thus, a dataset not linearly separable on the original attributes can be separated by mapping the attributes to a higher dimensional feature space. The KPCA does so by introduction a kernel function, which is a function capable of computing the dot product of $\phi(a)^T \cdot \phi(b)$ based on the original vectors **a** and **b**, not knowing the transformation ϕ. The most common kernel functions are the linear, the polynomial, the Gaussian radial basic function and the Sigmoid function, shown in equation 3.12, 3.13, 3.14 and 3.15.

$$\text{Linear:} K(\mathbf{a},\mathbf{b}) = \mathbf{a}^T \cdot \mathbf{b} \tag{3.12}$$

$$\text{Polynomial}: K(\mathbf{a},\mathbf{b}) = (\gamma \mathbf{a}^T \cdot \mathbf{b} + r)^d \tag{3.13}$$

$$\text{Gaussian RBF} K(\mathbf{a},\mathbf{b}) = exp(-\gamma \left\| \mathbf{a}^T \cdot \mathbf{b} \right\|^2) \tag{3.14}$$

$$\text{Sigmoid} K(\mathbf{a},\mathbf{b}) = tanh(\gamma \mathbf{a}^T \cdot \mathbf{b} + r) \tag{3.15}$$

3.2 Machine Learning

The term *Machine Learning* was first coined in 1959 by the American pioneer in the field of artificial intelligence Arthur Samuel. Samuel described ML as a field of study where computers can learn without being explicitly programmed (T. M. Mitchell 1997). In other words, without any

human interaction computers can improve their learning process based on experience. The question then arises; *what does it mean that a computer is learning?* An Engineer-oriented definition of machine learning was given by Tom Mitchell back in 1997

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

T. Mitchell 1997 Page 2

An example could be, given a task T of predicting stock prices. An ML model can learn from previous experience E, in this case, historical stock prices. If the model has successfully learned, it will then do better on predicting future stock prizes P. The way the model learns, is similar to the way humans are programmed to learn from experience. Consider the task, T, of tossing paper into a bin. The performance measure P is now the distance between the bin and the position of the paper. The first attempt might result in the paper landing way past the bin. The force is therefore decreased in the second attempt. This time the angle of the toss was not perfect, and therefore adjusted for in the third attempt. Similarly, when an ML model predicts stock prizes, an error is calculated and adjusted for in the next prediction attempt.

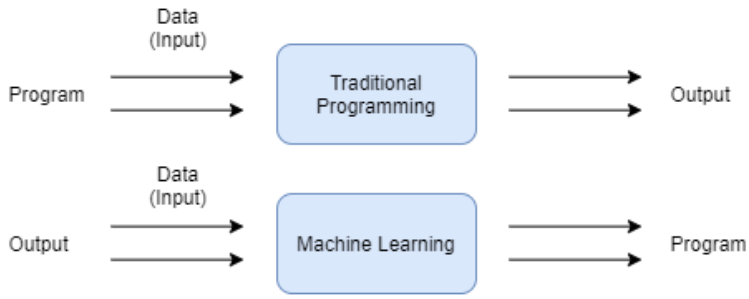

Figure 3.4: Comparative overview between traditional programming and machine learning.

The basic difference between ML and traditional programming is depicted in Figure 3.4. Traditional programming requires human interaction to create the program logic and decision rules. Then given some input data, the desired output can be obtained from the created program. In contrast, machine learning models create the program logic and decision rules by inferring the relationship between the input and the desired output. This program is then used to produce an output from new unseen inputs. Implicitly, this means that ML models require some learning response or output signal (T. M. Mitchell 1997).

3.2.1 Classification of Machine Learning

The different types of machine learning problems can be separated into three groups, depending on the nature of the learning response.

3.2.1.1 Supervised Learning

Supervised learning (SL) is used for the majority of practical machine learning problems (Brownlee 2016). As the name implies, supervised algorithms are trained under supervision. This means that the data fed into the algorithm also include the correct response variables, often referred to as *labels*. A definition provided by Liu and Y. Wu (2012) defines supervised learning in the following manner.

"Supervised Learning is a machine learning paradigm for acquiring the input-output relationship information of a system based on a given set of paired input-output training samples".

Liu and Y. Wu 2012

The dataset is partitioned into two parts, a *training set* and a *validation set*, in order to train the algorithm. The training set consists of all relevant data features, as well as the response variable. By using this dataset, the goal is to approximate a mapping function based on the input data (X) and use this function to predict the output variable (Y).

$$Y = f(X)$$

The learning process stops when the algorithm reaches the desired level of performance, which is measured by using the trained algorithm on the validation set, which does not contain the response or target variables. Compared to human learning, students are provided with correct approaches to solve different problems before an exam. The students derive rules and logic from this information and use it when solving new problems on the exam.

Supervised learning can be further categorized, depending on the nature of the problem it attempts to solve. These are *regression* and *classification* problems, where it is important to identify the properties of the response variable.

- **Regression:** Problems where the model tries to approximate a mapping function based on the input data (X) to continuous or numerical output variables (Y). For instance, predicting stock prices is a regression problem since the response or output variable, *stock price* is continuous.

- **Classification:** Problems where the model tries to approximate the mapping function based on the input data (X) to discrete or categorical output variables (Y). Image recognition is a type of classification, where the task could be to determine whether the image displays a car or a bike. The response or output variable is now discrete.

3.2.1.2 Unsupervised Learning

Unsupervised learning is when the model infers relationships on the input data when there is no response variable present. Since there is no response variable, this is not the type of algorithm that is used in prediction and decision making. However, by restructuring the input data, it can identify intricate patterns like associating features into classes or locating uncorrelated values (Ghahramani 2003). Deriving these patterns provides insight regarding the nature of the data, which is useful information when implementing a supervised learning model. This type of learning can be compared to when humans determine different classes from multiple objects, like categorizing a set of books into different genres. Unsupervised learning can be grouped into two subcategories, *cluster analysis* and *association rules*.

3.2.1.3 Reinforcement Learning

Reinforcement learning (RFL) is a learning system where an *agent* observes the environment and performs actions based on the observations. Different actions rewards different scores and the agent seeks to obtain the highest possible score. Hence, such a system is defined by the fact that there is less feedback, since it is not the proper action but only the evaluation of the chosen action that is given by the external expert. Such an algorithm learns the best strategy by itself, called *policy*, over time. Such a policy defines what action the agent should choose in a specific situation.

3.2.2 Data in Machine Learning

Machine learning models are only as good as the data from which they are built. Therefore, both the quantity and the quality of the data collected are crucial for developing high-performance models. In addition, the models require some response to determine if they have successfully learned. In other words, an indication of how well the model generalizes over the problem domain. This is done by dividing the collected data into a training, validation and test set. As displayed in Figure 3.5, both the training and validation set is used during training. The ML algorithm infers the relationship between the input variables and the target variable in the training set. This relationship is continuously evaluated while training by making predictions on the input variables of the validation set. The predictions are measured against the correct target variables from the validation set, which yields a prediction error. The ML algorithm then adjusts its parameters depending on the magnitude and direction of the calculated error. This process is repeated until the ML algorithm is satisfied with the prediction error or when it is not able to improve its performance. When the training has ended, the final model is evaluated by making predictions on the unseen input variables from the test set. The prediction error on the test set reflects the performance of the final model.

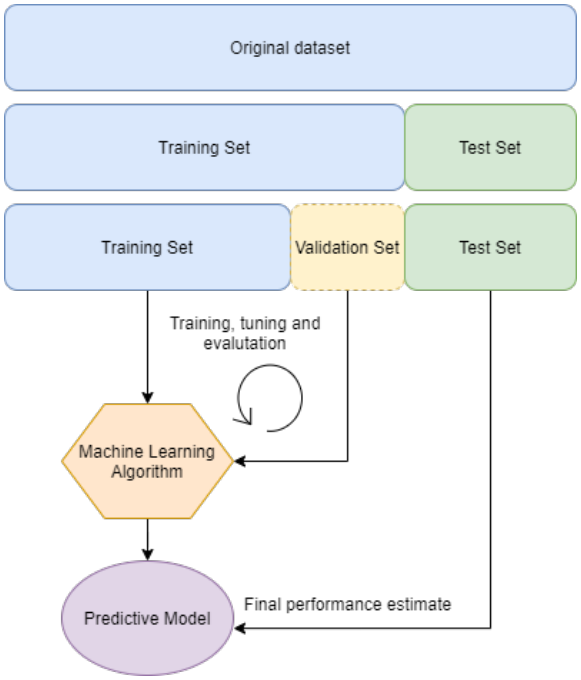

Figure 3.5: Explanatory illustration of the learning process in machine learning frameworks.

3.2.2.1 Cross-Validation

When a model has completed its training, en error estimate can be derived by the difference in the predicted values and the correct target values. This is referred to as the evaluation of the residuals Browne (2000) and reflects the prediction error on the training set. However, this error estimate does not reflect the trained model's ability to make a prediction on unseen data. *Cross validation* is used to evaluate how well the trained model generalize on new and independent data (Browne 2000). Cross-validation techniques differ in the way that the original dataset is divided into a training set and a test set. The validation set depicted in Figure 3.5 is only used during training and must not be confused with the unseen test set when referring to cross-validation.

3.2.2.1.1 Holdout Validation

Holdout validation is the simplest form of cross-validation. The original dataset is separated into a training and test set. During training, the model attempts to infer the relationship between the input variables and the target variables from the training set. The model performance is then evaluated by making predictions on the unseen test set. This performance measure is more accurate than the evaluation of the residuals because it, to some extent, describes how well the model generalizes on independent data. The downside of this validation technique is that it is heavily dependent on the samples in the training and test set. Ideally, both the training and test sets should represent the entire problem domain. However, this is often not the case in real-world examples. If the training set consists of observation such that it partially represents the problem domain, it will not be able to capture patterns that might occur in the test set.

3.2.2.1.2 K-fold Validation

K-fold validation is a validation technique used to improve the holdout validation by creating multiple instances of the ML model which train on different arrangements of the dataset. It divides the original dataset into k subsets and performs k iterations of the holdout validation method such that for each iteration one of the k subsets is the test set, and the training set is remaining $k-1$ subsets combined, as shown in Figure 3.6. The final model performance is calculated by averaging over the performance from each iteration. Training and validating across the entire dataset improves the model's inferred relationships between input and target variables (Fushiki 2011).

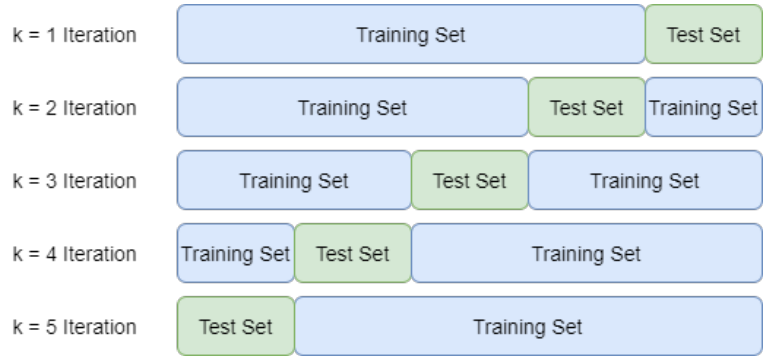

Figure 3.6: Dataset partition during k-fold cross validation. Five different arrangement of the original dataset are created, indicating five complete iterations of the entire data.

3.2.2.2 Cross-Validation with Time Series

Additional considerations need to be taken into account when performing cross-validation on time series data. It is critical to preserve the time series aspect of the data, such that the model can leverage this information (Bergmeir and Benıtez 2012). Hence, the data needs to be temporally separated into training and test sets. This means that the test set must contain observations that occur chronologically after the observations in the training set. One validation technique that preserves the time-series aspect is the holdout validation method, given that the test comes after the training set. However, as mentioned above, using holdout validation is not sufficient enough in order to obtain a robust predictive model. K-fold validation is not a good validation method on time series data, as it does not preserve the sequential aspect of a time-series.

3.2.2.2.1 Nested Cross Validation

Figure 3.7 displays how the data can be partitioned when using the nested cross-validation technique. Similar to the k-fold validation, the nested cross-validation technique performs multiple iterations on the original data (Hasselt 2013). However, to preserve the time series property of

the data, the test set of each partition must contain observations that occur after the observations in the training set. As a result, the model can not utilize the entire dataset in each partition as the k-fold validation technique. The final model performance is the combined average of the model performances on each partition.

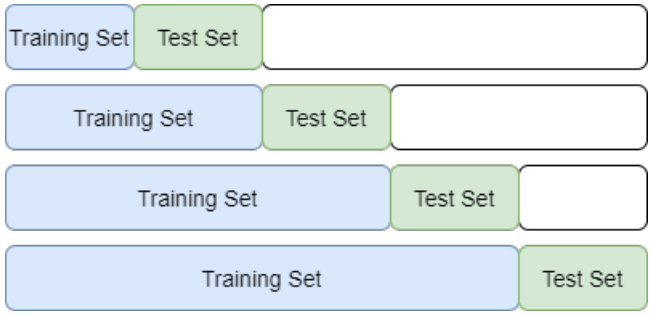

Figure 3.7: Dataset partition during nested cross validation. The datasets are arranged such that they preserves the time series aspect.

Another important consideration is the positioning of the validation set. As explained in Section 3.2.2, the validation set is continuously used during training to tune and evaluate the training parameters. In order to accurately train the model on time series data, the validation set must contain observations that occur after the observations in the training set, as well as before the observations in the test set. Figure 3.8 shows the entire process of nested cross-validation on time series data. The data is separated into three partitions with correct positioning of the training, validation and test sets. Three instances of the model are created and trained on the training and validation subsets from each partition. Its residuals and prediction error on the test set are evaluated for each model. The final model performance is then the combined average of the three individual models.

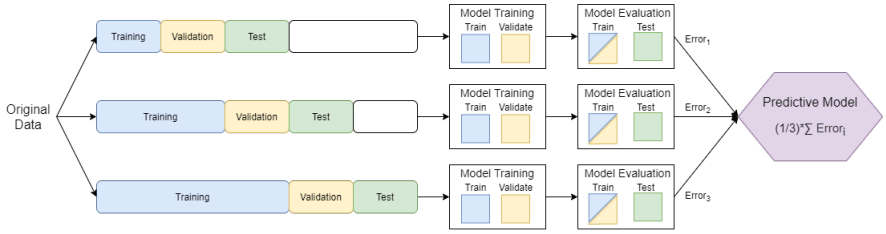

Figure 3.8: Modeling approach when targeting time series data with nested cross validation.

3.3 Machine Learning Models

3.3.1 Supervised Models

3.3.1.1 Linear Regression

Linear regression (LR) is one of the most popular and vastly used machine learning algorithms. As its name would suggest, LR solves regression problems which means that it predicts values within a continuous and numerical range.

$$Y = f(x) \tag{3.16}$$

Equation 3.16 shows the relationship between the independent input variable X and the dependent output variable Y. This results in a linear relationship between the input data and the output. LR aims to find a linear mapping function based on the observed input data in order to calculate the output variable Y. A Linear Regression model can be represented by equation 3.17.

$$Y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n, \tag{3.17}$$

where Y is the predicted value or the output, θ_0 is the bias therm, $\theta_1, \theta_2, ..., \theta_n$ are the model weights and $x_1, x_2, ..., x_n$ are the feature values. The LR model aims to obtain the set of weight $\theta_1, \theta_2, ..., \theta_n$, which forms a

line that best fits the data. This line is referred to as the *regression line* and is obtained by minimizing the mean of squares of the vertical deviation between each datapoint and the fitted line, otherwise known as *mean least squares* (Seber and Lee 2012). As an example, consider the simple linear regression $Y = \theta_0 + \theta_1 x_1$. Let the initial weights be $\theta_0 = 1.5$ and $\theta_1 = 0$, which forms the green line in Figure 3.9. In this particular example, the loss function is the *mean square error*, which is calculated in Equation 3.18.

$$MSE = \frac{1}{n}\sum_{i=0}^{n}(y_i - \overline{y_i})^2, \tag{3.18}$$

where n is the number of datapoints, y is the correct target values and $\overline{y}$ is the predicted target values. Inserting the simple regression function Y results in Equation 3.19.

$$MSE = \frac{1}{n}\sum_{i=0}^{n}(y_i - (\theta_0 + \theta_1 x_1))^2 \tag{3.19}$$

The MSE of the initial green line is 4.75, and by visual inspection of Figure 3.9, it is clear that this line is not the best fit of the data points. The LR algorithm attempts to find a better fit by using *gradient* methods which reduce the loss function (Ruder 2016). One gradient method is the *gradient descent* which minimizes the MSE by finding the partial derivative of the loss function concerning the model weights. Equation 3.20 finds the partial derivative with respect to the bias θ_0.

$$D_{\theta_0} = \frac{1}{n}\sum_{i=0}^{n} 2(y_i - (\theta_0 + \theta_1 x_1)) \tag{3.20}$$

Equation 3.21 finds the partial derivative with respect to the model weight θ_1.

$$D_{\theta_1} = \frac{1}{n}\sum_{i=0}^{n} 2(y_i - (\theta_0 + \theta_1 x_1))(-x_1) \tag{3.21}$$

The weight θ_1 and bias θ_0 are then updated by using Equation 3.22 and 3.23.

$$\theta_0 = \theta_0 - MSE \times D_{\theta_0} \tag{3.22}$$

$$\theta_1 = \theta_1 - MSE \times D_{\theta_1} \tag{3.23}$$

The algorithm repeats this process until the MSE is minimized. The orange line in Figure 3.9 is the final regression line which best fits the data.

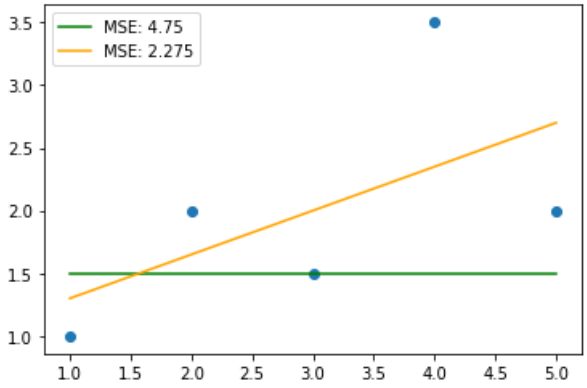

Figure 3.9: Illustrative example of a simple linear regression. The green line represents the initial regression line, whereas the yellow line represents the optimal regression line.

3.3.1.2 K-Nearest Neighbours

K-nearest Neighbours (KNN) is an algorithm used for both classification and regression problems and is also widely used for pattern recognition and data mining. When an unknown data point occurs, the algorithm aims to classify which class it belongs to based on its k nearest neighbours. An essential part of the algorithm is the distance measure. Such a measure is used to determine which of the k data points are closest to the data point being classified. The Euclidean distance in Equation 3.24, the

Manhattan distance in Equation 3.25 and the Minkowski distance given in Equation 3.26 are the most commonly used distance metrics.

$$\text{Euclidean Distance}: \sqrt{\sum_{i=1}^{k}(x_i - y_i)^2} \tag{3.24}$$

$$\text{Manhattan Distance}: \sum_{i=1}^{k} \mid x_i - y_i \mid \tag{3.25}$$

$$\text{Minkowski Distance}: (\sum_{i=1}^{k}(\mid x_i - y_i \mid)^q)^{\frac{1}{q}} \tag{3.26}$$

Figure 3.10 shows a categorical variable plotted, and the goal of this example is to classify a new data record as blue, red or green. Given a new data point x, the k most similar records from the training set are located. The similarity between the data records is measured according to different distance measures, and in the example seen in Figure 3.11, the Euclidean Distance is used. The unclassified data point x will be assigned to a group based on the observation of which group the majority of its k-nearest neighbours belongs to. The most important parameter to tune is k, the number of neighbours used to make the classification decision. Figure 3.11 shows the clusters with the k parameter tuned to 1, 10 and 50 respectively. As mentioned, the KNN-algorithm can also be used for regression tasks by returning the average value of all of the k-nearest neighbours of the data sample x.

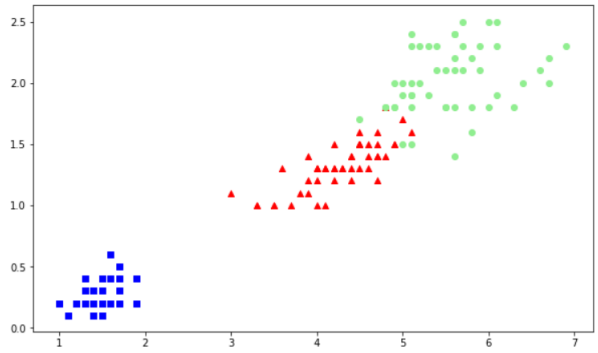

Figure 3.10: Result of assigning datapoints to three different classes using k-nearest neighbours.

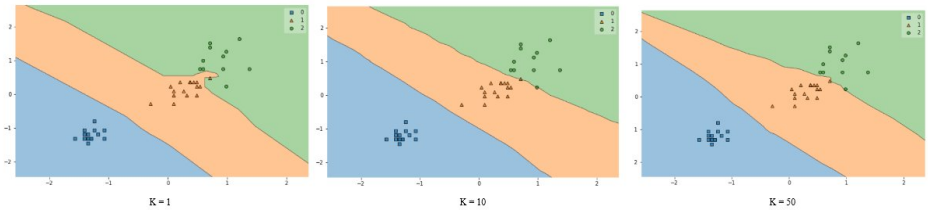

Figure 3.11: Illustration of the different decision boundaries for k = 1, 10 and 50 for the KNN algorithm

3.3.1.3 Support Vector Machines

A support Vector Machine (SVM) is a versatile ML algorithm, used for both linear and non-linear classification, regression, and outlier detection. The basic idea is to find a hyperplane in a n dimensional-space that best separates the classified instances. Shown in Figure 3.12 is the simplest version of such an algorithm. As shown, the dataset is linearly separable, and the decision boundary is created by separating the dataset with the largest possible margin, called *large margin classification* (Géron 2019). Only the data instances lying on the margin separating the classes are stored, known as the *Support Vectors*, marked in red in Figure 3.12 and 3.13. These margins are later used to classify a new datapoint as ei-

ther *green* or *blue*. *Large Margin Classification* is only applicable if the dataset is linearly separable, and it is also susceptible to outliers, as seen in Figure 3.13. The width of the decision boundary is significantly decreased due to one single outlier, as seen in Figure 3.13.

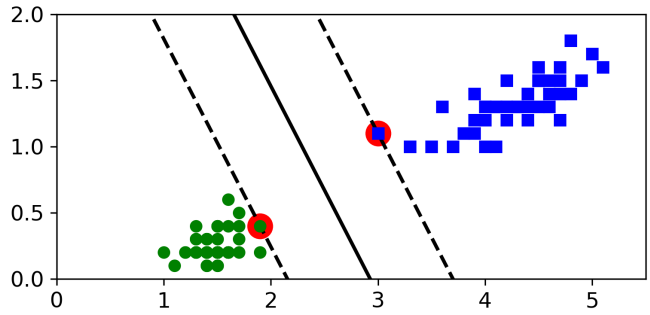

Figure 3.12: Illustrative example of SVM's decision margin for linearly separable classes

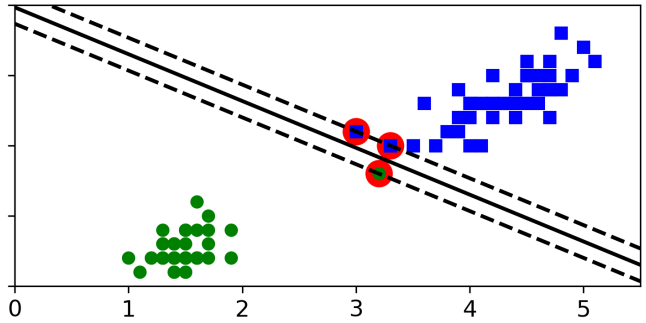

Figure 3.13: Illustrative example of SVM's sensitivity to outliers

One approach to handle nonlinear datasets is the same as the logic used in Section 3.1.2.2.2, by projecting the data to a higher-dimensional space, which can result in a linearly separable dataset. As seen in Figure 3.14, the data points in one dimension are not separable, but after being projected to a two-dimensional space, the categorical groups are perfectly separated by a line.

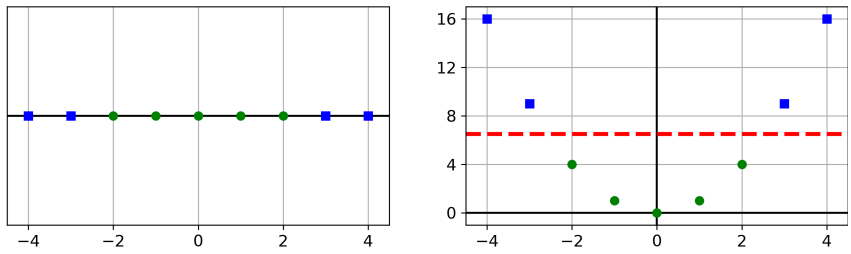

Figure 3.14: Illustration of a dataset linearly separable after scaling to a higher dimension.

3.3.2 Artificial Neural Networks

An artificial neural network is a computational model inspired by the brain's architecture in order to build an intelligent machine. ANNs are the very core of *deep learning* which is defined as large or complex neural networks. Such networks are sturdy, versatile and scalable and are used to tackle large and highly complex ML problems. A neural network consists of three different layers, the input, output and the hidden layers. The input and output layer is nothing more than what the word indicates, the input and the output of the computational model. The interesting aspect of a neural network is what happens between these two layers, in the hidden layer, often referred to as the black box layer. The number of hidden layers depends on the structure and complexity of the task to solve, and this also counts for the number of nodes, referred to as *neurons*.

3.3.2.1 Perceptron

The perceptron has one of the simplest architectures, and is based on an artificial neuron called a *linear threshold unit* (LTU). The input and output of a LTU are numbers and each input is associated with a *weight*. Figure 3.15 shows a simple example of a LTU, where $X = x_1, x_2, x_3, ..., x_n$ are the input variables, $W = w_1, w_2, w_3, ..., w_n$ are the corresponding weights and y is the calculated output.

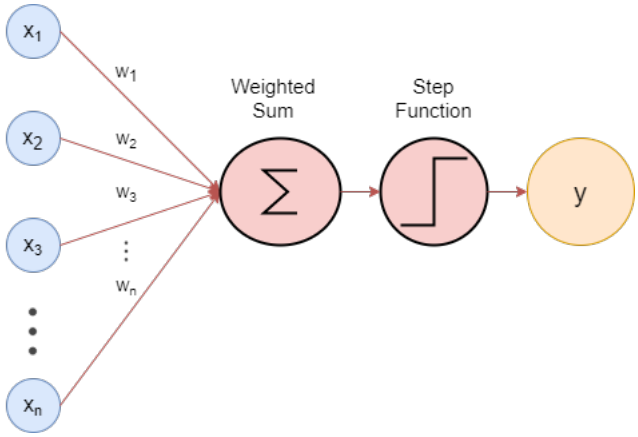

Figure 3.15: Illustration of a linear threshold unit with n inputs.

The LTU computes a weighted sum of its inputs as shown in Equation 3.27, and then applies a *step function* to that sum and outputs the results as shown in Equation 3.28.

$$z = w_1 x_1 + w_2 x_2 + w_3 x_3 + ... + w_n x_n = w^T \cdot x \tag{3.27}$$

$$y_w(x) = step(z) = step(w^T \cdot x) \tag{3.28}$$

Common step functions are:

- Heaviside step function
- Signum function

A simple perceptron architecture consists of multiple LTUs. In other words, the hidden layer consists of multiple neurons that are all connected to each input neuron.

3.3.2.2 Multi-Layer Perceptron

A Multi-Layer Perceptron (MLP) is composed of one *input layer*, one or more layers of perceptrons, called *hidden layers* and one finale layer called the *output layer*. When an artificial neural network has two or more hidden layers, it is called a *deep neural network* (DNN). Figure 3.16 displays a multi-layer perceptron with two hidden layers. Both the input layer and the hidden layers consist of n neurons, and each neuron is connected to every other neuron in the subsequent layers. The input neurons are now referred to as the set $a_1^0, a_2^0, a_3^0, ..., a_n^0$.

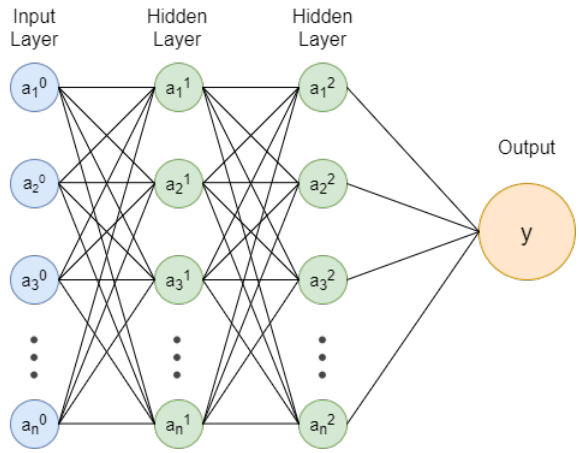

Figure 3.16: Illustration of a multilayer perceptron with one input layer, two hidden layers and one output layer.

In a forward pass, each value of a subsequent neuron is dependent on all preceding neurons with their corresponding weights as shown is Equation 3.29

$$a_n^k = \sigma(w_1^{k-1}a_1^{k-1} + w_2^{k-1}a_2^{k-1} + w_3^{k-1}a_3^{k-1} + ... + w_n^{k-1}a_n^{k-1}) = \sigma((W^{k-1})^T a^{k-1}), \tag{3.29}$$

Where k represents the hidden and input layers. The step function z is replaced with the *sigmoid* activation function, σ, such that the network

obtains properties which makes it able to perform *back-propagation* as explained later in Section 3.3.2.3. The *sigmoid* function takes an input, in this case, the weighted sum of all the preceding neurons and maps it to a value between 0 and 1 (Naduvil-Vadukootu, Angryk, and Riley 2017). The value a_n^k for each neuron, including the output neuron is calculated using equation 3.29. When the predicted output value is obtained, it can be measured against the correct target value, and the network can begin its *backward pass*.

3.3.2.3 Back-Propagation

Back-propagation is the heart of a neural network and is used to identify which adjustments to the network's weights that cause the most significant decrease to the cost function (Hecht-Nielsen 1992). It is essentially how the network learns. To illustrate this technique, the above network has been simplified into the network depicted in Figure 3.17. This network consists of 4 neurons, where a_1^0 is the input neuron, a_1^1 and a_1^2 represent the neuron in each respective hidden layer and a_1^3 is the output neuron

Figure 3.17: Illustration of a simplified multilayer perceptron with one neuron in each layer.

Given the correct output value of one training example y_1, the corresponding cost is calculated by the square error as shown in Equation 3.30.

$$C_0 = (a_1^3 - y_1)^2 \tag{3.30}$$

As a remainder, the a_1^3 is dependent on the the previous neuron a_1^2 and the corresponding weight w^2 as shown in equation 3.29. To further

simplify, consider only the two last nodes a_1^3 and a_1^2, and $a_1^3 = \sigma(z^{(2)})$, where $z^{(2)} = w^2 a_1^2$. The cost function is now rewritten in Equation 3.31.

$$C_0 = (\sigma(z^{(2)}) - y_1)^2 \tag{3.31}$$

The goal is to reduce the cost C_0 by adjusting the value of weight w^2. This is achieved by using the *chain rule* to find the partial derivative of the cost with the respect to the weight as shown in Equation 3.32.

$$\frac{\partial C_0}{\partial w^2} = \frac{\partial z^{(2)}}{\partial w^2} \frac{\partial a_1^2}{\partial z^{(2)}} \frac{\partial C_0}{\partial a_1^2}, \tag{3.32}$$

where each of the partial derivatives are shown in Equations 3.33, 3.34 and 3.35.

$$\frac{\partial z^{(2)}}{\partial w^2} = a_1^1 \tag{3.33}$$

$$\frac{\partial a_1^2}{\partial z^{(2)}} = \sigma'(z^{(2)}) \tag{3.34}$$

$$\frac{\partial C_0}{\partial a_1^2} = 2(a_1^2 - y_0) \tag{3.35}$$

Equation 3.36 shows the final cost function.

$$\frac{\partial C_0}{\partial w^2} = a_1^1 \sigma'(z^{(2)}) 2(a_1^2 - y_0) \tag{3.36}$$

Equation 3.37 shows the above example extended such that it regards not only 1 training sample, but n training samples.

$$\frac{\partial C}{\partial w^2} = \frac{1}{n} \sum_{k=0}^{n-1} \frac{\partial C_k}{\partial w^2} \tag{3.37}$$

The contribution from each neuron to the network cost is calculated until the algorithm reaches the input layer. This backward pass efficiently measures the cost gradient across all the connection weights in the net-

work by propagating the cost gradient backwards in the network, hence the name of the algorithm. In addition, for more extensive networks, like the multi-layer perceptron depicted in Figure 3.16, each component of the gradient vector need to be calculated. For each training instance, the backpropagation algorithm makes a prediction, the forward pass, measures the cost, then goes through each layer in reverse to measure the cost contribution from each connection, the backward pass, and finally, the weights are tweaked to reduce the cost.

3.3.2.4 Activation Functions

Each neuron in the neural network is assigned an activation function, which acts as a mathematical gate between the input fed into the neuron and the output transmitted to the next layer of neurons (Elsworth and Güttel 2020). Choosing a suitable activation function is critical in terms of model accuracy and computational efficiency. In addition, depending on the nature of the data, some activation functions result in a faster model convergence, while other activation functions might prevent the same model from converging at all.

3.3.2.4.1 Binary Step Function

One simple activation function is the binary step function, which is a threshold-based activation function (Elsworth and Güttel 2020). It takes the input or the weighted average from the previous layer, and if this signal is above a determined threshold, then the same signal is forwarded to the next layer of neurons. The main downside with such an activation function is that it does not allow for multi-value outputs, it just provides the binary signal "off/on" or "true/false".

3.3.2.4.2 Linear Activation Function

The linear activation function takes the input signal x and transform it to an output signal y by using the linear function $y = bx + c$ (Elsworth and

Güttel 2020). Contrary to the binary step function, the linear activation function is able to provide multi-valued outputs. The main downside of this activation function is that it prevents the use of *back-propagation*. Recall from Section 3.3.2.3, that the back-propagation step required the gradient of the cost function in order to identify the impact of adjusted weights. The gradient of the linear function $y = bx + c$ is constant, meaning that it has no relation with the input signal x. In other words, it is impossible to perform back-propagation. A neural network with the applied linear activation function is, therefore, just a simple regression model.

3.3.2.4.3 Non-linear Activation Functions

Non-linear activation functions are essential when it comes to learning complex data like audio, images and non-linear datasets with high dimensionality. The activation function presented below are non-linear and used in modern neural networks. They all address the main downside with a linear activation function, namely the ability to perform back-propagation.

Sigmoid Activation Function

The sigmoid activation function takes the input signal x and maps it to an output signal between 0 and 1, by using the formula in Equation 3.38.

$$\phi(x) = \frac{1}{1 + \exp^{-x}} \tag{3.38}$$

Figure 3.18 shows the plotted curve of the sigmoid function. This function is differentiable, which means that the slope of the curve can be obtained and it enables the use of back-propagation. The main downsides are that it is computationally expensive as well as it is suffering from the vanishing gradient problem elaborated in Section 3.3.2.5.

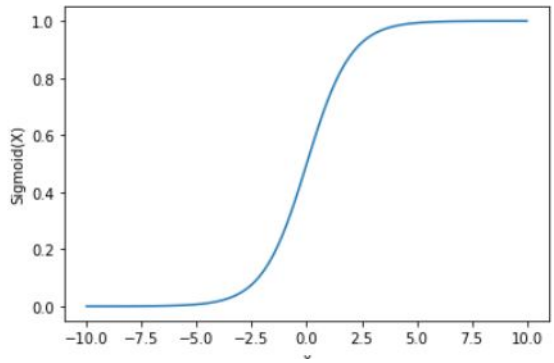

Figure 3.18: Illustrative curve of the sigmoid activation function.

Hyperbolic Tangent Activation Function

The hyperbolic tangent activation function is in many ways similar to the sigmoid function. It is differentiable, but instead of mapping the values between 0 and 1, it maps the input signal between −1 and 1. The output signal is obtained by feeding the input signal into the Equation 3.39. The curve of the activation function is depicted in Figure 3.19.

$$tanh(x) = \frac{2}{1 + exp^{-2x}} - 1 \tag{3.39}$$

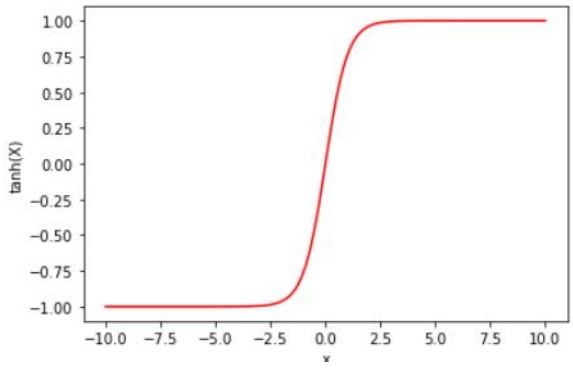

Figure 3.19: Illustrative curve of the tangent hyperbolic activation function.

The hyperbolic tangent function suffers from the same vanishing gra-

dient problem as the sigmoid function. The main advantage of the hyperbolic tangent function compared to the sigmoid function is that negative input signals are mapped more strongly negative. Also, it is zero-centred meaning that input signals close to 0 are also mapped near 0.

Rectified Linear Unit Activation Function

The rectified linear unit (ReLU) activation function is the most popular activation function due to its computational efficiency. As depicted in Figure 3.20, the ReLU is half-rectified, which means that when the input signal is below or equal to 0 then the output signal is mapped to 0. Although it looks like a linear function, the ReLU is differentiable and enables back-propagation.

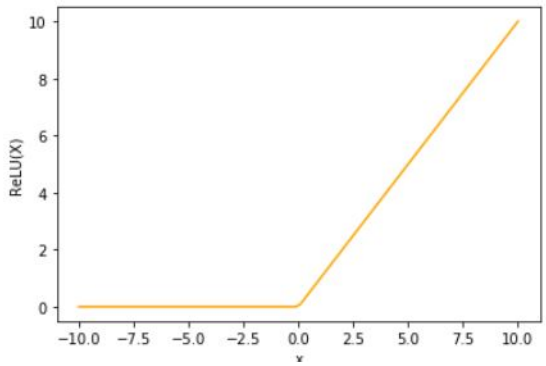

Figure 3.20: Illustrative curve of the rectified linear unit activation function.

3.3.2.5 Problem of Vanishing Gradient

Vanishing gradient refers to the problem where the gradient of the cost function of a neural network approaches 0 due to the use of some activation function in many layers of the network (Hochreiter 1998). For instance, the sigmoid function maps the input signals within the range $[0,1]$, which means that a significant change in the input signal yields a

small change in the output signal. In order words, the derivative of the activation function becomes small, as depicted in Figure 3.21.

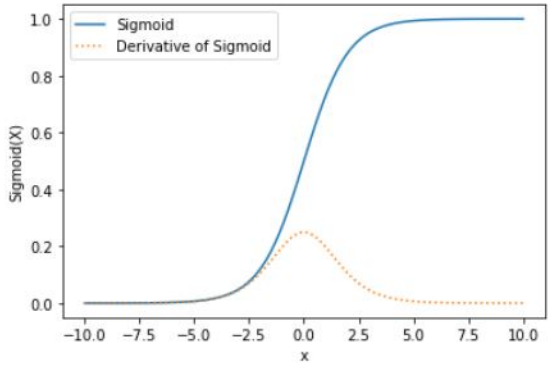

Figure 3.21: The Sigmoid activation function plotted against the derivative of the Sigmoid function.

This problem arises for more extensive neural networks with multiple hidden layers that use activation function like the sigmoid or the hyperbolic tangent. During back-propagation, the chain rule is used to obtain the network gradient by multiplying the gradient of each layer from the final layer down to the initial layer. As mentioned above, this gradient becomes small if the sigmoid activation function is applied and if the network also consists of many layers, then by the chain rule, the network gradient becomes a product of many small gradients. The network gradient decreases exponentially when propagating down to the initial layer (Hochreiter 1998). A small network gradient results in the network's weights not being correctly updated and the network would fail to train.

3.3.2.6 Stochastic Gradient Descent

Computing a back-propagation step on the entire dataset is computationally heavy if the dataset is quite large. Stochastic gradient descent increases the computational efficiency of back-propagation by randomly shuffling the data and dividing it into multiple subsets or *batches*. Then

it performs one forward pass and one back-propagation step on each batch. Put differently; the first batch is fed through the network, and predictions are made. The cost function is obtained, and the network's weights are adjusted by using back-propagation. This process is repeated until all batches have been processed.

3.3.2.7 Error Metrics

Error metrics are used to evaluate model performance during and after training. While training, the error metric defines the cost function, which is used during back-propagation. Different error metrics inherent unique properties which directly impact how the network learns. Elaborated in the following sections are the most used error metrics for regression problems.

3.3.2.7.1 Mean Square Error

As shown in Equation 3.40, the mean squared error (MSE) is the average of the squared difference between the predicted target values, $\overline{y}$ and the actual target values, y. Due to the squaring of the difference, this metric penalizes small errors.

$$MSE = \frac{1}{n}\sum(y-\overline{y})^2 \tag{3.40}$$

3.3.2.7.2 Root Mean Square Error

The root mean squared error (RMSE) is the most used error metric in regression problems. As shown in Equation 3.41, it takes the square root of the MSE, and the squaring of the difference before taking the average results in a high penalty on large errors.

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{n}\sum(y-\overline{y})^2} \tag{3.41}$$

3.3.2.7.3 Mean Absolute Error

Mean absolute error (MAE) is the average of the absolute difference between the predicted target value, $\overline{y}$ and the correct target value y, as shown in Equation 3.42. Compared to MSE, this error metric does not square the difference which means that it does not penalize errors as strong as the MSE and are more robust to outliers.

$$MAE = \frac{1}{n}\sum |y - \overline{y}| \tag{3.42}$$

3.3.2.7.4 R^2 or Coefficient of Determination

The coefficient of determination is a regression error metric that compares the model with a baseline. The baseline is obtained by taking the mean of the data, and the R^2 is then calculated as shown in Equation 3.43.

$$R^2 = 1 - \frac{MSE(model)}{MSE(baseline)} \tag{3.43}$$

3.3.2.7.5 Adjusted R^2

The adjusted R^2 is an improvement of the original R^2, which takes into account the number of encountered observations. By doing so, it provides a more realistic error metric which is always lower than the original R^2. The formula for the adjusted R^2 is found in Equation 3.44, where n is the number of observations and k is the number of independent variables.

$$AdjustedR^2 = 1 - \left[\left(\frac{n-1}{n-k-1}\right)\cdot\left(1-R^2\right)\right] \tag{3.44}$$

3.3.2.8 Overfitting

As explained in Section 3.2.2 neural networks are trained by dividing the dataset into a training set and a testing set. The model is trained using the

training data, and later the performance of the trained model is evaluated. For any machine learning algorithm, the "bias-variance dilemma" is highly relevant and involves finding a balance between the accuracy and the generalization of a model. An example of such a trade-off is that a model can perform exceptionally well on training data while showing poor accuracy when applied to a test set. This concept is called overfitting and happens when a model too well represents the training data, hence it will fail to generalize well on data outside the training set, characterized by a large variance. Hence overfitting the model to the training set will result in a training error converging to zero, while the test error will increase drastically. Such an error indicates that the model is not capable of making accurate predictions on observations not previously seen in training. The bias represents the inability of the physical model to accurately approximate the function, known as the approximation error. The variance represents the inadequacy of the empirical knowledge in the training sample, known as estimation error. There is a trade-off between these two, typically known as the loss function, shown in Equation 3.45.

$$\text{Loss function} = (Bias)^2 + Variance \tag{3.45}$$

The model with the optimal predictive capability is the one that leads to the best balance between bias and variance. Both can only be eliminated in the case when the number of training samples becomes infinitely large.

3.3.3 Recurrent Neural Networks

Recurrent neural networks (RNN) is a type of neural network that makes use of the sequential information that the input may have (Liang and Bose 1996). This is in contrast to the traditional feed-forward neural network, which assumes that all inputs are independent. Such an as-

sumption holds for many tasks, but for problems like sentence sentiment analysis or time-series predictions, there is a lot of information in the sequence of the input.

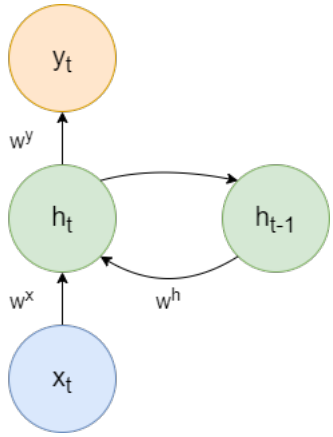

Figure 3.22: Simple illustration of a recurrent neural network that utilizes information from preivous timesteps.

Similar to feed-forward neural networks, the recurrent neural networks consist of an input layer, an output layer and hidden layers. The difference is that the hidden layers in a recurrent neural network have connections to back to themselves. As Figure 3.22 shows the hidden layer h_t has a connection to the hidden layer h_{t-1} that occurred at the previous time step t. In this case, the number of observations, previously noted as n, is replaced with t, which for easier explanation refers to the time when observation t occurred. With the architecture in Figure 3.22, the neural network is now able to use the information from previous time steps. W^x, W^h, W^y represents the weight vectors of the input layer, previous hidden layer and output layer respectively. The way the recurrent neural network evolves is shown in Equations 3.46 and 3.47.

$$h_t = f(w^x x_t + w^h h_{t-1}) \tag{3.46}$$

$$y_t = g(w^y h_t) \tag{3.47}$$

f and g are some suitable activation functions. Equation 3.46 shows that the state of the hidden layer at time t is dependent of the state of the hidden layer at previous timestep $t-1$ and the input signal at time t. Equation 3.47 states the the output signal at time t is only dependent of the state of the hidden layer at time t. Figure 3.23 shows the recurrent neural network when unfolded. The length of the unfolded recurrent network is dependent on the length of the input sequence and the output sequence. The length of the input sequence defines how much previous information the networks take into account when performing predictions. The output sequence defines how many time steps into the future, k, the model tries to predict, which is the hidden state at time $t+k$.

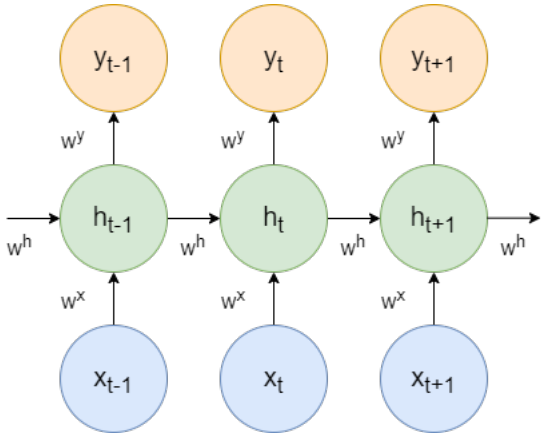

Figure 3.23: Illustration of an unfolded recurrent neural network.

3.3.3.1 Gated Recurrent Unit

Traditional recurrent neural networks suffer from the vanishing gradient problem explained in Section 3.3.2.5. Gated Recurrent Unit (GRU) is a type of recurrent neural network that solves this problem by introducing

a *reset gate* and a *forget gate* in each cell structure of the traditional recurrent neural network (Ravanelli et al. 2018). Figure 3.22 illustrated a RNN with only one cell per hidden layer. The number of cells in each hidden layer is equal to the number of previous time steps that the network remembers when performing predictions, in other words, the length of the input sequence. Figure 3.24 shows a RNN that remembers three previous states.

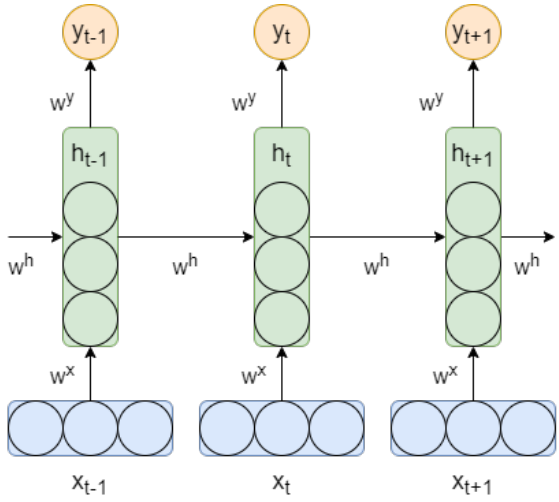

Figure 3.24: Illustration of an unfolded recurrent neural network with three cells.

The structure in each cell is depicted in Figure 3.25. Each cell contains an *update gate* and an forget gate, which are two vectors that decide what information to pass on. During the training of a GRU, these vectors are configured only to remember the relevant information of the past. Both the update gate, z_t and the forget gate r_t is defined in Equation 3.48. The update gate decides how much past information the network should remember, while the reset gate determines how much past information the network should forget.

$$z_t = r_t = \sigma(w^x x_t + w^h h_{t-1}) \tag{3.48}$$

The current memory content, h'_t, applies the reset gate to determine the relevant past information, as shown in Equation 3.49.

$$h'_t = \tanh(w^x x_t + r_t \circ w^h h_{t-1}), \tag{3.49}$$

where $\circ$ is the Hadamard product (Horn 1990) and tanh is the non-linear activation function. The final memory content, h_t contains the information that is passed to the next time step $t+1$ and is used to predict the output y_t. The update gate is then applied, as shown in Equation 3.50.

$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ h'_t \tag{3.50}$$

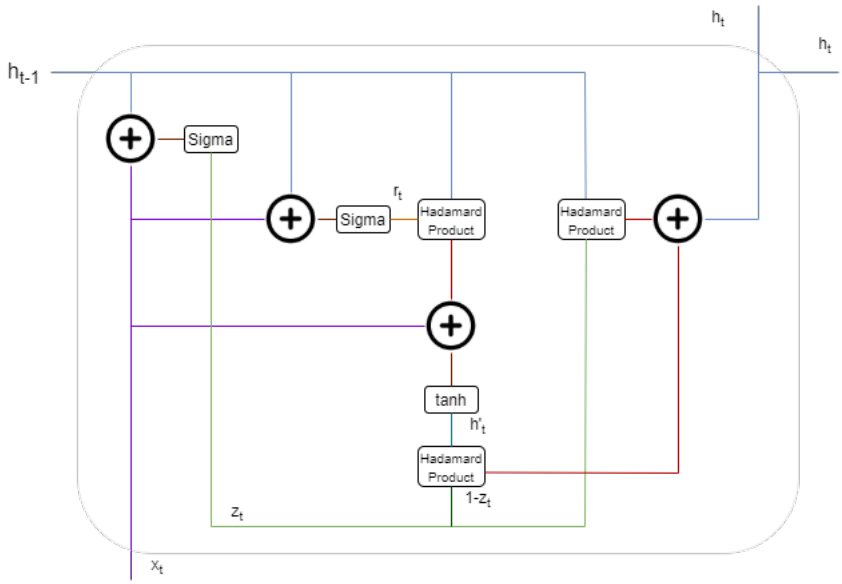

Figure 3.25: Internal architecture of one cell in a Gated Recurrent Unit network.

Figure 3.25 and the corresponding equations (3.48, 3.49 and 3.50) illustrate how the GRU network control the flow of information by using the update and reset gates. The problem of vanishing gradient is solved,

which makes the network better at using past information to predict the future.

3.3.3.2 Long-Short Term Memory

Long-Short Term Memory (LSTM) is another type of RNN, which similarly to the GRU controls the flow of information in order to solve the problem of vanishing gradient (Ravanelli et al. 2018). The GRU had two gates, the *update* and *forget* gates, whereas the LSTM implements three gates, namely the *input*, *output* and *forget* gates. Hence, the cell structure of the LSTM is a bit different from the GRU. The main difference is that the LSTM is more suitable when required to remember longer sequences of data, where the GRU is less complex and trains faster (Ravanelli et al. 2018).

3.4 Statistical Forecasting Approaches

3.4.1 Classical Time Series Models

3.4.1.1 ARIMA

The ARIMA model is a classical statistical tool for analyzing and forecasting for time-series data. Such models have the capabilities to represent stationary and non-stationary time-series to produce a forecast, based on the historical data of a single variable (Kumar and Anand 2014). *ARIMA* is an acronym for Auto Regressive Integrated Moving Average. The *AR* in the model stands for *Auto Regressive*, and is a linear regression model. The variable of interest is forecasted using a linear combination of the past values of the variable (Hyndman and Athanasopoulos 2018). Seen in Equation 3.51 is an auto regressive model of order p. The value of $Y(t)$ depends only on its own lags.

$$Y_t = c + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + \beta_p Y_{t-p} + \epsilon_t \tag{3.51}$$

MA is acronym for *Moving Average* and instead of using past values, a moving average model uses past forecast error in its model. Seen in Equation 3.52 is the model for moving average of order q, where ϵ_t is white noise.

$$Y_t = \alpha + \epsilon_t + \phi_1 \epsilon_{t-1} + \phi_2 \epsilon_{t-2} ... + \phi_q \epsilon_{q-1} \tag{3.52}$$

Chapter 4

Literature Review

In the project thesis, an extensive literature review of the use of machine learning for predictive maintenance and fault detection was conducted. Summarized in the following sections are findings and discoveries relevant for this master thesis, as well supplementary findings in the literature related to time-series, forecasting, and the new obstacles that arise when using real-life data compared to simulated data.

4.1 Time-Series and Forecasting

Being able to predict the future is a significant competitive advantage for any industry and business. Time series analysis and forecasting have been an active research area over the last decades, and there is still being put much effort into developing methods that can contribute to decision-making by performing time series analysis and forecasting. The accuracy of these methods is crucial, and the research of improving the precision and effectiveness of forecasting models has never stopped (G. P. Zhang 2003).

4.1.1 Time Series Models and Applications

In 1970, George Box and Gwilym Jenkins published their book *Time series analysis, forecasting and control*, where they proposed the Box-Jenkings method. For nonstationary time series, the approach starts with the assumption that the process that generated the time series can be approximated by the classical model Autoregressive Integrated Moving Average (ARIMA) (Box et al. 2015). The forecasting domain has been influenced for a long time by linear statistical methods such as the ARIMA model. Classical models have the advantage of being easily accessible, and during the past four decades, ARIMA has been the most popular method in time series forecasting. The model describes how the dynamics of the time series behave over time and continues the model dynamics into the future. However, in 1970 and 1980, it became increasingly clear that linear models are not adaptable to most real-world applications. Machine learning models became a severe contestant to classical statistical models by creating non-parametric nonlinear models using only historical data to learn about the stochastic dependency between the past and the future. Paul John Werbos, known as the pioneer of recurrent neural networks, found that classical artificial neural networks outperformed classical statistical methods such as linear regression (Bontempi, Taieb, and Le Borgne 2012).

Since the ARIMA model has been widely used in forecasting research and practice, comparing ARIMA to different deep learning models has a broad area of application. Particularly many studies have been applied to financial time series forecasting, and the results show that ANN models outperform the ARIMA models. This is also supported by a case study on Iranian poultry retail price done by Karbasi, Laskukalayeh, and Seiad Mohammad Fahimifard (2009) and a study on stock data obtained from the New York Stock Exchange, where the findings support this claim (Adebiyi, Adewumi, and Ayo 2014). ARIMA and ANN are often compared with mixed conclusions depending on the superiority of prediction and

forecasting performance.

Even though neural network often outperforms classical models, they are not always superior. S. Fahimifard et al. (2009) reviews the need for exchange rate forecasting in order to prevent its disruptive movements by comparing several models (GARCH, ARIMA, ANN, ANFIS). Briefly, the conclusion was that the ANFIS model had an effective result in forecasting the accuracy of the exchange rate. In financial time series, many studies have been using ANNs (G. Zhang, Patuwo, and Hu 1998). Nevertheless, K.-j. Kim (2003), F. E. Tay and L. Cao (2001) and L.-J. Cao and F. E. H. Tay (2003) highlights that SVMs also provides a promising alternative for financial time-series forecasting. Voyant et al. (2017) tries in his research to forecast the amount of solar radiation by exploring different machine learning models. By trying to predict the amount of solar radiation in different cities, multi-layer perceptron (MLP) in most cases outperforms ARIMA, but occasions of the opposite are also presented. Also, less typical predictors are created as a part of the research, such as SVM, SVR and k-means clustering. Voyant concludes that ARIMA and ANN as methods are almost equivalent in terms of quality of prediction, but the flexibility of ANN as universal nonlinear approximation makes it more preferable than the classical ARIMA model. However, the authors argue that the quality of the prediction depends highly on the quality of the data.

4.1.2 Time-Series Prediction From A Real-Life Perspective

Several authors argue in their research papers, discussing different predictors, that the quality of the predictor is equivalent to the quality of the data. Data preprocessing and reduction is essential when working with real-world datasets, which tend to be unstructured, inconsistent and noisy. The method of time series forecasting relies on the idea that historical data includes inherent patterns that convey useful information about future descriptions (Parmezan, Souza, and Batista 2019). These patterns are usually non-trivial to identify, and their discovery is one of

the primary goals of the time series processing. As mentioned, deep neural networks are known to be the state-of-the-art approach for time-series prediction, but from a real-life perspective, the preprocessing phase has a significant impact on the accuracy of the model and predictions.

In research on the importance of preprocessing Naduvil-Vadukootu, Angryk, and Riley (2017) creates a state-of-the-art ANN model, using data preprocessed with time-series specific techniques. Their research demonstrates how preprocessing leads to simpler architectures and more accurate predictions. The inherent complexity of time-series data stems from high dimensionality. Naudvil highlights the importance of exploratory data analysis, in which a deeper understanding of the data is achieved through visualization as histogram, box-plots and scatter plots, and by the use of correlation analysis which brings out the relationships and other patterns between the dependent and independent variables. Only by the use of simple preprocessing techniques such as scaling and interpolation, the model gets a much higher accuracy, than on a model processing the raw dataset. Elsworth and Güttel (2020) also highlights the importance of preprocessing in his research, as feeding unprocessed numerical data into a LSTM network is unlikely to forecast values outside the numerical range of the training set (Elsworth and Güttel 2020). Therefore it is essential to remove linear time-series trends, as the forecasting will be poor otherwise. By using an interval of length $N = 2000$, with values in the interval $[0, 0.5]$, the LSTM model trained on raw data fails to forecast values greater than 0.5, and the author suggests the solutions of differencing the data before feeding it into the LSTM model, thereby removing the linear trend. Machine learning methods trained on raw numerical time series data show fundamental limitations, such as high sensitivity to hyper-parameters and even initialization of random weights (Elsworth and Güttel 2020).

When working with raw data, the need for preprocessing is decisive. Nevertheless, since the dataset varies for each problem and domain, there

is no recipe for success to create a dataset that is free of noise, inconsistencies, outliers, and missing values. Amongst many other researchers, Ramırez-Gallego et al. (2017) highlight that data scientists spend most of their time preprocessing data, rather than mining or modelling. Additionally, N. Zhang and Lu (2007) state in their publication that it is a well-known fact that data scientists spend 80% of their time preprocessing, and 20 % modelling. Further, both authors claim that in the context of data preprocessing there is a long road ahead, in terms of reduction rates, computational time and memory usage in order to prepare a real-life dataset for mining or modelling.

Chapter 5

System Description

Teekay states in their maintenance manual that systems including compressor, turbines and diesel engines require a comprehensive data collection and calculation to produce an adequate state evaluation. Problems involving such components might be too intricate to be solved with the laws of physics and classical statistical methods, and state-of-the-art machine learning models might be a better solution to such problems. This chapter outlines the problem of research by giving a brief explanation of each component involved.

5.1 Problem of Interest

At the Piranema Vessel, Teekay is currently experiencing a problem with the gas compression system. It has been an ongoing issue since the start of production of the Piranema oil field in 2007. As seen in Figure 5.1, the position of the OPRA turbine exhaust outlet is somewhat close to the air inlet of the SOLAR turbine. The SOLAR turbine operates the three gas compressor trains, both seen in Figure 5.2. The exhaust from the OPRA turbines has been a lingering problem since the operation start, causing two adverse effects.

First and foremost, it creates a high temperature near the 2nd stage

separator area, as this area is located right above the OPRA Turbines outlet. Increased heat and polluted air affect the working environment for personnel performing regular maintenance operations. Secondly, on days with little wind, the exhaust from the OPRA Turbines goes into the SOLAR turbine air inlets. Polluted air into the compression system can lead to a significant loss in compressor efficiency, with the cascade effect of occasional process shut down (PSD) trips, with high cost due to penalisation. To be able to say something about the future performance of the compressors would be a great benefit for Teekay Offshore Production. With the use of such an indicator, a PSD-trip can potentially be avoided, and thereby increase the safety onboard the process plant and reduce costs.

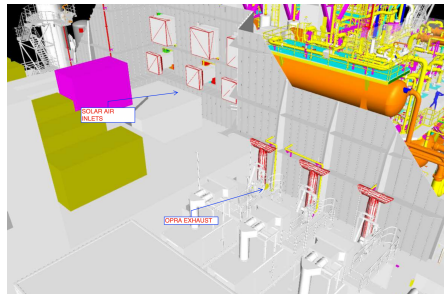

Figure 5.1: Illustrative overview of the position of the OPRA exhaust outlets and the Solar air inlets.

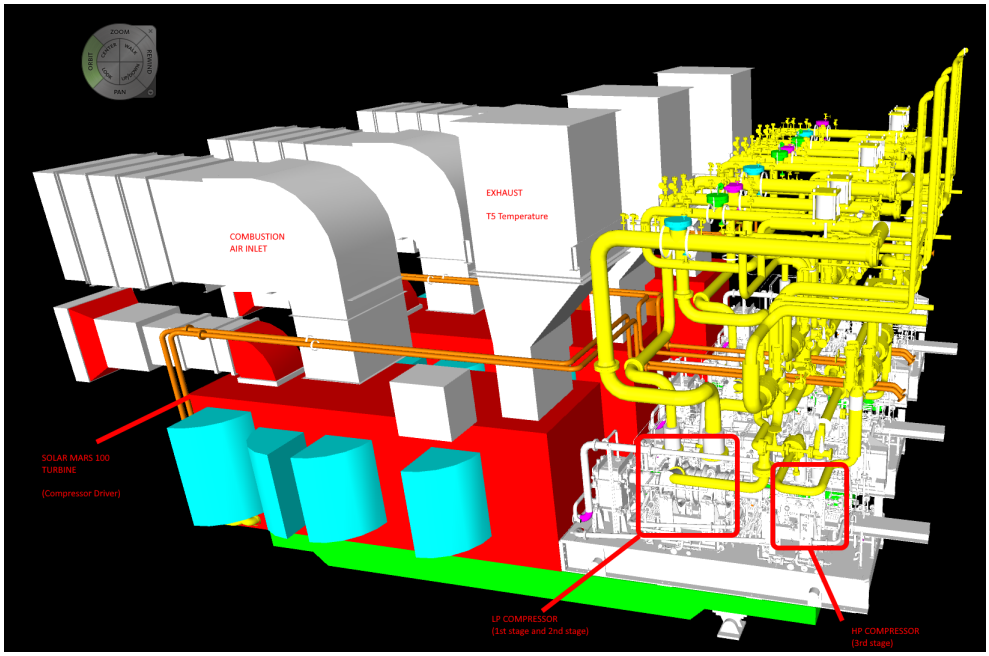

Figure 5.2: Overview of the Solar turbine and compressor trains. (Photo: Screenshot from a 3D model of the FPSO)

5.1.1 Possible Solutions

Different solutions to the problem briefly described in Section 5.1 have been proposed over the years by the engineers in Teekay. As sketched in Figure 5.3, a possible solution was to extend the exhaust outlets of the OPRA turbines with 5 meters to prevent the polluted air from entering the SOLAR Turbines air inlets. A downside of this solution would be an increase in temperature on the 2nd stage separator, which will prevent regular maintenance operations from being executed. Another disadvantage is related to the helipad and helicopter operations, as local and possibly quick changes in air temperature might cause turbulence.

Figure 5.3: Possible solution of extending the exhaust outlets of the OPRA turbines.

Another solution discussed among the maintenance engineers was to create an exhaust manifold routed overboard, as sketched in Figure 5.4. Such a solution could potentially have prevented the exhaust from coming into the air inlet of the compressors and will help regarding the hot exhaust on the 2nd stage separator. The downsides of such a solution would be the considerable cost and an exhaust outlet close to the lifeboat section.

Figure 5.4: Second possible solution of rerouting the exhaust outlets overboard.

5.2 Qualitative Analysis

As previously mentioned, polluted air into the SOLAR turbine air inlets can have the cascade effect of an occasional process shut down trip, due to low compression efficiency. To illustrate the possible combinations of critical events that potentially can cause a PSD-trip, a fault tree diagram (FTD) is drawn. The purpose of this diagram is to illustrate the logical interrelation between the basic events, that apart or together may lead to a PSD trip (Lewins 2013). From the root node, the possible failures and events causing a PSD trip are combined through logical *AND* and *OR* gates in a binary approach (Rausand and Høyland 2003). In Figure 5.5 the root node is defined as the critical event *PSD Trip*. A trip of the compressor system will result in extensive flaring, and too much flaring will harm the "up-time" for the production. As Teekay is restricted to keep the amount of monthly flaring under a predefined limit, a *PSD trip* might result in the need to reduce oil production, in order to stay within the monthly flaring allowance. As seen the Figure 5.5, a Process Shut Down trip can be caused by several different events being:

- The OPRA Turbines not delivering the required grid power.
- A reduced compressor efficiency due to either compression train shut down or reduced efficiency of the Solar Turbines.

A combination of events that will lead to a PSD Trip is called a *Cut Set*. A cut set is said to be minimal if the set cannot be reduced without losing its status as a cut sets (Rausand and Høyland 2003). The criticality of a cut set depends on its order, referring to the number of events making up a set. As seen in Table 5.1 the FTD narrows down to 11 minimal cut sets. The occurrence of these event(s) in each minimal cut set will result in the top event, a PSD trip. The qualitative analysis given is simplified, and each of the basic events can be composed into an individual fault tree diagram.

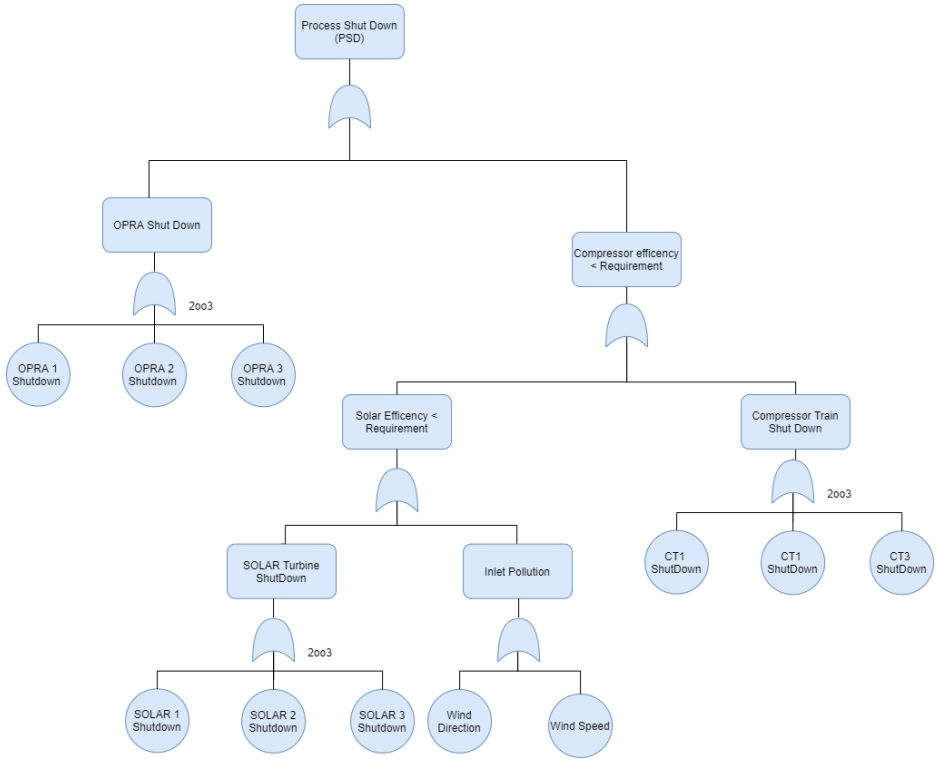

Figure 5.5: Fault tree diagram yielding cut sets regarding a process shut-down event.

Table 5.1: Minimal cut sets corresponding to the fault tree diagram.

Degree	Minimal Cut Sets
1	{Wind Direction}, {Wind Speed }
2	{CT1 Shut Down, CT2 Shut Down}, {CT1 Shut Down, CT3 Shut Down}, {CT2 Shut Down, CT3 Shut Down} {Solar 1 Shut Down, Solar 2 Shut Down}, {Solar 1 Shut Down, Solar 3 Shut Down}, {Solar 2 Shut Down, Solar 3 Shut Down} {OPRA 1 Shut Down, OPRA 2 Shut Down}, {OPRA 1 Shut Down, OPRA 3 Shut Down}, {OPRA 2 Shut Down, OPRA 3 Shut Down}

5.2.1 Component Description

The following section contains a brief explanation of the components that potentially could be involved in causing a PSD trip. A PSD trip is a result of the failure of either the OPRA Turbines, the SOLAR turbines or the components in the compression train. To gain insight into the

failure- modes, effects and causes for each component, a simplified Failure Modes and Effects Analysis (FMEA) were conducted. The simplified FMEA is seen in Table 5.2.

Table 5.2: Summary of the failure mode and effect analysis.

Component	Function	Failure Mode	Failure Effects	Failure Causes	Controls
OPRA Turbine	Power generator to the FPSO	Fail to start/stop on demand. Breakdown. Vibration, Noise, Overheating	Shutdown of the Solar Turbine	Fuel availability / quality	
Solar Turbine	Power generator for the compression trains	Fail to start/stop on demand. Breakdown. Vibration, Noise, Overheating	Reduced compressor efficiency. Shutdown of the compression trains	High temperature and pressure	Sensors
Gas Compressor	Compresses the gas	Fail to compress the gas to the required pressure	Reduced compressor efficiency. Potential PSD - trip	High temperature or pressure. To high flow rate. Leakage	Valves and Sensors
Heat-Exchanger	Reduce temperature of the gas. A cooling medium (water) is used to cool gas and to condensate out liquid	Fail to cool the gas to meet the required temperature	Reduced compressor efficiency. Potential PSD - trip	High temperature. To high flow rate	Valves and Sensors
Scrubber	Separate liquids and condensate from the natural gas	Fail to clean the natural gas.	Reduced compressor efficiency. Potential PSD - trip	To high flow rate and amount of liquids and condensate in the gas	Valves and Sensors
Flare Tower	Dispose gas from the compression trains when a shut down occurs	Fail to dispose the gas	Ignition. Fatal consequences for the FPSO	Reduced compressor efficiency.	EV
Valves	Adjust flow.	Fail to open/close on demand. Delay	Potential PSD - trip	To high flow rate	Sensors

5.2.1.1 OPRA turbines

The Piranema vessel is equipped with three OPRA Turbines, functioning as the main power generators onboard the Piranema Vessel. There is also one backup diesel generator and one emergency generator. As seen in the FTD in Figure 5.5, failure of two of the three turbines will cause the event of a PSD Trip. With a certain wind degree and wind speed the exhaust from the OPRA turbines will go into the air inlet of the SOLAR Turbines, a scenario that also can potentially cause a PSD trip due to the reduced compressor performance. Usually, the three power turbines operate in parallel at around 50% load, and the system load is managed by the process engineers through the SCADA system.

5.2.1.2 SOLAR Turbines

Each of the three gas compression trains consist of single shaft 3 stage Dresser Rand back to back compressor driven by SOLAR Mars 100 Gas Turbines. The SOLAR turbines provide the required mechanical power to the compressors in order to re-inject gas to the subsea wells at 284 barg.

5.2.1.3 Flare

The flare system functions as the central safety system onboard the production plant. The purpose of the flare system is to receive and safely dispose and burn gas from the plant, which is often under high pressure. The main flaring scenarios can be divided into three groups: production flaring, emergency flaring and process flaring (Emam 2015). Production flaring occurs during the exploration and production of oil/gas sectors. There is a need to evaluate the quality of the gas before heavily investing in that particular sector. During these evaluations, there is often no container for the gas, and it has to be burnt down instead. Emergency flaring refers to the burning of gas due to some process upset. Component failures, like a compressor failure or a broken valve, may lead to overpressurization, which needs to be relieved through burning the excess gas. These emergencies involve a high volume of gas with high velocity being burned. In case of a PSD trip, the emergency valve (EV), as seen in Figure 5.7, will be opened, and the gas burns off safely through the flare tower. If the gas is not fully burnt it is a chance it will ignite and explode.

5.2.1.4 Gas Compressor Train

The primary purpose of the gas compression system is to compress the produced gas up to 284 $barg$. The gas is reinjected to the subsea wells to enhance oil production and to extend the lifetime of the wells. The produced wells consist of gas-liquid (condensate) that enter the production plant at 10 $barg$. The main separators separate the gas from liquid, where the gas is routed to the compression train inlets. Here the gas is compressed from around 9 $barg$ up to well reinjection pressure of 284 $barg$ with intermediate cooling and scrubbing. Figure 5.6 illustrates the second compression stage, and an explanation of each component is given in detail in the following list.

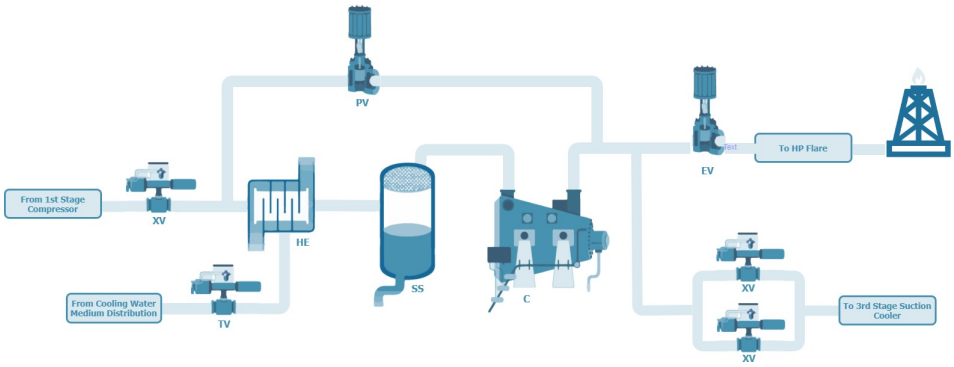

Figure 5.6: Schematic displaying one compressor and the surrounding equipment in the second compressor stage at the Piranema vessel.

C Compressor

Natural gas needs to be compressed for reinjection for oil production optimization to the subsea wells. In the compressor stage, the gas is compressed in a factor of three. Three compressor stages are operating in series on the Piranema vessel, in order to achieve the required reinjection pressure.

HE Heat-Exchanger

The compression phase generates much heat, which results in high gas discharge temperatures. In order to not exceed design temperature limitations and high volumetric flows into the subsequent compression stages, cooling is required between the compression stages. Heat exchangers cool the gas, which also results in liquid condensation as the operating point passes the dewpoint and into the two-phase region (gas + liquid). The gas coolers, Heat-Exchanger, operate with corrosion inhibited fresh water from the on-board cooling medium system.

SS Suction Scrubber

The suction scrubber is required to protect the compressor from seeing liquids on the inlet. The scrubber separates gas and con-

densed liquids (water + HC condensate) received from the upstream heat exchanger. Gas is routed to the next compression stage, while liquids are recycled back to the oil production plant. Upstream in the 3rd stage compressors, the gas is dehydrated by removing the water. The gas is dehydrated to ensure good quality fuel gas for the power and compressor gas turbines and also to protect the subsea wells from experiencing hydrate (ice) formation.

Flare Flare

If needed, the emergency valve (EV) will be opened so that the gas can be burned off in a safe manner through the flare tower. Unignited gas released to the atmosphere is called "cold venting", which is not desired and can lead to an explosion as a worst-case scenario.

V Valves

XV: Safety isolation valves ensure process volume segregation during a PSD. If an incident occurs, the valves can isolate the system and prevent errors.

TV: The temperature control valve opens if the gas temperature is too high or too low, in order to adjust the amount of cooling medium that is feed to the gas cooler.

PV: The Anti-Surge valve ensures that the compressor operates away from the surge line preventing equipment damage due to surge. These valves are usually fast and controlled by the compressor control system.

EV: The emergency valve will open towards flare (process volume blow-down) if the pressure or temperature in the system is too high.

The gas goes through three compression stages, as illustrated in Figure 5.7, before it is reinjected into the wells.

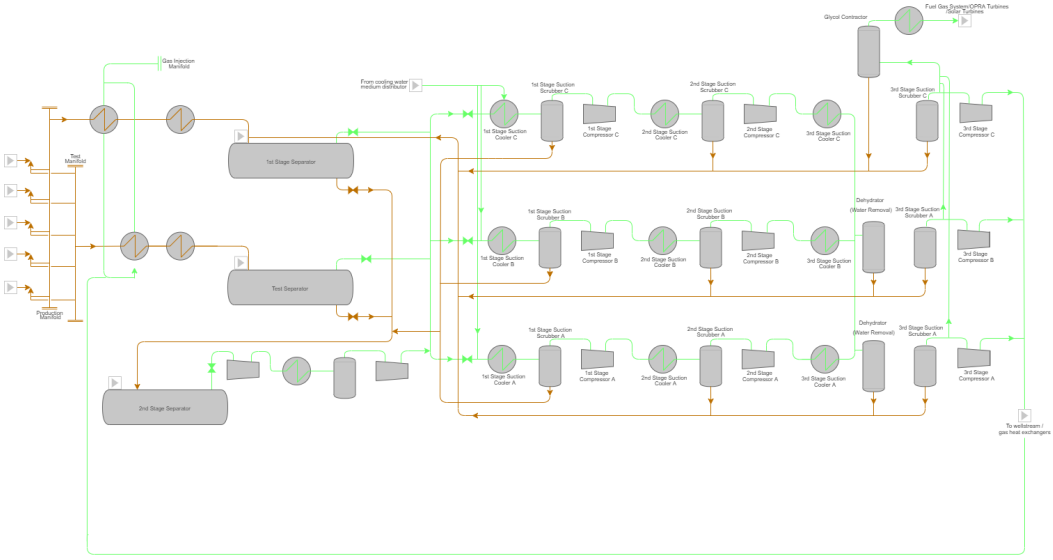

Figure 5.7: Schematic displaying the three compressor trains with corresponding components at the Piranema vessel.

Chapter 6

Preprocessing

Creating an accurate health indicator for a compression train requires a vast amount of historical data. The sensor data was retrieved from the Piranema-server stationed at the FPSO in Brazil. Similar to the data extracted from the Piranema-server, real-world dataset differs from simulated data in many ways and poses many more challenges when creating an accurate model. Such data is often unformulated, composed of missing data points, noisy data, outliers and inconsistent data. Hence, preprocessing is an essential yet time-consuming phase. In this chapter, a comprehensive preprocessing of the Piranema Dataset is performed in order to create an accurate predictive model. First and foremost by exploring the dataset to potentially find new points of interest, next by imputing missing values, detecting outliers, scale the data and select the most relevant features for a future ML model.

6.1 Data Exploration

The dataset obtained from Teekay's FPSO Piranema contains 70 different features containing data related to weather conditions, the Solar turbines, the OPRA turbines and the compression trains. Given the number of features, the goal of the exploration phase is to get a better un-

derstanding of the collected features and potentially find new points of interest among the data. After collecting the data from the server of the Piranema Vessel, a thorough exploration of the data set is required in order to gain valuable insight.

6.1.1 Limitations

6.1.1.1 Time Limitation

The Piranema Vessel started its production back in 2007, and the data obtained from its server is dated back to *01.06.2007*. The data series extracted from the IP21 information system has an hourly sampling rate. By exploring the dataset, there is a vast majority of sensors not logging at this point in time. Thus, the data set is limited to the time frame shown in Table 6.1.

Table 6.1: Table indicating the time frame limitation.

TimeFrame Limitation	
Start Date	End Date
13.05.2011 08:00	27.02.2020 00:00

6.1.1.2 Missing Values

Even after limiting the time frame, the dataset contains missing values, which is common for real-life datasets. A simple and basic strategy commonly used is to discard the entire row containing a missing value, often referred to as a *NaN*. *NaN* indicates that a value is undefined. The number of such values in the given dataset was 110839, meaning that a total of 2.05 % of values in the dataset is *NaN* values. Discarding all the rows containing such values could result in loss of valuable data and essential patterns. Thus, incomplete values must be imputed.

6.1.1.3 Label Encoding

All of the features in the dataset is of numerical type. Thus, no label encoding will be necessary. If the dataset contained categorical features, these must have been encoded, meaning that a feature of categorical type, for example, *ON* and *OFF*, would have been encoded to the numerical values 1 and 0.

6.1.2 Explanation of the Features

6.1.2.1 Weather Data

The features related to the weather are information regarding the wind speed and the wind direction at the FPSO. In all there are five features, with the names *Wind Speed Sensor 1*, *Wind Speed Sensor 2*, *Wind Direction Sensor 1*, *Wind Direction Sensor 2* and *Deg Heading*.

6.1.2.2 SOLAR Turbine Data

For each of the three Solar Turbines the *Air Inlet Temperature*, *T5 Temperature* which is the turbine combustion chamber temperature, *Discharge Pressure*, *Speed Set Point*, *Producer Speed* and the *RPM*, short for revolutions per minute, are the given features.

6.1.2.3 OPRA Turbine Data

The feature related to the OPRA turbines is the outlet exhaust temperature logged for each turbine and named *OPRA Gas Temperature*.

6.1.2.4 Compressor Train

As shown in Figure 5.7 in Section 5.2.1.4 the compressor train system consist of three individual compressors. All three compressors each have three stages. For each stage the *Suction Temperature*, *Suction Pressure*, *Flow Indicator*, *Discharge Temperature* and *Discharge Pressure* are logged.

6.1.2.5 Flare

HP Flare, short for High-Pressure Flare is the rate of flared gas. The flare tower has a safety function of burning and disposing excessive gas from the FPSO.

6.1.3 Target Variable Identification and Creation

Primarily, the scope of the ML model is to create an ML driven health indicator of the compression trains. Providing the current and future health would yield useful information such that the engineers of Teekay can adjust the operation on the vessel to avoid a PSD trip. A target variable which indicates the compression system's performance is therefore required. During the close collaboration with the process engineers at Teekay, several possible target variables were suggested and evaluated. This subsection covers the process of obtaining and constructing the different target variables, as well as evaluating their advantages and disadvantages.

6.1.3.1 Compressor Efficiency

Calculating a compressor's efficiency is a complex task, which differs significantly for compressor types as well as the fluid being processed. It combines multiple disciplines like chemistry, fluid dynamics, thermodynamics and mechanics. The assumption was that given the input variables to the compressor system, the difference between the expected compressor effect, $W_{expected}$, and the actual compressor effect, W_{actual}, could be a reliable target variable for indicating the compressor performance. The expected compressor effect is an estimate derived from the official performance document of the compressor, whereas the actual compressor effect is a theoretical estimate based on the physical properties of the compressor and its' medium.

The first step was to calculate the expected compressor effect. The

compressor performance documents consist of multiple performance curves, where one of them is pictured in Figure 6.1.

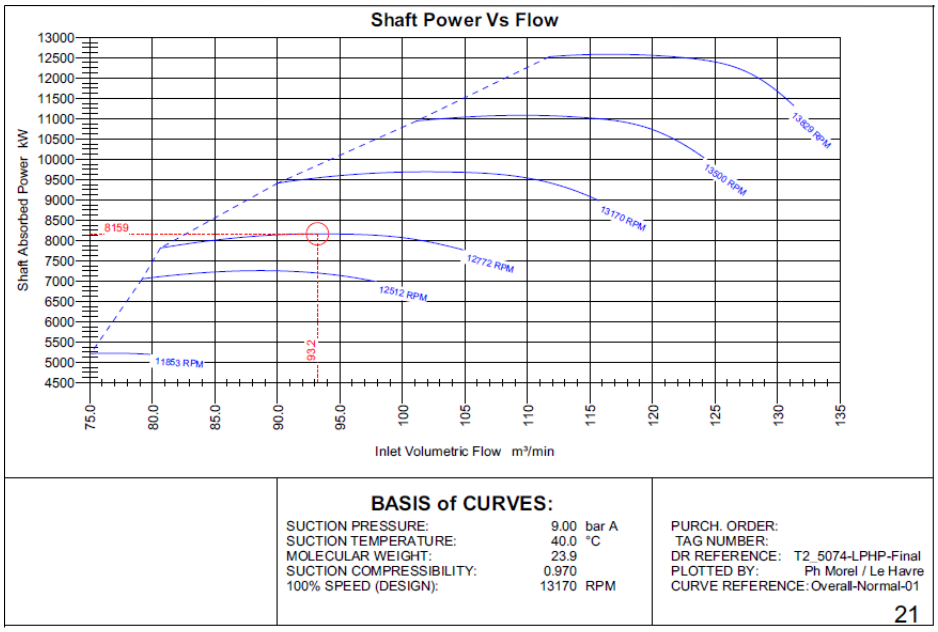

Figure 6.1: Official performance curves from Dresser-Rand, the provider of the compressors.

This empirical data is obtained by testing and measuring relevant parameters of the compressor during normal conditions. Hence, it will provide the expected compressor effect given the assumption that everything works under a normal condition. The effect is dependent on the volumetric inlet flow, as well as the RPM of the turbine running the compression train. A quick inspection of the RPM history from the turbines shows that when they are operating, they run on a reasonably steady RPM, ranging from 12700 to 13200. Therefore, the relevant values in the following calculations lie in between two of the given performance curves. Hence, the next step was to obtain the corresponding equations of the relevant performance curves.

These equations were not available and had to be reconstructed. *GetData Graph Digitizer* is a program used to digitize graphs and plots from

an existing pdf or image. The process of retrieving the data corresponding to the graphs is done manually by following the steps presented in Figure 6.2 and 6.3. Figure 6.2 illustrates how to select the start and end point of the x and y axis on the image, and followed by entering the respective min and max value. Figure 6.3 illustrates how to hover the cursor over the graph, to select the relevant datapoints.

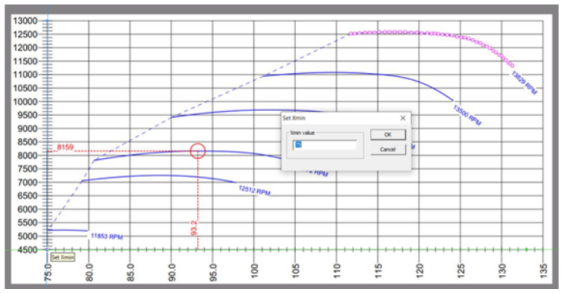

Figure 6.2: Illustration of the first step for retrieving datapoints with the *GetData Graph Digitizer* program.

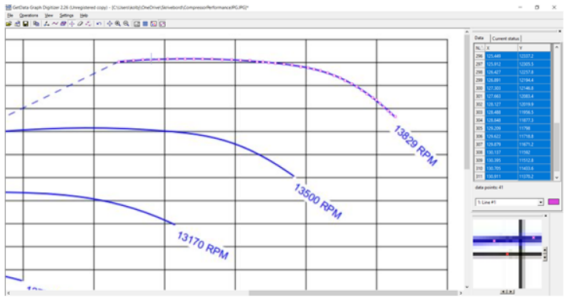

Figure 6.3: Illustration of the second step for retrieving datapoints with the *GetData Graph Digitizer* program.

Polynomial interpolation was used to obtain the corresponding equation to the collected data points. The interpolation was simply done by using Excels built-in interpolation function. With the equations obtained, the next step was to calculate the volumetric flow, which was the input value when determining the compressor effect.

Volumetric flow rate is the volume of fluid that passes per unit of time and is often represented by the letter Q. The Piranema database provides continuous monitoring of the mass flow, represented as volumetric flowrate in standard conditions (1 atm, 20$degC$). It is a common practice to use standard units for volumetric flow rates to perform easier comparisons between gaseous flows. Standard volumetric flow rates represent the actual volumetric flow corrected to the standardised properties of pressure and temperature. There exist many different standards, each having their baseline properties. The standard used at The Piranema Vessel is 101.2 $kPaa$ and 25 $degC$ (288 K).

The actual volumetric flow is calculated by applying the *Ideal Gas Law* shown in Equation 6.1, where Q_{actual} is the actual volumetric flow in $\frac{m^3}{h}$, Q_{std} is the standard volumetric flow $\frac{Sm^3}{h}$, P_{std} is pressure at standard conditions at 101325 Pa, P_{actual} is pressure at actual condition in Pa, T_{std} is temperature at standard conditions in Kelvin, 288.15 degK, T_{actual} is temperature at actual condition in Kelvin, Z_{std} is compressibility factor at standard conditions, and Z_{actual} is compressibility factor at actual conditions.

$$Q_{actual} = Q_{std} \cdot \frac{p_{std} \cdot T_{actual} \cdot Z_{actual}}{p_{actual} \cdot T_{std} \cdot Z_{std}} \tag{6.1}$$

The compressibility factor is defined as the ratio of actual volume at a given pressure and temperature to the ideal volume under the same conditions of pressure and temperature. In other words, the ratio of the molar volume of a gas to the molar volume of an ideal gas at the same temperature and pressure. Due to the small change in temperature and pressure between the standard conditions and the actual conditions, the compressibility factors are nearly equal, resulting in the fraction $\frac{Z_{actual}}{Z_{std}} \approx 1$ and the equation can be rewritten as Equation 6.2.

$$Q_{actual} = Q_{std} \cdot \frac{p_{std} \cdot T_{actual}}{p_{actual} \cdot T_{std}} \tag{6.2}$$

The variables Q_{std}, p_{actual} and T_{actual} are all values measured by the sensor system on the Piranema vessel, while p_{std} and T_{std} are both standard values. The compressor effect for each sample in the dataset can now be calculated by entering the actual volumetric flow into the obtained equations from the performance curves.

With the obtained expected compressor effect, the next step was to calculate the actual compressor effect for each sample. The calculations in the obtained equations from the performance curves had the assumptions that the compressor was operating during normal conditions. When estimating the actual compressor effect, the assumption no longer holds, where the operating conditions vary. The attempt was to calculate the theoretical compressor work for an ideal gas and then compensate for the assumption of an isentropic process by using the polytropic efficiency which is given in the compressor performance doc

Equation 6.3 expresses the specific compressor work.

$$w = -\frac{k}{k-1} \cdot p_1 \cdot v_1 \cdot \left[\left(\frac{p_2}{p_1} \right)^{\frac{k-1}{k}} - 1 \right] = -\frac{k}{k-1} \cdot m \cdot R_{specific} \cdot T_1 \cdot \left[\left(\frac{p_2}{p_1} \right)^{\frac{k-1}{k}} - 1 \right], \tag{6.3}$$

where k is the isentropic coefficient, T_1 is the temperature of the gas in Kelvin K, R is the specific gas constant and $\frac{p_2}{p_1}$ is the pressure ratio.

The specific gas constant is given by the universal gas constant R divided by the molar mass M of the gas mixture, as shown in Equation 6.4.

$$R_{specific} = \frac{R}{M} = \frac{8314}{23.90} = 347.87 \tag{6.4}$$

The specific work w is multiplied with the compressibility factor Z of the gas, expressed in Equation 6.5.

$$w_{ideal} = w \cdot Z \tag{6.5}$$

Using Equation 6.6 compensates for the assumption of the ideal gas, by dividing the specific compressor work with the polytropic efficiency.

$$w_{actual} = \frac{w_{ideal}}{polytropic efficiency} \tag{6.6}$$

The polytropic efficiency varies with the type of compressor as well as the operating conditions. The values are obtained from the compressor performance document.

Finally, the compressor effect is determined by multiplying the compressor work with the mass flow, as shown in Equation 6.7.

$$W_{actual} = w_{actual} \cdot m, \tag{6.7}$$

where $m = \rho \cdot Q_{actual}$. ρ refers to the density of the gas and Q_{actual} is the volumetric flow calculated in equation 6.2.

The difference ΔW between W_e, the expected effect if the conditions are normal, and W_{actual}, the effect during the actual conditions, can now be calculated and used as a metric for evaluating the operating conditions for the compressors. The hypothesis was that an increase in ΔW would indicate a weakened compressor performance.

6.1.3.2 High Pressure Flaring

Another potential variable that could represent the compressor performance was the amount of gas burnt off in the flare system. During normal operating conditions, the amount of flared gas is steady and low. A decrease in compressor performance naturally results in the compression trains not being able to compress the desired amount of gas per unit time. A decrease in compression efficiency leads to an over-pressurized system, where pressure needs to be relived through flaring. Thus, the dif-

ference in flaring burnt per unit of time can be an indication of normal vs abnormal condition for the compression system.

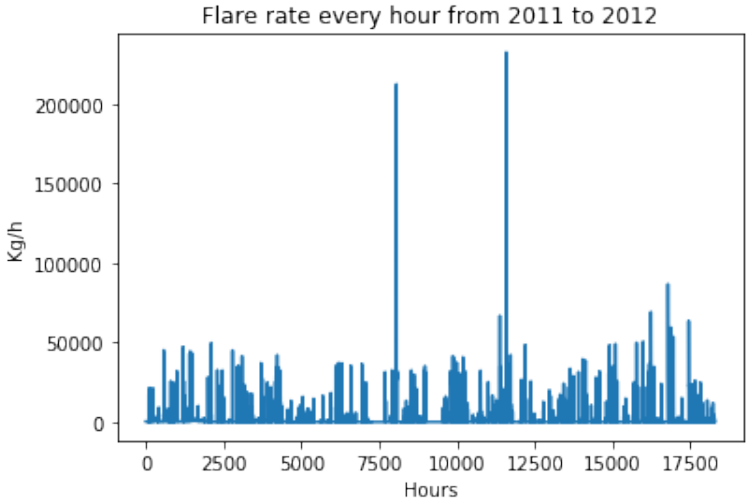

Figure 6.4: High pressure flare rate sampled every hour from 2011 til 2012.

Figure 6.4 shows the rate of gas being flared off every hour from 2011 to 2012. Gas flaring is a necessity to ensure safety onboard the vessel and to reduce the potential damage of components. However, excess flaring will result in significant economic loss and a negative impact on the environment. Reviewing the historical data shows that the amount of flared gas is relatively stable, though it does experience some extreme spikes.

6.1.3.3 Choice of Target Variable

Introducing gas flaring as a target variable can have multiple benefits. Firstly, it can function as a reliable indicator of the compression train performance. Having a well-describing target variable is crucial in order to create an accurate model, that can recognize patterns leading up to system damage and identify problem areas. As mentioned in Section 5.2, Teekay Offshore Production is assigned a monthly flaring allowance.

If the company exceeds this limit, the process plant needs to be shutdown. Thus, an indicator also predicting the future amount of flare can potentially contribute to optimizing the up-time for the plant. High pressure flaring are in most cases related to low compression efficiency, but can also be a result of the start or shutdown of a well or to check the quality of the oil. Despite this level of uncertainty, high pressure flaring is chosen as the target variable, due to an even higher level of uncertainty using ΔW. This is further discussed in Section 9.2.

6.2 Tentative Model

Being able to evaluate the performance of different preprocessing techniques is essential in order to create a dataset of exquisite quality. Several of the preprocessing techniques also require a model to perform their desired actions. Thus, a tentative ML-model is created in order to accomplish both cases. After an exploration of the dataset in Section 6.1.1.1, it became essential to create a model that leverages the properties of the data. As the data is sequential, an essential requirement was a model utilizing the time-series aspect of the data. Both the literature and theory affirm that the best performing machine learning model on time-series is a recurrent neural network. Keep in mind that the model created is only tentative and does not reflect a fully optimized model. Nevertheless, several optimizations steps are completed, like cross-validation and averaging over multiple runs to ensure valid results.

6.2.1 Tentative Model Architecture

The model used in the preprocessing phase is a *Gated Recurrent Unit* (GRU) network, which is a type of recurrent neural network. As explained in Section 3.3.3.1, GRU is a modified version of a simple recurrent neural network. The most significant strength of the GRU compared to a simple RNN is the ability to eliminate the *vanishing gradient problem*,

elaborated in Section 3.3.2.5. In short, by eliminating the vanishing gradient problem, the model is able to utilize previous relevant information, which is desirable when performing predictions on time series data. Due to the large dataset, the dataset is split into batches of 256 samples. Such a partitioning means that only 256 samples are used in each forward and backward pass in the network. Increasing the batch size increases the required memory space. Hence, using larger batch sizes or even the entire dataset in one forward and backward pass might exceed the memory capacity. The sequence length is set to 100 samples, which means that the model evaluates data from the 100 previous samples, in this case, the last 100 hours. The number of units in each hidden cell is set to 512 and the *sigmoid function* is used as the activation function. The *root mean square propagation* (RMSP) algorithm is used to update the model weights with a learning rate of 0.001. Finally, the monitored loss metric is set to the *mean square error*. The majority of the parameters mentioned above is determined by the most common values used in previous experiments on similar data.

Some further modifications have been implemented to ensure valid results. First, a nested cross validation technique was implemented such that the final model score is the combined average of the separate model scores on various partitions of the dataset. Five different partitions of the data were used as shown in Table 6.2. The partitions are arranged to preserve the time series aspect of the data. In example, the training set of partition 1 contains the first 50% of the data and the validation set contains the remaining 50%.

Table 6.2: Partition of dataset used for training and validation of the tentative model.

	% of training data	% of validation data
Partition 1	50	50
Partition 2	60	40
Partition 3	70	30
Partition 4	80	20
Partition 5	90	10

To even further validate the results, five separate models are trained on each partition, where the combined average scores reflect the model performance. Each separate model is trained on the entire dataset at least ten times, in other words, the epoch parameter is set to 10. Two callback functions, *early stopping* and *model checkpoint* were implemented in order collect the best results from each model. Each model performance is evaluated by predicting values on the unseen test set and calculating the root mean square error, RMSE. The entire source code is found in appendix A.2, and Algorithm 1 shows the pseudocode.

Algorithm 1 Tentative Model for Preprocessing

```
for each Partition do
    Train 10 separate models
    for each of the 10 separate models do
        Perform 10 iterations on the training set
        for each iteration do
            Stop training if the validation loss doesn't improve
            Save the weights from the best performing model
        end for
        Perform predictions on the unseen test set
        Calculate the RMSE of the prediction
    end for
    Calculate the average RSME over all models
end for
Calculate the average RSME over all partitions
```

6.3 Imputation of Missing Values

The goal of an imputation technique is to replace all the *NaN* values in order to create a complete dataset. Some ML models are capable of handling such values, but they are the exception, not the rule. In the exploration phase, a number of 110389 missing values were encountered in the Piranema dataset. These *NaN* values must be replaced with substituted values and is a way of avoiding the pitfall of discarding entire rows containing missing values. In Figures 6.5 and 6.6 a heat map illustrating the data instance on the y-axis and the variables on the x-axis is shown. The segments in white illustrate a *NaN* value and black illustrates a numerical data point. There are several variables with larger segments containing *NaN* values, and thus imputing these missing variables is an essential step in the preprocessing phase.

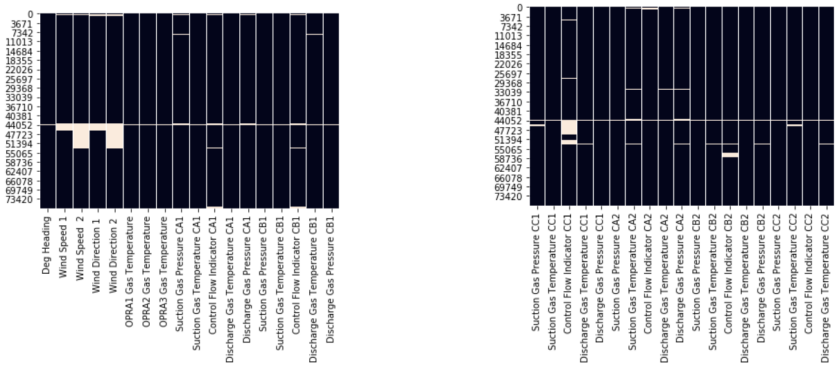

Figure 6.5: Heat Map illustration of NaN values for features 0 to 37

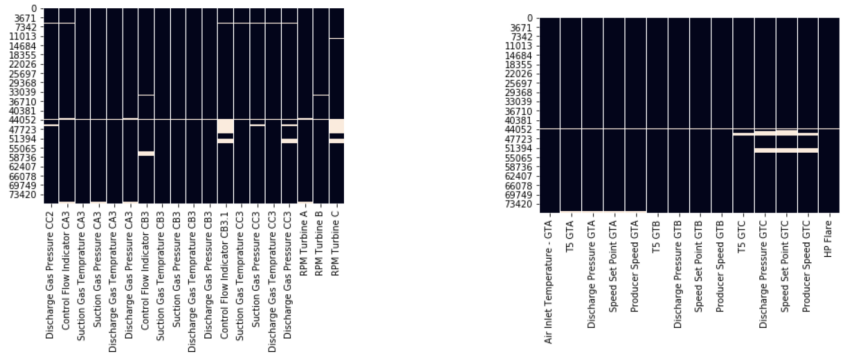

Figure 6.6: Heat Map illustration of NaN values for features 38 to 70

6.3.1 Comparison of Imputation Techniques

A partition of the dataset not containing any *NaN* values is used to compare the different imputation techniques. Some of the imputation methods take into account that the dataset is sequential, while others do not. Univariate methods such as *Mean*, *Median* and *Most Frequent* do not take the time-series aspect into account, while *Backward Filling*, *Forward Filling* and *Interpolation* do. As for the multivariate techniques, the time-series aspect is not considered.

After exploring the dataset, the segment within range [1500 - 5000] does not contain any *NaN* values and will be used to illustrate the different imputation techniques, called $D_{Complete}$. The original dataset contains 110 839 *NaN* values of a total of 5.395.880 datapoints, meaning that 2.05% of the datapoints are *NaN values*. Thus a total of 2.05 % of the existing datapoints will be removed to illustrate the accuracy of the different imputation techniques, creating $D_{Incomplete}$. In order to quantify the performance of imputation algorithms, the following steps have been done:

- Create the dataset $D_{complete}$ from the Piranema dataset.
- Create a new dataset $D_{incomplete}$ which is a replica of $D_{complete}$ but

with 2.05% of missing values.

- Apply the different imputation techniques to impute the missing values of $D_{incomplete}$.

- Compare the difference between $D_{complete}$ and $D_{incomplete}$

6.3.1.1 Univariate Imputation

All the *NaN* values in the $D_{incomplete}$ dataset is imputed using different univariate imputation techniques introduced in Section 3.1.1.3.1, in order to illustrate the bias that imputation techniques can introduce to the dataset. Table 6.3 shows the original value and the imputed value using the imputation techniques *Mean*, *Median*, *Forward Fill* and *Backward Fill*, *Linear* and *Polynomial* interpolation, respectively. To say something about the quality of the imputation method, the RMSE is calculated using the imputed value $\hat{Y}_{HPFlare}$ and the original value $Y_{HPFlare}$, as shown in table 6.4

Table 6.3: Table showing some instances of the original and imputed values for the variable HP-Flare.

	$D_{Complete}$	$D_{Incomplete}$ with Imputation Methods:					
Data Instance	**Original**	**Mean**	**Median**	**Forward**	**Backward**	**Linear**	**Poly**
1551	728.90	744.61	67.70	754.10	733.70	743.90	771.93
1632	737.80	744.61	67.70	725.70	737.90	731.80	675.33
1663	1443.60	744.61	67.70	994.20	1561.80	1278.00	875.32
1677	574.50	744.61	67.70	592.40	555.10	573.75	554.62
1766	575.60	744.61	67.70	596.10	575.60	585.85	610.12

Table 6.4: Mean square error and standard deviation between the actual and the imputed value.

Imputation Method	**RMSE**	**STD**
Mean	53.45	1291.29
Median	51.87	1312.77
Forward Fill	54.42	1342.08
Backward Fill	34.01	806.43
Linear Interpolation	43.92	1061.65
Polynomial Interpolation	55.68	1216.10

6.3.1.2 Comparison Univariate and Multivariate

In an attempt to compare the imputation techniques, a recurrent neural network will be trained to predict the target variable *HP Flare*. At random 2.5 % of the values in the dataset will be replaced with *NaN-values*, creating the dataset $D_{Incomplete}$. First all the *NaN* values in the $D_{Incomplete}$ dataset will be imputed using univariate methods that gave the most accurate results, which were the *Forward Filling*, *Backward Filling* and *Linear Interpolation*, as well as some multivariate imputation techniques provided in the *Scikit* ML library. The applied multivariate imputation techniques were an iterative *decision tree* and an iterative and non-iterative version of *k-nearest neighbour*. After imputing the entire dataset, the feature *HP Flare* will be predicted using the trained recurrent network explained in Section 6.2. The root mean square error and standard deviation is then calculated to give an estimate of the model's accuracy. The approach of comparing the different imputation techniques is summarized in following list. The attempt was executed ten times, all giving the same results as seen in Figure 6.7. Thus, linear interpolation was the technique yielding the most accurate predictions, and was chosen as the imputation technique to be used in the modelling phase.

1. Remove 2.5% of the values in $D_{Complete}$ to create $D_{incomplete}$

2. Apply several imputation techniques to calculate the missing variables of $D_{incomplete}$
3. Train the tentative model on all versions of the imputed dataset $D_{incomplete}$
4. Calculate the RMSE between the predicted $\hat{y}$ and the actual value y

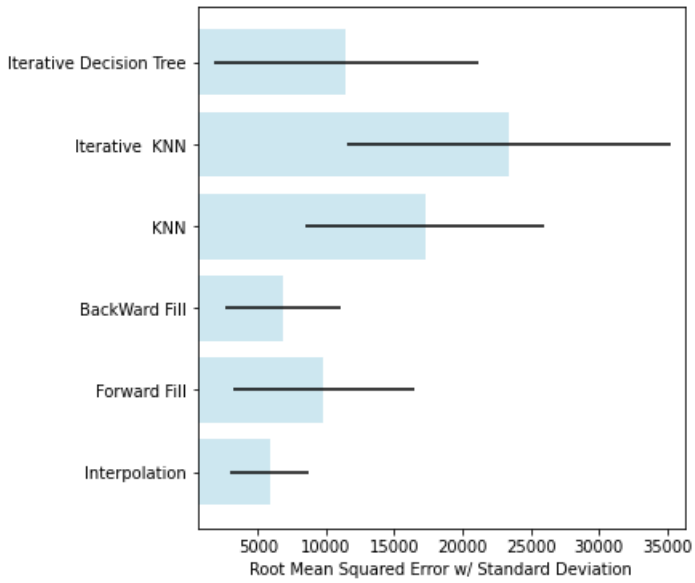

Figure 6.7: Comparison of different imputation techniques measured in terms of root mean square error illustrated in blue and corresponding standard deviation.

6.4 Identifying Outliers & Operational Modes

The DBSCAN algorithm was used to detect the different operational modes for each feature in the dataset and identify unwanted abnormal observations. As mentioned in Section 3.1.1.5, the input parameters of the

DBSCAN were the eps and min_points. These values were manually determined by inspection and were different for each feature due to the varying range of values. Figure 6.8 shows the result of DBSCAN used on four of the features. The dataset is reduced down to a month of data for easier visualization.

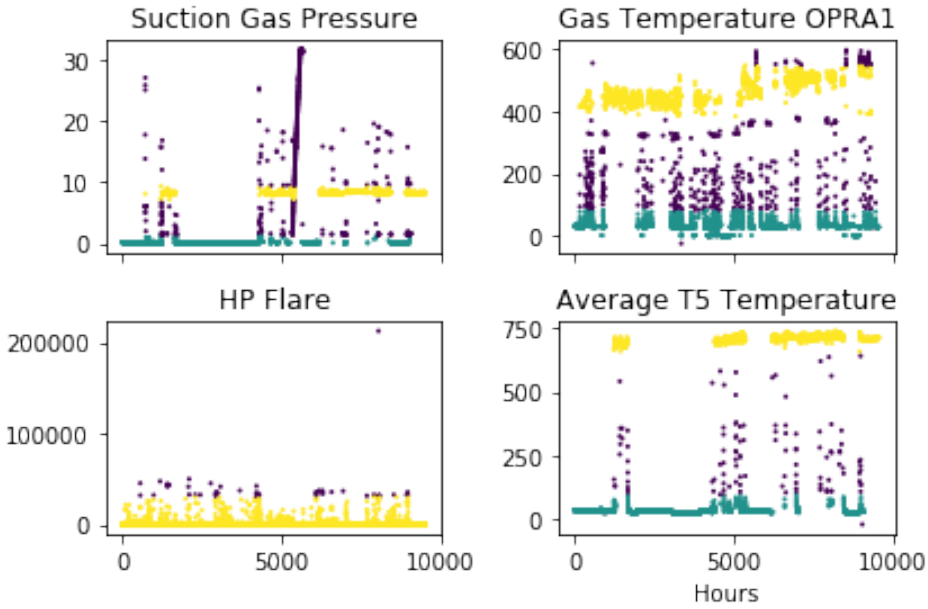

Figure 6.8: Results of DBSCAN applied on four features to determine their operational modes.

The algorithm can identify the different operational modes for each feature. The top right plot displays the gas pressure at the suction of the 1st stage of compression train A. The algorithm identifies two operational modes which are values around 0 bar, where the compressor is turned off, and values among 9 bars where the compressor is operating. The remaining values coloured in purple, as seen in Figure 6.8 are identified as outliers and require further analysis.

The outlier values, which lies in between the two operational modes, could indicate the starting and stopping of the compressor. If that is the

case, then these values should not be considered as abnormal behaviour. Though, it could also represent a smaller error where the compressor is not fed a gas with the correct properties. This must be investigated in comparison with relevant systems components. The outlier values above the 9 bar, shows that the gas entering the compressor has a much higher pressure than it is supposed to have. This is a clear indicator that the system is not working correctly. There could be multiple reasons for this result, like sensor failure or component failure.

The bottom left plot in Figure 6.8 shows the amount of gas that is flared off. There is no clear indication of multiple operational modes. However, the algorithm defines the range of the most common values, such that values located outside this range are identified as points of interest. These outliers are of great interest in order to explain abnormal behaviour, as well as providing information regarding potential future threats.

6.5 Feature Scaling

Another critical step during preprocessing is feature scaling, as discussed in Section 3.1.1.2. Though exceptions exist, most ML algorithms do not perform well when numerical attributes have different scaling. Therefore, multiple scaling techniques have separately been applied to the dataset and evaluated. As an example, Figure 6.9 illustrates five samples from the original dataset, whereas Figure 6.10 displays the same five samples after applied to three different scaling techniques. The plots combine two features that initially had a significant range difference. After applying the Min-Max-Normalization, Z-Score Normalization and Decimal Scaling Normalization, it is evident that the different feature ranges have significantly been reduced.

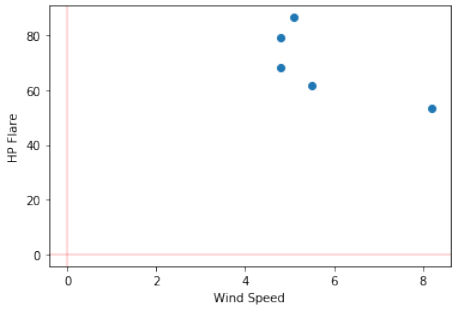

Figure 6.9: The original data points in the feature scaling evaluation.

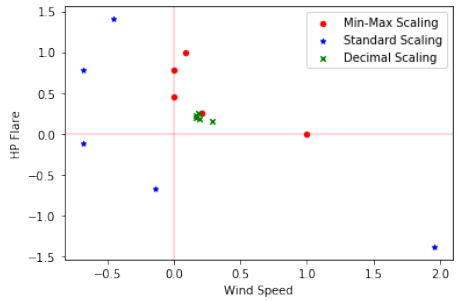

Figure 6.10: Data points after multiple scaling techniques.

Another example is displayed in Figures 6.11, 6.12, 6.13 and 6.14. Boxplots are used to visualize the distributions of features; in this case, five features were selected. Figure 6.11 shows the feature distributions from the original dataset and displays that the feature ranges are greatly unlike. As shown, the feature *RPM Turbine A* consists of data distributed over a range much larger than the remaining four features.

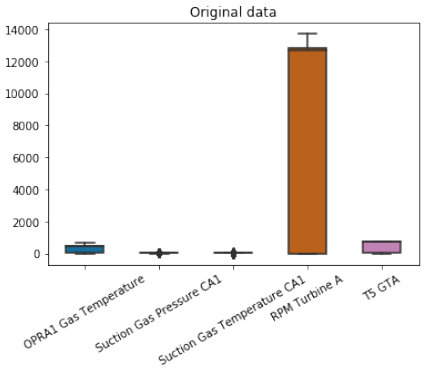

Figure 6.11: Boxplot visualizing the distribution of five features from the original dataset.

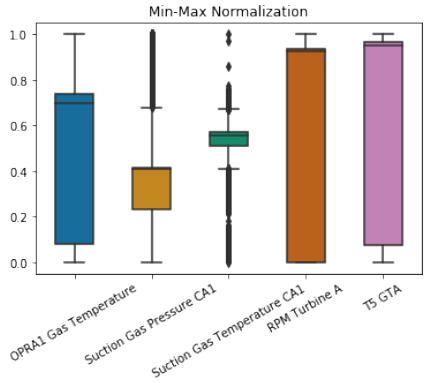

Figure 6.12: Boxplot visualizing the distribution of five features after the Min-Max normalization.

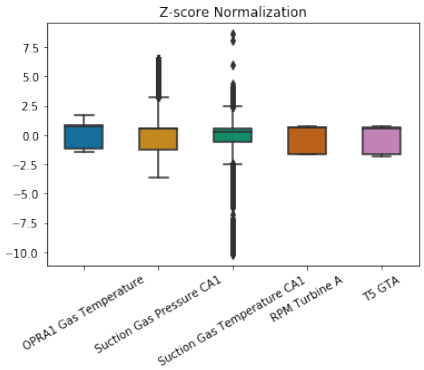

Figure 6.13: Boxplot visualizing the distribution of five features after the Z-score normalization.

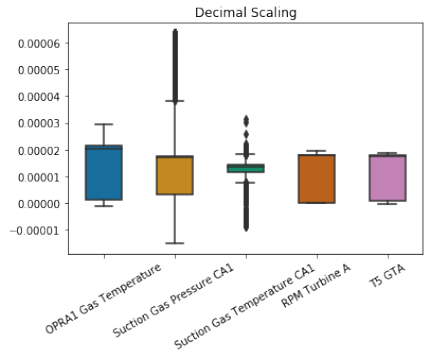

Figure 6.14: Boxplot visualizing the distribution of five features after the decimal normalization.

Figure 6.12 displays the distribution of the same features after being scaled with the Min-Max Normalization techniques. The feature ranges are now the same, where the data lies between the values 0 and 1. The Min-Max normalization technique is suitable for data that does not follow a Gaussian distribution. However, it is significantly sensitive to outliers. Figure 6.13 shows the same data after applied to the Z-Score normalization technique. The feature ranges are not the same, but more

similar than the ranges from the original dataset. In this case, the features are arranged in a way that they have a mean value of 0 and a standard deviation of 1. This scaling technique is best suitable when the data is close to normally distributed. The final plot, Figure 6.14 displays the features after being scaled with the Decimal normalization technique. Similar to the Min-Max normalization, the Decimal normalization technique produces values between 0 and 1.

The next step is to evaluate the scaling technique most suitable for the given dataset. In order to perform such an evaluation, the tentative model described in Section 6.2 was fed four different datasets. These four sets includes the original dataset before scaling and scaled using the three different techniques. These four dataset includes three normalized versions of the dataset described in Section 6.1.2, as well as the original dataset.The hypothesis behind the experiment was that the model error, would indicate the impact of the different scaling strategies. As explained in Section 3.1.1.2, scaling of data may have an impact on how fast models converge. It is therefore desirable to collect information regarding the models' convergence time to find the most suitable scaling technique. It is not the model error in itself, but the difference in error between the models that are of interest.

Table 6.5: Results from the tentative model on the original dataset, the applied Min-Max normalization dataset, the applied Z-score normalization dataset and the applied decimal normalization dataset.

	Mean RSME by Nested Cross-Validation	STD RSME by Nested Cross-Validation
Original Data set	4555.13	277.86
Min-Max Normalization	3575.68	356.38
Z-Score Normalization	4013.27	256.97
Decimal Scaling	4553.12	277.92

The tentative model is used on each of the four datasets, where the results are described in Table 6.5. It is a significant difference in terms of loss before and after feature scaling, except for decimal scaling. Notably, the model applied with the Min-Max normalization has performed

significantly better than the model with the original dataset. Not surprisingly, the model applied with the Z-Score normalization has not performed equally well since this scaling technique is best suitable on normally distributed data, and the original dataset does not follow such a distribution. Figures 6.15, 6.16, 6.17 and 6.18 displays the loss curves for both the training and validation set for each model. Again, the model with the Min-Max normalization techniques outperforms the remaining models in terms of reaching convergence faster. The Min-Max normalization model reaches convergence on the validation set after approximately three epochs, whereas the decimal scaling model reaches convergence after 12 epochs. The model with the Z-Score normalization and the model with the original dataset does not indicate any stable convergence. Overall, the Min-Max normalization is the highest performing scaling technique and is therefore chosen.

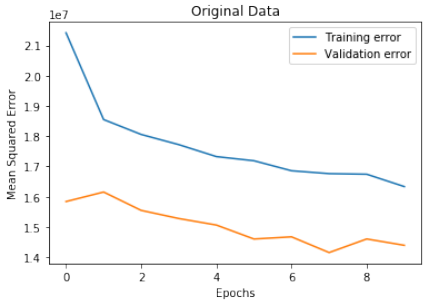

Figure 6.15: Mean square error on the training- and test set with the original data.

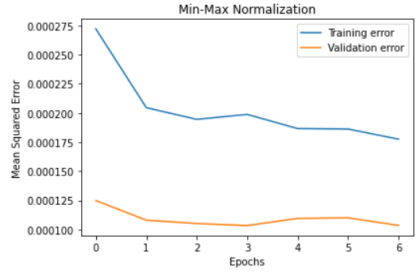

Figure 6.16: Mean square error on the training- and test set after Min-Max normalization

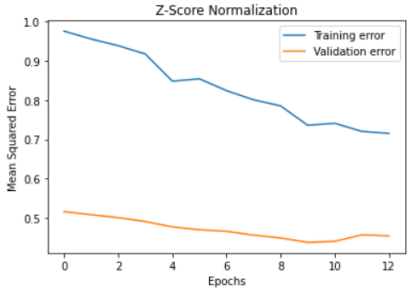

Figure 6.17: Mean square error on the training- and test set after Z-Score normalization

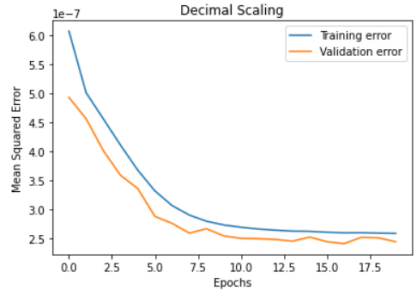

Figure 6.18: Mean square error on the training- and test set after decimal normalization

6.6 Feature Selection

The collected dataset from the IP21 database contains 70 features, where one is the target variable. As explained in Section 3.1.2.1, the purpose of feature selection is to reduce the dimensionality of the data. With reduced dimensionality follows reduced computational time, and in some cases increased model accuracy due to the removal of confusing information. In addition, it is desirable to remove features that have no correlation with the target variable or do not provide any new information. This section describes the feature selection steps applied to the dataset, starting with inspecting and visualizing the data. Further, each feature

is measured by its variance before all features are compared with each other and the target variable to reduce redundant information. Finally, a recurrent neural network has been used to select the last remaining features.

6.6.1 Percent of Missing Values

The first selection step was to inspect the features for missing values. A common rule of thumb is to remove features that contain 95% missing values and perform imputation techniques on features with less than 50% missing values. Features with 50 to 95% missing values should not be applied with an imputation technique, but rather be replaced by a binary variable representing the presence of the feature samples. Figure 6.19 displays the percentage of missing values for the features in the dataset.

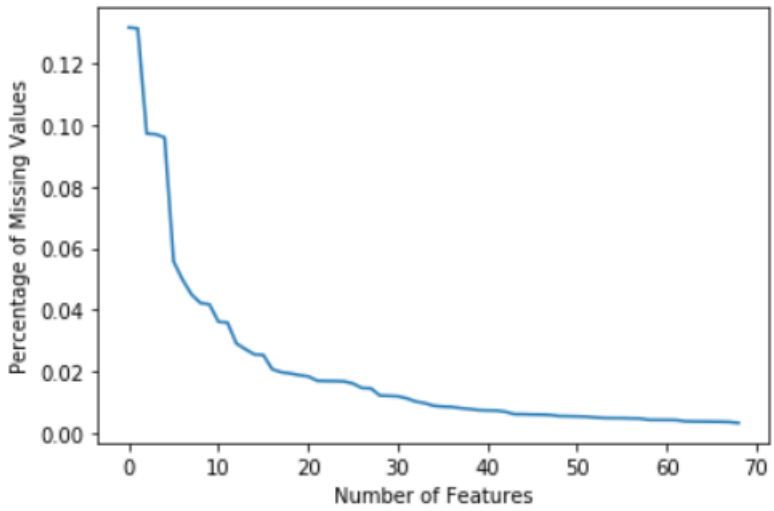

Figure 6.19: Displays the number of features with their percentage of missing values.

There are no features that have 95% missing values nor more than 50% missing values. The highest percentage of missing values is only

13%, which means that no features are removed or replaced in this step. By theory and by information gathered from previous experiments with machine learning models on real data, the missing values were dealt with by using different imputation techniques. This is explained in section 6.3.

6.6.2 Amount of Variation

The next step was to evaluate each feature by its variance. Features that have no variances will act like a constant and will have no impact on the model predictability and should be removed. Features with a very low variance are selected for further investigation. Figure 6.20 shows the variance for the each feature in the dataset.

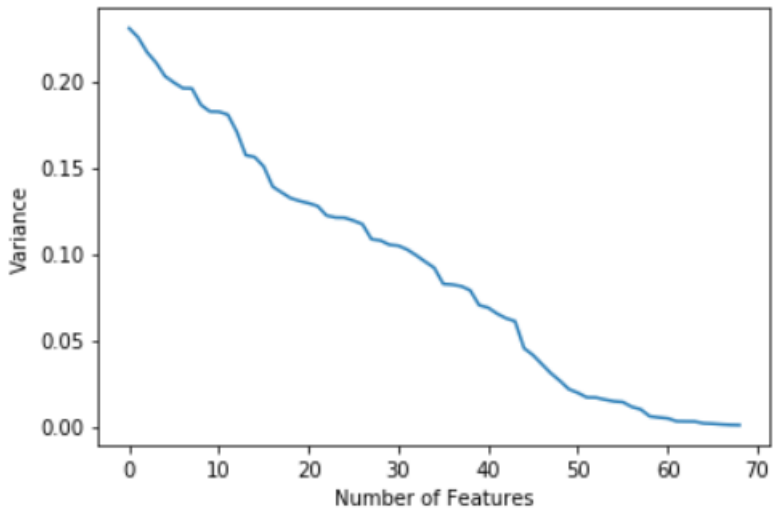

Figure 6.20: Displays the variance of 70 features in the dataset.

The feature variances range from approximately 25% to below 1%. As explained in Section 3.1.2.1.1, the goal is to remove features that do not vary much, which often means identifying the *long tail*. There is no clear indication of a long tail in terms of feature variance in this dataset, but

more of a linear trend. However, eleven features have a variance below 1%. All are selected for further investigation.

6.6.3 Correlation with Target

Before potentially removing features, whether it is due to missing values, variation or correlation, their correlation with the target variable needs to be evaluated. Even though their properties locally do not seem that significant, they can still have a great impact on the target variable. Therefore, removing these features will result in a less accurate model. Figure 6.21 shows the correlation between the entire dataset with the target variable. The features range from 29% to almost 0% correlation with the target variable. Figure 6.22 displays the correlation with the target variable among the eleven selected features from the variance step. The correlation method applied in this step is the Spearman Rank Method 3.1.2.1.2. Some of the selected features are significantly correlated with the target variable and should not be removed. Others are not heavily correlated and are removed. The selected features and their respective correlation with the target variable are displayed in Table 6.6, where the features marked in bold are the ones that were removed.

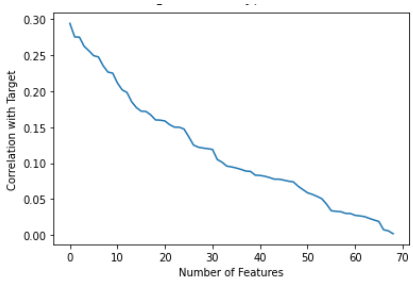

Figure 6.21: All feature's correlation with the target variable.

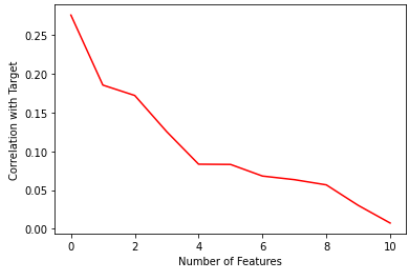

Figure 6.22: Eleven selected feature's correlation with the target variable.

Table 6.6: Showing eleven selected features and their corresponding correlation with the target variable.

	Absolute Correlation with Target Variable
Suction Gas Pressure CB2	0.275496
Suction Gas Pressure CC2	0.185307
Suction Gas Temperature CC1	0.171897
Suction Gas Pressure CA2	0.125283
Air Inlet Temperature - GTA	0.083347
Suction Gas Temperature CA1	0.083043
Suction Gas Temperature CA3	0.068078
Deg Heading	0.063430
Suction Gas Temperature CB3	0.056711
Suction Gas Temperature CC3	0.030285
Suction Gas Temperature CB1	0.007518

6.6.4 Pairwise Correlation

An important aspect of feature selection is to remove redundant information. Features that are heavily correlated provides a lot of the same information, which is unwanted with respect to computational time. The pairwise correlation technique identifies pairs of features that are heavily correlated by using the correlation matrix of the data. If the correlation coefficient between the two features is higher than a defined threshold, then one of the features are removed. In essence, this means that if the threshold is quite large, the data can be reduced without losing too much information.

For selecting features based on pairwise correlations a custom function, *Pairwise Correlation Selection* was created. The source code is located in Appendix A.1, and the pseudocode is found in Algorithm 2. The algorithm iterates through each feature of the remaining dataset after being processed in Section 6.6.1, 6.6.2 and 6.6.3. For each of the features, the correlation matrix of the dataset is calculated by using the Spearman Rank Method. All features that have a correlation coefficient greater than

0.95, with the currently evaluated feature are collected. Then each of the collected features, including the evaluated feature, are measured against the target variable. The feature with the highest correlation with the target variable is kept, while the remaining features are removed. This ensures that a minimal amount of information is lost, while the number of features is significantly reduced. After running the pairwise correlation algorithm on the dataset, 18 features were removed.

Algorithm 2 Pairwise Correlation Selection

1: **for** each feature, x, in Correlation Matrix **do**
2: Find the correlation coefficient, c, between x and each remaining feature
3: **if** $c > 95$ **then**
4: Select the current feature
5: **end if**
6: Calculate the correlation coefficient between the selected features and the target variable
7: Drop all features which don't have the highest coefficient
8: Update the Correlation Matrix
9: **end for**

6.6.5 Wrapper Methods

As explained in Section 3.1.2.1.3, wrapper methods select features by finding the subset of the original data that provides the highest model performance. The model used is the tentative model described in Section 6.2. The forward feature selection strategy is performed on two different feature arrangements. The first arrangement was by feature correlation with the target variable. Hence, the feature with the highest absolute correlation with the target variable was first inserted into the subset and run through the model. Then the feature with the second-highest absolute correlation was added to the subset and fed to the model. This process continued until all features were added into the subset and yielded the

result presented in Figure 6.23.

Figure 6.24 displays the result after using the second feature arrangement. In this case, all features have been selecting as individual datasets and run through the tentative model. The different model scores are now the metric that decides the arrangement of the features.

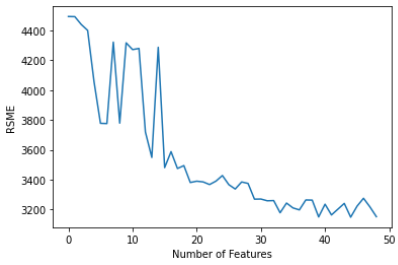

Figure 6.23: Result after performing the forward feature selection technique with a target correlation-based arrangement

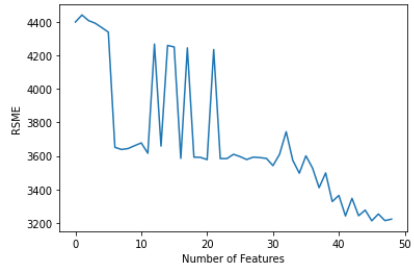

Figure 6.24: Result after performing the forward feature selection technique with a model performance-based arrangement

Reviewing the results, the arrangement after the target variable correlation yielded the smallest subset of features while remaining a high model accuracy. After completing all the feature selection steps, the original dataset was reduced from 70 to 49 features.

6.7 Feature Extraction

The original dataset contains 70 features, and the goal of the feature extraction phase is to reduce the dimensionality of the data but without losing important information. As explained in the theoretical framework, in real-world datasets, the data is not uniformly spread across all dimensions. Some features are almost constant, and others highly correlated. As a result, a majority of the instances might lie in a lower-dimensional subspace of the high-dimensional space. The Piranema dataset consists

of 70 features, and some of these data instances might be noisy data or data of little significance. A dataset containing more features than necessary might harm the learning phase of the ML algorithms. Thus, the goal is to find a simpler representation of the dataset in a lower dimension.

6.7.1 Principal Component Analysis

As explained in the theoretical framework in Section 6.7.1 the dimensionality is reduced by calculating the principal components and reducing the dimensionality of the dataset down to n dimensions, defined by the n first principal components. The *Explained Ratio* is a function in the *Scikit-Learn* library which indicates the proportion of variance in the original dataset that is preserved in the reduced dataset of n dimensions. Figure 6.25 is the cumulative explained variance plotted along the y-axis, and the number of components plotted along the x-axis. The curve quantifies how much of the 70-dimensional variance from the original Piranema dataset that is contained within the first n dimensions. Figure 6.25 indicates that approximately 95% of the variance is preserved within the first three components.

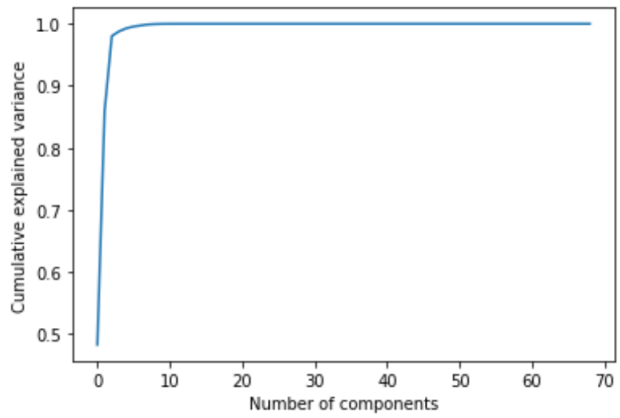

Figure 6.25: Variance contained compared to number of selected components.

According to the Principal Component Analysis, the 70-dimensional dataset can be projected into 3 dimensions, by projecting each data point along the direction which preserves the largest amount variance. In Figure 6.26 the dataset is projected into three dimensions, and preserving 95% of the variance within the dataset. Table 6.7 shows the amount of variance preserved given a different number of features or dimensions.

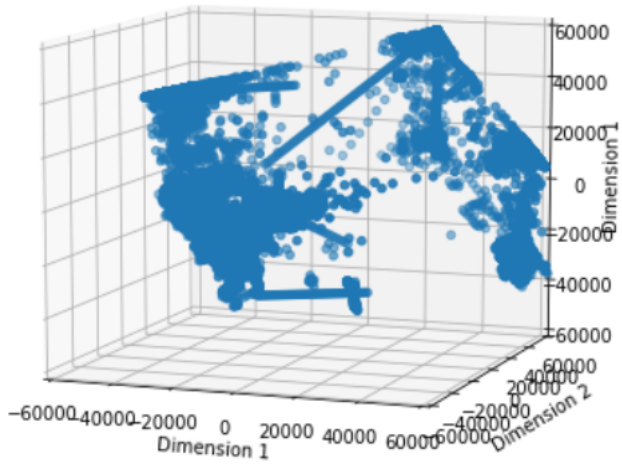

Figure 6.26: Graphical plot of the dataset projected into three dimensions.

Table 6.7: Number of dimensions required to preserve selected amount of variance.

Number of Features / Dimensions	Variance Preserved
3	0.9500
5	0.99
9	0.9999

The same recurrent neural network as described in Section 6.2 is trained on the original dataset and datasets reduced to 3, 5 and 9 dimensions,

respectively. Figure 6.27 shows a plot of the root mean square error and the standard deviation, for the predictions for the original dataset and datasets reduced down to 3, 5 and 9 dimensions. As seen, the variance between the different error rates is rather small, validating the fact that the PCA reduction algorithm preserved the variance of the original dataset. The dataset applied PCA preserving 0.9999 per cent of the variance yields the best result, reducing the dimensionality of the dataset from 70 to 9 dimensions.

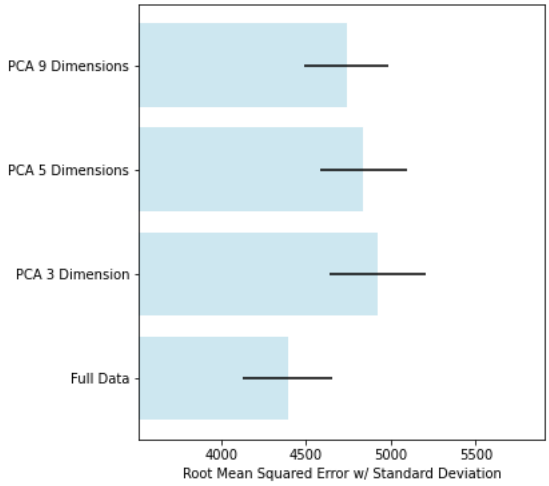

Figure 6.27: Illustrating RMSE in blue and STD as a black line for the dataset reduced with PCA and the original dataset

6.7.2 Kernel Principal Component Analysis

As explained in the theoretical framework by introducing the *kernel trick*, the principal component analysis can be used to represent a nonlinear structure that the PCA was not able to discover. The introduction of the kernel trick applied to the PCA makes it possible to perform nonlinear projection for dimensionality reduction. The kPCA is an unsupervised learning algorithm, and standard for unsupervised learning algorithm is that there is not a performance measure. As this step is a part of the pre-

processing for a regression problem, a simple *linear regression* model will be fitted to the original dataset, and by calculating the accuracy of this model the *kernel function* and the tuning of the hyperparameter γ will be made.
For this task a *grid search* is used, which is simply an exhaustive search through a specified subset of possible *kernel functions* and values for γ, to get the most accurate prediction when using a simple *linear regression* model to predict the variable of *HP Flare*. The optimal kernel function and tuning of γ for the same number of dimensions as in Table 6.7 was found, shown in Table 6.8.

Table 6.8: Choice of kernel function and tuning of the gamma parameter for KPCA corresponding to number of dimensions/components

Number of Components	Kernel Function	γ
3	RBF	0.0055
5	Cosine	0.0033
9	RBF	0.00166

Looking at the size of the Piranema dataset, the computation of a kernel principal component analysis would require a tremendous amount of random access memory (RAM). Similar to Support Vector Machines, the entire matrix for $K(x_i, x_j)$ needs to be calculated, where x_i are the sample points. Given that the Piranema dataset contains 5395880 data points, the kPCA-algorithm would have to compute $(5395880)^2$ terms and store it in matrix K. Such a calculation would require an enormous memory capacity, and even if the memory requirement was fulfilled, the matrix multiplication would take $O(n^3)$, resulting in $O(5395880^3)$ for the Piranema dataset. Hence, such an analysis is not applied to the Piranema dataset, due to the memory and computation complexity.

Chapter 7

Modelling

This chapter outlines the configurations of the models used to predict the future flare rate, *HP Flare*, on the Piranema vessel. A predictor for *HP Flare* can be interpreted as an indicator for compression efficiency, as extensive flaring is related to low efficiency. Thus, *HP Flare* can be used as an indicator for a potential *PSD-trip*, since a low compression efficiency can result in such a trip. The literature affirms that the state-of-the-art model in the ML-domain for time-series predictions are recurrent neural networks. Thus, three types of RNN are created. The first model created was a Simple RNN, which is the most basic version of such a network. Furthermore, a GRU network and an LSTM network are created. These networks have more advanced configurations than the first model. For comparison, two additional models are created, namely a multi-layer perceptron and an ARIMA model. The multi-layer perceptron functions as a baseline for comparing recurrent neural networks with non-recurrent neural networks. The ARIMA model is implemented in order to compare the performance of neural networks with traditional statistical methods. The two baseline models create the opportunity of assessing the potential value of increasing the complexity of the models. Finally, to combine the advantages of machine learning and statistical methods, a hybrid model was created. All implemented code and the

used dataframe is to be found on Github. [1].

7.1 Time-Series Forecasting

The models created are measured in their ability to predict the value of *HP Flare* one hour into the future. Such a prediction is done without any knowledge of future x variables and based simply on the historical values. This means that the models are trained on a dataset with *lagged* variables. Given a target instance y_t, then a lagged target instance y_{t+k} is obtained by shifting the data by k rows. This ensures that the inputs at time t corresponds to the output at time $t+k$. This concept is illustrated in Table 7.1. Regarding the models in this thesis, the lag constant k is set to 1. Due to the complexity of the targeted system, this thesis focuses on configuring models that yield accurate one-step-ahead predictions. A natural adaption for future work will be to extend the prediction horizon further. The error metric used to estimate the models' ability to make predictions is the *root mean square error*, as explained in Section 3.41.

Table 7.1: Example of a dataset with a lag constants k = [0,1,2].

x^1_t	x^1_t	x^3_t	y_t	y_{t+1}	y_{t+2}
50	2	10	1	2	3
25	32	1	2	3	4
2	45	78	3	4	5
22	3	67	4	5	-
12	66	45	5	-	-

7.2 Libraries

All models are written using Google Colab notebooks, which is an online cloud-based environment designed to train machine learning and

[1] `https://github.com/kolbjornf/Master-Thesis`

deep learning models. It leverages the computational power of Google hardware with high performing GPUs[2]. The programming language is *Python* which yields the *Numpy* library used for multidimensional matrix operations. Additional libraries like *Pandas* and *Matplotlib* are used for data visualization and inspection. *Tensorflow* is an open-source machine learning platform provided by Google. It contains all the necessary tools and function to build a state-of-the-art machine learning model.

7.3 Recurrent Neural Network

Due to the size of the data, all neural networks are trained using *mini-batch gradient descent*, elaborated in chapter 3.3.2.6. This ensures a more efficient training process by performing back-propagation on smaller subsets, rather than on the entire dataset. Subsets or batches are created with a custom made batch generator which preserves the time series aspect of the data. It was imperative to create a function that automatically yielded batches that were separated chronologically in order to provide valid predictions. The batch generator source code is located in appendix A.3. The data is divided into training, validation and test sets which are all divided into smaller batches. Similar to the tentative RNN model used during preprocessing in chapter 6.2, the nested cross validation 3.2.2.2.1 is used to ensure valid results and all the neural networks are following the same approach presented in Algorithm 1.

7.3.1 Architecture & Hyperparameters

The Simple RNN, GRU and LSTM follow the same architecture with some slight modifications. Multiple architecture designs and hyperparameter choices have been tested and evaluated, where Table 7.2 summarizes the setup of hyperparameters that yielded the highest performance for each

[2]Graphics Processing Units

model. Table 7.3 displays the network architecture that is equal for all the RNNs.

Table 7.2: Summary of the hyperparameter setups for the highest performing versions of Simple RNN, GRU and LSTM.

	Simple RNN	**GRU**	**LSTM**
Error Metric	Root Mean Square Error	Root Mean Square Error	Root Mean Square Error
Optimizer	Root Mean Square Propagation	Adam	Adam
Learning Rate	0.001	0.001	0.001
Regularization	Dropout	Dropout	Dropout
Batch Size	740	370	370
Lookback	100	200	200
Callback	Checkpoint & Early Stopping	Checkpoint & Early Stopping	Checkpoint & Early Stopping

Table 7.3: Summery of the corresponding architecture of the Simple RNN, GRU and LSTM.

	Input Layer	**Hidden Layer 1**	**Hidden Layer 2**	**Output Layer**
Number of Units	50	50	1	1
Activation Function	Tangent Hyperbolic	Tangent Hyperbolic	Linear	Leaky ReLU
Dropout	0.3	0.3	-	-

- **Optimizer**: Amongst many available optimizers, the root mean square propagation and the Adam yielded the highest performance.

- **Regularization**: A dropout regularization technique has been applied to the input and first hidden layer of the network. The dropout probability is set to 0.3, which means that every node in the layer has a 30% chance of being deactivated. Naturally, this means that the network must find alternative paths during training. Implementing dropout in neural networks are common practice, which often reduces the chance of overfitting (Srivastava et al. 2014).

- **Lookback**: Refers to the number of previous samples taken into account when performing predictions. A lookback value of 200 with a sampling rate of one hour means that the model leverages information from the last 8 days.

- **Callback**: Two callback functions are applied to each network. The *Checkpoint* function ensures that the set of feature weights with the highest performance on the validation set during training are saved and used when performing predictions on the test set. The *Early Stopping* function stops the training when the performance on the validation set is no longer improving. Initially, the number of epochs is set to 50. If three consecutive epochs with no improvement on the validation set occur, then the training is stopped.

7.3.2 Multilayer perceptron

The multilayer perceptron consists of an input layer, one hidden layer and an output layer. The input and hidden layers have a linear activation function which effectively makes it equivalent to a linear regression model. Similar to the RNNs, the output layer has the *Leaky ReLU* activation function, which provides a linear transformation of positive values, whereas negative values are transformed close to zero. The type of network is not recurrent, hence the lookback value is 0. The error metric, optimizer and learning rate are similar to the *Simple RNN*, and dropout and callback regularization techniques are applied.

7.4 ARIMA

The ARIMA model is constructed using the *statsmodels* library, which is a python module with functions designed for statistical modelling. The ARIMA function only uses the target variable *HP Flare* when performing its future predictions. As explained in Section 3.4.1, the ARIMA model is a combination of an autoregressive model, a moving average model and an integration term. The model parameters are p, d and q.

- p, refers to the autoregression (AR) term, which is the number of lagged samples used in each prediction.

- d, refers to the integrated (I) term, which is the number of times samples are differenced.
- q, refers to the moving average (MA) term, which is the size of the moving average window.

7.5 Hybrid Model

Reviewing the performance of the recurrent neural networks compared to the ARIMA model, we observed that RNNs were able to provide accurate predictions for abnormal values, but failed to capture small changes in the target variable. On the other hand, the ARIMA model was able to consistently recognize small changes in the target variable but missed greatly when abnormal values occurred. Hence, we propose a final model, combining the highest performing recurrent neural network, the LSTM, and the ARIMA model in order to merge the strengths of both individual models. The hypothesis was that the hybrid model would be able to predict the small changes in the target value during normal operation, as well as to detect the occurrence of abnormal behaviour. The hybrid model executes the ARIMA model on each batch of the data. The predicted values from the ARIMA model is then transformed into a new feature and inserted into the data batches. The first batch is then fed into the LSTM, where a forward and backward pass is completed and weights updated. The next batch is then inserted into the updated LSTM, and this process continues until all the batches are processed, which is equivalent to one epoch. Predictions on the validation set are then performed before the next epoch begins as illustrated in Figure 7.1. The pseudocode is found in Algorithm 3.

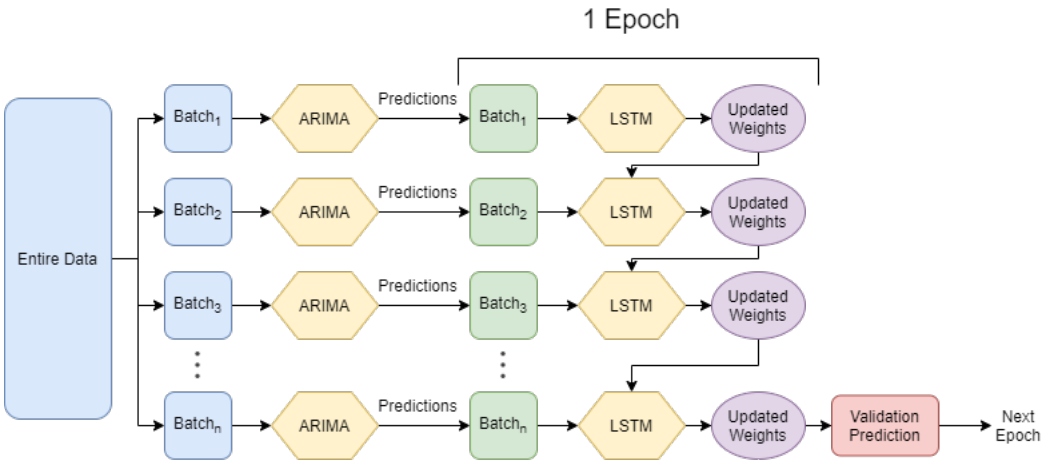

Figure 7.1: Illustration of the proposed hybrid model combining LSTM and ARIMA.

Algorithm 3 Hybrid Model

1: Split the dataset into training, validation and test set
2: Split the training set into n batches
3: **for** each batch x **do**
4: Perform predictions, Y with the ARIMA model
5: Transform x by inserting Y, yielding new batch x'
6: **end for**
7: **for** each epoch in the LSTM model **do**
8: **for** each new batch x' **do**
9: Train the LSTM model
10: Update weights
11: **end for**
12: Perform predictions on the validation set
13: **end for**
14: Perform predictions on the test set

Chapter 8

Results & Analysis

This chapter outlines the results of the highest performing configurations of each model presented in Chapter 7. The chapter presents and discusses the results of the three recurrent neural networks and the two baseline models. In addition, three different datasets are evaluated to assess the effect of data preprocessing. Model performance is defined as its ability to generalize well over unseen data. Translated to this particular problem, the performance of a model refers to the ability to accurately predict the future health condition of the compression train. To ensure valid and comparable results, each model follows the same approach. The training set provides samples which are used to adjust the feature weights according to the correct target variable. Obtaining a high performance on the training set is not difficult, but it is not desirable if it reduces the performance on unseen data. In this case, the model only memorizes the correct answers on limited information, rather than inferring patterns and relationships that are applicable to new data. The validation set is used to find the set of feature weights, obtained during training, that yields the highest performance on unseen data. This effectively means that it reduces the performance on the training set while optimizing the models' ability to generalize over new data. The final evaluation is determined on the test set, which reflects the model's actual

performance. All models are evaluated using the root mean square error metric.

8.1 Dataset Comparison

The importance of feature scaling and imputation is evaluated in Chapter 6. After evaluating every technique with a simple RNN-model, the min-max scaling and linear interpolation proved to be the best techniques for the dataset. This section assesses the effects of data reduction, more specific, feature extraction and feature selection. As explained in Chapter 6, reducing the number of features also reduces the complexity of the model. A reduction in dimensionality results in a more efficient and potentially more accurate model. The data reduction step yielded two datasets. The first consisted of 9 features after applying the principal component analysis. By reducing the dataset from 70 to 9 features 0.9999 per cent of the variance is still preserved in the dataset. The second dataset consisted of 49 features after the multiple selection methods described in Section 6.6. Both datasets are fed to three different RNNs alongside the original dataset for comparison. The result is displayed in Table 8.1, indicating the RMSE for each dataset.

Table 8.1: Results summarizing the effects of data preprocessing reflected in the root mean square error.

	RMSE for Entire Dataset		**RMSE for Feature Extraction (PCA)**		**RMSE for Feature Selection**	
	Training Set	**Test Set**	**Training Set**	**Test Set**	**Training Set**	**Test Set**
Simple RNN	4042	3359	4284	3740	4002	3365
GRU	4451	3358	4407	3686	4404	3288
LSTM	4328	3344	4476	3441	4146	3224

Compared to the models fed the original dataset, the models fed the dataset where the feature selection algorithm is applied are consistently performing better both in the training and test phase. One exception is that the simple RNN model performed slightly better when fed the original test set. On average, models with the feature selection dataset have

increased their performance of 1.82% on the test set and 2.08% on the training set compared to the models with the original dataset. Regarding the dataset with the feature extraction methods, the average performance decreased by 8.02% on the test set and 2.71% on the training set. In conclusion, the dataset that yielded the highest performance was the feature selection dataset and is the dataset used further for comparing the performance of the different models.

8.2 Model Comparison

The two baseline models, the ARIMA and the multilayer perceptron, yield the benchmark performances in order to assess the advantages or disadvantages of increasing the model complexity. To make the results comparable, every model follows the same approach and assumptions. Except for the ARIMA, which is a univariate model and the hybrid model, all models are fed the same dataset. Every model has the possibility of running 50 epochs, although training is stopped when 3 consecutive epochs show no improvement on the validation set. Each model is run 10 times, where the average performance reflects the final result. Nested cross-validation is applied to each model. Such a performance indicator ensures an even more valid representation of how well each model generalizes on unseen data. Table 8.2 provides the final result and the compared performances.

Table 8.2: Performance of each implemented model and summary of performance compared to baseline models.

	RMSE Score			**Baseline Comparison, %**	
	Training Set	**Validation Set**	**Test Set**	**ARIMA**	**Multilayer Perceptron**
ARIMA	4999	4444	3893	-	-6.78
Multilayer Perceptron	4541	4385	3629	6.78	-
Simple RNN	4002	3408	3365	13.56	7.27
GRU	4404	3394	3288	15.54	9.39
LSTM	4146	3327	3224	17.18	11.16
Hybrid	3625	3417	2822	27.51	22.23

All the recurrent neural networks have a higher performance than both of the baseline models. In the case of the ARIMA benchmark, the results state that neural networks, both recurrent and non-recurrent, have at least 6.78% increased performance. This illustrates the effect of creating more complex models. In the case of the second benchmark, all the recurrent neural networks are outperforming the multi-layer perceptron by at least 7.27%. That recurrent neural networks outperform the multi-layer perceptron model emphasizes the importance of leveraging past information and considering the time-series property of the dataset. Amongst the recurrent neural networks, the LSTM had the highest performance with an improvement of 17.18% compared to the ARIMA, and 11.16% compared to the multi-layer perceptron.

The performance of the hybrid model confirms our hypothesis of combining the strengths of classic statistical and machine learning methods. The average performance increased with 27.51% compared to the ARIMA and 22.23% compared to the multi-layer perceptron. In addition, the performance increased with 12.45% compared to the highest performing recurrent neural network, the LSTM. Figure 8.1 and 8.2 displays the predicted against the correct target variable for the LSTM and hybrid model respectively. The hybrid model is far more capable of predicting values in the range of 1000 to 15000 compared to the LSTM. Similar to the LSTM, the hybrid model remains robust enough to detect even larger target variables (>20000). Figure 8.3 and 8.4 illustrates the benefits of integrating the ARIMA into the hybrid model. The LSTM alone is, to some extent, able to recognize large spikes, but does not infer information regarding smaller values. On the other hand, the hybrid model is even better in its predictions of large spikes, as well as able to capture the changes in smaller values.

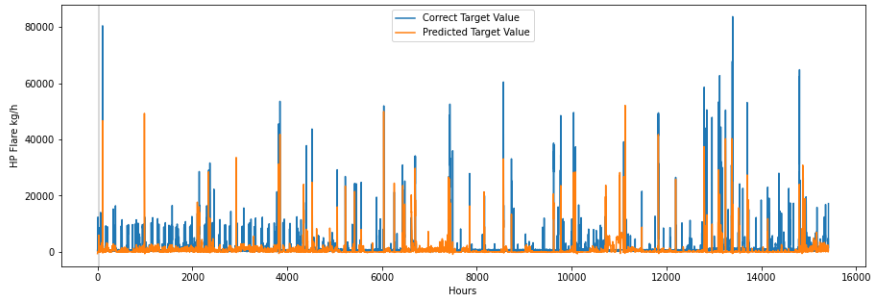

Figure 8.1: Illustration of predicted compared to correct target values of the LSTM model.

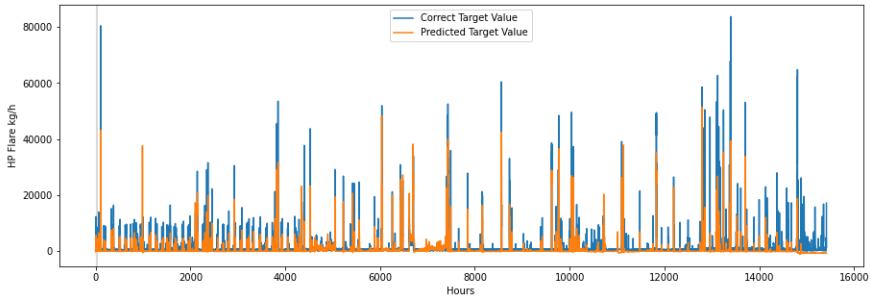

Figure 8.2: Illustration of predicted compared to correct target values of the hybrid model.

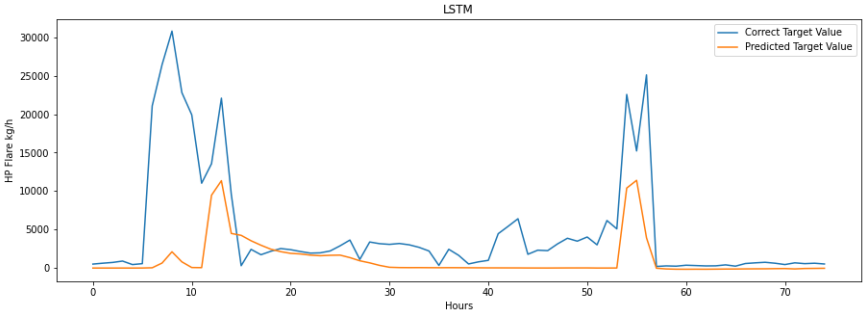

Figure 8.3: 70 sample illustration showing the performance of the LSTM model on smaller changes in the target variable.

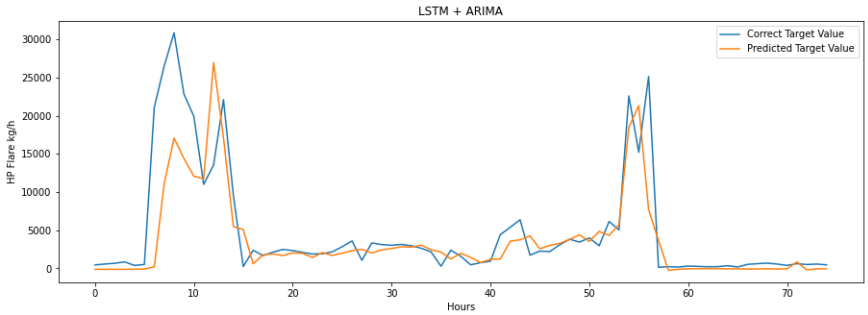

Figure 8.4: 70 sample illustration showing the performance of the hybrid model on smaller changes in the target variable.

Figure 8.5 illustrates the loss and required number of epochs to reach convergence of each model. The hybrid model outperforms the remaining models in terms of loss and reaches convergence later than the LSTM and GRU. The Simple RNN and the multi-layer perceptron are the worst-performing models, though the Simple RNN does converge, while the multi-layer perceptron shows no indication of convergence.

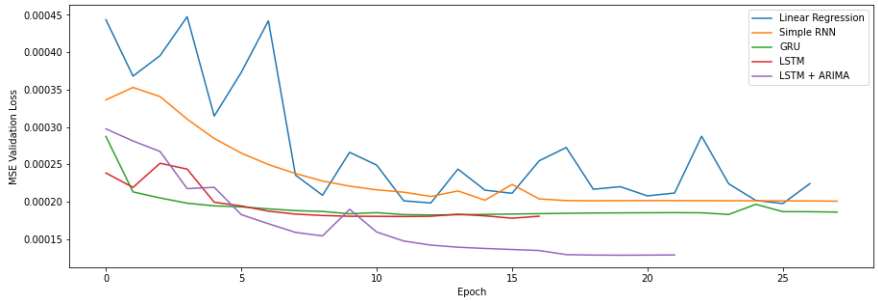

Figure 8.5: Illustration of the training loss per epoch for each implemented model.

Chapter 9

Discussion

This chapter includes a comprehensive reflection corresponding to the objectives introduced in Section 1.3. Section 9.1 highlights the importance of conducting qualitative analysis in order to gain essential domain knowledge of the problem of interest before going into the preprocessing phase. Reflections regarding the choice of the target variable and the importance of a well-describing dependent variable are found in Section 9.2. Section 9.3 discusses the necessity of data preprocessing, particularly important when the data is inconsistent and unreliable. The potential value of increasing the complexity of the model by introducing machine- and deep learning is reviewed in Section 9.4. Section 9.5 discusses on the benefits and challenges of combining multiple models. Conclusively, reflections concerning how Teekay Offshore Production can benefit from ensuing the digital transformation is found in Section 9.6.

9.1 Profound Domain Knowledge

Even though machine learning allows unbiased modelling of complex problems, incorporating domain knowledge is always beneficial. Performing the qualitative analysis in Section 5.2 gave a more in-depth per-

spicacity of the components on the FPSO, as the petroleum industry was a new domain for us. Conducting analysis such as the FMEA and FTD gave profound insight, even though all findings are not directly applicable to the models created. For instance, the shut-down of 2 out of 3 OPRA turbines will also cause a PSD-trip, and the same goes for the SOLAR turbines. If these failure times had been traceable in the IP21-system, this information could also have been incorporated in the models. After conducting the FMEA analysis, it became clear that the main reason for high-pressure flaring was low compression efficiency. Thus, we proposed *HP Flare* as the target variable for the predictive models. Truly understanding the data and the problem of interest is crucial. If there exists no relationship between the features and the target, no model would be able to provide any predictions.

9.2 Choice of Target Variable

As previously stated, in order to avoid PSD trips by obtaining a robust future health predictor of the compression train, an accurate performance indicator is required. Determining a target variable best describing the performance of the compression train was a complex task, and this thesis introduced several reasonable dependent variables.

Primarily, using ΔW as target variable, describing the difference between the expected compression efficiency W_E and the actual efficiency W_A, could potentially provide a strong indication of the compression performance. However, neither of the values, W_E or W_A, are monitored and logged in the SCADA-system. Thus, both values had to be derived from an official performance document and a theoretical estimate of the physical properties of the compressor and the processed fluid, respectively. Such an analysis would require broad domain expertise and combine several engineering disciplines. Process engineers at Teekay did some calculations on this matter a few years ago but were not able to

determine whether the derived calculations explained the performance of the compressors accurately. When applying theory in practice, many assumptions have to be made. Such assumptions added a lot of uncertainty to the actual performance variable, W_A, and it was hard to determine whether the calculations gave an accurate estimate of the performance. Deriving such a performance measure with high accuracy was attempted but proved to be far too complex. Due to the uncertainty surrounding the approach and calculations, we decided to discard this potential target variable.

As an alternative target variable, the amount of high pressure flaring, *HP Flare* was proposed. A high value of flaring is an indication of low compression efficiency, as the gas has to be burnt off to release pressure in the process plant. However, flaring is also a result of the process plant starting up or shutting down. Hence, the multiple reasons for flaring create a bias in the target variable. However, the highest peaks of flare, ranging from [50 000 - 200 000] is most likely a result of low compression efficiency, and being able to predict such peaks indicates the compressors health state. Although this target variable introduces a level of uncertainty, we evaluated it to be a more robust performance indicator since it is logged in the SCADA-system. Compared to the ΔW, choosing *HP Flare* would remove the uncertainty concerning data validity, whereas the ΔW calculations would not necessarily reflect the real world. That being said, comprehensive testing and calculations of ΔW performed by domain experts would probably yield a more robust indicator of the compressor performance. As mentioned several times, working on real-world applications is a multidisciplinary task, and it would be beneficial to incorporate an interdisciplinary strategy to further improve the work completed in this thesis.

9.3 Preprocessing

When dealing with real-life data, both the literature and theory regarding preprocessing agree on the fact that it is a time consuming and essential phase. Gan, Ma, and J. Wu (2007) stated in their research in 2007 that preprocessing consumed more than 80 per cent of the time of a modelling phase, and a decade later, Ramırez-Gallego et al. (2017) stated the same. Computer science is one of the most swiftly developing fields of discovery. Still, the researchers stated the same about the time consumption of preprocessing, even a decade apart. As real-life datasets will vary from problem domain and industry, and differ in quality, dimension and structure, there exists no state-of-the-art pipeline of preprocessing techniques to transform a raw dataset into a high-quality dataset. The most applicable techniques have to be determined based on the raw data available, as well as for the problem of interest. The data obtained from Teekay Offshore Production was of an extreme dimension, both in terms of features, samples and cardinality. To be able to create a dataset without noise, inconsistency, missing values and outliers, an extensive grid-search through different preprocessing techniques was executed. The quality of a preprocessing technique was validated using a simple RNN network, in order to determine which of the techniques improved the quality of the predictor. As stated in the literature, this was a time-consuming part of the research, as there exist multiple approaches for imputation, scaling, detecting outliers, selecting the most relevant features and for reducing dimensionality. The techniques that gave the best result for the problem of interest are by no means applicable for another real-life dataset, making it hard to determine an overall recipe to success for preprocessing. Even though researches state that preprocessing is time-consuming, Naduvil-Vadukootu, Angryk, and Riley (2017), Elsworth and Güttel (2020) amongst others emphasizes the importance of preprocessing in their research, as this leads to a better understanding of the dataset, simpler models and more accurate predictions. Also, the

models proposed in this research achieved a higher accuracy and a more simple structure, when fed preprocessed data rather than raw data, as seen in Table 8.1. One can demonstrate that the quality of a predictor is only as good as the quality of the data, emphasizing the importance of such a premodelling phase.

9.4 Modelling

According to the literary findings in Chapter 4, machine and deep learning models tend to outperform classical methods, due to their ability to derive complex relationships from non-linear and non-parametric data. The increase of available data combined with efficient models and hardware makes machine learning models capable of solving a more comprehensive range of intricate problems. However, due to the models' complexity, the creator is left with little to no understanding of how the model derives the relationships. Machine Learning models are also highly dependent on the quality of the data. Ideally, the available data should cover all the areas of the problem domain so that the model will generalize well on new data. Nevertheless, this is often not the case, especially in real-world applications. In our research, the main objective was to create an accurate health-indicator for the compression train onboard the Piranema vessel. In order to be accurate and valuable, the model must be able to detect the degradation of the component's state of health. Since ML models are highly dependent on the data available, the data must consist of observations from a degraded state, as well as data from a normal condition. In practice, this means running a component further than the recommended safety criteria in order to gather data from a degraded state. Such a scenario is both expensive and potentially dangerous.

The literature also emphasizes the value of implementing simpler models. Such models might not provide a result as accurate as an ML model,

but presents intuitive information regarding the causes of change in the target variable. For instance, by observing the direction and magnitude of the coefficients of a linear regression model, one can derive the features with the highest impact on the target variable. Performing a similar operation on deep neural networks yields a broad set of feature weights which are difficult to trace back to the original features.

In this research, a great emphasis is put on testing out multiple different models. Cross-validation and averaging over multiple executions ensure valid results that truly reflect its performance. Baseline models were also created to assess the benefits of increasing model complexity. The results in Chapter 8, show that the recurrent neural networks outperform both baseline models. The results display the advantages of choosing models designed for non-linear time-series. In addition, , looking at the convergence in Figure 8.5 in Section 8.2, the networks designed to overcome the common challenge of vanishing gradient are all reaching convergence faster, at least in terms of epochs. Such a property is beneficial when networks are required to compute predictions in real-time. Then again, the computation time per epoch for both the GRU and LSTM networks are higher than for the Simple RNN. Both GRU and LSTM incorporate more information regarding the past, which naturally increases the models' complexity. In terms of scalability, the tradeoff between accuracy and speed is crucial. Prediction accuracy has been the main performance measure for the created models. However, with today's rate of generated data and safety requirements, it might be desirable to sacrifice some accuracy for speed. For instance, a model that provides warnings after hazardous events occur has no value.

9.5 Hybrid Model

The different models created in this research tended to capture different changes in the target variable. Such an observation was the reason for

the proposition for the hybrid model, combining the strengths of classic statistics and state-of-the-art machine learning models. As previously mentioned in Chapter 8, the LSTM model was able to detect large spikes in the target variable but did not perform well on smaller changes under normal condition. On the other hand, the ARIMA model generalized well on smaller changes in the target variable but was not able to detect abnormal behaviour. The literature emphasizes the strengths of combining multiple models to create a more robust predictor. It is challenging in practice to determine whether a time-series is generated from a linear or a non-linear process. Never the less, real-world time-series are rarely pure linear or non-linear. In this thesis, we propose a hybrid-model, combing the strength of the linear ARIMA model and the non-linear LSTM-RNN network. These models are used jointly aiming to capture the different form of relations in the time-series data provided from Teekay. Such a hybrid model takes advantage of the unique strength of ARIMA and RNN in linear and non-linear modelling. The result shows that the hybrid model outperforms the highest-scoring RNN by 12.45%. More interestingly, inspecting figures 8.3 and 8.4 show that the hybrid model did inherit the advantages of both models. The hybrid model was able to provide accurate predictions during normal and abnormal behaviour. It especially improved the accuracy of detecting abnormal behaviour compared to the LSTM. The result of the hybrid-model initiates the proposal of incorporating additional models.

9.6 Gain for Teekay Offshore Production

Teekay monitors and stores a large amount of data. However, the data is not designed such that it is easily applicable in advanced machine learning analysis. Achieving the end goal of a fully automated condition monitoring system requires a restructuring of the database. Unstructured data, such as free text is not compatible with supervised machine learn-

ing models and should be transformed. Hence, parts of the maintenance strategy, such as reporting maintenance events and logging component health after testing or inspection, must be stored as structured data. Today the sensory data is stored in IP21, whereas the corresponding event data is stored in STAR IPS. Combining these two databases and creating the connection between the sensor and event data would yield data more suitable for advanced analysis. First, the occurrence of missing sensory data could be explained by the corresponding event. When performing analysis, it is desirable to know if the missing data is due to a planned event such as component maintenance, or unplanned events such as sensor malfunction or abnormal component behaviour. Secondly, identifying these events creates an opportunity for a profound analysis of specific scenarios. For instance, the models implemented in this thesis could be further specialized to obtain information of component behaviour prior to selected events. This increases the understanding and insight of the data, such that the decision-makers can apply actions with a higher level of certainty. Furthermore, it would be easier to expand the proposed model in this thesis to other components on the vessel. A reconstruction of the database will increase the complexity of the available data, which ML models are far more capable of accurately interpret, as the results of this thesis substantiates.

Chapter 10

Conclusion

The use of machine learning in condition-based monitoring is a field of study with great potential. Rapidly obtaining valuable information is beneficial in terms of competitive advantage, safety and sustainability. The increasing amount of available data creates the opportunity for companies to optimize their business strategies with precise predictive abilities. This thesis has explored the potential benefits of introducing machine learning in Teekay Offshore Production's maintenance strategy by creating a predictive health-indicator. We conducted a comprehensive review of the theory and literature regarding different approaches for implementing ML-models and preprocessing data when applied to the real-world domain. Creating an accurate health-indicator requires high-quality data and efficient data-driven models. Hence, the primary concern of this thesis has been to ensure high-quality data, as the obtained dataset was unstructured, inconsistent and noisy. The achievement of this goal required extensive experimental research in the preprocessing domain, in order to obtain the highest possible data quality. As the literature states,this was a time-consuming task, as it still does not exist a state-of-the-art pipeline of preprocessing techniques. By implementing several methods and validating the transformed dataset using a predictor, we were able to select the methods most suitable for our time-series.

The importance of preprocessing is further confirmed when executing the optimized final model, which performed significantly better when fed preprocessed rather than raw data. We emphasize that these techniques can by no means ensure high-quality for another real-life dataset, as real-life data vary significantly in terms of quality, dimension and complexity. The implemented predictive models reflect state of the art within the field of ML with innovative solutions for inconsistent and sophisticated systems. Traditional time-series forecasting has been challenged by comparing classical statistical methods and recurrent neural networks. The LSTM significantly outperformed the baseline models with an increased performance of 17.18%. Seeing that the literature emphasizes the advantages of hybrid models, we propose a predictive model which combines the strengths of the classical ARIMA and the LSTM. The hybrid model's predictive performance significantly outperforms the baseline models and RNNs with an increase of 27.51% and 12.45%, respectively. As real-life data is seldom purely linear or non-linear, it is natural that a model able to detect both relations performs better.

The implemented models are proposed as the first initial integration of data-driven health-indicators. An accurate prediction of *HP Flare* can predict the future performance of the compression efficiency. This would yield useful information that the maintenance engineers can incorporate to avoid unwanted events such as a PSD trip. Reviewing the results, we argue that implementing ML in Teekay's maintenance strategy is worth the investment. However, this requires comprehensive restructuring regarding data allocation with a greater emphasis on ensuring reliable and consistent data. Having sensor data logged in one system, and unstructured data in another is the main obstacle Teekay needs to overcome, in order to follow the digital transformation. Several of the preprocessing steps required in this research can be replaced with a well-structured database, providing applicable data for real-time analysis. Embedding predictive strategies into the maintenance strategy yields

more information that the engineers in Teekay can utilize in their daily operations and future planning. By reconstructing the database and introducing predictive strategies, the company's fundamental values can be strengthened, by optimizing all decisions considering people, planet and potential profit

Chapter 11

Further Work

Due to time limitations, interesting adaptions for the researched problem were left for future research. This thesis focuses on the obstacles of using real-life data for time series forecasting in order to create a health indicator to support Teekays maintenance routine. The research conducted in this thesis is a first step into the world of automated decision making. By creating data-driven models; the goal is to provide Teekay with an ultimate toolbox to perform such analysis and use the data generated over the past decade to gain valuable insights to the process. To reach this goal, a lot of research, new adaptions and testing of new approaches remain.

11.1 Extensions of Problem of Research

The problem study in this thesis is a complex one. As mentioned in Section 5.1.1, Teekay engineers have, over the years, proposed several possible solutions. The approach of this thesis was to research the possibility of solving this problem by creating a machine-learning model to be used as a health indicator. Due to the time constraints, other propositions have not been tested, but using other methods to attempt to solve this problem will be an interesting area of research.

First and foremost, identifying a better target variable could potentially give a result with higher accuracy for the time-series predictions. Using the compressor's efficiency, as discussed in Section 6.1.3.1 is a complex task, combining several engineering disciplines like chemistry, fluid dynamics and thermodynamics. A research collaboration between students in these fields, as well as computer engineering students, would have been an interesting continuation of this thesis. Using expert knowledge of the physics and chemistry behind the compressor trains could potentially result in a better and more accurate target variable describing the efficiency of the compressor. By having a more accurate target variable, known in classical statistics as the dependent variable, the model can potentially discover new relations and patterns between the dependent and independent variables.

Secondly, this research only explores the possibility of creating a health-indicator using data-driven models, both by using machine-learning and the classical statistical model ARIMA. As mentioned, combining students with multiple disciplines would add an extra dimension to this research. A clear understanding of the compressor trains efficiency, in combination with traditional statistical tools, would tackle the uncertainties and high variability in the system behaviour. Instead of building a time-dependent model, an interesting field of research would be to try to create an incremental degradation model, and model the degradation level between t_k and t_j. Then again, such a model requires that the probability density functions for the variables can be estimated using classic statistical tools. Given this requirement, one could try to model the degradation by a *Levy* or *Gamma* process etc., with a predefined threshold.

Thirdly, after creating an accurate health-indicator for the efficiency of the compression system, an interesting extension of this thesis will be to optimize the preventive maintenance routine. Such a routine could balance the risk of failure and the cost of maintenance, based on the indications from the data-driven model.

11.2 Future Collaborations with Teekay

This thesis is the first step into exploring the possibility of a fully-automated decision system for Teekay Offshore Solution. Exploring such opportunities is critical to maintaining relevance and competitiveness in a world more inclined to data-driven solutions. Teekay would greatly benefit from collaborating with students in the future and exploring other fields of study where data automation would be an interesting approach.

STAR IPS, as described in Section 2.3.4.1 is the program used for planning and administration of the maintenance work. After having executed both corrective and preventive maintenance, the engineer has to fill out a form in the program, describing the work details and the current health-status for the components. Everything is written in free-text, with no use of a predefined template. This issue limits the opportunities of taking advantage of historical knowledge because the status of a component is reported in various ways. The components sensor values are logged in the IP 21 system, while failure times are reported in STAR IPS. Applying deep learning to unstructured text might be a solution for extracting relevant information and organizing such information. This idea is a complex task but would be of great advantage for the company. A clear overview of the failure times of every component would simplify the work of creating accurate health-indicator models.

Appendix A

Source Code

A.1 Pairwise Correlation

```
#---------- Pairwise Correlation
 ↪  ----------------------------------------

pc_data = data.copy()
print('Original dataset shape: {}'.format(pc_data.shape))
if True: # If True, drop columns with low variance
  pc_data = pc_data.drop(features_low_variance_corr, axis
   ↪  = 1)
print('Dataset shape after dropping features:
 ↪  {}'.format(pc_data.shape))
pc_data_index = pc_data.columns

#Calculates the correlation matrix
pc_corr_matrix = pc_data.corr(method = 'spearman')
pc_corr_matrix_target = abs(pd.concat([pc_data, target],
 ↪  axis = 1, sort = False)
    .corr(method = 'spearman').iloc[-1])
```

```
temp = pc_corr_matrix_target
corrMatrix_final = pc_corr_matrix

#Method that finds features that are correlated above the
 ↪  threshold,
#then find the feature that are most correlated with the
 ↪  target variable,
#delete the remaining features.
for col in corrMatrix_final:
    if col in corrMatrix_final.keys():
        thisCol = []
        thisVars = []
        for i in range(len(corrMatrix_final)):

            if abs(corrMatrix_final[col][i]) == 1.0 and
             ↪  col != corrMatrix_final.keys()[i]:
                thisCorr = 0
            else:
                thisCorr = (1 if
                 ↪  abs(corrMatrix_final[col][i]) > 0.95
                 ↪  else -1) *
                 ↪  abs(temp[[temp.keys()[i]]].values)
            thisCol.append(thisCorr)
            thisVars.append(corrMatrix_final.keys()[i])

        mask = np.ones(len(thisCol), dtype = bool)

        ctDelCol = 0

        for n, j in enumerate(thisCol):
```

```
            mask[n] =  (False if ((j[0] != max(thisCol)) &
             ↪  (j[0] >= 0)) else True)

            if ((j[0] != max(thisCol)) & (j[0] >= 0)):
                corrMatrix_final.pop('%s' %thisVars[n])
                temp.pop('%s' %thisVars[n])
                pc_data.pop('%s' %thisVars[n])
                ctDelCol += 1
        corrMatrix_final = corrMatrix_final[mask]

#Find the correlation with target variable for the
 ↪  remaining features
pc_data = pd.concat([pc_data, target],axis = 1, sort =
 ↪  False)
print('Dataset shape after pairwise:
 ↪  {}'.format(pc_data.shape))
pc_remain_corr_matrix = pc_data.corr(method = 'spearman')

if True:
  pc_remain_corr_matrix =
   ↪  abs(pc_remain_corr_matrix.iloc[-1])
else:
  pc_remain_corr_matrix = pc_remain_corr_matrix.iloc[-1]

pc_remain_corr_matrix =
 ↪  pc_remain_corr_matrix.sort_values(ascending =
 ↪  False).iloc[1:]
plt.plot(np.arange(len(pc_remain_corr_matrix)),
 ↪  pc_remain_corr_matrix)
plt.xlabel('Number of Features')
plt.ylabel('Correlation with Target')
```

```
print(pc_remain_corr_matrix.index)
```

A.2 Tentative Model

```
#----------- Tentative Model --------------------

split = [0.5,0.6,0.7,0.8,0.9]
split = [0.8]
def run_model(data):
  final_mse = []
  for temp_split in split:
    x_data = data.values
    y_data = df_targets.values.reshape(-1,1)
    if False:# True if evaluating each feature seperately,
     ↪  otherwise False
      x_data = x_data.reshape(-1,1)

    #Split dataset
    num_data = len(x_data)
    train_split = temp_split
    num_train = int(train_split * num_data)
    num_test = num_data - num_train

    x_train = x_data[0:num_train]
    x_test = x_data[num_train:]
    y_train = y_data[0:num_train]
    y_test = y_data[num_train:]

    num_x_signals = x_data.shape[1]
    num_y_signals = y_data.shape[1]
```

```
#Scale the data
x_scaler = MinMaxScaler()
x_train_scaled = x_scaler.fit_transform(x_train)
x_test_scaled = x_scaler.transform(x_test)
y_scaler = MinMaxScaler()
y_train_scaled = y_scaler.fit_transform(y_train)
y_test_scaled = y_scaler.transform(y_test)

batch_size = 256
sequence_length = 100

#
generator = batch_generator(batch_size,
 ↪  sequence_length, num_x_signals, num_y_signals,
 ↪  num_train, x_train_scaled, y_train_scaled)

validation_data = (np.expand_dims(x_test_scaled,
 ↪  axis=0), np.expand_dims(y_test_scaled, axis=0))
model_mse = []
num_runs = 10
for i in range(num_runs):
  #Model Architecture
  model = Sequential()
  model.add(GRU(units=512,
                return_sequences=True,
                input_shape=(None, num_x_signals,)))
  model.add(Dense(num_y_signals,
   ↪  activation='sigmoid'))

  optimizer = RMSprop(lr=1e-3)
```

```
model.compile(loss=loss_mse_warmup,
 ↪  optimizer=optimizer, metrics = ['mse',
 ↪  r_square])

path_checkpoint = 'best_model'
callback_checkpoint =
 ↪  ModelCheckpoint(filepath=path_checkpoint,
 ↪  monitor='val_loss', verbose=1,
 ↪  save_weights_only=True, save_best_only=True)
callback_early_stopping =
 ↪  EarlyStopping(monitor='val_loss', patience=3,
 ↪  verbose=1)
callbacks = [callback_checkpoint,
 ↪  callback_early_stopping]


model.fit(x=generator,
          epochs=10,
          steps_per_epoch=100,
          validation_data=validation_data,
          callbacks = callbacks)
try:
  model.load_weights(path_checkpoint)
  print('Success')
except Exception as error:
  print("Error trying to load checkpoint.")
  print(error)

# Input-signals for the model.
x = np.expand_dims(x_test_scaled, axis=0)
```

```
        # Use the model to predict the output-signals.
        y_pred = model.predict(x)
        y_pred_rescaled =
         ↪  y_scaler.inverse_transform(y_pred[0])

        temp_mse = np.sqrt(np.mean(np.square(y_pred_rescaled
         ↪  - y_test)))
        temp_mse = temp_mse.item()
        model_mse.append(temp_mse)

        print('Model finished')

      print('Split finished')
      final_mse.append(np.mean(temp_mse))

    return_final_mse = np.array(final_mse)

    return return_final_mse
```

A.3 Batch Generator

```
def batch_generator_chron(batch_size, sequence_length,
 ↪  num_x_signals, num_y_signals, num_train,
 ↪  x_train_scaled, y_train_scaled):

    while True:
        x_shape = (batch_size, sequence_length,
         ↪  num_x_signals)
        x_batch = np.zeros(shape=x_shape,
         ↪  dtype=np.float16)
```

```
        y_shape = (batch_size, sequence_length,
         ↪  num_y_signals)
        y_batch = np.zeros(shape=y_shape,
         ↪  dtype=np.float16)
        idx = 0
        step_size = int(sequence_length/2)
        for i in range(batch_size-1):
            x_batch[i] =
             ↪  x_train_scaled[idx:idx+sequence_length]
            y_batch[i] =
             ↪  y_train_scaled[idx:idx+sequence_length]
            idx = idx + step_size

        yield (x_batch, y_batch)
```

Bibliography

Adebiyi, Ayodele Ariyo, Aderemi Oluyinka Adewumi, and Charles Korede Ayo (2014). "Comparison of ARIMA and artificial neural networks models for stock price prediction". In: *Journal of Applied Mathematics* 2014.

Ashour, Wesam and Saad Sunoallah (2011). "Multi density DBSCAN". In: *International Conference on Intelligent Data Engineering and Automated Learning*. Springer, pp. 446–453.

Bergmeir, Christoph and José M Benıtez (2012). "On the use of cross-validation for time series predictor evaluation". In: *Information Sciences* 191, pp. 192–213.

Bontempi, Gianluca, Souhaib Ben Taieb, and Yann-Aël Le Borgne (2012). "Machine learning strategies for time series forecasting". In: *European business intelligence summer school*. Springer, pp. 62–77.

Box, George EP et al. (2015). *Time series analysis: forecasting and control*. John Wiley & Sons.

Browne, Michael W (2000). "Cross-validation methods". In: *Journal of mathematical psychology* 44.1, pp. 108–132.

Brownlee, Jason (2016). "Supervised and unsupervised machine learning algorithms". In: *Machine Learning Mastery* 16.03.

Cao, Li-Juan and Francis Eng Hock Tay (2003). "Support vector machine with adaptive parameters in financial time series forecasting". In: *IEEE Transactions on neural networks* 14.6, pp. 1506–1518.

Duggal, AS, CN Heyl, S Ryu, et al. (2009). "Station-keeping of FPSOs In Extreme Environments". In: *Third ISOPE International Deep-Ocean*

Technology Symposium. International Society of Offshore and Polar Engineers.

Elsworth, Steven and Stefan Güttel (2020). "Time Series Forecasting Using LSTM Networks: A Symbolic Approach". In: *arXiv preprint arXiv:2003.05672*.

Emam, Eman A (2015). "GAS FLARING IN INDUSTRY: AN OVERVIEW." In: *Petroleum & coal* 57.5.

Fahimifard, SM et al. (2009). "Comparison of ANFIS, ANN, GARCH and ARIMA techniques to exchange rate forecasting". In: *Journal of Applied Sciences* 9.20, pp. 3641–3651.

Friedman, Jerome H (1997). "On bias, variance, 0/1—loss, and the curse-of-dimensionality". In: *Data mining and knowledge discovery* 1.1, pp. 55–77.

Fushiki, Tadayoshi (2011). "Estimation of prediction error by using K-fold cross-validation". In: *Statistics and Computing* 21.2, pp. 137–146.

Gan, Guojun, Chaoqun Ma, and Jianhong Wu (2007). *Data clustering: theory, algorithms, and applications*. Vol. 20. Siam.

Garcia, Salvador, Julian Luengo, and Francisco Herrera (2015). *Data preprocessing in data mining*. Springer.

Geisser, Seymour and Wesley O Johnson (2006). *Modes of parametric statistical inference*. Vol. 529. John Wiley & Sons.

Géron, Aurélien (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media.

Ghahramani, Zoubin (2003). "Unsupervised learning". In: *Summer School on Machine Learning*. Springer, pp. 72–112.

Gupta, Vijay and Ignacio E Grossmann (2011). "Offshore oilfield development planning under uncertainty and fiscal considerations". In:

Guyon, Isabelle and André Elisseeff (2003). "An introduction to variable and feature selection". In: *Journal of machine learning research* 3.Mar, pp. 1157–1182.

Han, Jiawei, Jian Pei, and Micheline Kamber (2011). *Data mining: concepts and techniques*. Elsevier.

Hasselt, Hado van (2013). "Estimating the maximum expected value: an analysis of (nested) cross validation and the maximum sample average". In: *arXiv preprint arXiv:1302.7175*.

Hecht-Nielsen, Robert (1992). "Theory of the backpropagation neural network". In: *Neural networks for perception*. Elsevier, pp. 65–93.

Hochreiter, Sepp (1998). "The vanishing gradient problem during learning recurrent neural nets and problem solutions". In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6.02, pp. 107–116.

Horn, Roger A (1990). "The hadamard product". In: *Proc. Symp. Appl. Math*. Vol. 40, pp. 87–169.

Hyndman, Rob J and George Athanasopoulos (2018). *Forecasting: principles and practice*. OTexts.

Jerez, José M et al. (2010). "Missing data imputation using statistical and machine learning methods in a real breast cancer problem". In: *Artificial intelligence in medicine* 50.2, pp. 105–115.

Jolliffe, Ian T and Jorge Cadima (2016). "Principal component analysis: a review and recent developments". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 374.2065, p. 20150202.

Karbasi, Ali Reza, Somayeh Shirzadi Laskukalayeh, and Seiad Mohammad Fahimifard (2009). *Comparison of NNARX, ANN and ARIMA Techniques to Poultry Retail Price Forecasting*. Tech. rep.

Khalid, Samina, Tehmina Khalil, and Shamila Nasreen (2014). "A survey of feature selection and feature extraction techniques in machine learning". In: *2014 Science and Information Conference*. IEEE, pp. 372–378.

Kim, Kyoung-jae (2003). "Financial time series forecasting using support vector machines". In: *Neurocomputing* 55.1-2, pp. 307–319.

Kim, Won et al. (2003). "A taxonomy of dirty data". In: *Data mining and knowledge discovery* 7.1, pp. 81–99.

Kumar, Manoj and Madhu Anand (2014). "An application of time series ARIMA forecasting model for predicting sugarcane production in India". In: *Studies in Business and Economics* 9.1, pp. 81–94.

Lewins, Jeffery (2013). *Nuclear reactor kinetics and control*. Elsevier.

Liang, P and NK Bose (1996). "Neural network fundamentals with graphs, algorithms and applications". In: *Mac Graw-Hill*.

Liu, Qiong and Ying Wu (2012). "Supervised Learning". In: *Encyclopedia of the Sciences of Learning*. Ed. by Norbert M. Seel. Boston, MA: Springer US, pp. 3243–3245. ISBN: 978-1-4419-1428-6. DOI: 10.1007/978-1-4419-1428-6_451. URL: https://doi.org/10.1007/978-1-4419-1428-6_451.

Meng, Xiao-Li, Robert Rosenthal, and Donald B Rubin (1992). "Comparing correlated correlation coefficients." In: *Psychological bulletin* 111.1, p. 172.

Mitchell, T.M. (1997). *Machine Learning*. McGraw-Hill International Editions. McGraw-Hill. ISBN: 9780071154673. URL: https://books.google.no/books?id=EoYBngEACAAJ.

Mitchell, Thomas M. (1997). *Machine Learning*. 1st ed. New York, NY, USA: McGraw-Hill, Inc. ISBN: 0070428077, 9780070428072.

Naduvil-Vadukootu, Sajitha, Rafal A Angryk, and Pete Riley (2017). "Evaluating preprocessing strategies for time series prediction using deep learning architectures". In: *The Thirtieth International Flairs Conference*.

OffShore-Energy-Today (2019). *Teekay Offshore's Piranema Spirit FPSO gets Petrobras extension*. URL: https://www.offshore-energy.biz/teekay-offshores-piranema-spirit-fpso-gets-petrobras-extension/ (visited on 05/15/2020).

Parmezan, Antonio Rafael Sabino, Vinicius MA Souza, and Gustavo EAPA Batista (2019). "Evaluation of statistical and machine learning mod-

els for time series prediction: Identifying the state-of-the-art and the best conditions for the use of each model". In: *Information Sciences* 484, pp. 302–337.

Ramırez-Gallego, Sergio et al. (2017). "A survey on data preprocessing for data stream mining: Current status and future directions". In: *Neurocomputing* 239, pp. 39–57.

Rausand, Marvin and Arnljot Høyland (2003). *System reliability theory: models, statistical methods, and applications*. Vol. 396. John Wiley & Sons.

Ravanelli, Mirco et al. (2018). "Light gated recurrent units for speech recognition". In: *IEEE Transactions on Emerging Topics in Computational Intelligence* 2.2, pp. 92–102.

Ruder, Sebastian (2016). "An overview of gradient descent optimization algorithms". In: *arXiv preprint arXiv:1609.04747*.

Seber, George AF and Alan J Lee (2012). *Linear regression analysis*. Vol. 329. John Wiley & Sons.

Srivastava, Nitish et al. (2014). "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1, pp. 1929–1958.

Tay, Francis EH and Lijuan Cao (2001). "Application of support vector machines in financial time series forecasting". In: *omega* 29.4, pp. 309–317.

Voyant, Cyril et al. (2017). "Machine learning methods for solar radiation forecasting: A review". In: *Renewable Energy* 105, pp. 569–582.

Wilcoxon, Frank (1992). "Individual comparisons by ranking methods". In: *Breakthroughs in statistics*. Springer, pp. 196–202.

Zhang, G Peter (2003). "Time series forecasting using a hybrid ARIMA and neural network model". In: *Neurocomputing* 50, pp. 159–175.

Zhang, Guoqiang, B Eddy Patuwo, and Michael Y Hu (1998). "Forecasting with artificial neural networks:: The state of the art". In: *International journal of forecasting* 14.1, pp. 35–62.

Zhang, N and WF Lu (2007). “An efficient data preprocessing method for mining customer survey data”. In: *2007 5th IEEE International Conference on Industrial Informatics*. Vol. 1. IEEE, pp. 573–578.