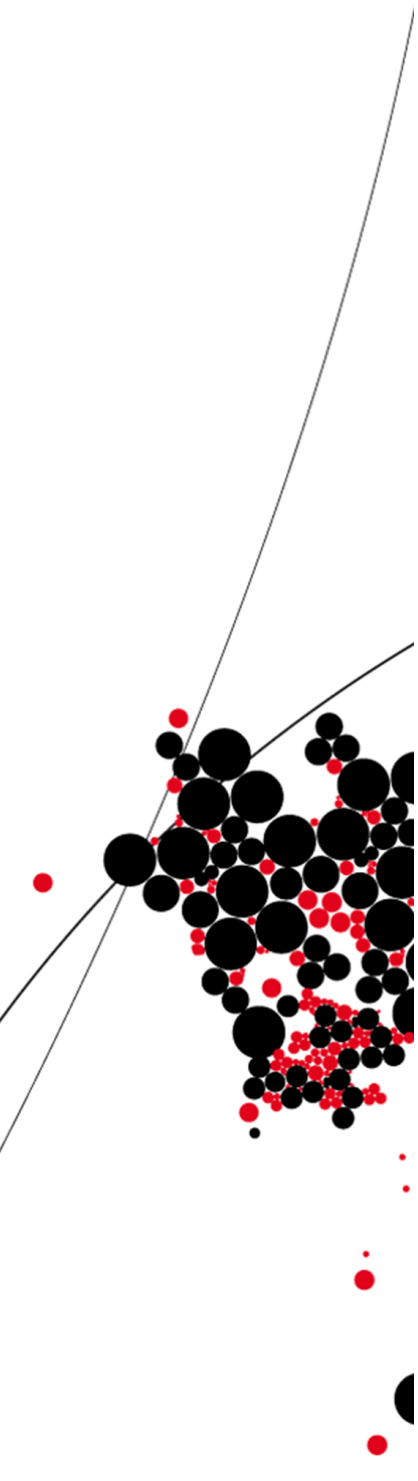
UNIVERSITY OF TWENTE.

Faculty of Industrial Engineering and Business Information Systems



Maintenance Optimization Through Remaining Useful Life Prediction

A Case Study For Damen Shipyards

Kevin J. Dekker

M.Sc. Thesis
February 2020

Supervisors:
First: dr. E. Topan
Second: dr C.G.M. Groothuis-Oudshoorn
Company: D. Mense & A. Jorritsma

Industrial Engineering & Management
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

Preface

Before you lies my thesis in which I share my findings on the topic of predictive maintenance. The research was undertaken at the Damen Shipyards headquarters in Gorinchem, The Netherlands, where I enjoyed working on my thesis. But first, I want to express my gratitude to my first supervisor Dr. Engin Topan for his detailed and useful feedback during the process. I also want to thank Dr. Karin Groothuis-Oudshoorn for her insights and feedback on my work. Furthermore, I would like to thank my company supervisors Danny Mense and Arnout Jorritsma for the weekly meetings. I learned a lot from their practical considerations which made the report more exhaustive. On a daily basis, I had contact with Jorrit de Jong, a fellow intern at Damen. Together we discussed our projects, concerns and resolutions. Therefore, I also want to thank Jorrit for our pleasant collaboration.

This thesis marks the ending of my time as student. Naturally, this comes with mixed feelings. On the one hand, I have an exciting new journey to embark on. I will enjoy to see the impact of my future work and I am sure that the skills absorbed in the University of Twente will play a core role in that. On the other hand, I will miss my time as a student because the university has felt like a fail safe environment where the focus lies on learning through experimentation. My aspiration is to keep the learning curve steep during my career and to implement the things I have learned to make my contribution. In fact, I want to encourage everyone to stay curious and keep learning. Explore and exploit!

Summary

Fast crew supply vessels are industrial assets with often high up-time requirements. This type of ship is used to transfer crew and equipment between shore and platforms at sea. If a fast crew supply vessel is not operational, this can cost an operator between €10.000 and €55.000 a day. To reduce the operational risks and minimize the costs related to keeping these assets operational, adequate maintenance policies are essential. This applies especially to assets for which maintenance execution options are restricted due to intensive use profiles, like the aforementioned vessels. Predictive maintenance offers a solution and this thesis explores how Damen can implement and benefit from it. We answer the research question:

'How can we enable Damen to predict remaining useful life of vessel components in varying circumstances with indirect condition-monitoring data and reduce costs?'

To find out, we investigate the structure of this predictive maintenance problem. This is key to understanding the problem and the modelling requirements that arise from it. We concluded that our predictive maintenance problem has a severe class imbalance, is a multivariate time-series regression, and has soft but obvious failures (alarms). We also analysed the root-cause of the problem and found that the degradation state of vessel components is unknown. Therefore, we can address the root-cause by creating models that predict the remaining useful life of vessel components. In order to do so, we investigate a remote monitoring dataset which includes indirect condition monitoring data and alarms. The alarms resemble soft failures. However, this data has various issues, like sequences of missing values and sensor error resulting in odd observations. Therefore, using this data would start a tedious trial and error process because when a predictive model does not work we are not sure what the reason is. Is it due to our modelling choices, due to the data quality, or due to the absence of any relationship between indirect condition-monitoring data and failure patterns? For that reason, we simulate a predictive maintenance dataset that allows us to train, validate and test predictive models. This simulated dataset has the same structure as the real dataset, yet the real dataset suffers from data quality issues (Damen is currently working on that). Our focus is the creation of a predictive maintenance method that creates remaining useful life prediction models and optimizes the resulting predictive maintenance policy.

From literature, we select two promising prediction models: a recurrent neural network with long-short term memory units, and a gradient boosting regression tree. We evaluate

these models based on the maintenance costs after optimizing a maintenance threshold (see Appendix C. We also report other evaluation metrics. The hyperparameters of the models are tuned with a particle swarm optimization. We concluded that a 1.4% cost reduction can be achieved on our simulated dataset by predicting the remaining useful life with a histogram gradient boosting tree when compared to the baseline time-based preventive maintenance model. The recurrent neural network did not outperform the baseline when compared on the cost-based metric.

We offer four main contributions to practice. First, we develop machine learning models to predict the remaining useful life of critical assets. Second, we provide a methodology that generalisese to other ships and predictive maintenance cases. Third, we formulate a maintenance threshold optimisation model which allows to act upon the remaining useful life predictions. Finally, we started the development of an application in which predictive models can be trained, evaluated and implemented. This application should use the Damen remote monitoring system as input, predict the remaining useful life using a chosen implemented model, and forward maintenance decisions to the client via the maintenance management system. Additionally, the academic contributions are twofold. First, a clear taxonomy of maintenance with a special focus on predictive maintenance is provided. Second, the applicability of two machine learning models (recurrent neural networks and gradient boosting trees) to predictive maintenance problems is explored.

We recommend Damen to proceed with predictive maintenance. When doing so there are various directions to explore and exploit. Specifically, the recommendation to Damen is:

- select a vessel and component with high maintenance costs or down-time penalty using input from clients and maintenance experts;
- in absence of labeled failure data gather root-cause knowledge regarding the failure and use design-out maintenance or create a rule-based expert system or outlier detection;
- in case there is sufficient labeled failure data continue with the predictive maintenance models from this research and further develop the predictive maintenance application;
- once predictive maintenance models are in place, look into prescriptive models through for instance case-based reasoning systems.

When following these recommendations, we expect Damen to advance their predictive maintenance capabilities and provide customer value by reducing maintenance costs and down-time penalties.

Contents

Chapter 1

Introduction

This chapter introduces the reader to the predictive maintenance case study. This is achieved by describing the problem at hand, the research questions, the data, and the scope. The last section of this chapter, provides an outline of the report.

1.1 General

Unforeseen asset failures are seen as a major operational risk by executives in asset-intensive industries, (LaRiviere et al., 2016). To reduce the operational risks and minimize the costs related to keeping critical assets operational, adequate maintenance policies are essential. This applies especially to assets for which maintenance execution options are restricted, like ships. Ships are moving assets with often high utilization rates that, when sailing, face reduced maintenance execution options. Furthermore, vessels are exposed to a large variety of weather and water conditions. This exposure increases the likelihood of unforeseen failures on vessels and therewith the added-value of predictive maintenance policies, which raises a business case for predicting the moment of failure of critical components at maritime conglomerates like Damen Shipyards. In this thesis, we develop a methodology that enables Damen to predict the remaining useful life (defined as the time to next failure) of a vessel sub-system and optimize the maintenance policy. We offer four main contributions to practice. First, we develop machine learning models to predict the remaining useful life of critical assets. Second, we provide a methodology that generalises to other ships and predictive maintenance cases. Third, we formulate a maintenance threshold optimisation model which allows to act upon the remaining useful life predictions and compare the predictive policy with a baseline preventive policy. Finally, we started the development of an application in which predictive models can be trained, evaluated and implemented. This application should use the Damen remote monitoring system as input, predict the remaining useful life using a chosen implemented model, and forward maintenance decisions to the client via the maintenance management system. The academic contribution of this thesis is two-fold. First, we provide

a clear taxonomy of maintenance with a special focus on predictive maintenance. Second, we explore the applicability of two machine learning models (recurrent neural networks and gradient boosting trees) to predictive maintenance problems.

1.2 Research objective & approach

In this research, we aim to develop the predictive maintenance models which enable Damen Shipyards to predict the remaining useful life of vessel components. By predicting the remaining useful life of components, the maintenance services portfolio can be further elaborated and enhanced maintenance policies can be brought to the client. The remaining useful life is defined as the time till failure of the respective component, where in our case failures are indicated through an alarm. These alarms signal reduced functionality of a vessel system and thus resemble a soft failure. Prediction of the remaining useful life can enable customers to prevent failures just-in-time, which contributes to the customer experience, reduces maintenance costs and increases asset up-time. Avoiding vessel down-time makes for a clear business case because the contractual penalties for a day of down-time can add up to 55.000 euros a day. To avoid down-time, we aim to predict the remaining useful life of a vessel component using the available remote monitoring data. The available data does not directly provide information regarding component health and can be regarded as indirect condition-monitoring data. Using this data, we intend to develop predictive maintenance capabilities that can be generalised to a multitude of components and vessels. To realize these objectives, we pose the following main research question:

'How can we enable Damen to predict remaining useful life of vessel components in varying circumstances with indirect condition-monitoring data and reduce costs?'

The research is structured through the decomposition of this main research question into four sub-questions. The answer to the main research question should follow logically from the answers to the four sub-questions. The sub-questions, together with the chapters which answer them, are the following:

- What is the structure of this predictive maintenance problem? (Chapter 2)
- Which solution approaches are reported in literature to tackle predictive maintenance problems and which are most suitable for the problem structure at hand? (Chapter 3)
- How can we implement, validate and evaluate the models? (Chapter 4)
- How do the developed predictive maintenance models perform? (Chapter 5)

Once we know the root-cause of the predictive maintenance problem and all its attributes, we can explore the literature for suitable solution approaches. For the most suitable solutions,

we have to consider the best way of implementation, validation, and evaluation. Considerations, such as how to choose hyperparameters and which evaluation metrics are applicable, have to be made. Finally, we compare the predictive maintenance models to a preventive maintenance model which serves as a baseline.

1.3 Data, scope & problem structure

1.3.1 Data

Damen Shipyards is an internationally operating vessel building conglomerate in the maritime industry. The newer line of Damen vessels is equipped with alarm notification systems that indicate failures of a sub-system of the vessel. Our case vessel is a high-speed crew transit vessel, which is equipped with amongst others a fuel separation, a fire safety and a low oil pressure alarm. To be explicit: the alarms provide dichotomous information as they can be either 'active' or 'not active'. This was, however, not labeled dichotomously, but as a 16-bit binary number. An active alarm indicates damage, reduced functionality or an endangered operating state which will cause damage if not acted upon. Our failures are thus not hard, but rather soft failures (Taghipour and Banjevic, 2012). The alarms are what Maillart (2006) refers to as obvious failures.

These alarm notifications provide useful diagnostic information, however, they do usually not enable the operator to prevent a failure due to the diagnostic nature of the alarm. Predicting the rise of such alarm can therefor be of added value. Our focus lies in predicting fuel separator alarms. This has three reasons. First, the functionality of fuel separators is critical to the survival of the engine, which is a costly component to replace. This criticality is reaffirmed by the observation of redundancy in fuel separators (there are two on each side). Second, after cleaning the dataset, the fuel separator alarms provide a reasonable number of alarms over the given time span of operation (see Appendix B). Many other alarms are raised too frequently to be interpreted as serious failures or not raised at all. Moreover, the choice for the fuel separator system over other alarms in the list is endorsed by maintenance experts within the company. Finally, the fuel separator system and the related variables are present on a multitude of other vessels. As a result, the value of this research is increased through the generalisability of the crafted prediction models to other vessels.

Aside from the alarms, we have a set of 9 sensors measuring engine load, fuel rate, generator load and wind speed (each sensor on the port side and starboard except for wind speed). These sensor measurements do not provide direct information regarding the degradation of the fuel separator system. We refer to this as indirect condition-monitoring (Si et al., 2011). Both, the alarms and the sensors, are tracked every ten seconds. A line of data thus enters the database every ten seconds, resulting in time series data. Using the sensor measurements of an interval $(t - \tau; t)$, we will try to predict the remaining useful life at time t.

Predicting remaining useful life with time series data is a complex task with many considerations to be made. As briefly mentioned earlier, vessels are moving assets which causes them to operate in varying weather and water conditions. Furthermore, vessels are operated at varying speeds and engine loads. We refer to this as a non-constant rate. Also, a vessel can be in different operating states. We define five operating states 'in-port', 'berthing', 'sailing', 'at-platform', and 'drifting'. Together, the variable rate and the multiple states make that the problem at hand is non-continuous. Thus, where Maillart (2006) look at continuously operating systems, we research a non-continuous operating system. This non-continuity causes a non-stationary time series.

Furthermore, we need to consider data impurities because the available data is not clean. Extended periods of sensor malfunctions and missing data are the most pressing issues. These data quality issues need to be considered, when choosing prediction models for the remaining useful life of the fuel separation system. The data quality is expected to improve in the future, however, we cannot expect it to be perfect. Therefore, we need to take into account the resilience of the models under the following data issues:

- missing values;
- sensor errors;
- imbalanced dataset (less than 1 in 6.000 observations contains a failure event).

In our dataset, these issues are severe and the evaluation of predictive maintenance models created with this data is ambiguous. Using this data would start a tedious trial and error process because when a model does not work we are not sure what the reason for that is. Is it due to our modelling choices, due to the data quality or due to the absence of any relationship between indirect condition-monitoring data and failure patterns? To exclude this ambiguity, we first simulate a dataset (based on the clean parts of the observed dataset) with the structure of a predictive maintenance problem (in short that is: a multivariate time series with a severe class imbalance and obvious failures). In this simulation, we can ensure the existence of a relationship between indirect condition-monitoring data and failures. We also avoid any data quality issues. Therefore, if a model does not work, we can attribute this to our modelling choices. When a model does work, we can later evaluate the performance of these models on a real dataset with sufficiently clean data. In short, we simulate the data to develop a robust methodology to build, validate and evaluate predictive maintenance models. During the execution of this thesis, Damen has been improving the quality of the data gathering. However, currently only two months of clean data is available which is not enough to train and validate our models.

1.3.2 Scope

The research is scoped to a data set from an FCS2710, which is a fast crew supply vessel of 27 meters long and 10 meters wide. Its function is to supply offshore platforms with crew

and materials. This type of vessel was selected due to its high utilization rates which makes proper maintenance a necessity. When extracting data, the complete time interval (from launch to extraction date) was selected in order to retrieve all observations available. This resulted in a raw dataset with close to 2 million observations (rows) and 50 columns between the 28th of August 2018 till the 1st of October 2019. 9 columns contain sensor readings, 40 contain various alarms and the remaining 2 columns contain the timestamp and the yard number (which is the unique identifier of a vessel). Our focus on the fuel separation system directs the research to four alarms, two fuel separators on each side of the vessel. We further limit the research to starboard fuel separator with most alarm observations.

1.3.3 Problem structure

In this section, we define the problem at hand as a multivariate time series regression. Yet, the problem structure can be further decomposed. Not only do we face indirect condition information regarding the degradation, but we also observe an imbalanced dataset and a system that is not continuously operating. When choosing a suitable solution approach, we have to take into account the following problem structure:

- Diagnostic vs. **predictive**
- Uni-variate vs. **multi-variate**
- **Time-series** vs. time independent
- Categorical vs. **Regression** (we convert the problem in a regression)
- **Obvious** vs. silent failures
- **Soft** vs. hard failures
- Balanced vs. **imbalanced**
- Direct vs. **indirect** condition information
- Fixed vs. **variable** operations

1.4 Report organization

The remainder of this report is organized as follows. Chapter 2 describes the current process of maintenance services, investigates the problem, and analyses the consequences of the problem. The current system analysis rounds up with a descriptive data analysis and concluding remarks regarding the status quo. Chapter 3 is a literature review in which we explore the available set of predictive maintenance modeling solutions. We provide a taxonomy of maintenance with a focus on predictive maintenance and conclude with a choice for the most suitable models. Chapter 4 discusses the methodology of the chosen predictive maintenance models. It describes modelling choices and hyper-parameter tuning in depth and describes

the creation of a simulated dataset on which models can be trained, tested and validated. Furthermore, it analyses the applicability of various error metrics to evaluate our models. Chapter 5 reports the results of the remaining useful life prediction models and compares the performance of predictive models to preventive maintenance. We finalize the report with conclusions and recommendations in Chapter 6.

Chapter 2

Status Quo

This chapter answers the question: *'what is the problem structure of the predictive maintenance problem at hand?'* It contains a description of the current (predictive) maintenance related activities within Damen Shipyards Gorinchem's Services department, the problem analysis and an analysis of the available data based upon which the simulation and predictive maintenance models are created.

2.1 Description of the current state

This section elaborates on three current processes within Damen which have essential ties to predictive maintenance. First, predictive maintenance can contribute to the current servitization strategy. Second, predictive maintenance should be an addition to the current maintenance policies. Third, predictive maintenance capabilities should be developed while considering the remote monitoring system.

2.1.1 Servitization

Damen is making a shift from a pure vessel building company to a maritime services provider. Therefore, new services have to be developed. Predictive maintenance can be a core component of these services. This has implications for the way in which the services department operates. To succeed in the delivery of servitized products, it is important to ensure the alignment of the Damen Services strategy with that of the remainder of the organization. The current Damen Services strategy[1] outlines three pillars:

- provide better customer support;
- develop total life cycle support;
- get ready for guaranteed technical availability.

[1] Internal Damen 2020 Services Strategy documents

Successful predictive maintenance inherently contributes to the three pillars of this strategy. Firstly, predictive maintenance enables Damen to enhance its relevance as a maintenance partner for its clients. Secondly, by linking the predictive maintenance application tot the maintenance management system we can inform customers when maintenance is required, which drives the customer experience. Finally, predicting failures enables the scheduling of execution of otherwise corrective maintenance activities, which allows customers to increase the utilization of their vessels. This is especially beneficial for customers with high vessel utilization rates like the fast crew suppliers which are the scope of this research. Damen sees predictive maintenance as a critical component in their servitization strategy.

2.1.2 Maintenance plans

Vessel maintenance plans provide customers with instructions on maintenance activities which their vessels require. The aim of the maintenance plan is to upkeep the full functionality of the vessel and to extend the asset lifetime. Each vessel has hundreds of component suppliers (OEMs), many of which provide their own maintenance manuals. Damen queries the original equipment manufacturer maintenance documentations and combines relevant maintenance activities in one bundle. Following up on this, the bundle of maintenance activities is reviewed and in special cases, the maintenance plan is altered to fit the specific needs of the customer's use profile. These maintenance plans typically result in a list of preventive and corrective maintenance actions with the detail of descriptions depending on the customer demands. The interval of the preventive maintenance activities is a calendar-based interval, determined by the number of operating hours of the vessel or a combination of the two. Predictive maintenance can add intelligence to the timing of the maintenance action by leveraging available remote monitoring data.

2.1.3 Remote monitoring

Damen has introduced an IoT remote monitoring platform for its customers. This platform consists of sets of sensors and alarms notification systems that are installed on a selection of new Damen vessels. The sensors provide measurements such as speed, fuel consumption, engine load, and wind speed, while the alarms notify the operator of a failure in a subsystem of the vessel. The locally collected vessel data is sent to a cloud service when an internet connection is available. A selection of this data is presented to the customer in a web-based user interface. Furthermore, the data can be used to analyse historical use patterns, inspect warranty claims and develop services like predictive maintenance. To do so, an extraction request needs to be send to the cloud service. The remote monitoring system should serve as the maintenance data source for predictive maintenance services.

2.2 Core problem & consequences

In this section, we describe the problem at hand and analyse the root-cause in more detail. We also shed light on the consequences of the problem.

2.2.1 Problem description

Vessel components are prone to degradation. The degradation of multi-components systems can be complex to monitor, especially when the asset is expected to operate under varying and rough circumstances such as at sea. Damen is looking to translate their remote monitoring solutions to predictive maintenance services. The context in which this problem takes place is depicted in Figure 2.1. We choose the absence of knowledge about the remaining useful life (RUL) of a component as our core problem because it is out of our hands to prevent components to degrade in the first place.

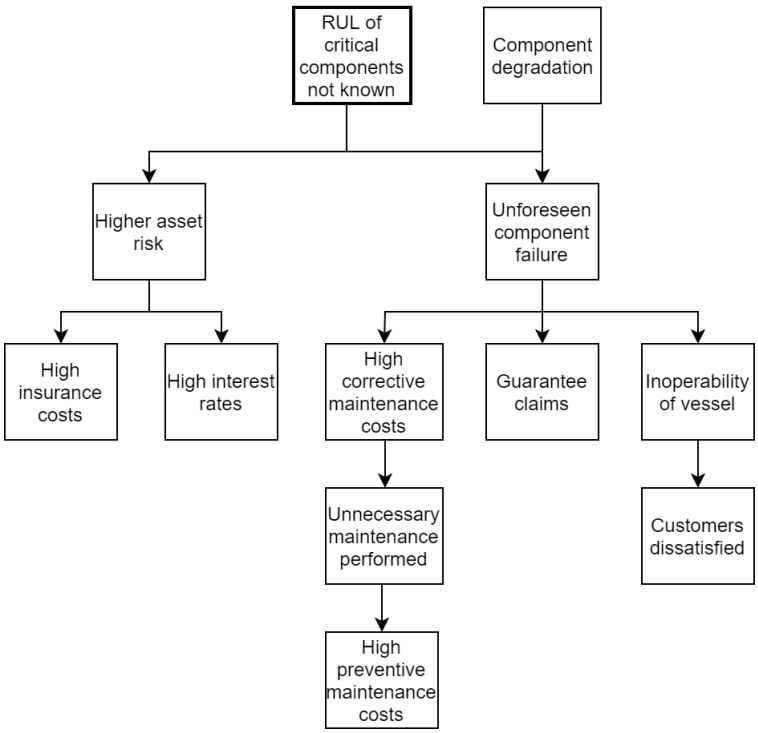


Figure 2.1: Problem Cluster level 1

The sensors on FCS2710 vessels provide high-level information and do not allow to directly model the degradation of the vessel's components. The alarms indicate failures of certain sub-systems, and can, therefore, be used as a proxy-variable for excessive degradation. Alarms provide dichotomous outputs, that is: an alarm can be active or not active. Using this observation, we model the remaining useful life as the time to next alarm. To address the core problem, we develop models that predict the RUL. Such predictive maintenance models would solve the core problem.

Sikorska et al. (2011) explored prognostic modelling for predicting remaining useful life (RUL). They observe that most manufacturing plants are not ready for full-scale predictive maintenance and that predictive modeling depends on proper fault diagnostic systems. Therefore, they suggest three levels of increasingly complex prognostic modelling for RUL predictions based on ISO13381. Our predictive maintenance models are built on top of the alarm notification system which acts as the fault diagnosis system. This project falls under Level 1 and Level 2 of the conceptual diagnostic–prognostic process, see Figure 3 in Sikorska et al. (2011, p.1808).

Looking at the scoped case vessel data, we observe 9 sensors and one fuel separation alarm. Currently, no sensors provide direct condition information. Our data can thus, only indirectly or partially model the underlaying degradation of the fuel separation system with which the remaining useful life is to be predicted. This type of data is called indirect condition monitoring data by Si et al. (2011).

The absence of knowledge about the remaining useful life of components results in higher asset risks and therewith higher insurance costs and interest rates. The opportunity to reduce these through predictive maintenance is an interesting case, but most benefits arise from the case for reducing maintenance costs. Solving the core problem is expected to relieve these problematic symptoms. We describe these consequences of the core problem in more detail in Section 2.2.3.

2.2.2 Problem Analysis

In this section, we further decompose the core problem in sub-problems. Why is the remaining useful life (a measure for degradation) of a component not known? We found that this is mainly due to unknown relations between the available sensor data and the fuel separation alarm. It is unknown if, and how, certain patterns in sensor data might result in future failure observations. If the relation between the indirect condition monitoring data and the remaining useful life of the fuel system could be exposed through our predictive maintenance models, we would effectively solve the core problem. The decomposition of our core problem results in Figure 2.2.

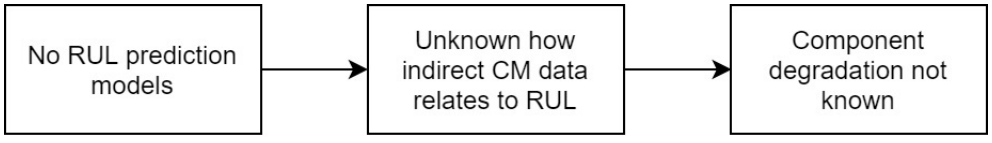


Figure 2.2: Problem Cluster level 2

So, which models should be considered to predict the RUL? To answer this question, we first consider the problem structure. The multi-variate time-series regression problem at hand has the following problem structure:

- **Obvious** vs. silent failures (Maillart, 2006)
- Hard vs. **soft** failures (Taghipour and Banjevic, 2012)
- Balanced vs. **imbalanced** class distributions
- Direct vs. **indirect** condition monitoring (Si et al., 2011)
- Fixed vs. **variable** operations

The problem at hand is a multivariate time-series prediction with an originally categorical target variable and indirect condition monitoring. The categorical target variable contains only two classes: failures or non-failures. These failures are signalled through alarms which make them obvious failures. Since a fuel separation alarm is an indication of an endangered operating state and not an immediate physical failure, we classify the problem at hand as a soft failure, (Taghipour and Banjevic, 2012). The fact that failures are rare events leads to observing relatively few failures in our dataset compared to non-failures. Therefore, we call failures the minority class. This class-imbalance makes it harder to generalise a model to new observations. Typically, models will be biased towards the majority class. To partially address the class imbalance, we transform our categorical failure label to a numerical time to failure label. Our time-series is further complicated by the variable of operations. We can classify the continuity of operations with two aspects. First, the continuity in the operating state, i.e. whether the system is performing the same task under the same conditions continuously. Second, the continuity of the usage rate, i.e. the variability in the intensity of use within operating states. See Figure 2.3.

		Usage rate		
		Continuous	**On or off**	**Variable**
State	**One**	Continuous operation	Interrupted operation	Interrupted variable operation
	Multiple	Continuous operation with state transitions	Interrupted operation with state transitions	*Interrupted variable operation with state transitions*

Figure 2.3: Continuity of operations matrix

It is important to make a distinction in the continuity of operations, because the way

in which a vessel is operated heavily influences the required maintenance tasks. Also, the time-series data is influenced by the continuity of operations. Variable operations lead to non-stationary time-series. Sensor measurements and survival rates are likely dependent on the type and intensity of operations. Since the use rate is highly variable in the case of vessels and we have multiple states, we classify our problem as having variable operations. Especially, when using simulated datasets to create pre-trained models or when using resampling techniques to address class imbalances, it is important to take this behavior into account.

2.2.3 Consequence analysis

The main consequence of not knowing the current RUL of components is an increase in maintenance costs. Components failing despite the preventive maintenance schedule result in corrective maintenance costs. The unit costs of corrective maintenance often exceed that of preventive maintenance. This often results in the tendency to apply more intensive preventive maintenance policies, which is also costly. Thus, the main benefit when it comes to predictive maintenance is the reduced maintenance costs as a result of 'just-in-time' failure prevention. This cost reduction works twofold. First, through predictions of failures, a corrective maintenance action can be replaced by a just-in-time preventive action. Second, predictive maintenance helps to postpone unnecessary (preventive) maintenance activities. Preventive maintenance on healthy components is not only costly in terms of labor costs but might also increase the failure probability of the component. In literature, this is known as 'infant mortality' which is reported to be the cause of 68% of failures, (Nowlan and Heap, 1978). Some possible reasons for components failing soon after maintenance are: product manufacturer defects, installation errors, incorrect commissioning, and maintenance that is too invasive. Predictive analytics might also assist in assessing whether the replacement was successful or is likely to be a case of infant mortality.

Besides the high maintenance costs, assets might be perceived to be at risk by third parties like insurance companies and financial institutions. As a result, unknown RUL and the lack of maintenance control lead to higher interest rates on loans and higher premiums for insurance. Furthermore, service contracts between the vessel operators and their clients often include penalty clauses which vary from 5.000-10.000 euros per day of inoperability for an FCS5009 to about 55.000 euros per day of inoperability for a platform supply vessel (PSV).

2.3 Data analysis

In this section, we explore and visualize the data in order to gain more insights about the dataset. We use these insights in the simulation. We also list the data issues which are reason for our choice to simulate a dataset.

2.3.1 Data explanation

The full list of alarms is shown in Appendix B. Our models focus on predicting the RUL (or time to alarm) of the fuel separation system. The three criteria for the component choice were: sufficient observations of the alarm being activated, not too many observations of the alarm being activated, and relevance of alarm. We need sufficient failure observations to draw inferences on failure behaviour. Simultaneously, we cannot expect that predicting an alarm that is triggered very often (e.g. daily) offers value because this alarm is not likely to be related to failure with high preventive or corrective maintenance costs. Finally, we look at the meaning of the alarm and its underlying system to assess the likelihood to contain relevant failure information. Proper fuel separation, for example, is critical to the engine's survival. Predicting low fuel alarms, on the other hand, adds little value aside from the case of fuel theft or leakage. This led to the selection of one interesting group of alarms: the fuel separator alarms. It is critical for fuel separators to operate effectively. When fuel is not cleaned before it is fed to the engine, this might cause costly engine failures. The FCS2710 has two fuel separators for each engine, which indicates redundancy to prevent inoperability of the vessel in case of failure (or replacement) of one of the fuel separators. This redundancy further stipulates the criticality of the fuel separator system. We observe 372 failure cases for starboard (SB) fuel separator number 1, and 13 failure cases for number 2. When this alarm occurs, the separator drip tray is full. Thus, impurities can no longer be filtered out of the fuel. This might indicate reduced functionality of the filter and can lead to impurities in the fuel mixture reaching the engine which can result in damage to the engine. Next to this alarm, our analysis includes the following sensors:

- PS generator KW load (%);
- SB generator KW load (%);
- ME PS engine speed;
- ME PS fuel rate;
- ME PS Percent Load At Current Speed;
- ME SB Engine Speed;
- ME SB Fuel Rate;
- ME SB Percent Load At Current Speed;
- Wind speed;

** Note that PS stands for port side while SB stands for starboard and that we normalise all sensor measurements and thus these have no units.*

2.3.2 Data issues and challenges

The following issues in our dataset prevent us from training predictive maintenance models on the dataset in its raw form:

- ambiguous alarm labels;
- missing values and odd observations;
- dataset unbalanced.

First, the alarm labels were binary codes for more than just the indication of 'alarm active' or 'alarm not active'. In most cases, the state of the alarm could easily be retrieved by looking at the second number from the right of the binary transformation of the observed label. For example, the labels 1024 and 1026 could be two of the two million observations of the chosen starboard fuel separation alarm. These values do not directly provide insight into whether an alarm was active or not. 1024 corresponds to the 16-bit binary digit: 0000010000000000, while 1026 corresponds to 0000010000000010. The second number from the right of this sequence indicates an active alarm when equal to 1 and inactive alarm when equal to 0. Thus, in our example 1024 is an inactive alarm while 1026 is an active alarm. See Figure 2.4.

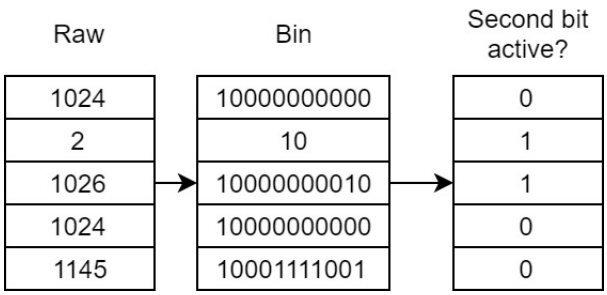


Figure 2.4: Process of cleaning the alarm raw data

Second, missing values can be handled in a multitude of ways. In our dataset, we observe two different types of missing values: missing observations (all values corresponding to an expected timestamp) and missing values for a specific sensor or alarm on a timestamp for which other sensors and alarms have values. If we have very few consecutive missing values we can impute these values with for instance a moving average. But, when we have many consecutive missing values, imputation methods might be unreliable and we should consider dropping the variable from the analysis, see Figure 2.5.

Horizontal lines in this graph indicate missing data. Missing values are especially problematic in time-series problems when modelling the temporal dependencies is key to accurate predictions. We can handle odd observations in a similar way as missing values. In practice, most odd observations are sensor errors. After addressing this issue for the fuel rate sensor, we observe Figure 2.6 and see an interesting pattern which indicates the variability of operations.

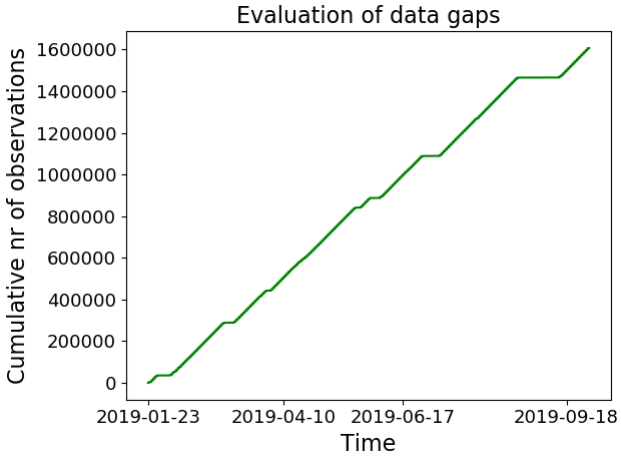
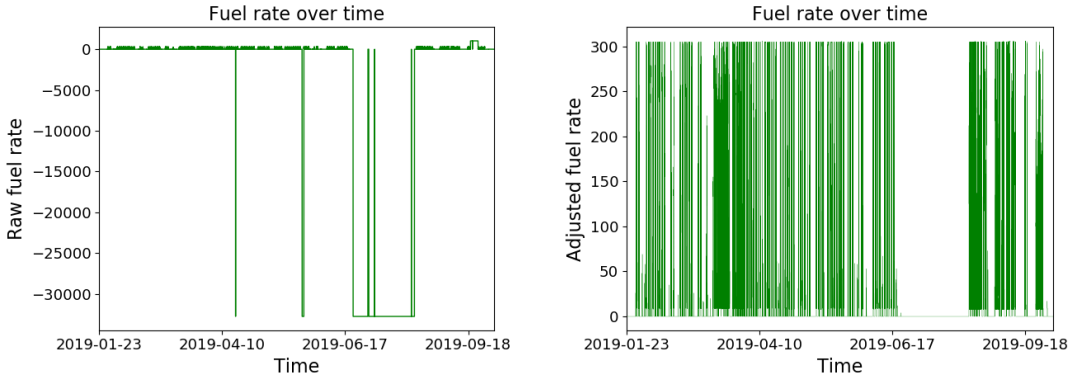


Figure 2.5: Missing observations in the dataset



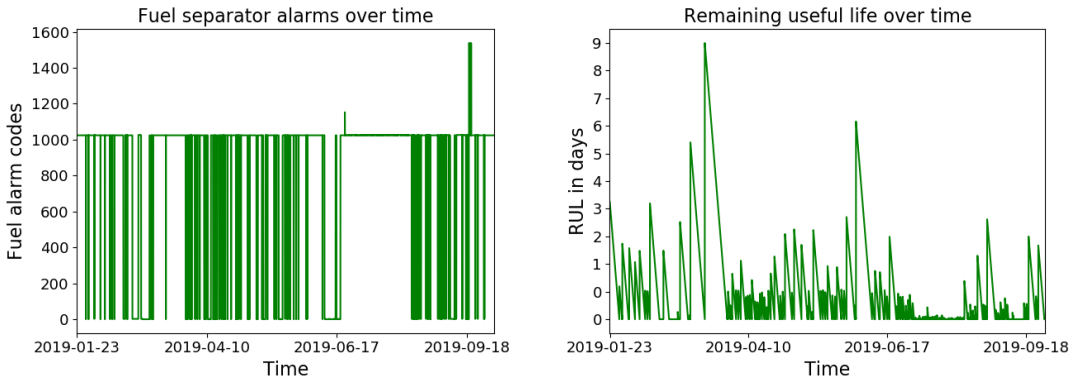
a) Raw fuel rate b) Fuel rate imputed with zeros on errors

Figure 2.6: The fuel rate contains large consecutive sections with errors

Lastly, the unbalanced nature of the dataset is due to the scarcity of failure observations. If not addressed properly, most models will be biased to the majority class. In order to partially resolve this issue, we transform our alarm observations in 'time-to-alarm' values. This is equivalent to remaining useful life, which we want to predict, see Figure 2.7.

2.3.3 Ratio analysis

In order to simulate realistic failure behaviour of a case vessel, we investigate the descriptive statistics of our dataset. To maximize the value of the insights gained from these descriptives, we pose the question: how do the descriptive statistics vary with respect to the RUL? Specifically, we are interested in whether relative sensor measurements (for example the engine load relative to the engine speed) show different medians when the RUL is small than when the RUL is large. To analyse this we define three equally sized groups: low RUL, medium RUL, and high RUL.



a) Fuel separator alarm binary encoding

b) Remaining useful life (RUL)

Figure 2.7: From fuel alarm codes to regression

We define ratio variables for each possible pair in the 9 previously mentioned sensors. This leads to $\binom{9}{2} = 36$ unique sensor ratios. We observe higher median 'fuel rate to engine load at current speed' ratios for short RULs than for long RULs. The low RUL group has a 'fuel rate to engine load at current speed' ratio (7.0) that is almost twice as high as that of the other two groups (3.8 and 4.1 respectively). This is an interesting finding from our ratio analysis, especially since the analysis regards the RUL of the fuel separation system. The observation suggests that the fuel usage is less efficient when a fuel separator is about to fail.

2.4 Status quo conclusion

In this chapter, we have seen that we can classify our problem specifically as an imbalanced multi-variate time-series regression with indirect condition monitoring data. This predictive maintenance problem can further be characterized by variable operations and soft, obvious failures. The absence of information about the RUL of a vessel component results in high maintenance costs for critical components. We investigate how we can enable Damen to predict fuel separation failures using their indirect condition monitoring data. These predictions should allow for just-in-time prevention of critical failures, thereby reducing both corrective and preventive maintenance costs. The algorithms developed to create the RUL prediction models should be generalisable. Meaning, we should be able to use the same algorithms to create RUL prediction models for other critical components or vessels. We have also seen that predictive maintenance fits with the current processes and observe an alignment between strategic goals and the envisioned benefits of predictive maintenance. Furthermore, we analysed the available data and found that fuel consumption is less efficient when a fuel separator is about to fail. We take this finding into account when simulating a clean dataset upon which a modelling algorithm can be developed.

Chapter 3

Literature

This chapter contains a description of the relevant literature. It answers the question: *'which solution approaches are reported in literature to tackle predictive maintenance problems and which are most suitable for the problem structure at hand?'* Predictive maintenance is defined and a taxonomy of predictive maintenance approaches is presented. Furthermore, we describe models suitable for predictive maintenance in detail. This allows us to make a well-informed choice for a prediction model while identifying future research opportunities. We also describe a time series extension for addressing class imbalance and finalize with a concluding model choice.

3.1 Predictive maintenance

3.1.1 Maintenance taxonomy

We intend to find a systematic grouping method (a taxonomy for maintenance), such that maintenance policies with common characteristics are placed in the same group. Najim et al. (2004) define a maintenance strategy as *"(...) decision rule[s] which establish a sequence of actions to be undertaken with regard to the operat[ing] state of the considered system."* Predictive maintenance makes use of condition monitoring data to predict the remaining useful life of a system under consideration. This prediction is used to optimize the maintenance policy.

Trojan and Marçal (2017) describe a maintenance taxonomy that helps to choose a suitable maintenance policy given an asset and its maintenance-related characteristics. Their taxonomy is based on five criteria which should be scored on a scale from 0 to 100. The resulting classification method prescribes a maintenance policy to apply. They do, however, not provide a clear maintenance taxonomy in which predictive maintenance can be unambiguously placed. Khazraei and Deuse (2011) developed a more exhaustive taxonomy shown in Figure 3.1. They list a set of twenty-three maintenance policies differentiating between reactive and preventive ones. The reactive policies are further split into corrective

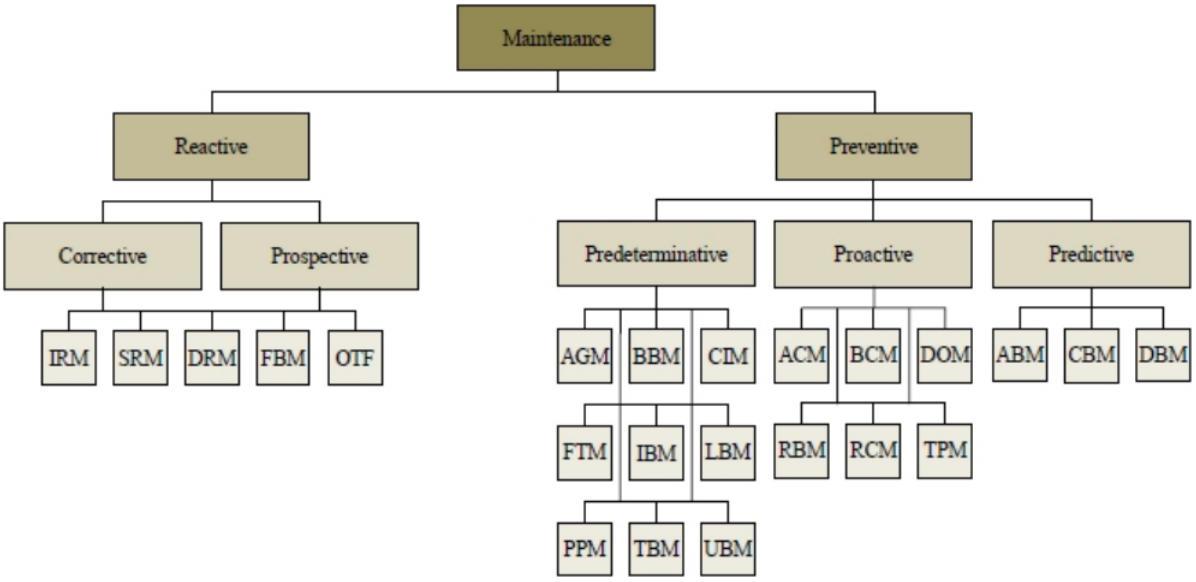


Figure 3.1: Taxonomy of Maintenance adopted from: Khazraei and Deuse (2011)

and prospective policies, while the preventive policies are decomposed in predeterminative, proactive and predictive policies. Their taxonomy sheds light on which maintenance policies can be classified under which category of maintenance policies. According to this classification, predictive maintenance is a container for three 'tactics'. Avoidance-based maintenance, condition-based maintenance, and detective-based maintenance are scaled under predictive maintenance policies. If these tactics are to compose predictive maintenance, it is important to define them. First, avoidance-based maintenance is defined by Khazraei and Deuse (2011) as *"[m]aintenance [that] is focused on the avoidance of a failure rather than detection of it. Failure is prevented by act of refraining from it."* A use-case for this would be to provide operational advice to the user regarding the way in which the vessel is handled. Second, they define condition-based maintenance as *"[m]aintenance [that] relies on the fact that the majority of failures do not occur instantaneously, and they can be predicted by condition monitoring."* Finally, they define detective-based monitoring (DBM) as: *"[m]aintenance [that] is undertaken as a consequence of condition monitoring done only by the human senses."* This breakdown of maintenance is not mutually exclusive and collectively exhaustive, (Rasiel, 1999, p.6-8). Especially, an overlap between DBM and reactive-prospective tactics and the absence of other predictive maintenance and condition-based methods cause concern for the predictive maintenance branch in the taxonomy. The taxonomy by Li et al. (2016) is regarded to have less overlap between different maintenance policies. This taxonomy splits maintenance policies in corrective, preventive and predictive policies. Corrective maintenance policies can be deferred, (meaning the maintenance action is scheduled to take place somewhere in the near future) or immediate. Preventive maintenance policies are decomposed in reliability-centered (Nowlan and Heap, 1978), time-based, opportunity, and

design-out policies. Finally, predictive maintenance policies are split up in statistical-based and condition-based.

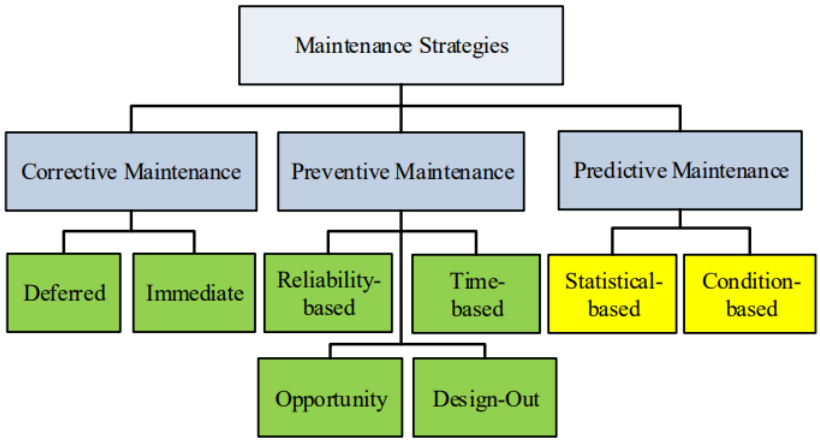


Figure 3.2: Taxonomy of Maintenance adopted from: Li et al. (2016)

We add to the exhaustiveness of the predictive maintenance taxonomy by incorporating the classification made by Okoh et al. (2016). This maintenance taxonomy breaks down predictive maintenance in data-driven, knowledge-based, model-based and hybrid approaches. Data-driven techniques are build upon historical 'run-to-failure' data. Pure data-driven approaches require quality data about prior failures and the circumstances in which these failures occur. Knowledge-based techniques solve problems by maintaining a knowledge base composed of expert knowledge. In the context of maintenance, knowledge-based techniques would keep track of prior failures and compare those to the current state of the system. Physical-model-based techniques rely on physics to determine relationships between usage, fatigue, and failure. Finally, hybrid techniques combine approaches to improve model performance. Physical-models are out of scope for this thesis, due to data unavailability on the level of detail required for those methods.

The union of available literature on predictive maintenance taxonomies results in five predictive maintenance strategies: statistical, data-driven, knowledge-based, physical-model-based, and hybrids. These maintenance strategies differ among the following three aspects: type of modelling techniques, data requirements and knowledge requirements, see Table 3.1. The outcome of this classification is in line with the literature review of Jardine et al. (2006).

Table 3.1: Predictive maintenance strategies

Type	Model type	Data requirements	Knowledge requirements	Applied by
Statistical	Statistical distribution based models.	Time-to-failure distributions, dependencies of components.	Prior probabilities of failure, misproductions, imperfect maintenance etc.	(Horenbeek and Pintelon, 2013)
Data-driven	Typically, machine learning models for failure pattern recognition, degradation or other inferences.	Labeled failure history and large amounts of observations of related variables.	Domain-knowledge, machine learning.	(Pecht and Kang, 2018, Chap. 21), (Si et al., 2011)
Knowledge-based	Expert-systems which keep track of a knowledge base to which the state of the system is compared.	Documented cases with context and action taken.	Domain-knowledge, basic rule logic.	(Butler, 1996)
Physical-model-based	Models based on physics. Formal relations between usage, fatigue and failure.	Detailed observations of physical use-dependent wear patterns.	Domain-knowledge, physics.	(Pecht and Kang, 2018, Chap. 3)
Hybrids	Combinations of various predictive maintenance strategies.	See data requirements of the relevant strategies.	See knowledge requirements of the relevant strategies.	

Resulting from this maintenance literature analysis, we propose the following maintenance taxonomy with focus on predictive maintenance, see Figure 3.3.

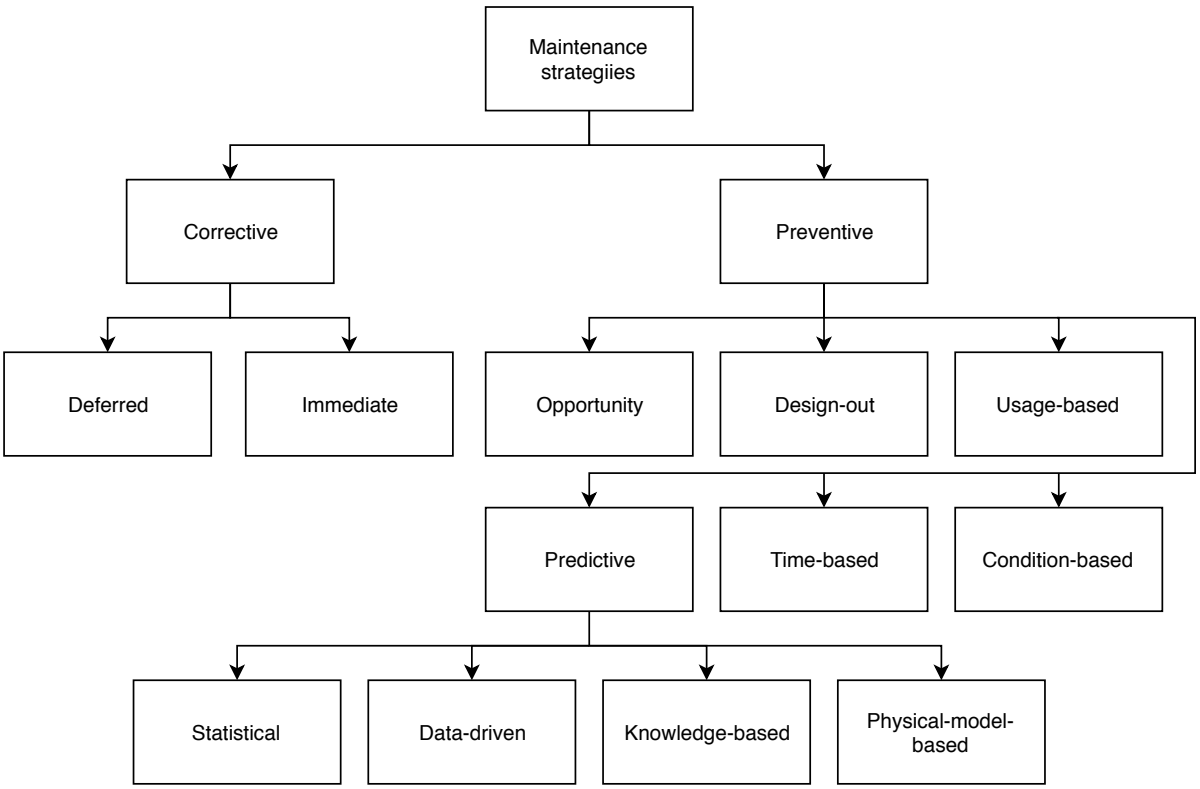


Figure 3.3: Own elaboration Maintenance Taxonomy

The aim of this taxonomy is to provide insight into the relation predictive maintenance has to other maintenance policies.

3.1.2 Predictive maintenance definition

Based on Table 3.1, we can conclude that there is a multitude of approaches to predictive maintenance, some of which are less applicable to our case than others. The various approaches and definitions of predictive maintenance can make the subject ambiguous. Yet, we want to clearly define the main concept: 'predictive maintenance'. Therefore, we review various definitions of predictive maintenance.

Trojan and Marçal (2017, p.565) define predictive maintenance as *"(...) nothing more than preventive maintenance based on the equipment condition."*Although, it gets the core, this definition might be too generic and it lacks specificity. A more specific definition is given by Kuzin and Borovicka (2016, p.123). Whom define predictive maintenance as *"(...) rule-based maintenance grounded on on-line condition monitoring, which relies on an appropriately chosen set of external sensors."* This provides a better explanation and mentions the necessity of online remote monitoring systems. Yet, itlimits the definition

to rule-based expert-systems, a specific model for knowledge-based strategies in Table 3.1. The subjectively most suitable definition in existing literature, for predictive maintenance, is *"(...) an approach that utilises the condition monitoring data to predict the future machine conditions and makes decisions upon this prediction"*, (Schmidt and Wang, 2018, p.5). This definition is generic enough to overarch all predictive maintenance policies in Table 3.1. At the same time, it covers 'machine condition' which is a concept one can relate to failures, behaviour, degradation, RUL and probably any other predictive maintenance-related desired outcome. Furthermore, it mentions that decisions should be made based upon the predictions. Yet, one extension could improve the definition. That is the inclusion of information other than data directly related to the condition, distinguishing it from direct CBM. We, therefore, suggest the following definition of predictive maintenance for this project. Predictive maintenance is a maintenance policy that utilises indirect condition monitoring data, as well as other contextual data, to predict the future asset failures based upon which maintenance decisions can be made. Here, contextual data is defined as data which provides further context about the operating circumstances of an asset, such as the location or current mission information. A predictive maintenance model consists of two essential components: (1) a RUL prediction model and (2) a maitenance optimization model. We focus on the predictive model, but also provide an optimization model, see Appendix C.

3.2 Predictive maintenance models

In this section, we provide an oversight of several models deemed suitable. For each type of predictive maintenance strategy (see Table 3.1) several implementation options exist. Physical models are out of scope because no data relevant for these models has been gathered. Therefore, the predictive maintenance models described here fall under statistical, data-driven, and knowledge-based strategies. To enhance readability, we first provide a brief explanation of the modeling jargon used.

Feature engineering: creation of new variables that should provide more information.

Loss function: generally differentiable function to be minimized by a model.

Underfitting: model is not learning enough from the training set.

Overfitting: model learned relations which are not generalisable to new observations.

Regularization: constraining a model to reduce the risk of overfitting.

Ensemble: combines several models to enhance performance, reduce bias or variance.

3.2.1 Statistical models

The first type of models described in this literature review are statistical models. We define statistical models as white-box approaches that allow the user to understand the decisions made by the model. This is in contrast to data-driven models which will often learn complex, deep patterns from the data resulting in a black-box model. The logic of statistical models is therefore easier to assess, which is a key strength of these models. A negative implication of the relative simplicity of these models, is the necessity of distributional assumptions which might not always hold.

Logistic regression

A (multiple) logistic regression is a simple statistical model which tries to place a discriminating hyperplane such that two (or more) classes are optimally split. This hyperplane is represented by a set of weights. These weights are often optimized by maximizing the log-likelihood (the loss function) of the data. Once the weights are optimized, we can retrieve the probability of an observation belonging to a certain class by applying the Sigmoid function to the dot product of the feature vector and the weight vector.

$$f(x) = \frac{1}{1 + e^{-xW^T}} \tag{3.1}$$

The classes this model predicts could, for example, be 'A': 'Failure', 'B': 'No failure'. Frequently, in a binary-class problem, an observation is classified as 'class A' if the outcome of the Sigmoid activation is more than 0.5 and 'class B' otherwise. However, a good choice for this threshold depends on the relative severity of different mistakes the model can make. If classifying an observation which belongs to class 'A' as class 'B' is worse than verse visa, the threshold can be lowered. This results in a higher chance to classify an observation as class 'A'. See Berkson (1944). It is good practice to normalize the features before training. This avoids complications with large weights when applying regularization techniques. A logistic regression can be a suitable alternative for diagnostics for simple non-time series failure diagnostics. When the target classes are lagged in time and time series features are engineered, the logistic regression and other classification models can be used as predictive models.

Simple classification and regression trees

A classification or regression tree (also frequently referred to as 'decision tree' in machine learning literature) consists of decision nodes, branches, and leaves. At the decision nodes, a split is made on a feature, the branches lead to the next decision node until a stopping criterion is reached. At that point, a leaf indicates the label of the target. A tree algorithm has to decide on which feature to split and what the threshold of the split should be. One way

to do this, for classification trees, is by calculating the 'information gain' of splitting the data on each feature. The information gain is defined as the difference in entropy of the data set before and after the split on a feature, see Hyafil and Rivest (1976).

Just like a logistic regression, a simple classification tree can be used for simple non-time series failure prediction. Although this model does not optimize weights, it is possible to say something about the feature importance by calculating the total entropy reduction that can be attributed to a feature. Regression trees are a variation on classification trees focusing on predicting a numerical variable rather than a class. Regression trees are therefor suitable for simple RUL predictions.

Perceptron algorithm

A perceptron is composed of a neuron, a vector of weights and a bias. A vector of input variables is multiplied (dot product) by a vector of weights. This scalar value is then inserted in a differentiable activation function which provides an output. The output is compared to the actual target which results in an error. The error is used to update the weights of the perceptron. See Rosenblatt (1957).

A perceptron algorithm can contribute to failure prediction in similar ways as the logistic regression and simple classification tree. Yet, the perceptron can also be used to predict RUL when altering the loss function. This algorithm forms the very basics of a neural network and is therefor a useful concept to understand.

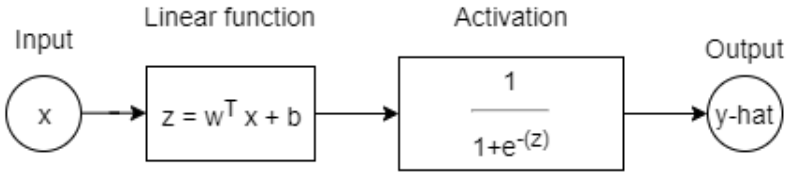


Figure 3.4: How a perceptron works

Note that the difference between a logistics regression and a perceptron with a Sigmoid activation, is the training procedure.

Naïve Bayes classifier

A naïve Bayes classifier is a probabilistic classifier which assumes independence between features. It computes the probability for an observation to belong to a certain class given the data (evidence). This is computed as the multiplication of the probability of belonging to the class (prior) with the probability of observing the data given the class, see Equation 3.2.

$$P(Class|Data) = \frac{P(Data|Class) * P(Class)}{P(Data)} \tag{3.2}$$

This probability (called 'posterior') is calculated for each class. Usually, the observation is classified following the maximum a posteriori (MAP) rule. See Maron (1961). If the independence assumption holds, training of naive Bayesian models converges in O log(n), where n is the number of features. This is one main advantage of this model. However, our time series features are not independent. It can be used for simple non-time series failure prediction, which could be the case if data is gathered through inspections with irregular intervals.

Proportional hazards model

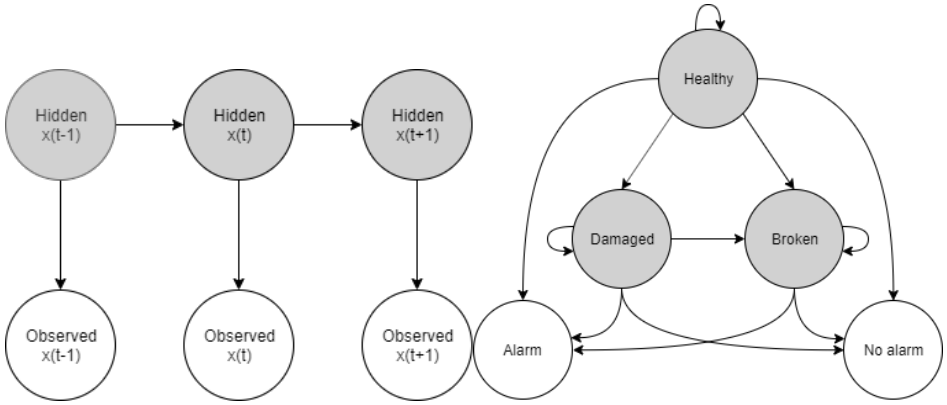
A proportional hazards model (PHM) is a survival regression model which relates time to failure to a set of variables called 'covariates'. The PHM is composed of the multiplication of a baseline hazard rate $h_o(t)$ and a function of covariates. This baseline hazard at (t) is multiplied by the covariates and their coefficients to obtain the proportional hazard rate. Formally, the baseline hazard function is the hazard function one obtains from the PHM when all covariates have no effect on the hazard rate. The Weibull distribution is frequently used to specify the hazard baseline function, (Si et al., 2011).

Si et al. (2011) reveal two main limitations to this type of model that are relevant in our case. Firstly, the calculation of the covariate coefficients for the regression might suffer from reverse causality, meaning a failure could influence the covariates rather than the covariates causing a failure. Secondly, the estimation of these covariate coefficients needs to be done concurrently because of its multiplicative nature. This means that a large amount of failure observations is required.

Hidden Markov model

A hidden Markov model (HMM), is a Markovian model with hidden states. The 'hidden' states are not directly observable by the model and therefore referred to as hidden. These models can be used to draw inferences about the hidden states as well as the observed states. This can be beneficial when a variable (feature or target) cannot be observed or has historically not been observed (imputation of missing values), see (Kinghorst et al., 2017; Simões et al., 2017). Formally, a HMM can be used (1) to find the probability of an observation (or sequence of observations), (2) find the most likely sequence of hidden states (Viterbi, 1967), and (3) learn, given a set of observed sequences, the most likely transition, and outcome (emission) probabilities.

In the context of our predictive maintenance case, these types of models can be used to identify the most likely state of the degradation. The sensors and alarm notifications would function as the observed process, while we try to identify the state of degradation, see Figure 3.5. The main weakness of this model is the assumption of the underlying Markovian degradation process.



(a) HMM unfolded overtime *(b)* HMM maintenance example

Figure 3.5: Hidden Markov model visualization

Kalman filters

A Kalman filter can be used to estimate a system state when it cannot be measured directly. A Kalman filter is a two-step process with a prediction step and an update step, much alike the expectation-maximisation algorithm for HMMs. The prediction part first produces an initial estimate of the observed variable, called the a priori state estimate and the error covariance p. These estimates are then used in the update of the model to produce the a posteriori estimate. Si et al. (2011) name three limitations of these types of models which are relevant to our case study. First, the models are one-dimensional. This causes problems since we have multiple variables in our data set. One solution would be to use a principal component analysis and only use the first principal component in the model. The second limitation is that such models assume no preventive maintenance actions are taken during the time interval of the data set. Finally, a large amount of event data is required to train these types of filters.

Partially observable Markov decision process

Partially observable Markov decision process (POMDP) is a generalized Markov decision process (MDP) where the states of the model are not completely observable, (Åström, 1965), see Figure 3.6. We try to find a policy that optimizes a reward function. Note that a repair action does not necessarily lead to a healthy state (not all possible actions are denoted in the figure to enhance readability). In a POMDP, belief states are updated to learn the most likely state of the model upon which an optimal policy should be built. The belief states can be interpreted as the perception the model has of its current state. That is, the belief state is the set of probability distributions for state variables. It is important to have an accurate perception of the current state because the 'best' course of action depends on the actual state. There are various techniques to optimize the belief states. When the belief states are learned and updated through applying machine learning techniques, we refer to it as reinforcement

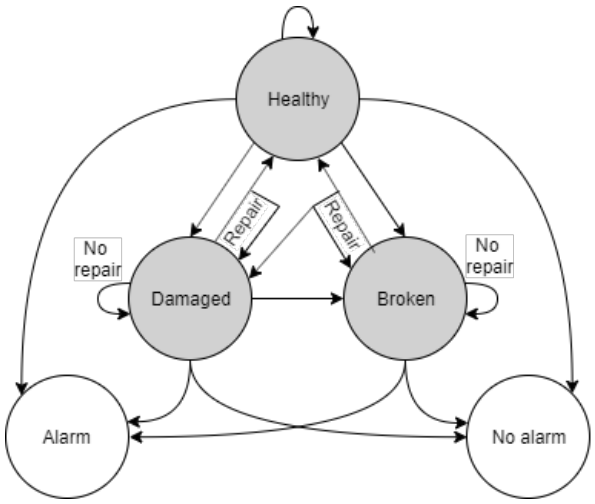


Figure 3.6: POMDP decisions influence transition probabilities

learning, Martinez et al. (2018). For reinforcement learning, the state space does not have to be guided by a Markovian process and thus no Markov property has to hold. However, both try to optimize a Bellman's equation, see Equation 3.3. We will describe reinforcement learning briefly hereafter.

$$V(s) = max_a\{R(s,a) + \gamma * V(s')\} \tag{3.3}$$

POMDP models can be used to model the theoretical failure process. Just like HMM models, we can infer the most likely state of degradation with these models. In addition to inference about the most likely state (belief state), the model can take maintenance decisions. Like this, an optimal maintenance policy can be defined based upon historic data or in a simulated environment.

3.2.2 Data-driven models

The second type of models discussed in the literature review are data-driven models. Data-driven models are black-box approaches which learn deep patterns from intensive training on a data set. The strength of these models is to capture complex relations between large amounts of variables. Capturing these complex relations comes at the cost of model simplicity, which makes the decision logic of data-driven models harder to understand, increases the risk of overfitting, and increases training duration.

Random forests

We have already discussed tree-based models in Section 3.2.1. Random forests are an extension of these classification trees. In essence, a random forest is a decision tree making use of bootstrap aggregation, Breiman (2001). This means a random sample with replacement

is taken repeatedly from the training set to which, each time, a tree is trained. The trees in the 'forest', make individual predictions of the expected target, which can be combined into an ensemble prediction. Thus, if a random forest exists of 1000 trees, there are 1000 predictions for each observation in the test set. The aggregation of the predictions made by the individual trees results in the prediction of the random forest. Random forests correct for simple tree's tendency to overfit on a training set when the maximum depth is increased. Yet, proper regularization is still crucial.

Random forests can capture more complex relations in data sets than simple classification trees, and might, therefore, be more suitable for predicting failures on our data set. As for any non-time series model described here, time-series feature engineering is required to create models which can incorporate time dependencies.

Boosted trees

Where a random forest uses a multitude of deep trees, a boosted tree trains a multitude of shallow trees (weak learners), Breiman (1997). Each consecutive tree is specialized in the previously misclassified training samples. This makes them very suitable for predictions on unbalanced data sets. Boosted trees are grown sequentially, while random forests are grown in parallel. Furthermore, boosted trees are sensitive to noisy data and overfitting, making adequate regularization techniques imperative.

Support vector machine

A support vector machine (SVM) is a non-probabilistic binary linear classifier which can be extended to usage for multi-class problems and non-linear classification. An SVM maps its inputs such that a hyperplane forms an optimal separation between classes. The mapping of inputs leads to higher-dimensional spaces in which the features may be linearly separable (whereas they were not in lower-dimensional spaces). The model can be expressed as a subset of the features. This subset is called 'the support vector', see, Boser et al. (1992).

Multilayer perceptron

The multilayer perceptron (MLP) is an extension of the perceptron algorithm. MLPs can have multiple layers of multiple neurons. These, 'vanilla' neural networks, are able to discriminate between targets which are not linearly separable by learning deep and non-linear relations between features and targets. These networks have an input layer, at least one hidden layer, and an output layer. If multiple hidden layers are used, the MLP is a form of deep learning. Various algorithms exist to optimize the weights of the MLP, the most popular amongst which is backpropagation (in combination with stochastic gradient descent), (Gardner and Dorling, 1998). Backpropagation uses the chain rule to compute the gradient of the loss with respect

to the weights. This is done one layer at the time, starting at the output layer and results in the updates for the weights.

$$W_i^{new} = W_i^{old} - \eta * \frac{\partial Error}{\partial W_i} \tag{3.4}$$

[h!] A few weights are shown in Figure 3.7 as an example of the regular numbering of weights. In our predictive maintenance case, output 1 could return the probability of an observation belonging to the class 'alarm active', while output 2 could provide the probability of an observation belonging to the class 'alarm not active'. Since the problem is modelled as a RUL regression, we limit the output neurons to 1 and change the loss function.

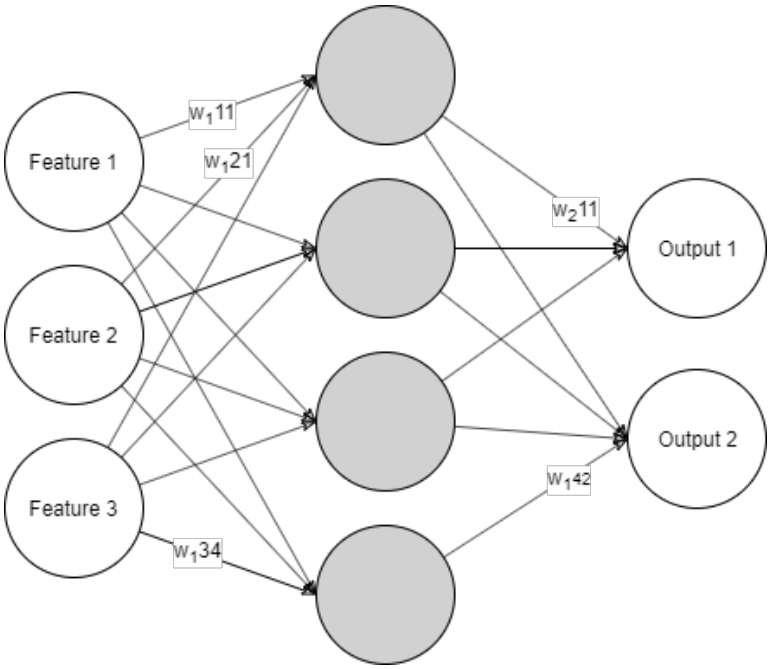


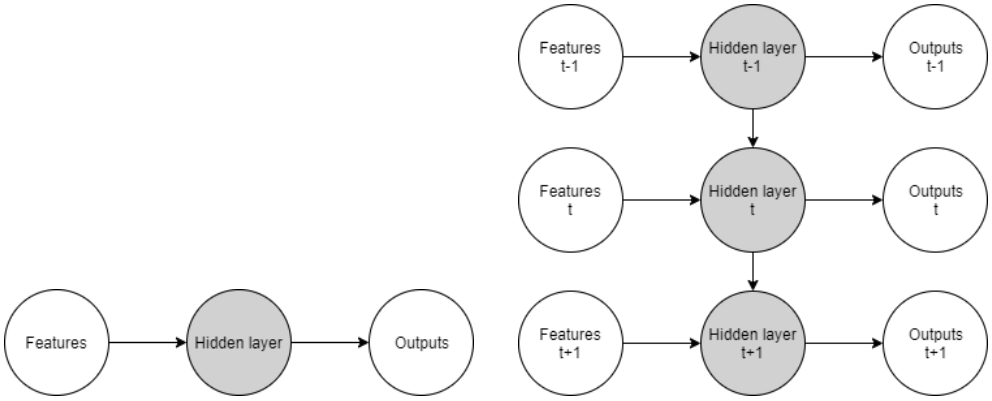
Figure 3.7: Neural network visualization

MLPs are often used in the prediction of time series, but in essence, are not a time series model. That is, MLPs do not keep track of time-dependent relations unless explicitly modeled.

Recurrent neural network

Recurrent neural networks (RNN) are an extension of MLPs which allow the modelling of time dependencies. The activation of a neuron not only depends on the input of this timestep (t) but also on the weighted activation of the previous time step (t-1). If we compress our MLP in Figure 3.7, to be represented as Figure 3.8 (a), we can depict an RNN as Figure 3.8 (b).

When we apply backpropagation to an RNN, the chain rule often results in many multiplications of gradients with respect to the weights. Two problems derive from this long chain



(a) Compressed visual of MLP *(b)* RNN

Figure 3.8: Recurrent neural network visualisation

of gradient multiplications. First, the gradient can 'vanish' as a result of many gradients being below one. This is known as the vanishing gradient problem. Second, the gradient can 'explode' as a result of many gradients being above one. This second problem is known as the exploding gradient problem. The result is exponentially growing or shrinking gradients which complicate learning. Exploding gradients prevent convergence by imposing too large weight updates while vanishing gradients prevent convergence by imposing too small weight updates which makes learning very slow.

Long-short term memory unit

Long-short term memory units are a way to address the vanishing gradients problem when modelling recurrent neural networks. This is realized through the introduction of an input gate, output gate and a forget gate which manage the flow of information within the LSTM-cell, see Malhotra et al. (2015) and Hochreiter and Schmidhuber (1997) for details. LSTM-cells in recurrent neural networks for time-series modeling have gained popularity in recent years due to their promising performance.

Reinforcement learning

Reinforcement learning is a model in which an agent is trained to perform tasks while receiving feedback (rewards) from the environment about its performance. The environment provides information which can be translated into features. In a game, this information can be the raw pixels of the screen or the random-access memory (RAM). The agent wants to maximize its reward. This is done by optimizing a value function, much alike Bellman's equation. One common algorithm for optimizing the value function is Q-learning, the value iteration updating of which can be observed in Equation (3.5). Many reinforcement learning algorithms use deep neural networks (like RRNs with LSTM units) to predict which action

leads to the highest reward.

$$\underbrace{Q^*(s,a)}_{\text{New Q-Value}} = \underbrace{Q(s,a)}_{\text{old Q*-Value}} + \alpha[\underbrace{R(s,a)}_{\text{Reward}} + \underbrace{\gamma}_{\text{Discount rate}} \overbrace{\max Q'(s',a')}^{\substack{\text{Maximum predicted reward, given} \\ \text{new state and all possible actions}}} - Q(s,a)] \tag{3.5}$$

Although, reinforcement learning is often related to Atari games, there are application possibilities for predictive maintenance as well. Examples of these are: an application of reinforcement learning to unbalanced data sets for classification problems by Lin et al. (2019), and a reinforcement learning approach for early classification for time series by Martinez et al. (2018).

Reinforcement learning can also be used to approximate the POMDP problem when the state space becomes too large to model the belief state in an exact matter or the transition probabilities are time-dependent. In predictive maintenance for vessels, the state belief of a POMDP should not merely be reflected by the prior state (t-1), because the way a vessel is used influences the RUL over time. That is, the Markov property might not hold. Therefore, it can be beneficial to represent the states through time series modeling like RNN-LSTMs which can be integrated into a reinforcement learning approach to predictive maintenance. A reinforcement learning algorithm incorporates both the RUL prediction and the maintenance optimization in one model.

3.2.3 Knowledge-based models

The third and last type of models evaluated in this literature review are knowledge-based models. Knowledge-based models, as described by Okoh et al. (2016), refer to expert-systems. Literature reveals three variants of expert-systems, which we describe here. These are:

- rule-based reasoning;
- case-based reasoning;
- model-based reasoning.

The main strength of this type of model is the inclusion of expert domain-knowledge, while the main weakness is the intensity of maintaining the knowledge base.

Rule-based reasoning

Rule-based reasoning models are composed of sets of if-then rules (rules are the knowledge base) defined by domain experts included in a model (inference engine) which, formally,

provides a consequent if an antecedent is true, see Clancey (1983). That is, it provides a decision, classification or prediction based on a set of predefined rules. This could be used to set a threshold for sensor measurements beyond which a failure is likely to occur. Experts could identify critical threshold values of one sensor reading or a combination of different sensors. As such, a rule can be composed which indicates maintenance is required.

Case-based reasoning

A case-based reasoning system is a set of cases and responses (cases are the knowledge base) gathered or defined by domain experts included in a model (inference engine) which provides a set of similar cases to the current case and possibly a prescriptive decision, Yoon et al. (1993). It retrieves similar cases, reuses those to prescribe a solution to the current case, compares the prescribed solution with reality, then retains a solution if successful or revises the knowledge-base if necessary. Predictive maintenance models can be assisted by a case-based reasoning system. This allows making a model partially prescriptive while taking into account expert knowledge.

Model-based reasoning

A model-based reasoning system is a model which mirrors the working of the underlying process, Ifenthaler and Seel (2013). It uses knowledge about the objects (for instance a vessel), assets (for instance the engine and gearbox) and relations in a specific field or industry to create a computer model representing the workings of the system. Model-based reasoning simulates an object, process or entity and is 'synced' by introducing several behavioural rules to the model. This type of model could be seen as a digital twin of the actual object.

Much like physical models this type of model is out of scope for this thesis. Yet, these models might provide valuable insight into the maintenance necessities of a vessel when validly modelled. The power of such model is that various policies can be simulated and tested before deployment without exposing an actual asset to substantial risks. It also allows to pre-train reinforcement learning models.

3.3 Class imbalance

Class imbalance is characterised by observing one class label, less often than another. Formally, *"an imbalance occurs when one or more classes have very low proportions in the training data as compared to the other classes"*, (Kuhn and Johnson, 2013, p.419). An intuitive example of naturally occurring class imbalance is fraud detection since most claims are (hopefully) just. A class imbalance results in biased models if not appropriately adjusted for because regular loss functions penalise mistakes on all targets equally. The reason for this becomes apparent when we look at an extreme class imbalance with 1 observation of

class A and many of class B. Any model could easily score 'well' by predicting all cases to belong to class B. This is especially harmfull when class A is of special interest e.g. because it is a fraudulent case or a costly engine failure. Thus, models are generally biased to the majority class if not properly adjusted for. Common techniques to correct for this bias, are undersampling (downsampling), oversampling (upsampling) and Synthetic Minority Oversampling TEchnique (SMOTE). These techniques are, however, not originally developed for time-series problems. Moniz et al. (2017) extend these three methods to address imbalanced time-series datasets. They implement each method in three ways: (1) basic resampling, (2) resampling with temporal differences, and (3) resampling with temporal differences and relevance bias. First, the basic resampling method extends undersampling, oversampling and SMOTE to the time series domain. Moniz et al. (2017) report an enhanced performance compared to priorly developed methods. Second, the temporal differences case takes into account concept drifts. Concept drifts are changes in the statistical relations between predictor and target variables over time. We do expect concept drifts in our current dataset because the operations are non-continuous. The relations between the predictors and target variable could be different when a vessel sails in cold waters than when the same vessel sails in warm waters. Thus, we could observe a concept drift if the location of the vessel's operation changes. Additionally, a concept drift could be observed when a fuel separation system or other critical component is replaced with another brand/make. Thirds, a relevance bias can be introduced to indicate how important one target value is relative to another. In the case of predicting RUL, this could boil down to assigning higher importance to low RUL values, because we are mostly interested in the last days prior to a failure. Implementation of SMOTE with temporal differences and relevance bias (SM_TPhi) method from Moniz et al. (2017) is suggested as future research.

3.4 Model choice

In this literature review, we have seen a taxonomy of maintenance policies which aims to provide a clear overview of the relation predictive maintenance has to other policies. We have defined predictive maintenance as 'a maintenance policy that utilises condition monitoring data, as well as other contextual data, to predict the future asset failures based upon which maintenance decisions can be made.' Furthermore, we shed light on four categories under which models for predictive maintenance can fall. In this thesis, we will employ models from the data-driven category to predict the remaining useful life of a fuel separation system. Specifically, the models used are a recurrent neural network with long-short term memory units, and a boosted regression tree. Recurrent neural networks with long-short term memory units can model highly non-linear data, handle the temporal dependencies of time series, are designed to address the vanishing gradient problem, and are often reported to perform well. Boosting trees excel at learning on imbalanced datasets because they are trained iteratively on

the subset of the dataset on which the model makes mistakes to build an ensemble model. This learning process makes the boosting tree especially strong for learning on imbalanced dataset, a strength that other models in our literature might lack. Furthermore, boosting trees are generally well suited to learn from large datasets due to their training speed. Yet, tree-based models lack the inherent capacity to model time dependencies and therefore require feature engineering. These data-driven machine learning approaches have the advantage that they are built upon minimal assumptions while possessing the power to resemble highly complex relations in data. Besides, the recurrent neural network with long-short term memory units is a time series model, reducing the necessity of time series feature engineering to enable the model to capture time dependencies. The recurrent neural network with long-short term memory units is the only model in our literature review which models time dependent relations autonomously (assuming that reinforcement learning does not use a recurrent neural network to estimate state-action-reward relations). To our best knowledge, we are the first to report on recurrent neural networks with long-short term memory units, and boosting trees for predictive maintenance. The exploration of their performance in this case study provides an academic contribution.

The literature review also revealed that the user might benefit from the support of a case-based reasoning expert system, which allows expert augmentation of the model. This type of model can be explored in future research. The added value of a case-based reasoning system is that prior failures or prevented failures are recorded with a responsive action. When a new failure is predicted to occur, the current pattern in sensor readings can be compared to prior case patterns. Similar cases are returned to the user, providing him or her with prescriptive information. The case-based reasoning system can thus be seen as an extension to make prediction models prescribe maintenance actions based on past experiences. When predicting the remaining useful life of new ship systems it is unlikely that sufficient failure data is captured to use data-driven approaches described in this thesis. In such cases, we advise to study the root-cause of the failure. Once the root-cause is known one would preferably alter the system to permanently prevent this failure (design-out maintenance). If the failure cannot be prevented by redesigning the system and there is very few failure observations, we recommend looking into rule-based expert systems or outlier detection models. In future research, it could also be interesting to have a look at synthetic minority over-sampling techniques for time series as proposed by Moniz et al. (2017), model-based expert systems and reinforcement learning.

Chapter 4

Methodology

This chapter aims to enhance reproducibility of the research and answers the question: *'How can we implement, validate and evaluate predictive models that are proposed for the prediction of remaining useful life of vessel components?'* It contains a description of the methods used to retrieve the results presented in this research. Specifically, we describe the chosen models and the implementation design of these models in more detail. Furthermore, we describe how we simulate a dataset which allows us to create a generalisable model tuning and choice technique. We also describe various evaluation metrics.

4.1 Conceptual framework

In order to guide the methodology, we provide a conceptual framework, see Figure 4.1. It outlines the course of action required to implement the chosen remaining useful life (RUL) prediction models.

The models have been chosen with the data structure and quality in mind, yet still require a certain data quality to assure performance. Therefore, the first check is on data quality. If data quality (sufficient number of observations and not too much unimputable missing or erroneous data) is acceptable, the data can immediately be prepared for modelling. However, if the quality is not sufficient to build, train and evaluate models, we are directed towards alternative methods. Our choice is to simulate a dataset including: (1) vessel behavior (operating states and state transitions), (2) sensor measurements (time series patterns), (3) failure behavior (degradation over time and failures). The preparation includes: (1) splitting the data in a train and test set, (2) splitting the train data in sub-train and validation sets, (3) feature engineering, (4) modelling temporal dependencies for the histogram gradient boosting tree, and (5) preparing the data dimensions for the recurrent neural network. After preparation, it is critical to choose proper hyperparameters for both models. The search for hyperparameters can be limited based on relevant literature, or practical considerations. The reduced search space is explored and exploited using a particle swarm optimization (PSO).

That is, we train and validate models with various hyperparameter settings through a PSO. Good hyperparameter settings result in models with good validation scores and fast training convergence. Our PSO stores each model, including its weights, identifier, training history, and predictions on the validation sets. This is useful for evaluating and debugging the search procedure and allows us to easily reload potentially good models because they score close to best.

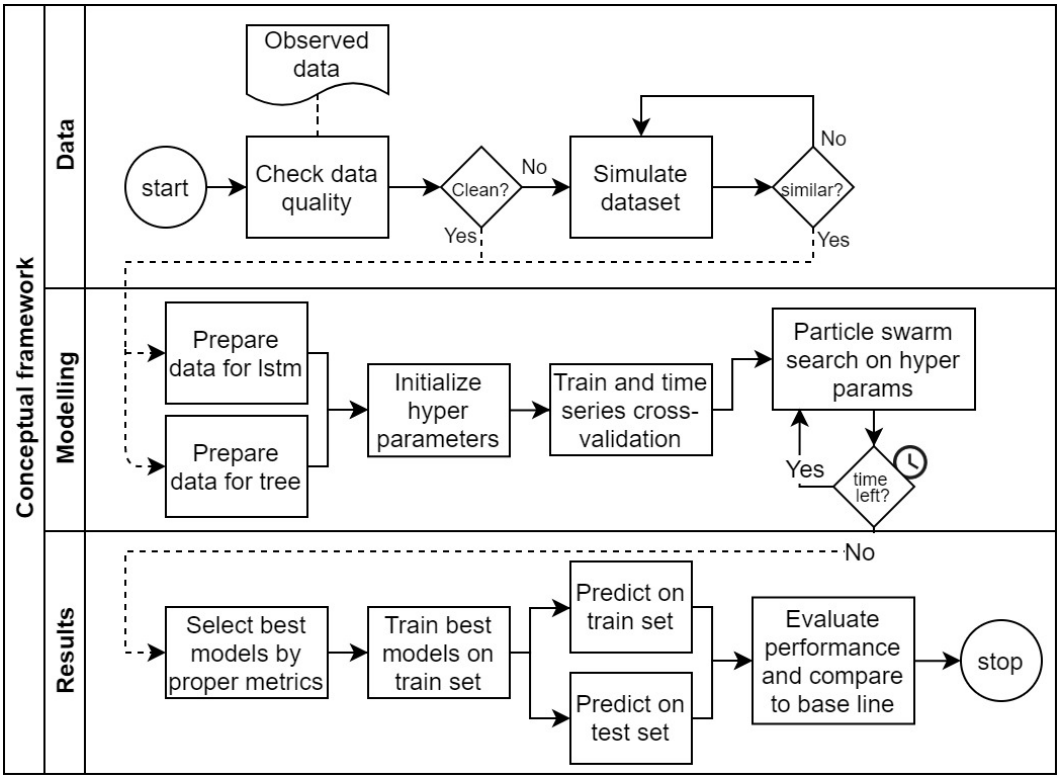


Figure 4.1: The conceptual framework

Once we have found proper hyperparameters, we use these to train a model on the complete training set (priorly, models have been trained on a subset of the training set to allow for evaluation of the generalisation error through a validation set). The trained model can then be used to predict on the test set and the final results are evaluated. Note that we also make a prediction on the training set. This is to evaluate overfitting and underfitting of the final model.

4.2 Simulated data

In this section, we explain how we simulate failure events, sensor measurements, and state transitions. In the simulation, we can ensure the existence of a relationship between indirect condition-monitoring data and failures. We also avoid any data quality issues. Therefore, if a model does not work, we can attribute this to our modelling choices. When a model does

work, we can later evaluate the performance of these models on a real dataset with sufficiently clean data. In short, we simulate the data to develop a robust methodology to build, validate and evaluate predictive maintenance models.

4.2.1 Simulation input and output

The simulation requires various inputs to imitate the behaviour of a ship on a specific mission. The user can adjust the state transition matrix probabilities, the state durations, mission duration, and the number of different missions sampled. For our simulated dataset, we sample one mission with a duration of 365 days. The used state durations and transition probabilities are specified in Section 4.2.3. The output of the simulation is a dataset with the following sensors: engine load, engine speed, fuel rate, wind speed, and generator load. Since we scope our research to one side of the vessel, we have 5 instead of 9 variables. The output also includes the remaining useful life as deducted from the timestamps when failures occurred. To assure that each entry in our dataset is accompanied with a RUL measurement, we delete the lines after the last observed failure in the dataset. From that point forward we cannot measure an historical RUL. Therefore, we work with truncated data. This dataset will be input to our RUL prediction models.

4.2.2 Insights from the observed dataset

In Section 2.3.2, we have explained the data issues and approaches to address these issues. After cleaning the raw data in 2.3.2, we analysed the ratios between each pair of variables and found that the fuel rate divided by the engine load deviates from the norm when the remaining useful life is low. Therefore, we explore the relation between the fuel rate and the engine load. Note that the variables are normalised between 0 and 1 and therefore have no units. We observe a correlation between the two variables (Pearson's correlation = 0.971, Spearman's Rho = 0.881).

We use this observations of high correlation for several features in our simulation. First, we use it to simulate the fuel rate through a process described in Section 4.2.4. Essentially, this process entails three steps: (1) fitting regressions through the scatterplot, (2) calculating a fuel rate for every engine load we observe in the simulation using the regressions, (3) sampling a residual. Second, the correlation is used to hypothesize a relation between indirect condition-monitoring data, degradation and remaining useful life. When apply the same process for engine speed, we find similar correlation values. Therefore, we incorporate the fuel rate and engine speed in our assumed relationship between indirect condition-monitoring and degradation (and thus RUL). The remaining variables generator load and wind speed are assumed not to relate to the failure of the fuel separator and therefore do not influence the degradation process in our simulation. They are however included in the simulation and used in our RUL prediction models because in future situations it might not always be known

Figure 4.2: Scatter plot observed load vs fuel rate

which variables contain relevant information and which do not. In future research, smart variable selection algorithms may be evaluated.

Our simulation should realise vessel behaviour which resembles that of the real dataset. Therefore, we look at an internal analysis (De Jong, 2020) of operating state dependent engine load distributions of the FCS2710 case vessel, see Figure 4.3.

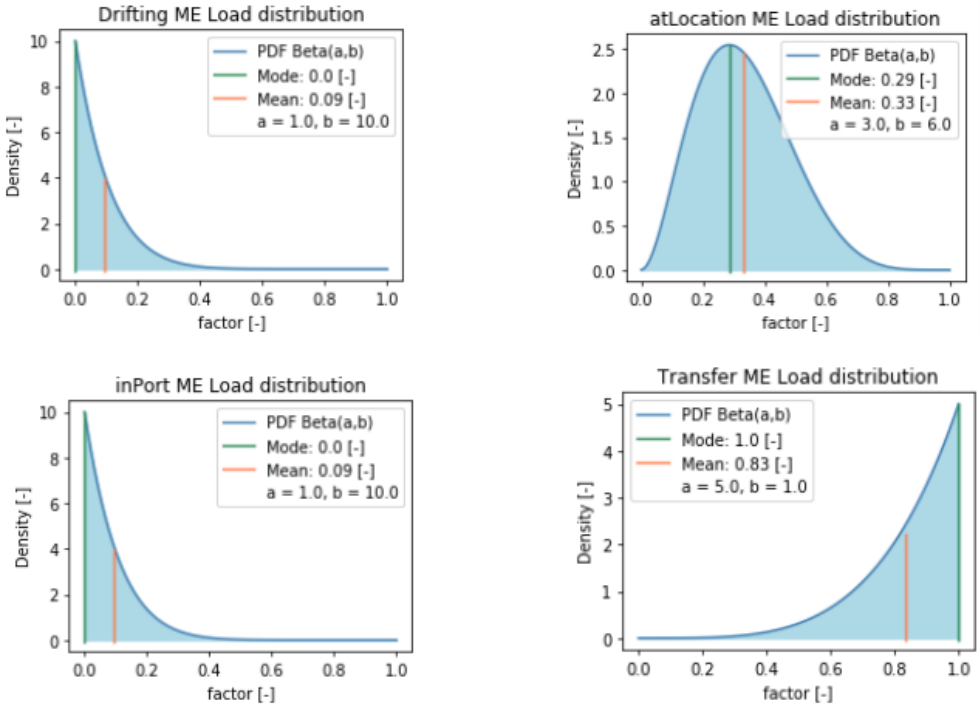


Figure 4.3: Engine load for states except berthing

In contrast to De Jong (2020), we assume an engine load of zero in the 'in-port' operating

state. The right tail observed in Figure 4.3 (left-bottom) is a result of the berthing operating state, which we thus intend to simulate as a separate state. We also looked into typical mission routes vessels embark on. In Figure 4.4, we see the mission routes of two FCS2710 over the last year (actual sailed routes in green and pink). The observed routes traverse between a port on shore and a wind farm (East Anglia One on top, Thorntonbank Wind Farm on the bottom).

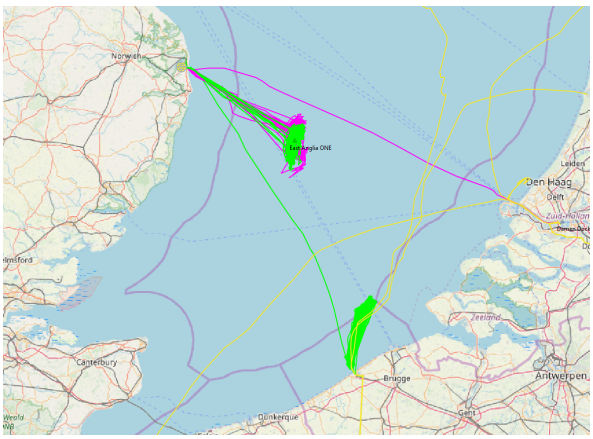


Figure 4.4: Typical mission routes FCS vessels

Figure 4.5: East Anglia One wind farm

In our simulation, we aim to reproduce the structural behaviour inherent to these typical mission routes. That means, we look at the typical distances and define a typical mission route length. We also model the observed operating states: in port, berthing, sailing, drifting, and at destination. The relation between failures and the available variables cannot be simply derived from the observed dataset. If this were possible, we would not have to do this research in the first place or would at least not simulate a dataset. However, we can recreate the structure of a real dataset and RUL prediction problem by incorporating the same failure density. The time to failure distribution in the observed dataset is visualised in Figure 4.6.

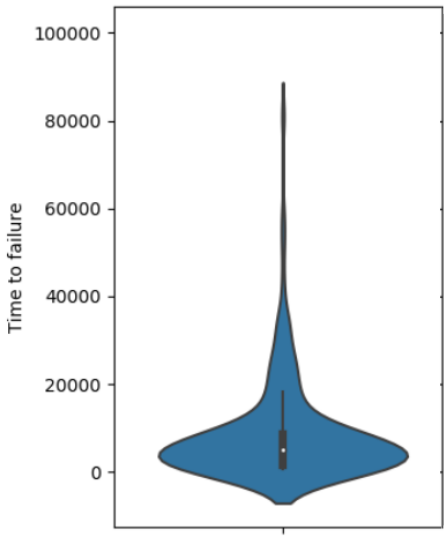


Figure 4.6: Observed failure density (time to failure distribution)

We observe a mean time to failure of 15839 time steps (1.8 days) and a median of 6569 (0.76 days). The long tail indicates a skewed time to failure distribution with large remaining useful life observations being relatively rare. Note that the y-axis can get negative in this plot because a (theoretical) Gaussian distribution was fit on the observed data. We do not observe negative time to failures in the empirical distributions in our datasets.

4.2.3 Simulating state transitions

In the Problem Analysis, Section 2.2.2 we described our specific predictive maintenance instance. One important notice was the absence of continuity of operations. Vessels do not always perform actions, neither do they operate in the same state continuously. Therefore, we model the state transitions of a vessel. Furthermore, the condition monitoring variables depend on the operating condition. This resembles the manner in which a vessel is being used over time, which influences the failure behaviour.

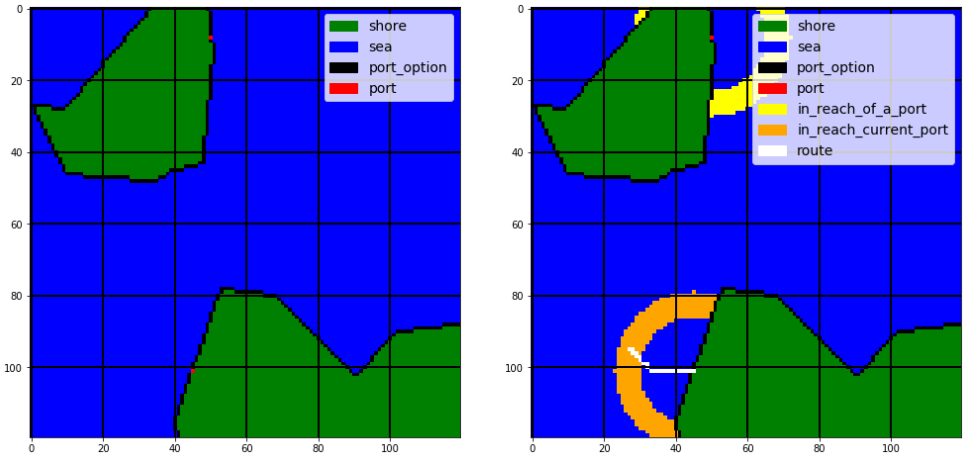
We model the failures of the fuel separation system as binary variables dependent on a set of predictor variables. Also, we add a predictor variable (wind speed) which has no direct nor an indirect relation to the alarm in order to check the robustness of our models. Robust models should not be influenced by predictors that do not possess information. That means: their performance should be similar when such predictors are added or removed.

Our simulated model is built as follows. At t=0, the ship is in port. The ship fuel separation system is as good as new at this time. From that point forward, we sample operating states [$o \in O$] which represent the behaviour of a ship and operator. The five operating states are:

- **in-port**: the ship is not sailing and the engine is turned off.
- **drifting**: the ship is waiting at sea, the ship's drift is corrected every now and then.

- **sailing**: the ship is sailing between the port and the destination platform with high engine load.
- **at-platform**: the ship is sailing engaging a platform to allow crew to board the platform with generally high engine load.
- **berthing**: the ship is approaching or leaving the port or platform with low engine loads.

The state sequence follows the logic of the observed vessel behavior. This is modelled as a Markov renewal process which samples states and state durations. In order to define how long a vessel is in the sailing state in our simulation, we sample a mission. Conceptually, we limit a mission to be the transfer of crew from shore to a platform and back. We define it by the shore and platform coordinates to define the sailing length, $m = (p_1, p_2) : r_{min} \leq |p_1, p_2| \leq r_{max}$, where r_{min} is a minimum distance after which the use of the ship is viable, r_{max} is the reach of the ship, $p_1 = (x_1, y_1)$ is the pair of coordinates of the destination port, and $p_2 = (x_2, y_2)$ is the pair of coordinates of the platform. We assume the ship is used for a given mission for $100 \leq n \leq 360$ days. Figure 4.7 a) shows one generated landscape. Figure 4.7 b) shows a generated mission route r (in white). The reach of the ship from any given port is presented in yellow. The reach from the port associated with the sampled mission is indicated in orange. After n days of operating on a given mission m, we sample a new mission. Thus, we sample one of the available ports and a platform that is within the reach of this port. A newly sampled state o_t depends on the prior states. When a ship is sailing to shore, the



a) sampled land and sea b) sampled mission route

Figure 4.7: Sampling the missing

state 'in-port' is expected after berthing, while when the ship is sailing to a platform at sea, the state 'at-platform' is expected after berthing. Thus, the state 'sailing' does not suffice to execute accurate state transitions. Therefore, we introduce states which indicate the reason for berthing and sailing. We refer to these states as Markov Renew Process states (MRP

states). The MRP states have a duration and engine load distribution depending on their 'parent' operating state. These are listed in Table 4.1.

Table 4.1: States for transition modelling

MRP state	Label	Operating state
in-port	p	port
berthing at port to leave	bpl	berthing
sailing to platform/destination	sd	sailing
berthing at destination	bd	berthing
at platform/destination	d	platform
drifting	dr	drifting
sailing to port	sp	sailing
berthing at port to arrive	bpa	berthing

The transition matrix is given in Figure 4.8 and visualised in Figure 4.9. Note that we sample a new state and a duration spend in that state as a tuple $s, duration_s$, effectively making the state transitions a Markov renewal process.

$$\mathbf{P} = \begin{array}{c|cccccccc} & p & dr & sp & sd & d & bd & bpl & bpa \\ \hline p & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ dr & 0 & 0.1 & 0.4 & 0 & 0.5 & 0 & 0 & 0 \\ sp & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ sd & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ d & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ bd & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ bpl & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ bpa & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

Figure 4.8: Transition matrix

Each mission is executed for n days by continuously following the rules of the transition matrix P. The time spent in a state is dependent on the newly sampled state and in the case of sailing also on the current mission route length $|r|$. With every state transition, a state duration $\phi(o)$ is sampled from the related state duration distribution, see Equation 4.1. If a mission route is longer, the ship spends a longer time sailing. The route in the simulation is generated by iteratively taking a step in the direction of the destination platform. The step can be in the x or y direction (chosen is the direction in which the highest distance is

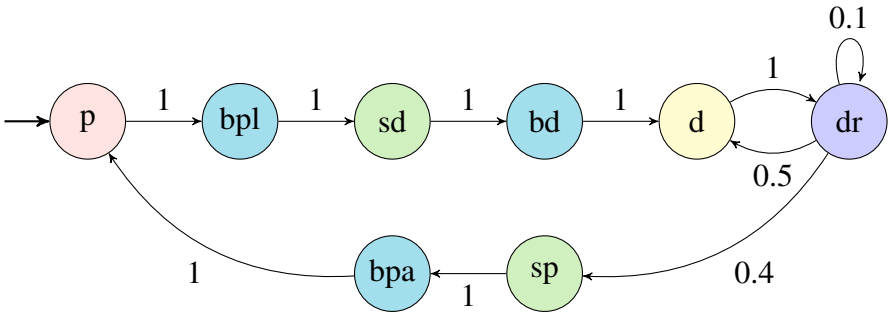


Figure 4.9: Transition diagram

still to be bridged), thus resulting in a Manhattan distance route length in a deterministic case. However, a deterministic step is altered with a chance of ρ percent. As a result of this random perturbation, some routes are longer than would be realistic. Therefore, routes are discarded when $|r| > UB_{dist_m/dist_e} * R_{max} * (1 + \rho)$, because this is the Manhattan upper bound (including a stochastic factor). Let x be the distance travelled in a horizontal direction following a Manhattan route. The remaining vertical distance to travel to reach a point on a circle with radius $dist_e$ is then: $\sqrt{dist_e^2 - x^2}$. The upper bound of the ratio between Manhattan distance and Euclidean distance $UB_{dist_m/dist_e}$ to reach any point on that circle is derived in Appendix A.1. This upperbound is used to define a maximum vessel reach when sampling new missions in the simulation. A typical mission, such as that depicted in 4.4 has a mission route length of 70km. The average speed of an FCS2710 is 40kph and thus we model the sailing duration state as the mission route length (in km) divided by 40kph.

The state durations are modelled as deterministic values which were queried through interviews within the firm. The state durations are:

$$\begin{array}{r||r||} operating state & state duration \\ \hline\hline port & 2 \\ drifting & 1.5 \\ sailing & |route|/speed \\ platform & 0.2 \\ berthing & 0.2 \\ \hline\hline \end{array} \tag{4.1}$$

Note that the MRP state durations follow the operating state durations.

4.2.4 Simulating sensor measurements

We have already seen that our state transitions follow a Markov renewal process. The evolution of sensor measurements in our simulated dataset is based on the evolution of sensor measurements in the observed dataset. Each of the sensor measurements in our simulated dataset is provided as a value normalised between 0 and 1. Since the vessel has a port side

and starboard which operate independently from each other, we limit the analysis to one side of the vessel (the starboard). Sensors included in our analysis are: engine load, engine speed, fuel rate, wind speed, and generator load. In every state, the engine load has a mean towards which the engine load will evolve within that state. This modelling choice is best argumented through an example: while sailing (=state), the most frequently observed engine load is its cap. Thus, we let the engine load evolve towards its cap once the vessel enters a sailing state. We also introduce a variability factor in the evolution of the engine load. The evolution of the engine load is modeled as $\Delta_{load,t} = (N \sim (\mu, \sigma)$ where $\mu = load\_goal_o - load\_value_{t-1})$ and $\sigma = 0.01$ where o is the state and t the is time step of the simulation.

For the evolution of the fuel rate and engine speed values, we follow the following procedure. We have fit a regression on the observed engine load (x) and the observed fuel rate or engine speed (y) while splitting the data in 50 engine load-dependent bins and a train and test set. This results in 50 models predicting the fuel rate and 50 models predicting the engine speed. After fitting the regressions on the training sets, predictions are made on the test sets and the residuals are stored. In the simulation, the simulated engine load is used to predict a fuel rate or an engine speed with the trained regression models. To this prediction, a random residual from the respective engine load-dependent bin is added. The wind speed is modeled as a random walk and the change in generator load follows the change in engine load plus a random normal variability component.

4.2.5 Simulating failure behaviour

The simulation assumes that the evolution of the indirect condition monitoring variables influences the failure behavior of the fuel separation system. This failure behavior is modelled to be degradation-based. That means, that using our indirect-condition monitoring sensor measurements, we construct a degradation variable. Our degradation variable is an abstract variable with no unit, but we can see it as a percentage. It is scaled between zero and one (where zero means no degradation and one is the failure threshold). We model degradation over time such that simulated failure frequency is in line with that observed in the real dataset.

Each time step a deviation from the norm is calculated which determines the degradation. This degradation is compared to a predefined total health (total health is one in the simulation) of the vessel component. When the total degradation is larger than the total health, a failure occurs. At that point, a corrective action takes place and the degradation is reset to zero. We also model a preventive action that resets the degradation to zero every three days. We model the degradation as a compounding multiplicative process, where a new degradation is multiplied by the existing degradation.

$$degradation_t = degradation_{t-1} * degradation\_factor_t \tag{4.2}$$

The assumption in our simulation is that the engine load and the fuel rate (also engine speed) have a positive correlation and that relatively large deviations result in degradation.

Our analysis in Section 2.3.2 has shown that when the remaining useful life is low (≤ 3 days), the fuel usage is likely to be less efficient under equal engine loads than when the remaining useful life is high.

$$degradation\_factor_t = 1 + |deviation_t|^{10} \tag{4.3}$$

Equation 4.3 computes the degradation at each time step in the simulation. Higher exponents flatten the curve, meaning less extreme deviations result in less severe degradation. The exponent was chosen such that the number of failures in the simulation is (on average) equal to that in the observed dataset. See Figure 4.10 for a visualization of the behaviour of this function. It shows exactly the behaviour, we would like to see under the assumption that high deviations result in high degradation.

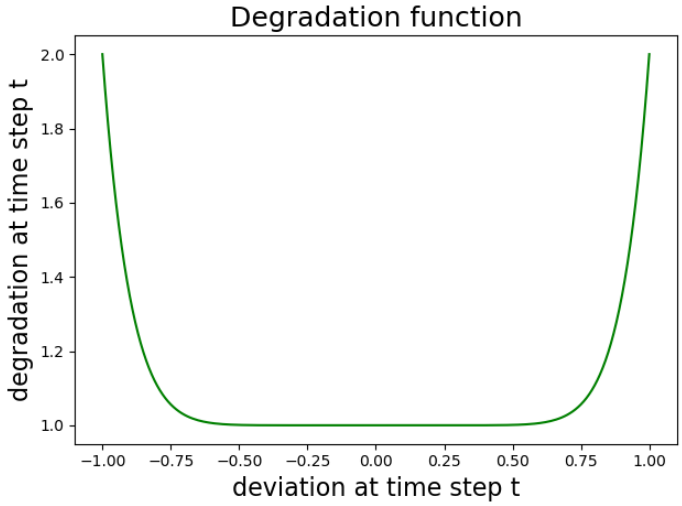


Figure 4.10: Degradation function visualisation of Equation 4.3

More specifically, the degradation is assumed to be dependent on the average of the difference between the fuel rate and engine load, and the difference between the engine speed and engine load.

$$deviation_t = \frac{(speed_t - load_t) + (fuel_t - load_t)}{2} \tag{4.4}$$

The reason for modelling the degradation like this is that we assume it to be feasible to distract a degradation pattern from the available indirect-condition monitoring data. If we were to model a degradation pattern which is independent of the available indirect condition-monitoring data, there would be no use in creating a simulation which follows the observed vessel behaviour. However, if we assume a relation to exist between the degradation and the available indirect condition-monitoring data, it is likely to be a complex relation. Recall that models should learn the relationship between indirect-condition monitoring and the remaining useful life. The models do not observe the degradation process directly and have

to learn the degradation function (as a hidden/latent variable) from the indirect-condition monitoring data in order to accurately predict the remaining useful life. Thus, the deviation factor follows a formula used to map a set of sensor measurements to degradation. This is done to introduce complexity in the failure behaviour of our simulation such that prediction of the remaining useful life is not an easy task. Note that we do not know how the sensor measurements relate to a real degradation pattern as degradation is not an observed variable in the real dataset.

We also model a time-based preventive maintenance action in our simulation because it would not be realistic to assume that a critical component is not preventively maintained. In practise, we observe a routine maintenance inspection procedure called 'the daily round'. Included in this procedure is the fuel separator inspection every three days. Assuming that obvious, soft failures can be detected by inspection, we assume perfect maintenance. Thus, after a preventive maintenance action, the component is as good as new. The resulting degradation pattern (with the modelled failure threshold) is shown in Figure 4.11. The component fails when degradation is higher than the total health. The degradation in Figure 4.11 is normalised between 0 and 1. Note that the degradation is zero right after a failure and right after a preventive maintenance action, while it only reaches one in case of a failure (excess degradation). This process is a modelling choice to simulate the soft failures. We observe a soft failure approximately every three days.

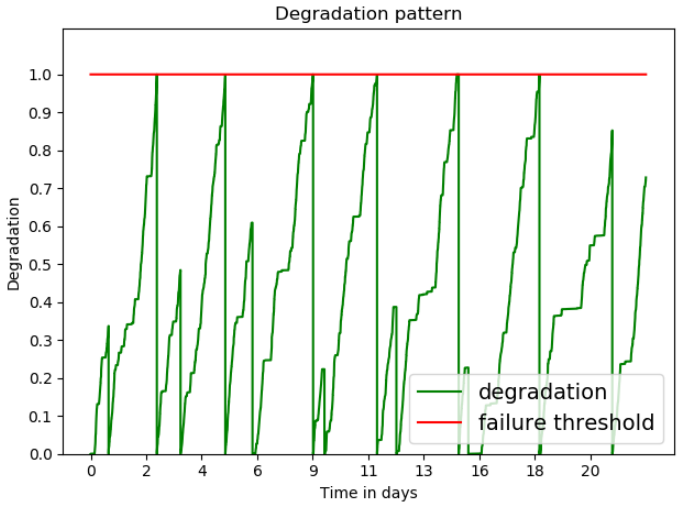


Figure 4.11: Degradation pattern visualisation

The resulting dataset is described and analysed in Chapter 5.

4.3 Model validation strategy

In this section, we describe the way in which we validate our models. This includes the used evaluation metrics and an assessment of their suitability, and the splitting of our dataset to prevent overfitting on the dataset.

4.3.1 Evaluation metrics

The models of which the performance on this predictive maintenance case will be evaluated are a recurrent neural networks with long short term memory units and a boosted trees. We compare the performance of our models with a simple baseline model. The baseline is $max(0, mtbf - tf))$, where tf is the time since last failure and $mtbf$ the mean time between failure are derived from a training set. Note that the baseline resembles an optimal time based preventive maintenance policy if the maitenance threshold is optimized. The evaluation of the models is not trivial because some errors are more costly than others. In this subsection, we try to find relevant evaluation functions to assess and compare the performance of our predictive maintenance models. The core of the evaluation problem is that predicting lower than actual RUL might result in unnecessary maintenance, while predicting higher than actual RUL might result in unprevented failures (this is visualized in Figure 4.12). We say might, because the actions taken in a maintenance cycle and the severity of prediction errors are not independent. We evaluate the following metrics in this section: mean absolute deviation, mean squared logarithmic error, root mean squared relative error, R2, and a cost-based optimization metric. We use these metrics to evaluate the performance of the models after predictions are made. Thus, do not develop cost-dependent loss-functions to use during the training of prediction models.

In the following equations, $\hat{y}$ represents the vector of RUL predictions in seconds/10 (the measurement interval in the real dataset), y represents the vector of RUL actualizations in seconds/10. The mean absolute deviation is a simple metric, which we can use to get an idea of the general error size. The units of this measure is the same as that of our target variable (RUL): number of times 10 seconds.

$$mad(\hat{y}, y) = \frac{1}{T} \sum_{t=0}^{T} |y_t - \hat{y}_t| \tag{4.5}$$

The mean absolute deviation penalises errors that result in failures (too high RUL predictions) equally hard as errors that result in too early maintenance (too low RUL predictions). Also, predictions that do not influence our maintenance decisions are penalised by the mean absolute deviation, which is not ideal and makes a good evaluation more difficult. Is the mean absolute deviation high because we have many harmless under- and overpredictions or because we have some influential under- and overpredictions? Both the msle and the rmsre penalize errors more severely when actual or predicted RUL is low. Since low actual RUL

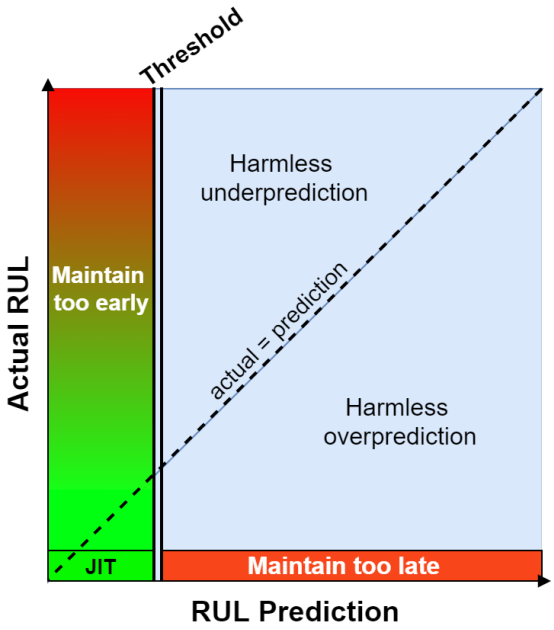


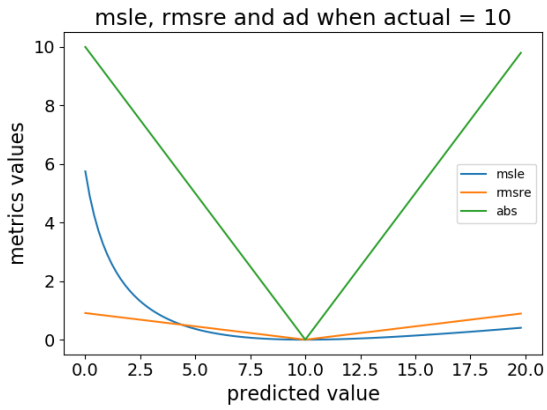
Figure 4.12: Caption

can result in failures and low predicted RUL in a maintenance decision, this is the area of interest for error evaluation.

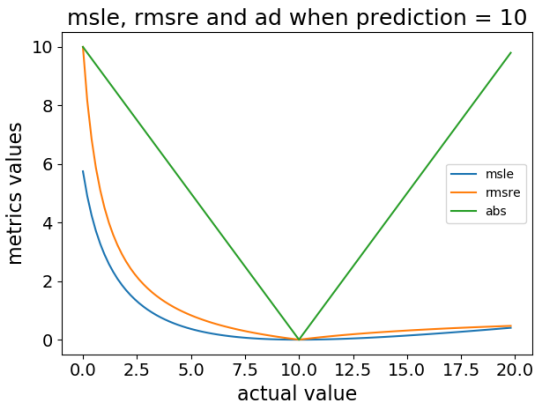
$$msle(\hat{y}, y) = \frac{1}{T}\sum_{t=0}^{T}\left[\left(\ln(y_t + 1) - \ln(\hat{y}_t + 1)\right)^2\right] \tag{4.6}$$

$$rmsre(\hat{y}, y) = \sqrt{\frac{1}{T}\sum_{t=0}^{T}\left[\left(\frac{\hat{y}_t - y_t)}{y_t + 1}\right)^2\right]} \tag{4.7}$$

Now, lets further evaluate the behavior of these error metrics and compare them to the absolute deviation. In Figure 4.13, we observe a fictive case of predictions of the RUL and the actual RUL values. In Figure 4.13 a), we observe the behaviour of the error metrics with a fixed actual RUL of 10 while varying the predicted RUL. The mad (green), msle (blue) and rmsre (orange) all penalise predictions that are lower than actual. Note, that we exclude the R2 metric and the cost-based optimization metric from the visualization because we cannot objectively quantify tuples of a prediction and an actual. The R2 metric is dependent on the variation of the actual which cannot be calculated from a single actual value, and the cost-based metric depends on a full history of predictions and decisions. R2 and the cost-based metric are discusses hereafter. In Figure 4.13 b), we observe the behaviour of the error metrics with a fixed RUL prediction of 10 while varying the actual RUL. All three metrics penalise overpredictions with low actual RUL ≤ 5. An interesting observation is that the msle penalises underprediction (4.13a) the same as it penalises overprediction (4.13b), whereas the rmsre penalises overprediction more severly. The reason for this is the division by the actual RUL in the rmsre.



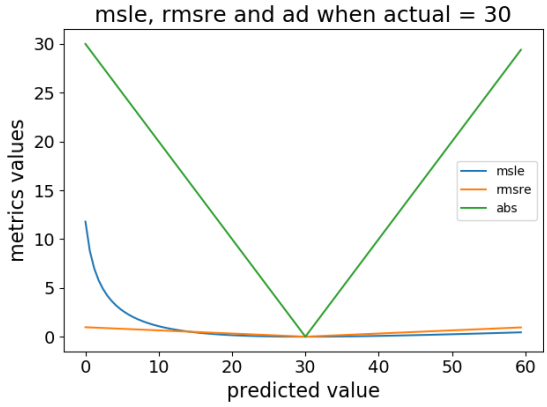
a) prediction from 0 to 20 actual = 10



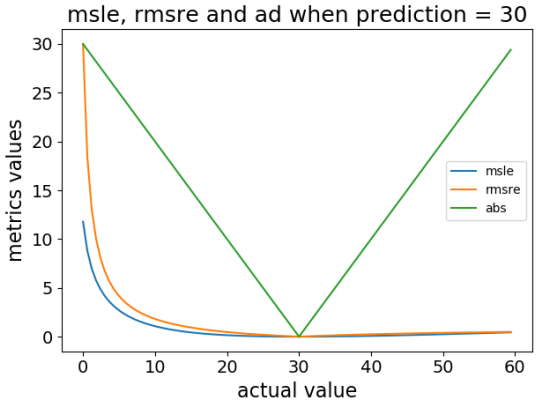
b) actual from 0 to 20 prediction = 10

Figure 4.13: Metrics with low actual RUL

For exhaustiveness, we show the behaviour of our metrics with higher RUL values in Figure 4.14. We immediately see that the penalty assigned by the absolute deviation is independent of the magnitude of the RUL, while the msle and rmsre are not.



a) prediction from 20 to 30 actual = 30



b) actual from 20 to 30 prediction = 30

Figure 4.14: Metrics with high actual RUL

From these three metrics, we can derive that the msle might be a better choice if the costs corrective failures are about the same as that of preventive maintenance, while the rmsre might be a good choice in the case where corrective maintenance is more costly than preventive maintenance. However, neither metric takes into account the decisions or events that have already taken place in a real maintenance cycle. If a preventive action took place due to an accurate prediction of the RUL, the costs of a corrective action should not be incurred. For this reason, we develop a cost-based maintenance metric. Additionally, we report the R2 value of the models to provide a more complete and unbiased image of the

model performance irrespective of the maintenance cost structure.

$$R2(\hat{y}, y) = 1 - \frac{\sum_{t=0}^{T} \left[(y_t - \hat{y}_t)^2\right]}{\sum_{t=0}^{T} \left[(y_t - \bar{y})\right]^2} \tag{4.8}$$

What we can conclude from the evaluation metric analysis thus far, is that good scores on the rmsre likely result in low corrective maintenance costs, because this metric penalises overpredictions harsher than underpredictions. However, since the rmsre barely punishes underprediction it allows very preventive maintenance policies to be evaluated as good policies. The msle corrects this behaviour, but penalises overpredictions and underpredictions equally, resulting in costly maintenance policies if corrective costs are higher than preventive costs.

An ideal metric would penalise errors on RUL predictions only when there is a consequence for the costs of the maintenance policy. For this reason, we develop an evaluation metric that incorporates the costs of prediction errors. Future research could look into ways to combine the optimization of the maintenance costs in concurrence with the prediction model like described in Elmachtoub and Grigas (2017). Equations 4.9-4.12 explain the construction of our cost-based evaluation metric. The total maintenance costs are:

$$cost\_metric(\hat{y}, y) = \sum_{t=0}^{T} \sum_{s=0}^{S} \left[prev_{s,t} * C_{prev,s,t} + cor_{s,t} * C_{cor} \right] \tag{4.9}$$

In this formula, s is the cycle identifier, t the time step in a cycle, C_{prev} the earliness dependent preventive costs and C_{cor} the corrective costs.

We define two binary variables, $prev_t$ and cor_t to model the points in time when costs are incurred. The costs of a preventive maintenance action are accounted for when the model predicts a RUL beneath than a maintenance threshold. Also, the system should not have failed before the RUL prediction passes the threshold, because then a corrective event was already present in that cycle. Furthermore, a maximum of one preventive maintenance action will be accounted per maintenance cycle, see Equation 4.10. Therefore, the costs of a preventive maintenance action depend on how much too early the action took place. Like that, we incorporate the costs of a missed exploitation opportunity.

First, the preventive costs are incurred when a prediction $\hat{y}_t$ is below a threshold and there was no other maintenance action earlier in that cycle.

$$prev_{s,t} = \begin{cases} 1 & \text{if } \hat{y}_t \leq thres \quad \wedge \quad \sum_{\tau=0}^{t} prev_{s,\tau} + cor_{s,\tau} = 0 \quad for \quad t = [0, T_s], \quad \forall s \\ 0, & \text{otherwise} \quad \forall t, \quad \forall s \end{cases} \tag{4.10}$$

The corrective costs are incurred when the actual RUL reaches zero before the RUL

prediction crosses the defined maintenance threshold.

$$cor_t = \begin{cases} 1 & \text{if } y_t = 0 \quad \wedge \quad \sum_{\tau=0}^{t} prev_{s,\tau} + cor_{s,\tau} = 0 \quad for \quad t = [0, T_s], \quad \forall s \\ 0, & \text{otherwise} \quad \forall t, \quad \forall s \end{cases} \tag{4.11}$$

Finally, we define the preventive maintenance costs as a function of the prediction error, see Equation 4.12. Larger errors result in higher costs because overly-preventive maintenance shortens the exploitation of the component at hand.

$$C_{prev,\,t} = prev\_costs + \frac{earliness * mean\_cycle\_costs}{mean\_cycle\_time} \quad \forall t \tag{4.12}$$

Two numerical examples:

- costs of a preventive action are 200 euros for both cases;
- maintenance threshold is 8 time steps for both cases;
- RUL prediction is 5 time steps for case 1 and 10 for case 2;
- actual RUL is 10 time steps for case 1 and 500 for case 2;
- mean cycle time is 1000 time steps for both cases;
- the mean cycle costs are 280 for both cases;

In the first case, preventive costs are $200 + 5 * 1000/280 \approx 217$, while in the second case the preventive maintenance costs are $200 + 500 * 1000/280 \approx 1986$. However, since the maintenance threshold is 8 time steps, and the predicted RUL for case 2 is 10, no preventive maintenance action is executed in case 2. Therefore, these costs are not incurred. Note that these equations could be transformed into an integer linear program by which we can optimise the threshold to minimize the resulting costs of the predictive maintenance policy. The linear program for this can be found in Appendix A.2.

4.3.2 Validation split

To avoid overestimation of the performance of our models, we split the dataset into a train and a test set. The train set will also be used to validate our models. For validation, the train set is further split in k sub-train and sub-validation sets. The sub-train and sub-validation sets are used to evaluate the performance of the two models with different hyper-parameter settings. The best performing hyper-parameter settings for each of the two models are then evaluated by training on the complete training set and evaluating on the test set. This procedure is followed to increase the performance of the models on unseen cases.

When validating a model it is important to keep the dependencies in the dataset in mind. That is because the model is not allowed to learn anything it would not be able to learn in practice. To be explicit, the model may not learn from data which is to be retrieved in the

future, but only from historical data. For our time series problem we distill the following requirements for proper validation:

- a training set may not have overlap with a validation or test set;
- the training set, as well the validation and test set must be chronologically ordered;
- the timestamps of the training set must precede the validation and test set;

To satisfy these requirements, we split the dataset as depicted in Figure 4.15. Note that

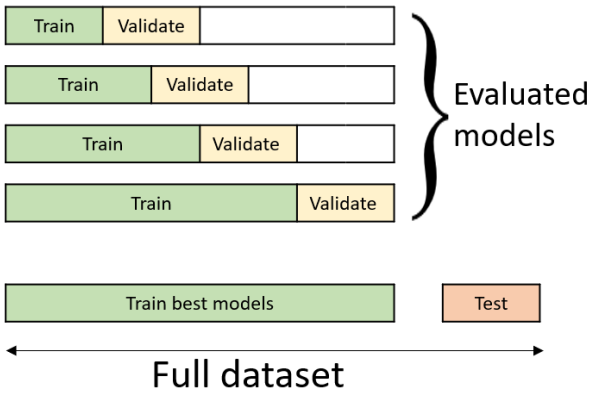


Figure 4.15: Train, test and validation splits. Own elaboration.

the test set is kept aside and never exposed to our models until the final evaluation of a few chosen models with the best validation scores. Algorithm 1 shows the pseudo-code for the implementation of our validation strategy. However, the choices for hyper-parameter settings pose an infinite solution space for each of the two models. As training a model to evaluate its hyperparameters is time-consuming, a smart hyperparameter tuning strategy is required. That is, we search an algorithm that minimizes the total computational time (number of times to execute Algorithm 1) while retrieving a good set of hyper-parameters.

To achieve this objective, we implement a particle swarm optimization, see Algorithm 2 in Appendix A.3. This algorithm optimizes several parameters of our models, but not all. Tuning all model parameters would be too time-intensive. Besides, literature and practical considerations can limit the search space considerably.

The hyper-parameters to be tuned differ for the three models we implement. In the following, we will discuss the choices made regarding each of the model implementations including which of the parameters are tuned through the particle swarm optimization.

4.4 Modelling choices

This section describes the modelling choices to be considered when building recurrent neural networks and boosting trees.

Algorithm 1: hyper-parameter evaluation procedure

```
1:  procedure EVALUATE(model, training_sets, validation_sets)
2:      validation_scores ← empty_list
3:      for all training_sets do
4:          trained_model ← train_model(model, training_input, training_target)
5:          prediction ← predict(trained_model, validation_input)
6:          model_score ← evaluate(prediction, validation_target)
7:          append(validation_scores, model_score)
8:      end for
9:      validation_performance ← mean(validation_scores)
10:     return validation_performance
11: end procedure
```

4.4.1 Recurrent neural network

Unlike vanilla artificial neural networks, RNNs with LSTM units (LSTM for convenience) process sequences of inputs with arbitrary length while keeping memory of prior inputs. In this process, LSTMs aim to learn temporal relations between inputs and outputs. With an LSTM, we model the relations between sensor behavior and RUL over time.

The input of an LSTM is different from that of time-independent models in that it requires the input the be a tensor with shape (samples, time steps, features) rather than a matrix with shape (samples, features). Thus, instead of a sample being a line of observed sensor measurements, a sample is a set of sequentially observed sensor measurements. See Figure 4.16 for a visualization of the required data structure. We chose to let sample $i + 1$ start where sample i ends.

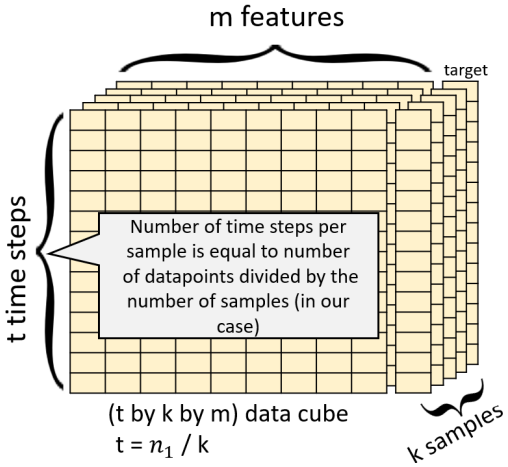


Figure 4.16: The input shape for an LSTM units. Own elaboration.

To achieve decent performance, we tune some hyper-parameters with Algorithm 2. How-

ever, according to (Hochreiter and Schmidhuber, 1997, p.23) the performance of LSTMs is relatively insensitive to hyper-parameter settings. Therefore, we argue for some of the hyper-parameters through the use of literature or practical considerations. See Table 4.2 for a list and description of considered hyper-parameter settings. The hyper-parameters remaining to be tuned through the particle swarm optimization are: the number of hidden layers and dropouts. Dropouts are a popular regularisation technique. The idea is that each neuron in a neural network is 'dropped out' during the training procedure with a certain probability. This requires the neural network to be more robust.

See Figure 4.17 for a visualisation of the validation procedure. This procedure is followed for each set of hyper-parameter settings evaluated in the particle swarm optimization.

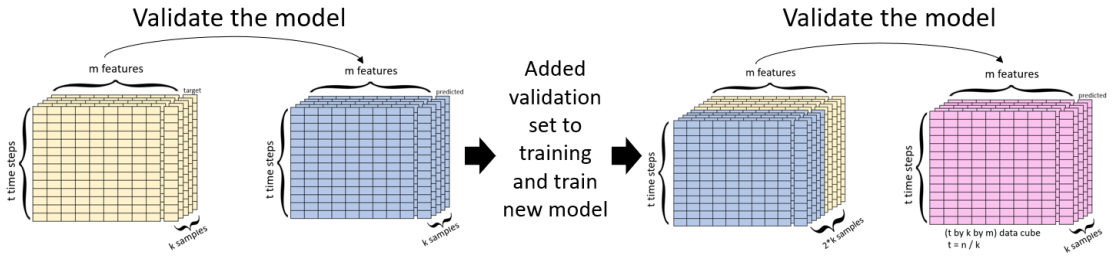


Figure 4.17: The validation of LSTMs. Own elaboration.

4.4.2 Boosting tree

Various implementations exist of boosting trees, examples are: XGBoost, LightGBM, LogitBoost, and AdaBoost. The implementation we choose to use is 'Histogram-based Gradient Boosting' from Pedregosa et al. (2011). The authors' claim that this implementation can be orders of magnitudes faster than regular gradient boosting algorithms and is especially aimed at prediction on larger datasets with more than tens of thousands of samples. This implementation is based on LightGBM Ke et al. (2017), which aims to improve the accuracy of gradient boosting trees while reducing memory usage and increasing speed.

We scope our search for proper hyper-parameters of the gradient boosting tree to the most relevant ones: the learning rate, the number of iterations, and the size of each tree, see Table 4.3.

Hastie et al. (2001, p.365) state that *"(...) the best strategy appears to be to set v to be very small ($v < 0.1$) and then choose M by early stopping."* Here, v refers to the learning rate in gradient boosting and M to the number of iterations. We, therefore, search a proper value for the learning rate between 0 and 0.1, through particle swarm optimization. Early stopping evaluates the model on a given scoring metric after each training iteration, which requires a validation set. In the implementation by Pedregosa et al. (2011), only a validation split rate can be provided, based upon which the train and validation sets will be randomly sampled. However, we cannot use early stopping for the gradient boosting tree as we did in

Table 4.2: LSTM hyper-parameter choices

Param.	Class	Tuned through	Settings	Important for	Reference
Learning rate	Speed	Literature	0.001	The learning speed of weight optimization (too high = local optima, too low = slow learning)	(Kingma and Ba, 2014, p.2)
Optimizer	Algorithm	Literature and practical	Adam	Global search for optimal weights.	(Kingma and Ba, 2014)
Weight initializer	Initialization	Literature	Xavier (Glorot uniform)	Learning speed & avoid exploding gradient.	(Glorot and Bengio, 2010, p.251)
Number of iterations	Intensity	Practical	1000 epochs	Too many would results in overfitting, to few in underfitting.	NA
Hidden layers	Complexity	Practical, particle swarm	Between 1 and 3	In conjunction with number of neurons determines the complexity a model can take on.	NA
Neurons per layer	Complexity	Practical	2^h where h is the hidden layer number counting from the last	In conjunction with number of layers determines the complexity with which relations	NA
Time steps	Temporal dependencies	Particle swarm	Between 5 and 5000	Higher number of time steps allows to remember information from longer ago but is more costly to train.	NA
Early stopping	Regularisation	Practical	Yes, when validation score does not improve.	Reportedly successful regularisation technique.	Girosi et al. (1998)
Dropouts	Regularisation	Literature, particle swarm	Yes, particle swarm decides where and the rate	Reportedly successful regularisation technique.	Krizhevsky et al. (2012)
Activation function	Gradient computation	Practical	Exponential linear unit (ELU)	Fast learning without dying neurons or vanishing gradient	NA
Gradient clipping	Gradient computation	Practical	Clipped at [-10,10]	Avoids exploding gradient problem	NA

the LSTM, because the random split implementation of Pedregosa et al. (2011) does not take into account time dependencies and would violate all three of our validation requirements stated earlier. Therefore, we add the number of iterations to the list of hyper-parameter to be chosen through a particle swarm optimization. Finally, we consider the size of each tree (weak learner) which is to be a component of the final model. The size can be limited in two ways: setting the maximum depth or setting a minimum number of observations per leaf. We limit the search by only considering the maximum depth because both should have the

Table 4.3: Gradient boosting tree hyper-parameter choices

Parameter	Class	Tuned through	Settings	Important for	Reference
Learning rate	Speed	Literature and particle swarm	< 0.1	The learning speed of weight optimization (too high = local optima, too low = slow learning)	(Hastie et al., 2001, p.365)
Number of iterations	Intensity	Practical and particle swarm	between 10 and 1000	Too many results in overfitting, to few in underfitting.	NA
Maximum depth	Regularisation	Literature and particle swarm	Between 2 and 9	Too many results in overfitting, to few in underfitting.	Hastie et al. (2001, p.412)

same effect. Hastie et al. (2001, p.412) set the maximum depth of their models between 2 and 9. This range will be input to our particle swarm optimization for this hyper-parameter. Besides these hyper-parameters, we will use Ridge-Regression (Hoerl and Kennard, 1970) as a regularisation method to mitigate overfitting.

The training, evaluation and testing procedure described here is implemented in Sci-kit learn in Python. The results are reported in Chapter 5.

4.5 Methodology conclusion

From our methodology, we can conclude that the best evaluation metric is a cost-based one. This metric evaluates the cost of implementing a predictive maintenance policy. We also conclude that we can limit the search for good modelling parameter settings to a selected amount of hyper-parameters for both the recurrent neural network and the boosting tree. Good hyperparameters are searched with our implemented particle swarm optimization, see Appendix A.3.

Chapter 5

Results

This chapter shares the results of the research. It answers the question: *'How do the chosen predictive maintenance models perform?'* To address this question, we first describe the simulated dataset. Hereafter, we report on the performance of the prediction models.

5.1 Simulation results

The (used) simulation output is a dataset with 2.6 million time steps and 9 features. These are: time stamp, engine load, fuel rate, engine speed, wind speed, generator load, sensor deviation, time since last failure, and remaining useful life. The simulation results in datasets which resemble the observed vessel behaviour. The benefit of the simulation is that we can generate clean datasets of arbitrary length which enlarges the set of actions available while training and validating the models.

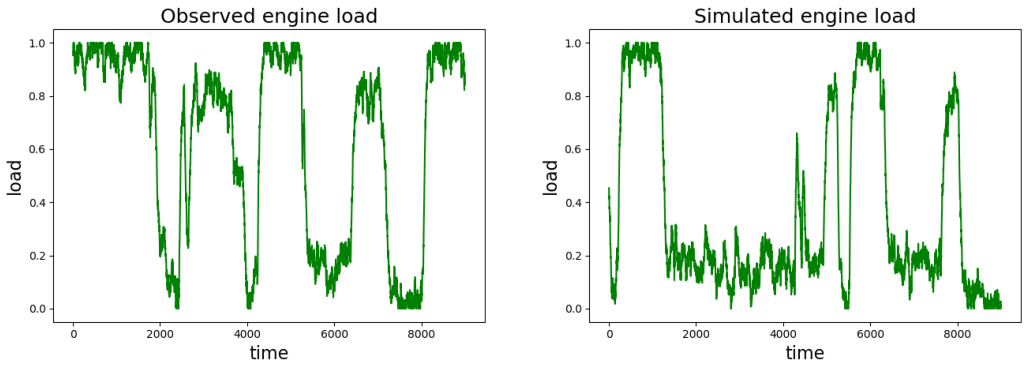
5.1.1 Sensor resemblance

We simulate the vessel behavior through sampling the indirect condition monitoring data. The resulting simulated sensor measurement means shows strong similarity with the observed sensor data, see Table 5.1. Values in parenthesis are the mean sensor measurements observed in the real dataset. First, we examine the engine load on the basis of a time series visualisation,

Table 5.1: Mean and standard deviation stats

Variable	Mean	Standard deviation
Engine load	0.54 (0.49)	0.37 (0.37)
Fuel rate	0.43 (0.39)	0.36 (0.36)
Engine speed	0.45 (0.42)	0.29 (0.30)
Generator load	0.50 (0.16)	0.31 (0.17)
Wind speed	0.50 (0.11)	0.36 (0.14)

see Figure 5.1. In reality, the state durations are not deterministic and the actual state



a) Observed engine load over time b) Simulated engine load over time

Figure 5.1: Engine load simulated and observed over time

transitions do not always follow our assumptions. Therefore, the observed patterns deviate from our simulation. Yet, even if our assumptions would hold, we can expect to see different patterns over time. The extent to which the observed engine load and simulated engine load are similar over time depends on the time window which is considered. Figure 5.3, provides a better visualisation of the similarities and differences of the observed and simulated dataset and Figure 5.2 shows the development of the engine load with the states over a selection of the data. As intended the engine load is zero in the port. Hereafter, the vessel is berthing for a small period of time before sailing towards the destination.

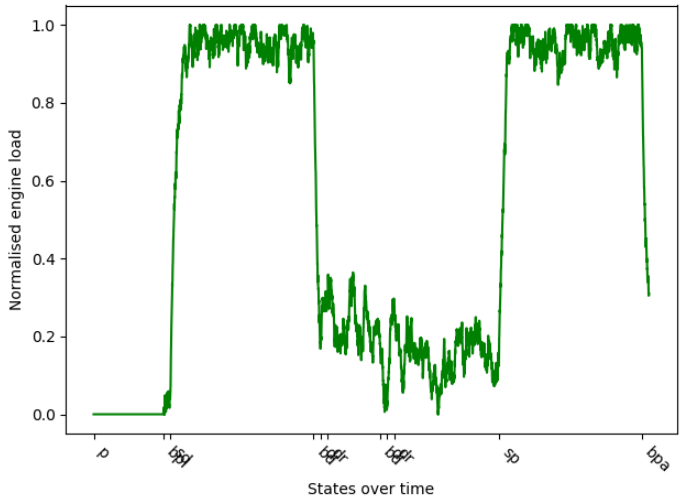


Figure 5.2: Engine load simulation evolution over time

The engine load exposes vessel behavior in concordance with the various operating states. We observe a higher density for low and high engine loads. Low engine loads are observed while berthing and drifting when the engine is mostly running stationary. High engine loads

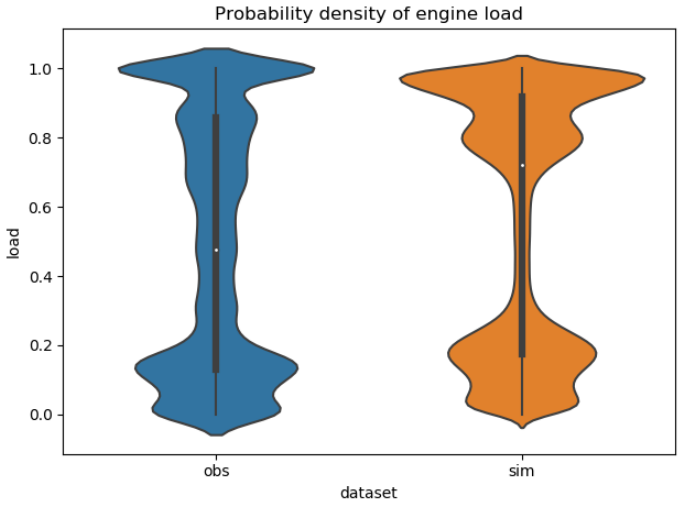
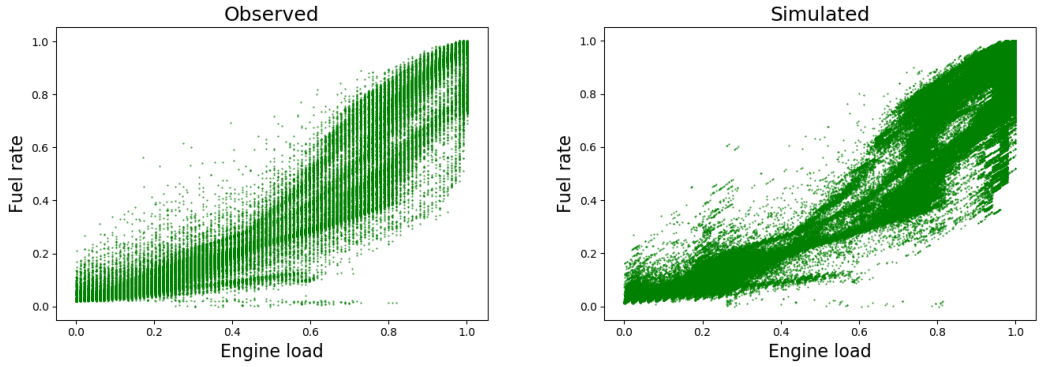


Figure 5.3: Violin plot: engine load (n_{obs} = 332.156, n_{sim} = 2.939.872)

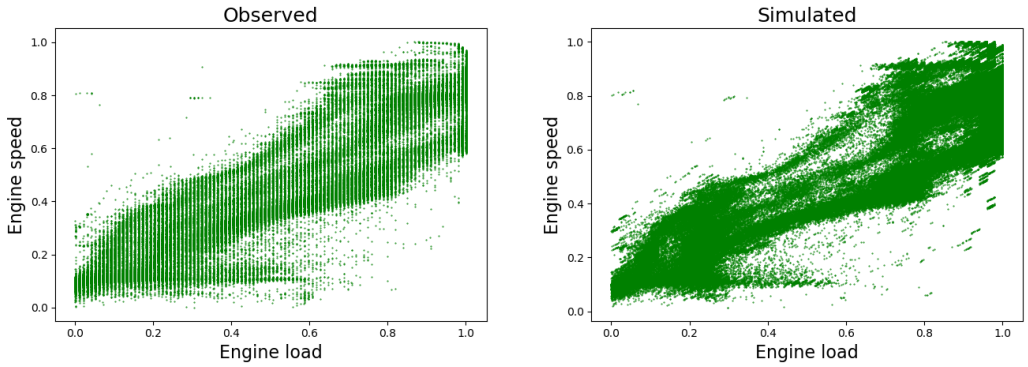
indicate a vessel is sailing, which happens mostly at full throttle. The main differences between the observed and simulated engine load are in the moderate and high engine loads. First, the observed engine loads show slightly more density in the moderate engine load levels (between 0.3 and 0.6). In our simulation, we have not defined states with a moderate mean engine load. We use moderate engine loads mostly as a transition between a low engine load state (berthing / drifting) and a high engine load state (sailing). Second, the simulated dataset contains relatively more high engine loads. This is because our simulated vessel is always in demand and thus operated 100% of the time. As stated in the methodology, we simulate the



a) Observed normalised load vs. fuel rate b) Simulated load vs. fuel rate

Figure 5.4: Scatterplots comparing simulation with observation

fuel rate, speed and other variables based on the engine load. The deviation of the speed and fuel rate from the engine load determines the degradation. We, therefore, compare the engine load versus fuel rate and the engine load versus engine speed of the observed and simulated datasets, see Figure 5.4 and 5.5.



a) Observed normalised load vs. speed b) Simulated load vs. speed

Figure 5.5: Scatterplots comparing simulation with observation

We observe that our methods achieve engine load versus fuel rate scatters that are very similar to those observed. Also, the engine load versus engine speed scatters are similar. The figures of the simulated scatters show more unique values are reached. This has two reasons. First, the simulation scatter was made up of more values than the observed scatter. Second, the simulated load and fuel rate have more decimal numbers than the observed values. We assume this is not influential for the further modelling process.

5.1.2 Failure behaviour

A comparison of the failure behaviour in the simulated and observed dataset can be seen in Figure 5.6. We conclude that the simulated failures follow a time to failure distribution similar to that of the observed case component. The y-axis of the plot show the time to failure in seconds/10 (the interval at which measurements are taken/received).

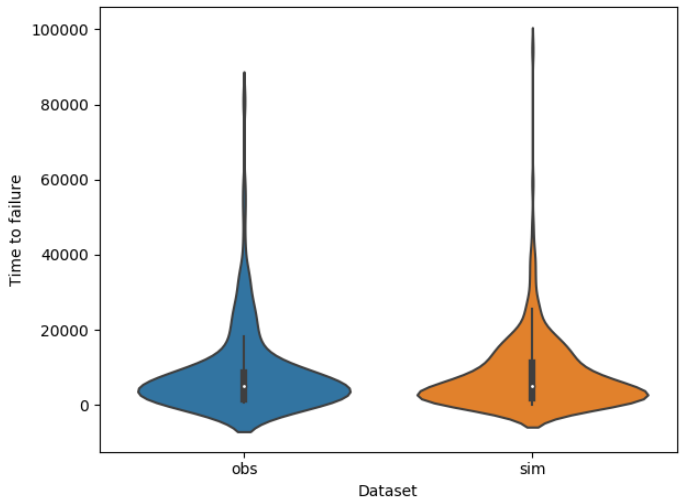


Figure 5.6: Violin plot: time to failure ($n_{obs} = 324, n_{sim} = 383$)

Most time to failure observations are around 8000 time steps (0.9 days). The tail of the simulated dataset is slightly longer and thinner than that of the observed dataset. Aside from that, the simulated dataset presented here has encounters 17.9% more failures than the observed dataset over the same duration. This can be explained by the fact that our simulated vessel is operational the whole year, while the observed case vessel did not track data for 1.5 months over a total duration of 9 months (16.67%), see Figure 2.5 in Chapter 2.

5.2 Baseline results

The baseline with an optimized maintenance-threshold (Section 4.3.1) represents a time-based preventive maintenance strategy. We compare the performance of our models with a baseline on various error metrics. The most important one, defined in Chapter 4, is the cost-based metric. Figure 5.7 shows the raw baseline prediction on a section of the test set. Note that the mean time between failure was derived from the training set.

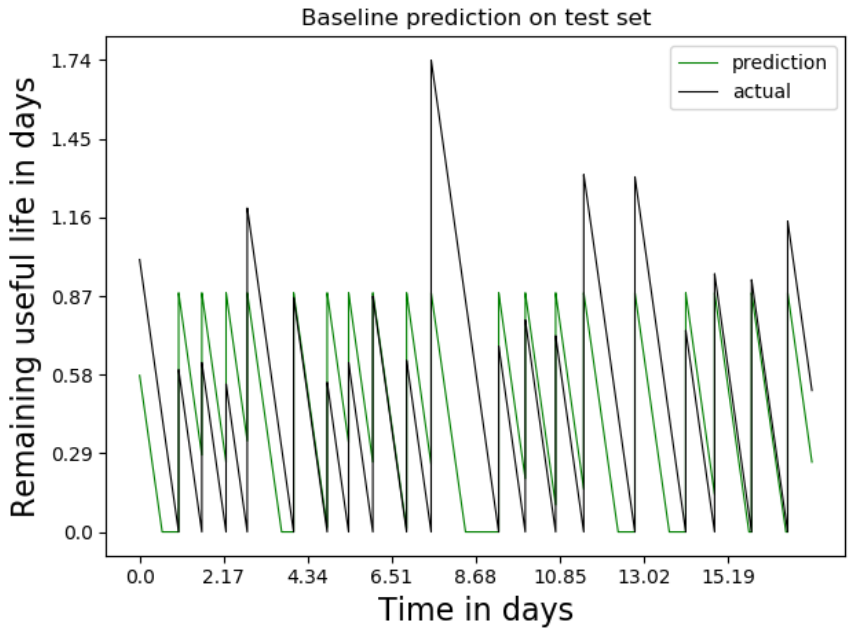


Figure 5.7: Baseline prediction on the test set

Table 5.2 shows the error metrics of the baseline model. Both the training and test metrics are reported. This is to evaluate the model performance. If scores on the training set are good, but scores on the test set are bad, the model could be overfitted on the training set. If both, train and test scores are bad, the model is likely not able to capture the relations.

The baseline explains 17% of the variance of the RUL variance in the training set. It explains more of the total RUL variance (26%) when used to predict on the test set, which can be explained by the observation that there are fewer outliers in the test set. When we optimize

Table 5.2: Error metric evaluation of baseline

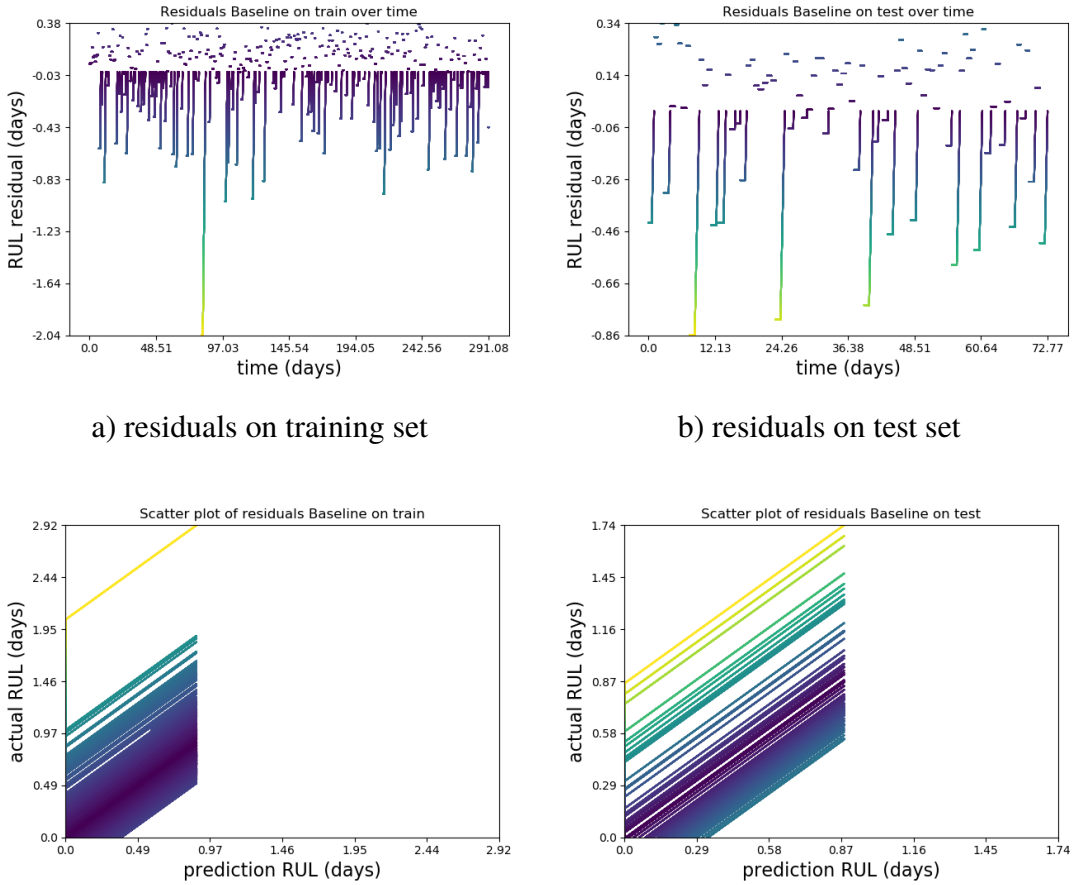
Metric	Train	Test
Mean absolute deviation	2004	1870
R2	0.17	0.26
Mean squared logarithmic error	7.59	7.08
Root mean squared relative error	20.03	18.96
Cost-based metric (thres = 0.33 days)	551.3	135.5

the maintenance threshold for the baseline model (on the training set), we find the optimal threshold to be 2870 time steps which is equal to about 0.33 days. Using this threshold, the resulting maintenance costs are 551.3 on the training set and 135.5 on the test set. It is important to optimize the threshold on the training set and not on the test set to adequately assess the generalisation performance. When the model is implemented in real-time, we cannot optimize the threshold on unobserved cases either. That means, we can only estimate the best preventive maintenance time interval based on data that is realistically available. Note that the costs are higher for the training set than for the test set because the training set is longer and thus requires more maintenance tasks to be performed.

Now, we evaluate the residuals of the baseline model. We do so by plotting the residuals of the model over time and the actual versus predicted scatter plot. The residuals plots, see Figure 5.8 a) and b), show the errors made by the baseline model over time. In the baseline model, the difference between actual and prediction will remain the same until the actual reaches zero, see Figure 5.7. That is why we observe flat lines only with positive residuals of the baseline model. When the baseline prediction is lower than the actual RUL, the (negative) residual will remain flat until the baseline prediction hits zero. At that point, the residual will decrease until the actual RUL hits zero.

For most regression problems, good models have low residuals which are equally distributed over time and normally distributed around zero. However, for our predictive maintenance regression problem, errors are not equally weighted because some errors have consequences and others do not. Errors with consequences are RUL predictions under the maintenance threshold when the actual RUl is high, and RUL predictions above the maintenance threshold when the actual RUL is zero (failure). This results in the necessity to compare the actual versus the prediction in a scatterplot, see Figure 5.8 c) and Figure 5.8 d). In these graphs, a diagonal line (where actual = predicted) indicates a perfect prediction of the RUL. The straight lines can be expected from the baseline model. When the model predicts RUL to be zero and the actual RUL is more than zero, this results in a straight vertical line, all other predictions result in diagonal lines. Distance from the predicted RUL to the actual RUL is indicated by giving smaller distances darker colours. In the baseline model, the only error type is a bias. What is interesting to see, is that we have one outlier in the training set

with a high remaining useful life, yet the baseline model does not consistently overpredict the RUL on the test set. We also see a cutoff at the mean time between failure, because the prediction will never exceed the mean time to failure (recall that the baseline is the mean time between failure minus the time since last failure).



a) residuals on training set

b) residuals on test set

c) Scatter on training set

d) Scatter on test set

Figure 5.8: Baseline evaluation

We summarise the baseline results by three core aspects. First, the maintenance costs are 135.5 on the test set. These costs are to be compared to the maintenance costs of the other two models. Second, the maintenance threshold these costs go along with is 0.33 days which is about one third of the mean time to failure. This is in line with our expectations, because a corrective maintenance actions was modelled to be three times as expensive as a preventive action. Third, the model can explain about 26% of the variance.

5.3 Recurrent neural network results

The best LSTM found in the particle swarm optimization is a model with 4 hidden LSTM layers and a recurrent dropout rate of 1.6% and 50 timesteps which model temporal depen-

dencies. This means that each output is predicted by letting 50 prior observations propagate through the LSTM.

The predictions by the LSTM on the test set are shown in Figure 5.9 (green = prediction, black = actual). Although the model is not always right, we can see that it has learned that the RUL is a decreasing function of the time since last failure. So, how well did it learn the relation between sensor measurements and RUL? Learning this relation is the core advantage of the recurrent neural network and the boosting tree should have over a simple preventive maintenance policy. We compute the evaluation metrics of our models to report the extent to which the models encapsulate the relation between sensor measurements and RUL, and to assess the impact of learning this relationship on the minimization of maintenance costs.

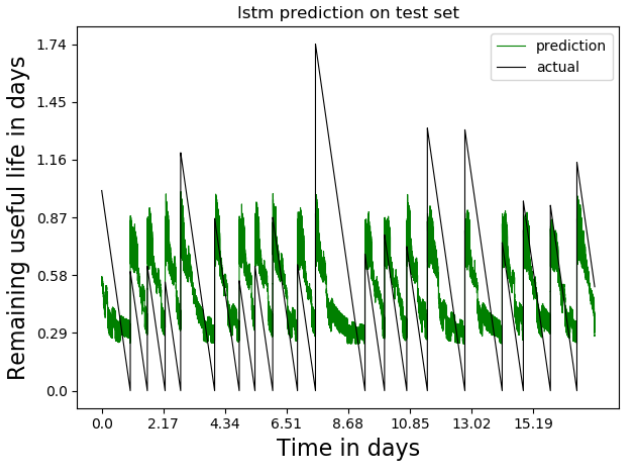


Figure 5.9: LSTM prediction on the test set

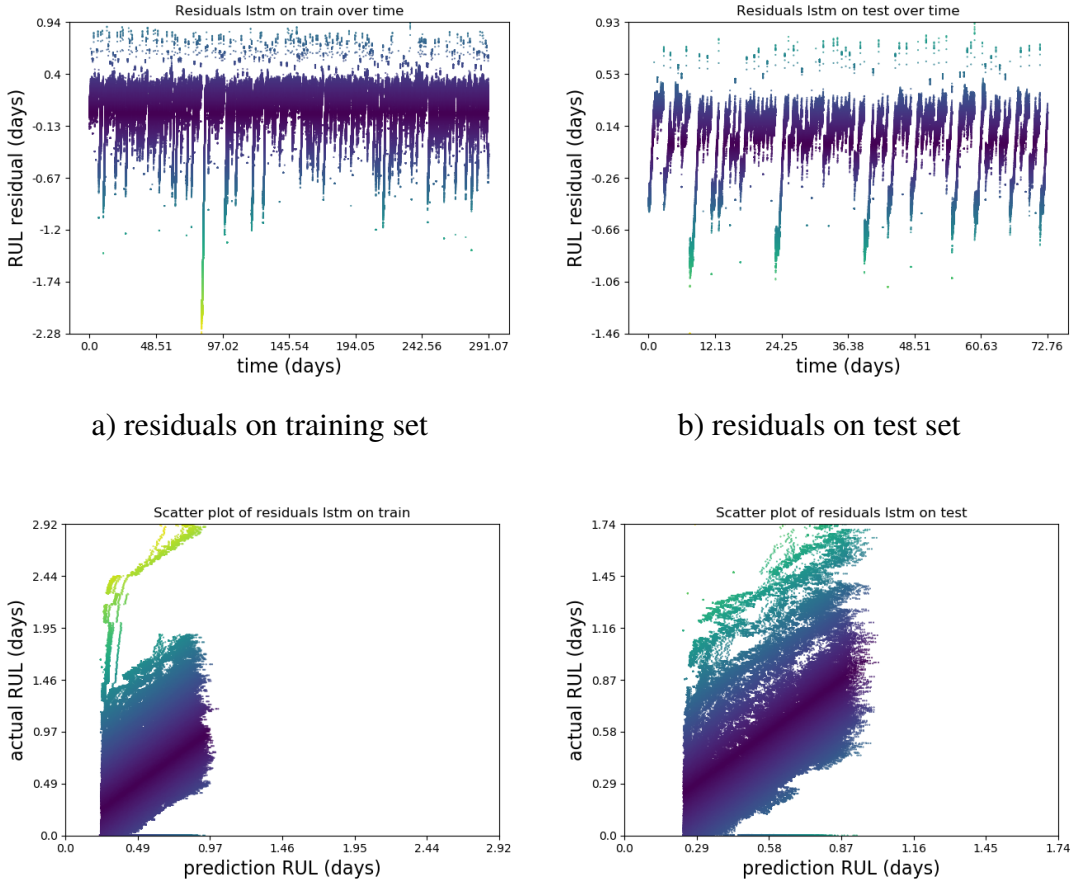
The metrics in Table 5.10 suggest that the model is a reasonable model for predicting the remaining useful life of the fuel separation system. This conclusion is drawn by observing relatively good scores on both the training and test set.

Table 5.3: Error metric evaluation of LSTM

Metric	Train	Test
Mean absolute deviation	1758	1819
R2	0.47	0.274
Mean squared logarithmic error	0.75	0.82
Root mean squared relative error	35.4	31.5
Cost-based metric (thres = 0.32 days)	553.16	138.43

The LSTM has a higher explained variance (R2) than the baseline model on both the training and the test set. It also scores better mean absolute deviation and on the mean squared logarithmic error. The latter is, however, likely due to the usage of the mean squared logarithmic error as a loss function for the LSTM model. The LSTM does not outperform

the baseline on the cost-based metric but is also not severely worse than the baseline (LSTM: 138.43, baseline: 135.5).



a) residuals on training set

b) residuals on test set

c) scatter on training set

d) scatter on test set

Figure 5.10: LSTM tree evaluation

Unlike the baseline, the LSTM model has adjusted its weights to reduce the error on the outlier in the training set on low actual RUL cases. We see this in Figure 5.10 c). The outlier is the array of most severe prediction errors (here in green). When the actual RUL reduces, the prediction gets better. This can best be observed when compared with Figure 5.8.

We summarise the LSTM results by three core aspects. First, the maintenance costs on the test set are 138.43. These costs are only slightly higher than that of the baseline model. Second, the maintenance threshold these costs go along with is 2340 (about 0.27 days), which is 0.06 days shorter than that of the baseline. Since we maintain when the prediction is under the maintenance threshold for the first time, this is not an improvement nor a decline in performance compared to the baseline. The slack that is left when the threshold is reached depends on the variance of the model's error. Since R2 represents the degree to which the variance of the outcome variable (RUL) is captured by the model, this is a good metric to evaluate the slack at first sight. The LSTM model can explain about 27% of the variance,

which is 1% points more than the baseline model.

5.4 Boosting tree results

The best histogram gradient boosting three found by the particle swarm optimization iterates through the dataset 167 times. It has a learning rate of 0.0074 and a maximum depth (that is the number of stacked weak learners or small trees) of 7. The predictions made by the boosted tree on the test set are shown in Figure 5.11. In green, we see the RUL prediction of the model, while the actual is RUL is black. See Table 5.4 for the metric scores of the boosting tree.

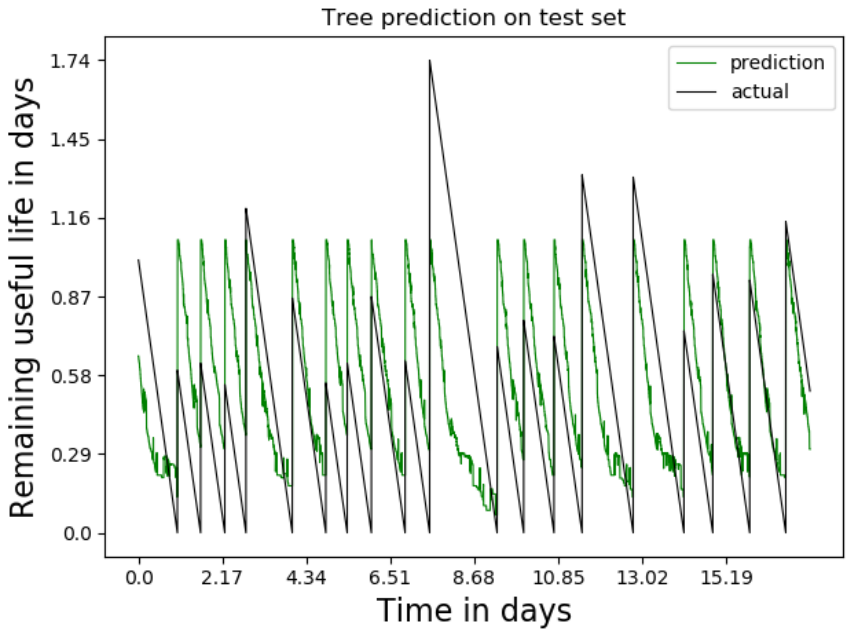


Figure 5.11: Boosting tree prediction on the test set

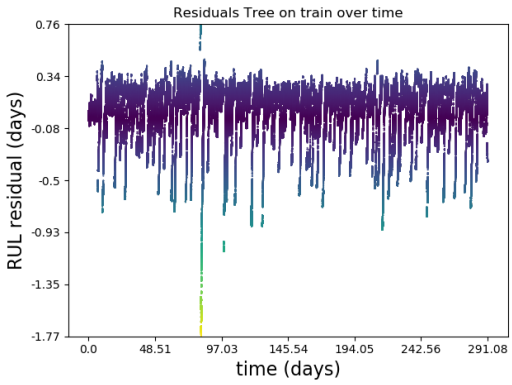
When evaluating the metrics in Table 5.4, we observe good scores on both the training set and the test set. Literature warned for a risk of overfitting with this tree-based model. We indeed observe slight indications of an overfit in the metrics. The R2 score, for example, is significantly higher for the training set than for the test set. Also, the mean absolute deviation is somewhat lower on the training set than on the test set. However, this model does achieve a 1.4% better score on the cost-based metric than the baseline model (133.6 compared to 135.5). The mean squared logarithmic error of the boosting tree is lower than that of both other models for the training and the test set. The same goes for the mean absolute error. Yet, the root mean squared error is worse than that of the baseline model (38.6 and 18.96 respectively).

We show the residuals and scatterplots of the boosting tree model like we showed the baseline model to visually compare their behaviour and performance, see Figure 5.12. These

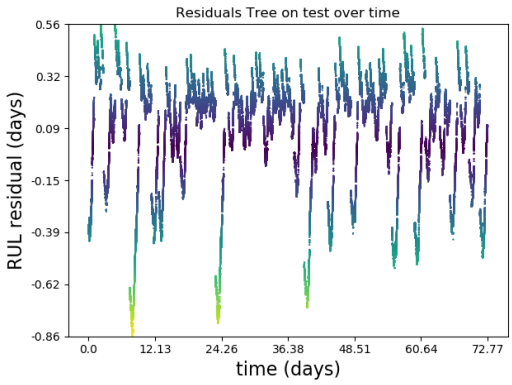
Table 5.4: Error metric evaluation of boosting tree

Metric	Train	Test
Mean absolute deviation	1775	1872
R2	0.419	0.353
Mean squared logarithmic error	0.866	0.880
Root mean squared relative error	39.8	38.6
Cost-based metric (thres = 0.32 days)	534.1	133.6

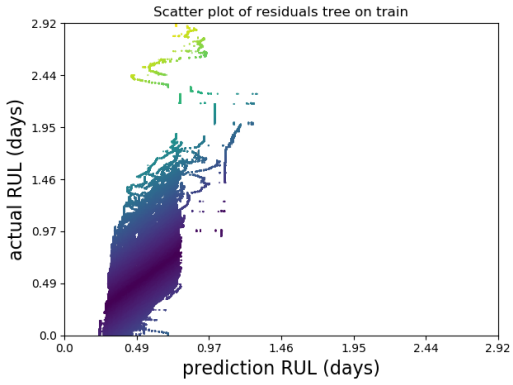
figures confirm our suspicion of the slight overfit, because 5.12 c) is close to a diagonal line where actual = prediction, whereas 5.12 d) is not. However, comparing 5.12 d) to 5.8 d), we see that the spread of the tree model is smaller especially on lower actual RUL cases. Another interesting observation is that the model never predicts RUL to be lower than about 5000 time steps (about 0.6 days). This is likely due to the maximum depth which restricts the tree to a limited number of output values.



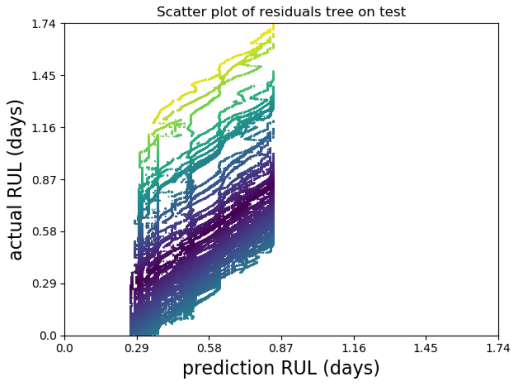
a) Residuals on training set



b) Residuals on test set



c) Scatter on training set



d) Scatter on test set

Figure 5.12: Boosting tree evaluation

Like the other two models, we summarise the boosting tree results by three core aspects.

First, the maintenance costs on the test set are 134. These costs are about 1.4% lower than that of the baseline. Second, the maintenance threshold of the boosting tree model is 0.32 days, which is 0.01 days shorter than that of the baseline. Once again, this is not an improvement nor a decline in performance compared to the baseline, because the slack that is left when the threshold is reached also depends on the variance of the model's error. If the variance of the error is high, the slack is also very volatile. This is incorporated in the cost-based metric, because models with high error variance are likely to cross the threshold too early, resulting in higher preventive maintenance cost than would be optimal. The boosting tree model can explain about 35.3% of the variance, which is 9% points more than the baseline model. We can, therefore, conclude that the boosting tree seems a suitable model to address the predictive maintenance model at hand. However, it does not strongly outperform the baseline model.

5.5 Discussion of results

We have seen that the boosting tree outperforms the other models on each metric except the root mean squared relative error. The baseline performs best on that metric. That might be due to the boosting tree never predicting lower than 0.6 days, resulting in high penalisation when the actual is low (although, the prediction might be below the maintenance threshold earlier in which case this should not be penalised). Furthermore, from the scatterplots we can derive that the boosting tree model achieved a better fit on the train and test data. This led to the conclusion that boosting trees can be suitable as remaining useful life prediction models. The maximum mistake is made where the RUL prediction is about 21000 and the actual RUL is about 36000, see Figure 5.12 d (in light green - yellow on the top). This mistake on the test set is also visible in Figure 5.12 b and in Figure 5.12 at time 20000 (in seconds/10). It is the highest remaining useful life observation in the test set and in fact, the mistake on this outlier is visible as well in the baseline model graphs. Interestingly, this outlier would be a very high actual RUL observation in the training set too. However, the training set contains an even more extreme outlier. The models did not learn to predict high actual RUL. At the start of a maintenance cycle a component is as good as new. There is no information regarding how long a component will survive at the start of cycle except for the timestamp. From the time stamp (and the RUL target) the models might learn the preventive maintenance interval which resets the degradation to zero (thereby extending the life time if this action takes place shortly after the start of the cycle). In short, it is particularly hard for the models to predict high actual RUL cases because (1) the only degradation information they might have at that point is their timing of a preventive action which they should learn from hidden relations, and (2) we avoid penalising the models on high actual RUL cases because this does not influence the maintenance decision through choosing appropriate loss functions.

The models should be easy to train, evaluate and deploy through the developed dashboard.

Therefore, it is important to take into account the training speed. The recurrent neural network with long-short term memory units requires about 26 minutes per trained and evaluated model, whereas the boosting tree requires about 1 minute. This allows the boosting tree to try more hyperparameter configurations in the same time and thus a more exhaustive particle swarm search. This might be one explanation for the boosting tree outperforming the recurrent neural network despite the temporal dependencies modelled by the latter.

Chapter 6

Conclusions and recommendations

This chapter consists of three sections. The first contains the general answer to the research question while the second describes the recommendations to Damen Shipyards. Finally, in the last section we discuss the limitations and implications of the research findings.

6.1 Conclusions

In Chapter 1, we have introduced our main research question as: *'How can we enable Damen to predict remaining useful life of vessel components in varying circumstances with indirect condition-monitoring data?'* We have shown that machine learning techniques can be suitable for remaining useful life predictions. According to our results, a cost reduction can be achieved by predicting the remaining useful life with a histogram gradient boosting tree. This remaining useful life prediction should then be used in deciding whether or not a system should be maintained at any point in time through considering whether the prediction is lower than an optimized maintenance threshold. The answer to our four sub-questions are the foundation for this conclusion. In Chapter 2, we posed the first sub-question: *What is the structure of this predictive maintenance problem?* The structure of the maintenance problem is key to understanding the problem and the modelling requirements that arise from it. We looked at the root-cause of the problem and its consequences. We then defined the root-cause as the absence of information regarding component degradation and thus the remaining useful life is uncertain. To address the root-cause of the problem, we predict the remaining useful life of vessel components. Before we did that, we reviewed the structure of the problem. We concluded that our predictive maintenance problem has a severe class imbalance, is a multivariate time-series regression, and has obvious failures (the alarms). In Chapter 3, we studied the literature to find fitting approaches to address the predictive maintenance problem with its structure as specified in Chapter 2. We concluded two models to be especially suitable for our problem structure: a recurrent neural network with long-short term memory units, and a gradient boosting tree regression. The first is suitable to model and remember long term

temporal dependencies, and the second is suitable to address the class imbalance and train and predict on large dataset. The recurrent neural network and the gradient boosting tree models needed to be implemented, validated and evaluated. We asked ourselves how this would best be done in Chapter 4. We concluded that the dataset is not suitable to train, validate and evaluate our models due to data impurities. Therefore, we simulated a dataset similar the original dataset, yet without data quality issues. The methodology used should, therefore, be generalisable to real predictive maintenance datasets. We also concluded that the best way to evaluate and compare various models is through a cost-based metric. This metric essentially follows a 'first predict, then optimize' strategy to define a maintenance threshold. Finally, we concluded that our predictive maintenance methodology realised a 1.4% cost reduction on the simulated dataset with a histogram gradient boosting tree. Whether this finding is generalisable outside of the simulation and to other vessels and components remains to be seen when clean data is available.

6.2 Recommendations

Our recommendations essentially answers two questions. First, we describe the lessons learned by answering the question: *how could the presented research be improved if it were repeated?* Second, we suggest directions for future research by answering the question: *what would be the appropriate question(s) for future research starting from what is presented here?* We hope to outline options to continue the research into predictive maintenance and enable Damen to further develop predictive maintenance services.

6.2.1 Lessons learned

Select assets for predictive maintenance

This research was initiated partially based on the availability of largely unused remote monitoring databases. The company's motivation to apply data-driven techniques to these databases to develop predictive maintenance capabilities led to the research presented here. The alarms used as a proxy for failures do not resemble severely costly failures, but rather soft failures. This research could be improved if it was motivated by a clear business case. That means that a vessel and component with high maintenance costs or down-time penalty have to be selected using feedback from clients and maintenance experts. Suitable assets for predictive maintenance have high maintenance costs or down-time fees and have sufficient failure information available to make a failure mode and effect analysis. Once a pre-selection of suitable components has been made, a failure mode and effect analysis can be used to expose the causes of each potential failure. Finding variables that relate to the root-cause of a potential failure (and ideally provide causal information) is an essential ingredient for successful predictive maintenance, because it helps making a good model choice. After identifying the right assets and

components for predictive maintenance, Damen has to select good monitoring equipment. Measuring variables related to the cause of failure is what enables predictive maintenance services that go beyond condition monitoring. Ideally, also a degradation variable would be made measurable through appropriate sensor installations. This allows Damen to model and predict the remaining useful life more accurately.

Develop and deploy predictive maintenance services

Once relevant sensors are installed, new predictive maintenance models can be made. The outcome of these models should eventually be made available to the client. Depending on the objective and available data, the chosen prediction model can be a rule-based expert-system, a remaining useful life prediction, an anomaly detection, or an other model type. If the choice is to make remaining useful life prediction models or the choice is to predict another form of degradation (remaining useful life is essentially a measure of degradation too) with a labelled dataset, Damen can make use of the modelling methodology developed in this thesis. Each modelling choice has its considerations regarding the use of the predictions made by the model. All should have a common goal with that of this thesis, that is: to exploit the full component life time while minimizing the risk to observe an unexpected failure. We summarize the options for new research directions that can be used to achieve this objective in Table 6.1.

Table 6.1: Model types with their requirements

Method	Type	Required	To optimize
Rule-based expert system	Condition-based	Expert maintenance and engineering knowledge & failure mode & effect analysis	Rules & actions prescribed when rule is true
Remaining useful life prediction	Predictive	Failure or degradation data, data science, maintenance engineering knowledge, optimization knowledge	Maintenance threshold & actions prescribed when threshold reached
Outlier detection	Predictive or condition-based	Data science, feedback system	Priorities of detected anomalies & actions prescribed when anomaly detected

After the development of a prediction model or reuse of the current models, one can look into the optimization of the usage of the predictions. Alternatively, the optimization can be taken into account in the prediction modelling phase using a 'smart predict, then optimize' paradigm (Elmachtoub and Grigas, 2017). The optimization should lead to, for instance, a threshold beyond which a maintenance action is prescribed. Once the models have been trained, validated and evaluated, Damen can start a predictive maintenance pilot. In this pilot, the prediction models should be linked to the existing data warehouse and maintenance management system.

Table 6.2: Prescibed models under various data scenarios

Measurement data	Observed failure cycles	Recommended model	Actionable result
Variables related to failure modes	None	Outlier detection	Maintenance or inspection when abnormality detected
Variables related to failure modes	Few	Rule-based expert system	Maintenance prescribed when rule violated
Variables related to failure modes	Many	Histogram gradient boosting trees (our models)	Maintenance when prediction is beneath optimized threshold
Direct degradation variables	None-few	Rule-based expert system	Maintenance or inspection when rule violated
Direct degradation variables	Many	Histogram gradient boosting trees (our models)	Maintenance when prediction is beneath optimized

Repeating this research

Although the data structure of the simulated dataset and the real dataset are similar (an imbalanced time series regression), we are not sure if we can predict failures using indirect condition-monitoring variables. Relations between the two could not be established due to data impurities. As a consequence, it would be a good start to reproduce the research using real data once sufficient clean data is gathered. When doing so, it is recommended to hypothesize which variables have a causal relationship with failures and to install sensors that measure those variables. If this research were repeated, larger datasets will be available. The newly appended data will also benefit from data quality improvements that Damen is currently working on. Therefore, more modelling options can be considered and models are less likely to be obstructed by data quality issues.

Concluding what we have learned

We can conclude that a clear objective, business case and program have to be defined before starting a research into predictive maintenance. The program execution is a team effort requiring expertise from fields like maintenance engineering and data science, see Appendix C for a project plan.

6.2.2 New directions

Continuing predictive maintenance

The new directions can partly be based on the lessons learned. A vessel and a component with high down-time penalties or maintenance costs have to be selected using input from clients and maintenance experts. Once a component is chosen, hypotheses should be made

regarding the variables which relate to its failure. This leads to the first suggested future research question:

- Which vessel and which component provide a business case for preventive maintenance and which variables are likely to be causally related to the failure of that component?

The modelling approach to accompany the newly chosen vessel component depends on data availability. If no data is available at all, Damen should start gathering data on both failures (if possible) and the variables which are hypothesized to be related. Yet, this should only be done if the project can be expected to financially beneficial given the risk of the project being unsuccessful or the project is of strategic importance. Once the sensors are installed, a rule-based expert system can be the first consideration for predictive maintenance. This leads to the second suggested future research question:

- If there is no historical data, which prefixed rules could indicate maintenance requirements prior to failure better than simple preventive maintenance.

A rule-based expert system requires an expert understanding of the physical relations between measured variables and component failure. If this knowledge is not available Damen could recede to outlier detection models. Once there is sufficient data, the methodology presented in this research can be used to predict the remaining useful life. Another way to identify failures is through anomaly (or outlier) detection models. These models compare the behaviour of a system to a 'norm' to identify odd observations that are likely to require attention or maintenance actions.

Prescriptive maintenance

With a prediction model in place, we suggest equipping the user with a tool to further benefit from the prediction. 'A maintenance action is required' might be a too generic message for an operator to act upon. This can partially be resolved by creating prediction models on the 'failure mode level'. However, more can be done to prescribe a relevant action. One way to do so is to create a case-based reasoning system. This expert system recalls cases which are similar to current failure predictions as to provide contextual information about the failure. The contextual information contains for instance a description of the problem priorly observed (in the recalled case). A case-based reasoning system is useful especially when an operator does not have much experience with the type of failure at hand. One way to implement this is by comparing the features of the currently predicted failure to those in the case-based reasoning system's historic dataset and returning the top n most similar observations (with or without failure). However, the best indicators upon which the case-based reasoning system decides which cases are similar and which are not should be researched. This leads to the third suggested future research question:

- How can a case-based reasoning system be built to extend predictive maintenance and enable prescriptive maintenance?

Integration with existing maintenance software and data warehouse

The data-driven predictive maintenance methodology presented in this research should not lose practical value once new sensors are installed or new ships are built. Therefore, the models should be highly generalizable and easy to design, train and deploy by Damen. To realize this, we recommend to further develop the predictive maintenance application which we have built. This should enable Damen to create new predictive maintenance models with data sets incorporating new sensors, new components of interest or even other ships. The application can be linked with the existing data warehouse through ETL (extract, transform, and load). This greatly enhances the practical value of the created modelling methodology. The final recommended future research question suggested here is as follows:

- How can the predictive maintenance models be presented to the end-user to enhance practical value?

Most logical is to research the feasibility of linking existing maintenance management systems with the main data warehouse as well as linking the developed predictive maintenance software with the data warehouse. Then maintenance can be prescribed directly to the client based on predictions by the predictive maintenance models. Additionally, Damen could integrate the prediction model with the spare part ordering process. As soon as a remaining useful life prediction (or degradation) falls beneath a threshold, the inventory position reduces by one because a spare part will be committed to the prescribed maintenance action. Whether doing so is useful depends on the reordering policy (for instance whether it is digitalised or not). An overview of the recommended predictive maintenance program including deliverables and required capabilities can be found in Appendix C. When following these recommendations, we expect Damen to advance their predictive maintenance capabilities and provide customer value by reducing maintenance costs and down-time penalties.

6.3 Discussion

This section highlights the contributions and limitations to the research.

6.3.1 Contributions

We offer four main contributions to practice. First, we develop machine learning models to predict the remaining useful life of critical assets. These models can be used to predict the remaining useful life of the fuel separation system and slightly outperform simple time-based preventive maintenance when compared on a cost-metric. Second, we provide a

methodology that generalisese to other vessel, components, and predictive maintenance cases. The models can be used to predict time-to-failure as well as the degradation or other variables of interest. Third, we formulate a maintenance threshold optimisation model which allows Damen to act upon the remaining useful life predictions. When predictions drop under the optimized threshold, maintenance should be performed. The maintenance threshold in our fuel separation was 0.3 days. This should be enough to preventively replace a filter because average sailing times are 3 hours after which there is time to replace a filter or perform small maintenance tasks. Finally, we started the development of an application in which predictive models can be trained, evaluated and implemented. This application should use the Damen remote monitoring system as input, predict the remaining useful life using a chosen implemented model, and forward maintenance decisions to the client via the maintenance management system. Additionally, the academic contributions are twofold. First, we provide a clear taxonomy of maintenance with a special focus on predictive maintenance. This can be used by academics or practitioners that want to consider the complete landscape of maintenance options while thinking about implementation of predictive maintenance. Second, we explore the applicability of two machine learning models (recurrent neural networks and gradient boosting trees) to predictive maintenance problems. We found gradient boosting methods to be suitable and cannot confirm the applicability of recurrent neural networks for our predictive maintenance case.

6.3.2 Limitations

Our models are dependent on the availability of clean data. Although both models can be trained with a few missing values our real dataset had too many impurities and missing values to train and validate models upon. Thus, we revert to a simulated dataset that should resemble the structure of the real dataset and thus the behavior of the fast crew supply vessel. This makes that the performance of developed models is dependent on the quality of our simulation. Since we do not know the true degradation process, the current models are not trained to perform quality prediction on real dataset. However, we cannot model the true degradation process. In fact, if we knew how to model the true degradation process using only the available indirect-condition monitoring data, prediction models would not be necessary anymore. We can conclude from this, that the current models would have to be retrained when sufficient clean data is available. The particle swarm search procedure can re-tune the parameters to create models that better suit real dataset and are able to model the true degradation process as a hidden variable to accurately predict the remaining useful life. So currently, the main limitation is the absence of sufficient clean data to test the performance of the modelling methodology in practice. We highlight two other limitations. First, the assumptions regarding behavior and state transitions made in the simulation might not reflect a generalisable operational profile. That means that other vessels might require other types of

models. Second the focus on remaining useful life predictions leads to the necessity of many failure observations to make reliable predictions. Not all component types fail frequently enough to use the type of regression models discussed in this thesis. Other models for these components are prescribed in our recommendations but were left out of scope for this thesis.

Bibliography

Berkson, J. (1944). Application of the logistic function to bio-assay. *Journal of the American Statistical Association*, 39(227):357–365.

Boser, E., Guyon, I., and Vapnik, V. (1992). A training algorithm for optimal margin classifiers. Colt.

Breiman, L. (1997). Arcing the edge. Technical report, Technical Report 486, Statistics Department, University of California at . . . .

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

Butler, K. L. (1996). An expert system based framework for an incipient failure detection and predictive maintenance system. In *Proceedings of International Conference on Intelligent System Application to Power Systems*, pages 321–326. IEEE.

Clancey, W. J. (1983). The epistemology of a rule-based expert system —a framework for explanation. *Artificial Intelligence*, 20(3):215 – 251.

De Jong (2020). Servitization in the shipbuilding industry: A research into the relation of user profiles and service constracts of high speed transport vessels. *Delft University*.

Elmachtoub, A. N. and Grigas, P. (2017). Smart "predict, then optimize". *ArXiv*, abs/1710.08005.

Gardner, M. and Dorling, S. (1998). Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric Environment*, 32(14):2627 – 2636.

Girosi, F., Jones, M., and Poggio, T. (1998). Regularization theory and neural networks architectures. *Neural Comput*, 7.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*.

Hastie, T., Tibshirani, R., and Friedman, J. (2001). The elements of statistical learning. *Aug, Springer*, 1.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9:1735–80.

Hoerl, A. E. and Kennard, R. W. (1970). Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67.

Horenbeek, A. V. and Pintelon, L. (2013). A dynamic predictive maintenance policy for complex multi-component systems. *Reliability Engineering System Safety*, 120:39 – 50.

Hyafil, L. and Rivest, R. L. (1976). Constructing optimal binary decision trees is np-complete. *Information Processing Letters*, 5(1):15 – 17.

Ifenthaler, D. and Seel, N. M. (2013). Model-based reasoning. *Computers Education*, 64:131 – 142.

Jardine, A., Lin, D., and Banjevic, D. (2006). A review on machinery diagnostics and prognostics implementing condition-based maintenance. *Mechanical Systems and Signal Processing*, 20:1483–1510.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. (2017). Lightgbm: A highly efficient gradient boosting decision tree. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30*, pages 3146–3154. Curran Associates, Inc.

Khazraei, K. and Deuse, J. (2011). A strategic standpoint on maintenance taxonomy. *Journal of Facilities Management*, 9:96–113.

Kinghorst, J., Geramifard, O., Luo, M., Chan, H., Yong, K., Folmer, J., Zou, M., and Vogel-Heuser, B. (2017). Hidden markov model-based predictive maintenance in semiconductor manufacturing: A genetic algorithm approach. In *2017 13th IEEE Conference on Automation Science and Engineering (CASE)*, pages 1260–1267.

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

Krizhevsky, A., Sutskever, I., and Hinton, G. (2012). Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*, 25.

Kuhn, M. and Johnson, K. (2013). *Applied predictive modeling*, volume 26. Springer.

Kuzin, T. and Borovicka, T. (2016). Early failure detection for predictive maintenance of sensor parts.

LaRiviere, J., McAfee, P., Rao, J., Narayanan, V. K., and Sun, W. (2016). Where predictive analytics is having the biggest impact. *Harvard business review*, 05(1):3–18.

Li, Z., Wang, K., and He, Y. (2016). Industry 4.0 - potentials for predictive maintenance.

Lin, E., Chen, Q., and Qi, X. (2019). Deep Reinforcement Learning for Imbalanced Classification. *arXiv e-prints*, page arXiv:1901.01379.

Maillart, L. (2006). Maintenance policies for systems with condition monitoring and obvious failures. *Iie Transactions*, 38:463–475.

Malhotra, P., Vig, L., Shroff, G., and Agarwal, P. (2015). Long short term memory networks for anomaly detection in time series.

Maron, M. E. (1961). Automatic indexing: An experimental inquiry. *J. ACM*, 8:404–417.

Martinez, C., Perrin, G., Ramasso, E., and Rombau, M. (2018). A deep reinforcement learning approach for early classification of time series. *2018 26th European Signal Processing Conference*, pages 2030–2034.

Moniz, N., Branco, P., and Torgo, L. (2017). Resampling strategies for imbalanced time series forecasting. *International Journal of Data Science and Analytics*, 3(3):161–181.

Najim, K., Ikonen, E., and Daoud, A.-K. (2004). Chapter 3 - optimization techniques. In *Stochastic Processes*, pages 167 – 221. Kogan Page Science, Oxford.

Nowlan, F. S. and Heap, H. F. (1978). *Reliability-Centered Maintenance*. San Francisco: Dolby Access Press.

Okoh, C., Roy, R., and Mehnen, J. (2016). Predictive maintenance modelling for through-life engineering services. *Procedia CIRP*, 59:196–201.

Pecht, M. and Kang, M. (2018). *Introduction to PHM: Fundamentals, Machine Learning, and the Internet of Things*, pages 1–37.

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

Rasiel, E. (1999). *The McKinsey Way*. McGraw-Hill Education.

Rosenblatt, F. (1957). *The Perceptron: A Perceiving and Recognizing Automaton*. Cornell Aeronautical Laboratory, Buffalo, New York.

Schmidt, B. and Wang, L. (2018). Cloud-enhanced predictive maintenance. *The International Journal of Advanced Manufacturing Technology*, 99(1-4):5–13.

Si, X.-S., Wang, W., Hu, C.-H., and Zhou, D.-H. (2011). Remaining useful life estimation – a review on the statistical data driven approaches. *European Journal of Operational Research*, 213(1):1 – 14.

Sikorska, J., Hodkiewicz, M., and Ma, L. (2011). Prognostic modelling options for remaining useful life estimation by industry. *Mechanical Systems and Signal Processing*, 25(5):1803 – 1836.

Simões, A., Viegas, J., Farinha, J., and Fonseca, I. (2017). The state of the art of hidden markov models for predictive maintenance of diesel engines: Hmm for predictive maintenance of diesel engines. *Quality and Reliability Engineering International*.

Taghipour, S. and Banjevic, D. (2012). Optimal inspection of a complex system subject to periodic and opportunistic inspections and preventive replacements. *European Journal of Operational Research*, 220:649–660.

Trojan, F. and Marçal, R. (2017). Proposal of maintenance-types classification to clarify maintenance concepts in production and operations management. *Journal of Business Economics*, 8.

Viterbi, A. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269.

Yoon, Y., Acree, A. D., and Peterson, L. L. (1993). Development of a case-based expert system: Application to a service coordination problem. *Expert Systems with Applications*, 6(1):77 – 85. Special Issue: Case-Based Reasoning and its Applications.

Åström, K. (1965). Optimal control of markov processes with incomplete state information. *Journal of Mathematical Analysis and Applications*, 10(1):174 – 205.

Appendix A

Technical

A.1 Derivation of Manhattan upperbound

$$dist_m(x) = x + \sqrt{dist_e^2 - x^2} \tag{A.1}$$

$$dist_m(x)' = 1 - \frac{x}{\sqrt{dist_e^2 - x^2}} \tag{A.2}$$

$$dist_m(x)'' = -\frac{dist_e^2}{(dist_e^2 - x^2)^{3/2}} \tag{A.3}$$

$$Finding\ \ a\ \ maximum\ \ implies:$$

$$dist_m(x)' = 0,\ \ dist_m(x)'' < 0 \tag{A.4}$$

$$dist_m(x)' = 0\ \ for\ \ x = \sqrt{dist_e^2 - x^2} \tag{A.5}$$

$$x^2 = dist_e^2 - x^2 \tag{A.6}$$

$$thus:\ \ x = \frac{dist_e}{\sqrt{2}}\ \ or\ \ x = dist_e \tag{A.7}$$

$$dist_m''(\frac{dist_e}{\sqrt{2}}) = -\frac{dist_e^2}{(dist_e^2 - \frac{dist_e}{\sqrt{2}}^2)^{3/2}} \tag{A.8}$$

$$dist_m''(\frac{dist_e}{\sqrt{2}}) = -\frac{2\sqrt{2}}{dist_e}\ \ for\ \ dist_e > 0 \tag{A.9}$$

$$dist''_m(\frac{dist_e}{\sqrt{2}}) = -\frac{2\sqrt{2}}{dist_e} < 0 \quad for \quad dist_e > 0 \tag{A.10}$$

$$Thus\ x = \frac{dist_e}{\sqrt{2}}\ is\ a\ maximum,\ and\ UB_{dist_m/dist_e} = dist_m(\frac{dist_e}{\sqrt{2}})/dist_e = \sqrt{2}.$$

$$dist''_m(dist_e) = undefined\ \ due\ \ to\ \ division\ \ by\ \ zero \tag{A.11}$$

$$Note,\ however,\ that\ dist_m(dist_e) = x\ and\ dist'_m(dist_e) = 1\ and\ domain:\ 0 < x < dist_e$$

$$Thus\ x = dist_e\ is\ a\ cornerpoint\ local\ minimum.$$

A.2 Threshold optimization ILP

Problem description We want to minimize the total maintenance costs by optimizing the maintenance threshold. A preventive maintenance action is executed when the predicted remaining useful life (RUL) is lower than the maintenance threshold earlier than the actual RUL is zero. Likewise, a corrective maintenance action is executed when the actual RUL is zero earlier than the predicted RUL is lower than the maintenance threshold.

Indices

s :	the maintenance cycle: from failure + 1 to next failure	[real]	$s = 0...S$
t :	the time step within a maintenance cycle s	[real]	$t = 0...T_s$

(A.12)

Parameters

$p_{s,t}$:	RUL prediction of time step t in cycle s	[real]
$a_{s,t}$:	actual RUL of time step t in cycle s	[real]
mcc :	mean cycle costs	[real]
mct :	mean duration of a maintenance cycle	[real]
$pmbc$:	preventive maintenance basic costs	[real]
$cmbc$:	corrective maintenance basic costs	[real]
ttf_s :	time to failure in cycle s	[real]
M_s :	Big M auxiliary parameter for each cycle s	[real]

(A.13)

Variables

$$
\begin{aligned}
K: &\ \text{threshold} && \text{[real]}\\
CC_s: &\ \text{corrective maintenance costs incurred over cycle s} && \text{[real]}\\
PC_s: &\ \text{preventive maintenance costs incurred over cycle s} && \text{[real]}\\
COR\_BIN_s: &\ \text{1 if corrective maintenance in cycle s, 0 otherwise} && \text{[bin]}\\
PREV\_BIN_s: &\ \text{1 if preventive maintenance in cycle s, 0 otherwise} && \text{[real]}\\
TTT_s: &\ \text{(Time To Threshold) time steps it takes for the prediction to get below threshold K in cycle s} && \text{[real]}\\
R_{s,t}: &\ \text{auxiliary variable (Reached) to define whether or not a prediction at time t is under the threshold K for every s} && \text{[real]}\\
RFT_{s,t}: &\ \text{auxiliary variable (Reached First Time) to define whether time t is the first time a prediction is under the threshold K for every s} && \text{[real]}\\
NR_s: &\ \text{auxiliary variable 1 when threshold K is Not Reached in s, 0 otherwise} && \text{[real]}\\
RWPRT_s: &\ \text{remaining useful life when prediction reaches threshold} && \text{[real]}
\end{aligned}
\tag{A.14}
$$

Objective and constraints

$$
\min \quad \sum_{s=0}^{S} CC_s + \sum_{s=0}^{S} PC_s
$$

subject to:

$$
\begin{aligned}
1)\quad & PC_s + M_s * (1 - PREV\_BIN_s) && \geq && \tfrac{mcc**RWPRT_s}{mct} + pmbc && \forall s\\
2)\quad & CC_s && \geq && cmbc * COR\_BIN_s && \forall s\\
3)\quad & M_s * PREV\_BIN_s && \geq && RWPRT_s && \forall s\\
4)\quad & M_s * COR\_BIN_s && \geq && 1 - RWPRT_s && \forall s\\
5)\quad & RWPRT_s && = && ttf_s - TTT_s && \forall s\\
6)\quad & TTT_s && \geq && RFT_{s,t} * t && \forall s, \forall t \leq T_s\\
7)\quad & TTT_s && \leq && RFT_{s,t} * t + M_s * (1 - RFT_{s,t}) && \forall s, \forall t \leq T_s\\
8)\quad & TTT_s && \geq && NRs * ttf_s && \forall s\\
9)\quad & \sum_{tau=0}^{t-1} RFT_{s,tau} && \geq && RFT_{s,t} && \forall s, \forall 1 < t \leq T_s\\
10)\quad & RFT_{s,t} && \leq && R_{s,t} && \forall s, \forall t \leq T_s\\
11)\quad & \sum_{t=0}^{T_s} RFT_{s,t} + NR_s && = && 1 && \forall s\\
12)\quad & COR\_BIN + PREV\_BIN && = && 1 && \forall s
\end{aligned}
\tag{A.15}
$$

More constraints

$$
\begin{aligned}
13) \quad M_s * R_{s,t} &\geq K - p_{s,t} + 1 && \forall s, \forall t \leq T_s \\
14) \quad NR_s &\leq 1 - R_{s,t} && \forall s, \forall t \leq T_s \\
15) \quad NR_s &\geq 1 - \sum_{t=0}^{T_s} R_{s,t} && \forall s
\end{aligned}
\tag{A.16}
$$

$$
\begin{aligned}
\text{Sign constraints)} \quad CC_s,\ PC_s,\ TTT_s &\geq 0 && \forall s \\
RWPRT_s,\ TTF_s,\ K &\geq 0 && \forall s \\
Z_{s,t},\ R_{s,t} &\in \{0, 1\} && \forall s, t
\end{aligned}
$$

Constraint explanations

1) The costs of preventive maintenance incurred in cycles
2) The costs of corrective mainteance incurred in cycles
3) Preventive maintenance action when RUL when prediction reaches threshold is more than zero
4) Corrective maintenance action when RUL when prediction reaches threshold is zero
5) RUL when prediction reaches threshold is time to failure minus time to threshold
6&7) Time to threshold is equals auxiliary for first reach times t
8) If threshold not reached, time to threshold equals time to failure (thus failure)
9) Restricting the first time the threshold is reached auxiliary variable to the earliest reach
10) First time reached is no later than any reach
11) If no first time reach, then not reached
12) Either corrective or preventive maintenance each cycle
13) Reached is one when prediction is less than threshold
14) Not reached is zero when any reach is one
15) Not reached is 1 if no reach is one

(A.17)

A.3 Particle Swarm Optimization

Algorithm 2: Hyperparameter tuning procedure

```
1:  procedure TUNE(swarm_size, data)
2:      training_sets ← get_training_sets(data, k)
3:      validation_sets ← get_validation_sets(data, k)
4:      particle_positions ← initilize
5:      velocities ← initilize
6:      for all positions do
7:          model ← build_model(particle_positions)
8:          particle_values ← Evaluate(model, training_sets, validation_sets)
9:      end for
10:     best_swarm_value ← max(particle_values)
11:     best_swarm_position ← max(particle_position)
12:     best_particle_values ← particle_values
13:     best_particle_position ← particle_position
14:     while stopping criteria not met do
15:         velocities ← update_velocities(velocities, particles, swarm)
16:         particle_positions ← update_positions(particle_positions, velocities)
17:         for all positions do
18:             model ← build_model(particle_positions)
19:             particle_values ← Evaluate(model, training_sets, validation_sets)
20:             if particle improved then:
21:                 update_best_particle(particle, value)
22:             if swarm improved then:
23:                 update_best_swarm(particle, value)
24:         end for
25:     end while
26: return best hyperparameters
27: end procedure
```

Appendix B

Context

B.1 Alarm names

Table B.1: Alarm relevance

Alarm name	Number of failures
SHORE DOUT24 - CB OPEN_DS	12
TANK 3 PERCENTAGE ALARM LOW_AS	172811
TANK 3 PERCENTAGE ALARM LOW LOW_AS	109839
TANK 4 PERCENTAGE ALARM LOW_AS	176563
TANK 4 PERCENTAGE ALARM LOW LOW_AS	112210
TANK 13 PERCENTAGE ALARM LOW_AS	178192
BILGE LEVEL SWITCH BILGE WELL ER SB AFT_DS	296
BILGE LEVEL SWITCH EMERGENCY BILGE SB_DS	58
BILGE LEVEL SWITCH COFFERDAM_DS	12
BILGE LEVEL SWITCH ER DIRECT BILGE PS_DS	5
BILGE LEVEL SWITCH BILGE WELL E.R PS AFT_DS	12
BILGE LEVEL FLOODING ALARM_DS	832
BILGE LEVEL SWITCH VOID BELOW SWBD ROOM_DS	9
BILGE LEVEL SWITCH VOID BELOW ROPE STORE_DS	3
BILGE WATER TANK LEVEL SWITCH DS_DS	6
Bilge Condition 10	9
Fire Safety Condition	16
PS FUEL SEPERATOR 2 DIESEL CONSUMERS_DS	82
SB FUEL SEPERATOR 2 DIESEL CONSUMERS_DS	13
SB FUEL SEPERATOR 1 DIESEL CONSUMERS_DS	372
PS FUEL SEPERATOR 1 DIESEL CONSUMERS_DS	7

Table B.2: Table Alarm relevance (continued)

Alarm name	Number of failures
ME PS Engine Cylinder Cutoff_DS	561
ME SB Engine Cylinder Cutoff_DS	410
PS generator Remote_DS	269
SB generator Remote_DS	171
TANK 13 PERCENTAGE ALARM LOW LOW_AS	113093
TANK 14 PERCENTAGE ALARM LOW_AS	183015
TANK 14 PERCENTAGE ALARM LOW LOW_AS	168064
TANK 2 PERCENTAGE ALARM LOW_AS	71295
TANK 2 PERCENTAGE ALARM LOW LOW_AS	48786
Thruster 1	35
Thruster 2	35
Thruster 3	7
Thruster 4	1
Thruster 5	36
Thruster 6	187
Thruster 7	5
Thruster 8	0
Thruster 9	0
Thruster 10	0

Appendix C

Recommended action

C.1 Project plan

Figure C.1: deliverables

Figure C.2: capabilities