UiO : **Department of Mathematics**
University of Oslo

Weakly Supervised Learning for Predictive Maintenance:

An Extended Random Forest Approach using Imbalanced Event Data from Hybrid Ships

Nicolay Bjørlo Kristensen

Master's Thesis, Spring 2021

Abstract

With the advent of Industry 4.0, *Predictive Maintenance* (PdM) has garnered a lot of interest, both academically and in the industry. This thesis will be developing and using machine learning methods for PdM, using real world event-log data gathered from hybrid marine vessels, equipped with electric propulsion systems. The methods that will be used were chosen for their abilities to solve particular problems, such as data imbalance through the use of Balanced Random Forest, weakly labelled data through the use of Multiple Instance Learning, and maintaining interpretability through the use of interpretable pre-processing techniques, such as window aggregation.

Acknowledgements

I would like to extend my thanks to my main advisor, Azzeddine Bakdi, for the enormous amounts of help he has provided me with throughout the whole writing process of this thesis. His assistance and thorough review of my work has been very helpful to me, as someone who has never written a large academic work before.

Secondly, I would like to thank my co-advisor Morten Stakkeland, for providing us with the data sets from ABB, and for his help with questions relating to the data sets and the ships they originate from. I would also like to thank my other co-advisor; Ingrid Kristine Glad, as well as Morten, for reading through parts of my thesis and providing valuable feedback.

Finally, I would like to thank my family for their support throughout the writing process, especially my mother for allowing me to borrow her office to write parts of my thesis, during the periods when the University of Oslo was closed due to the Covid-19 lockdown.

Contents

Abstract i

Acknowledgements ii

Contents iii

List of Figures v

List of Tables vi

1 Introduction 1

2 Predictive maintenance 4
- 2.1 Alarm management and event data management 4
- 2.2 Reactive, preventive and predictive maintenance 7
- 2.3 Predictive maintenance approaches 8
- 2.4 Common PdM algorithms 10
- 2.5 Recent trends in PdM 13

3 Log events based PdM in Electric Propulsion Systems 15
- 3.1 System description and data description 15
- 3.2 Pre-processing 17
- 3.3 Aggregating over windows 19
- 3.4 Random indexing 21
- 3.5 Overall feature engineering approach 24

4 PdM through Random Forest 26
- 4.1 Classification and Regression Trees 26
- 4.2 Random Forest 30
- 4.3 RF-Based approach for PdM 32

5 Multiple Instance Learning through Balanced RF 36
- 5.1 Imbalanced Data Classification and Cost Sensitive Approaches 36
- 5.2 Balanced RF 38
- 5.3 Weakly Labelled Data 39
- 5.4 Multiple Instance Learning 41

5.5 Novel Approach: Multiple Instance Learning through Balanced Random Forest 45

6 Results, discussion and analysis 47
6.1 Training and testing data 47
6.2 Hyperparameter analysis 49
6.3 PdM results 51
6.4 Discussion and comparison 58

7 Conclusions and future works 62

Bibliography 65

Appendices 70

Code 71
GenerateTimelines.py 71
RandomIndexing.py 73
RollingWindows.py 74
MILBRF.py 77

List of Figures

2.1 Alarm activation 5
2.2 An overview of maintenance strategies 9

3.1 Example of an EPS vessel 16
3.2 Example of signal data from a vessel 18
3.3 Illustration of window aggregation approaches 20
3.4 Event type occurrences 22

4.1 Example of nodes in a a CART 27
4.2 Node impurity measures with different cost functions 28
4.3 Random Forest illustration 31
4.4 Visualization of the full Predictive Maintenance process 33

5.1 Weakly supervised learning overview 40
5.2 Multiple Instance Learning algorithm 44
5.3 MIL-B-RF Demonstration 46

6.1 Estimated number of samples with $\hat{y} = 1$ in training set 51
6.2 Labelling of the training data 52
6.3 Labelling of the training data, zoomed 53
6.4 Classification confidence for test set 54
6.5 Classification confidence for test set, zoomed first bag 54
6.6 Classification confidence for test set, zoomed bag cluster 55
6.7 Feature importance 57
6.8 Feature importance, Random Indexing 57
6.9 Time-to-failure prediction 58

List of Tables

3.1 Table of event data from one source on a vessel 17
3.2 Summarization of data sets 19
3.3 Example of a data set after pre-processing 19
3.4 Data set after window aggregation and Random Indexing 25

4.1 Confusion matrix results from a naive labelling approach 34

6.1 Proportion of alarms 48
6.2 Initial reasonable parameter values 48
6.3 Imbalance reduction after window aggregation 49
6.4 Initial model hyperparameter values 50
6.5 Training/testing set split information 52
6.6 Confusion matrix for test set, vessel 1 55
6.7 Confusion matrix for test set, vessel 2 55
6.8 Confusion matrix for test set, vessel 3 56
6.9 Confusion matrix for test set, vessel 4 56
6.10 Most important features 56
6.11 Comparison of models using subsets of data 59
6.12 Comparison with similar applications 60

List of Algorithms

1 Aggregating over time windows and rolling windows 21
2 Modified Random Indexing algorithm for use with time based data 23
3 Overall approach combining window aggregation and Random Indexing 24

4 Multiple Instance Learning 43

CHAPTER 1

Introduction

In this thesis, the goal is to predict future failures on board ships equipped with *Electrical Propulsion Systems* (EPSs), using *Machine Learning* (ML) methods on historical event-log data. We will be trying to predict a particular class of failures in the variable speed drives on these ships. The industry is currently going through what is referred to as the '4th Industrial Revolution', or 'Industry 4.0'. Predicting failures to perform maintenance is known as *Predictive Maintenance* (PdM), and is one of the main points of interest in this revolution.

Until recently, there were two major maintenance strategies; *Run-to-Failure* (R2F) and *Preventive Maintenance* (PvM). As the name would suggest, a R2F strategy consists of not conducting repairs on the system until a part breaks. This strategy is not optimal, as unexpected breakdowns of a system can result in major economic losses because of the resulting downtime. PvM involves replacing parts regularly before they reach the end of their *Remaining Useful Life* (RUL). While it is generally a superior strategy to R2F, parts are often replaced while still usable, resulting in increased part costs. Components can also potentially break earlier than expected, resulting in the same downtime as would be experienced as with R2F. PdM aims to solve these problems by only replacing parts when they are nearing the end of their life, by predicting when they are going to fail.

PdM is an important field because of the many benefits of being able to detect when a failure may occur. Predicting and preventing a failure can reduce system downtime, and hence provide economical benefits, but also in some cases protect from environmental damage and loss of life. While the current revolution is mainly happening within the manufacturing industry, numerous other industries could also benefit from PdM, like the marine industry. For vessels equipped with EPS, an electrical failure can lead to the drives turning off, resulting in loss of control of the vessel. In a worst case scenario, this can lead to the vessel running aground, or a collision, potentially leading to loss of the entire ship. Alternatively a fuel spill can occur, damaging the environment significantly.

The PdM field is currently experiencing increasing interest, with the number of yearly publications growing each year, especially when used in combination with ML (Çınar et al. 2020). The field is not only experiencing academic interest however, more and more companies are performing work related to PdM (Scully 2019).

The scope of the thesis is limited in the sense that the main focus is to accurately predict the time of failure. The optimal time of maintenance would depend on the specific part and on board maintenance regime, and falls outside the scope of this thesis. This simplification is common between most academic papers on PdM methods.

The work presented in this thesis will use categorical event-log data from four vessels equipped with EPSs. This data will be pre-processed and used to train ML models. Importance will be placed on interpretable methods to ensure that the cause (event type, unit generating the event, time window) of a future failure is known, such that preventive actions can be taken. It is difficult to augment expert knowledge into the data sets, as such the causes of each failure are unknown, because of this the data sets are unlabelled aside from the time of the failure. Methods that attempt to uncover the underlying labels of which samples containing events that cause the failures will therefore be developed.

There are some common problems facing the PdM field, and a few of these will be the main focus of this thesis. An important component of the vast majority of PdM applications is that the results must be *interpretable*. This is necessary to ensure that proper actions can be taken to prevent the potential future failure. Because of this, an interpretable pre-processing method using window aggregation will be designed, and *Random Forest* (RF) will be the model of choice.

The lack of expert knowledge leads to the data sets only being partially labelled. While the data sets contain information about when failures have occurred, they do not contain any information about what sequence of the events caused the failures. Uncovering the true labels of the data sets will be accomplished using an algorithm known as *Multiple Instance Learning* (MIL). The data sets heavily imbalanced due to the fact that failures are rare. This is a common problem within the PdM field, particularly for those using ML, as failures are generally rare. Because of this, an extension to RF called *Balanced Random Forest* (BRF) will be employed in an attempt to work around this.

In this thesis, a novel approach to PdM will be described, with improved ability to handle the challenges described above, related to lack of labelling and imbalanced data. The novel approach will be tested on operational data, and its performance will be compared to previously described methods. A final goal is to be able to effectively predict future failures, in order to produce an algorithm that has future usage in a real operational system.

Structure

The structure of this thesis will be as follows;

- As a start, a general introduction to alarm systems and maintenance strategies will be given in Chapter 2. The current trends within the PdM field will be discussed, along with the major challenges the field is currently facing. The different PdM approaches will also be also be introduced, as well as some common ML algorithms that are used within the field.

- In Chapter 3, the data sets will be introduced, together with a high level description of the ships that produced the data. The pre-processing techniques that are applied on the data will also be described in this chapter. These techniques are window aggregation and *Random Indexing* (RI).

- Then, in Chapter 4, an introduction to *Classification and Regression Trees* (CARTs) will be given, before diving into RF in particular. Here, RF and its benefits over other methods will be discussed, and the motivation for choosing this particular method will be given.

- After which, in Chapter 5, the concept of imbalanced data will be described along with different ways of handling this problem. BRF in particular, will then be focused on as a solution to this problem. The chapter will continue by discussing *weakly labelled data* and how the problem can be solved using MIL. BRF and MIL will then be combined into a method that will be referred to as *Multiple Instance Learning through Balanced Random Forest* (MIL-B-RF).

- Finally, in Chapter 6, the models resulting from using the discussed techniques for training will be presented. Additionally, comparisons between models trained on subsets of the data sets will be conducted, as well as comparisons with the results presented by other related papers.

CHAPTER 2

Predictive maintenance

Overview and motivation

This chapter will introduce *alarms systems* and *events*, before discussing the major challenges that these systems are facing currently. It will be shown how some of these can be alleviated through data automation. Second, a comparison between *Predictive Maintenance* (PdM) and other maintenance strategies will be explained. The main challenges that the current state of the art PdM implementations are facing will be outlined. Third, a few classes of algorithms used for PdM will be evaluated. Finally, some of the recent trends within the predictive maintenance field will be discussed. This will lay the foundations for the work that will be presented in the later chapters of the thesis.

2.1 Alarm management and event data management

Alarm systems are an important component of industrial systems, they track numerous *events* and signals within the system, and they are designed such that a human operator can take action at the appropriate time to maintain the operation and functionality of the system. An event is a general term that refers to any information about the happenings in the system at a given time, and can belong to one of three different *types*, these are referred to as *information*, *warnings* and *alarms*.

Information, as the name implies are strictly informational events that primarily describe the termination of tasks or changes of setpoints in the system. Warnings signify that while there is no problem yet, there may be indications of a future issue in the system. Unlike warnings and information, alarms are events that require an action to be taken by the operator monitoring the system. When an alarm turns on it is said to be *active*, and an *active alarm event* is generated. When the alarm turns off it is said to be *inactive*, and an *inactive alarm event* is generated. An alarm is generally activated when a measured condition or estimated state exceeds a preset value, and remains active for the time between the generation of these two events. Alarm activation is illustrated in Figure 2.1, here a measured condition exceeds a given threshold, resulting in the alarm becoming active.

Alarms are generally assigned one of multiple *severities*. Their definitions depends on the alarm implementation, but they are commonly categorized into: low, medium and high (DeltaV 2016) (Goel, Pistikopoulos et al. 2019), or into: warning, serious, fatal (European Southern Observatory (ESO) 2007). An alarm

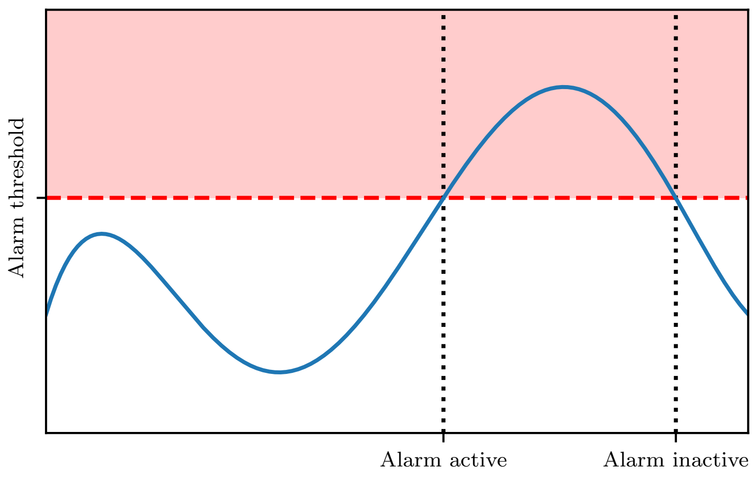


Figure 2.1: An alarm becoming active when a measured condition reaches some preset threshold, and becoming inactive after falling below the threshold again.

with a low severity means that something anomalous has happened, but the system can continue to function. A medium severity means that the system cannot perform properly, but a full shutdown is not required and recovery can be attempted. If an alarm has a high severity, the system is unable to recover and loss prevention must be performed to reduce potential economic losses and potential loss of human lives (DeltaV 2016).

Each alarm has an *acknowledgement status* stating whether or not the operator has *acknowledged* that the alarm is active (Goel, Pistikopoulos et al. 2019) (European Southern Observatory (ESO) 2007). When an alarm first activates, it has an acknowledgement status of *unacknowledged* and therefore enters a state of 'Active Unacknowledged'. If the operator notices the alarm, they will acknowledge the alarm resulting in its state changing to 'Active Acknowledged'. When the operator has taken action and the alarm becomes inactive it enters its final state of 'Inactive Acknowledged'. If however, the alarm either goes inactive without the operator taking any action or the operator takes action without acknowledging the alarm first, it will enter a state of 'Inactive unacknowledged', the operator might then acknowledge the alarm after it has become inactive, changing its state to 'Inactive acknowledged'.

An alarm can have multiple *levels* explaining why it has become active. The level of an alarm can be either 'low' or 'high', this refers to whether the alarm is active because the monitored value or condition is lower or higher than a preset threshold or safety limit, e.g., whether the temperature of a piece of equipment is too low or too high. Alarms can be divided into multiple levels to describe how far outside the normal bounds the current value is, e.g., 'low-low', 'low', 'high', 'high-high', etc. (Goel, Pistikopoulos et al. 2019).

Using the concepts of alarm- severity, acknowledgement and levels, one way of defining an event is,

$$E = (p, t, m, v, c, l). \tag{2.1}$$

p is the type of the event,

$$p \in \{\text{Information}, \text{Warning}, \text{Alarm}\}.$$

t is the timestamp of the event, and m is the event message, this can be a string containing various details about what has happened. The rest are only used if $p = \text{Alarm}$, v is the severity of the alarm,

$$v \in \{\text{Severity}^1, \dots, \text{Severity}^V\},$$

where ^{V} is the number of different severities. c is the acknowledgement of the alarm, either unacknowledged or acknowledged. l is the level of the alarm,

$$l \in \{\text{Level}^1, \dots, \text{Level}^L\},$$

where L is the number of alarm levels for the alarm.

There are many challenges facing current alarm systems, one of the most important is known as *alarm flooding* (Goel, Datta and Mannan 2017). Alarm flooding is a scenario where the amount of alarms becoming active at the same time is too high for the human operator to respond to. A human performance model study by Reising, Downs and Bayn (2004) showed that the amount of alarms an operator can respond to in a 10 minute window is around 10. This highlights the importance of good alarm system design to ensure that the operator is never presented with more alarms than he or she can reasonably respond to in a short amount of time.

This leads to another crucial part of an alarm management system, that is the *human-machine interface* (Goel, Datta and Mannan 2017). For the operator to be able to properly maintain the system and quickly respond to alarms, the alarms and other events must be presented in a way that allows him or her to understand the cause of the current system state. If too much information is presented at once without the ability for the operator to filter out other unrelated events, they will be unable to quickly resolve the issue potentially increasing damage costs.

The *tuning* of alarm variables and settings is another important task when designing an alarm system (Goel, Datta and Mannan 2017). Process setting values are generally set by the licensor or engineering company, but over time these values may require tweaking because of changes in the operation or maintenance operations. Poorly calibrated alarms can result in both false positives and false negatives, with false positives contributing to the alarm flooding problem.

Other issues are often encountered in alarm systems, one example being a lack of comprehensive documents which describe the philosophy behind the alarm design. Inadequate operating procedures can be caused by the operator being unaware of how to manage specific situations. Lack of resources can occur, because justifying the costs of a proper alarm system implementation to stakeholders can be an issue (Goel, Datta and Mannan 2017).

With the enormous amounts of data now becoming available from industrial systems, both in the form of numerical signal data and event logs, the opportunity to use data mining and analysis techniques to make more informed decisions about these systems is emerging (Goel, Pistikopoulos et al. 2019). The use of these techniques can result in discovering previously unknown connections in the system which can lead to better prediction of future failings of the system. This data can also be used to perform intelligent maintenance on parts of the system that are at risk of failing in the future, which will be the main point of discussion for the rest of the chapter.

2.2 Reactive, preventive and predictive maintenance

The way maintenance is performed on equipment in industrial systems can have major economic implications based on the length of the resulting downtime and cost of repairs. For a long time there were only two prevalent maintenance strategies, *reactive maintenance* also known as *Run-to-Failure* (R2F) wherein parts are only replaced after they break and *Preventive Maintenance* (PvM) where parts are replaced at regular intervals to minimize downtime of the system. In recent years however, with the increase in available data and computational power, a third maintenance strategy using *Machine Learning* (ML) algorithms has become of both industrial and academic interest, which is called *Predictive Maintenance* (PdM) (Susto et al. 2015) (Sakib and Wuest 2018). PdM is a method relying on statistical methods and data analysis to predict when a part is nearing failure or the end of its *Remaining Useful Life* (RUL).

R2F is the simplest maintenance strategy, and for this reason it is still frequently used. This strategy does not require any maintenance scheduling in advance or estimation of when a part requires repairs or replacement. While this saves money upfront by using parts for their entire lifespan, the economic losses from increased downtime of equipment and immediate scheduling of maintenance generally outweighs this benefit (Susto et al. 2015).

If the expected lifetime of a part is known, PvM can be implemented as an alternative. With a PvM strategy, parts are regularly replaced based on knowledge of how long a given part is likely to last before requiring repairs or replacement, this maintenance strategy is therefore also commonly referred to as scheduled maintenance (Carvalho et al. 2019). PvM has a higher upfront cost than R2F as replacement parts must be bought more often, because they are often replaced before the end of their RUL. The downtime of the system is however shorter because it only remains down while the maintenance is performed, unlike R2F where the system might have to stay down from the time of the part breaking until the maintenance is started as well. Occasionally a part can however break earlier than expected, due to unforeseen events, resulting in the same downtime and economic costs as using R2F.

With increasing amounts of available signal-, event-, maintenance data, and computing power, PdM with ML, statistical inference methods can be used to accurately predict when a part will require maintenance (Ran et al. 2019). If properly implemented, PdM saves money by reducing both replacements of healthy parts and the number of unexpected failures that occur. PdM does however come with a few challenges that need to be overcome in order to have a well-functioning maintenance strategy.

- The PdM strategy for any single system generally has to be *tailored* for that particular system, requiring *expert knowledge*, which increases the upfront cost of implementation (Susto et al. 2015). For ML methods, expert knowledge may also be required to avoid *unlabelled* data, because an expert may be required to determine what caused a failure.

- Modelling the health of a part using *quantitative indicators* to determine whether maintenance is currently required based on operating costs and failure risk (Susto et al. 2015).
- Many systems rarely fail, even with a R2F strategy, a PvM strategy makes failures an even rarer occurrence, as parts are generally replaced before they break. This results in a lack of maintenance data, leading to the data being very *imbalanced*. The data being imbalanced means that there is significantly more sensor/event data for normal operation than for failures (Susto et al. 2015).
- If ML methods are used, *insufficient* data can be detrimental to the accuracy of predicting when maintenance is required.
- The *interpretability* behind the reason the system requires maintenance can be important. Many ML methods, while providing accurate predictions, cannot be easily interpreted such that a human can understand the reasons maintenance may be required.

2.3 Predictive maintenance approaches

There are many *approaches* for predicting the likelihood of failure. These approaches can be grouped in many ways, however this thesis will be classifying them based on the work of Schmidt and L. Wang (2015).

- *Physical-based model* approaches require the modelling of the system as a whole using e.g. differential equations and comparing the simulated model to the real system. If the observed state of the real system diverges from the simulated model, there may be signs of maintenance being necessary.
- *Knowledge-based* (also referred to as *Rule-based*) approaches that are designed by an expert knowledgeable about the system, its failures mechanics and failure modes. The expert(s) set rules that let the model predict failures. This is generally either a list of manually designed 'IF-THEN' statements that determine if the system has entered an abnormal state. The use of *fuzzy systems* also falls under this type of approach, these use fuzzy logic to determine the current state of the system using potentially noisy or imprecise measurements.
- *Data-based model* approaches involve modelling the system using stochastic, statistical or ML algorithms with the data generated from the system (generally sensor- and/or event data) to determine when maintenance is required. These approaches can be further divided into two sub-approaches:
  - *Condition-based* approaches (Sakib and Wuest 2018) (also commonly referred to as *Condition-Based Maintenance* (CBM)) monitor different system states, these can either be sensor data, or more

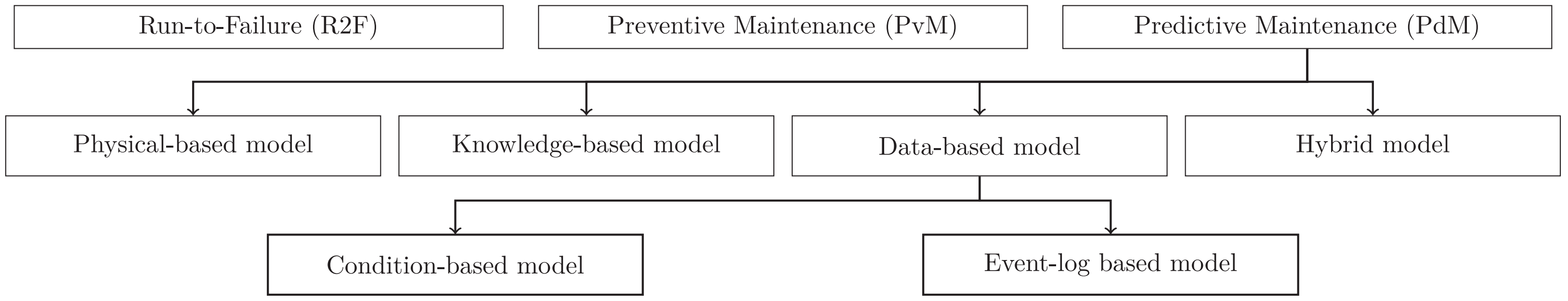


Figure 2.2: A flowchart showing a general overview of maintenance approaches

complex states estimated by other algorithms, such as the health of a part.

  - *Event-log based* approaches (J. Wang et al. 2017) rely only on analyzing the raw event-log data generated from these systems. These event data generally include information, warnings and alarms, as described in Section 2.1.

- *Hybrid model* approaches combine multiple of the above-mentioned approaches, e.g. using a combination of knowledge-based models and data-based ones.

Physical-based model approaches do not require any data from the system to be implemented and used effectively, thus avoiding the issues of imbalanced, unlabeled, and insufficient data (Ran et al. 2019) (Lughofer and Sayed-Mouchaweh 2019, p. 5) (Tinga and Loendersloot 2019, pp. 331–332). However the models used in these approaches require expert knowledge to be constructed, often resulting in high costs, they also need to be tailored to the specific system, and can be difficult to create for large complex systems.

Knowledge-based approaches also have the benefit of not requiring any data to be collected before their use (Ran et al. 2019) (Maciel and Ballini 2019, p. 406). In addition these approaches are very interpretable, because for one can simply look at which strict- or fuzzy 'IF-THEN' statements have led to the model determining that maintenance is required. As with physical-based model approaches, these approaches can be expensive to implement because of the expert knowledge required in their creation, they can also be difficult to extend when new failure types are encountered.

Both condition- and event-log based approaches do not necessarily have to be completely tailored to the individual system, as they rely on data generated by the system, rather than expert knowledge, and can therefore be cheaper to design and implement (Peng, Dong and Zuo 2010). These approaches also scale easily to large systems with thousands of components and failure modes, as using a physical-based or knowledge-based approach in these scenarios may not be feasible. Data-based approaches do however rely on the availability of large amounts of data generated from the system to train models that are able to reliably predict the likelihood of failure. Interpretability, unlabelled data and imbalanced data can all be issues depending on the choice of algorithm.

Hybrid model approaches' advantages and disadvantages depend completely on which other approaches are combined to create the hybrid model (Tinga and Loendersloot 2019, p. 332). They are generally more complex to design and implement than just using a single model approach, but often provide superior results.

An overview of the maintenance strategies and approaches discussed so far in this chapter is provided in Figure 2.2.

2.4 Common PdM algorithms

When using data-based model approaches for PdM, numerous statistical and ML algorithms were reported in the literature for various PdM approaches over the recent years. These all have their advantages and disadvantages, and can be grouped into classes of similar algorithms. The most important distinguishing factors between different algorithm classes are whether they are used for *regression* or *classification*, their interpretability, and whether they are robust to imbalanced data sets. Algorithms can also be distinguished by how prone they are to *overfitting*. Some algorithms can model unlabelled data, and they are called *unsupervised*, algorithms that require labelled data are similarly referred to as *supervised*.

Support Vector Machines

Traditionally a supervised algorithm, *Support Vector Machines* (SVMs) are a class of algorithms that can be used for either classification or regression. The standard SVM is used for binary classifications, but the algorithm can be extended for use in multi-class problems via *Multiclass SVM* and for regression as well via *Support Vector Regression* (SVR). This algorithm attempts to find a hyper-plane or other non-linear border that divides the classes in the data set with the largest margins. SVMs are widely used within PdM because of their good prediction ability, even if they struggle with interpretability.

SVM was used by Praveenkumar et al. (2014) for binary classification of whether or not a failure was likely to occur in an automobile gearbox, using vibration sensor data. Faults were successfully predicted with a high precision rate of $> 90\%$ for every gear in the gearbox. A modified version of SVR was also used by Mathew, Luo and C. K. Pang (2017) on simulated time-series data, in an attempt to estimate the RUL of a component.

Neural Networks

Another class of algorithms are the *Neural Networks* (NNs). These algorithms are generally used for condition-based maintenance and are known for their good prediction ability, even with high-dimensional data, but often require large amounts of available data, they have poor interpretability, and take a long time to train. They also rely heavily on randomized initial weights leading to the algorithms converging locally, but rarely globally (Leite 2019). *Artificial Neural Networks* (ANNs) are some of the most commonly used ML algorithms for PdM and are generally used for complex systems without requiring expert knowledge (Carvalho et al. 2019). *Deep Neural Networks* (DNNs) is a term used for more complex NNs, where the features are automatically extracted through multiple *hidden layers*. *Convolutional Neural Networks* (CNNs) are a type of DNN which are traditionally used primarily for image analysis, but they have shown success when used with sensor data for PdM, e.g., in Silva and Capretz (2019) and Janssens et al. (2016). *Recurrent Neural Networks* (RNNs) are a type of NN that are well suited for processing time-series data, *Long Short-Term Memory Neural Network* (LSTM) in particular is an RNN algorithm that is well suited for PdM because of its ability to make connections between temporally distant inputs, which other RNN algorithms generally struggles with.

In Scalabrini Sampaio et al. (2019), an NN was compared to the Random Forest and SVM algorithms for regression, to predict the time until failure using sensor data. While the other algorithms used for comparison, performed roughly equivalent to the NN in short term predictions, the NN performed noticeably better for long term predictions.

A comparison between a CNN, an ANN, a *Random Forest* (RF) and a SVM, for classification, was conducted in Silva and Capretz (2019), by predicting whether a failure in the near future is likely or not. Because CNNs are most commonly used for image analysis, the input data was transformed to be two-dimensional first. In 3 out of 4 metrics presented, the CNN outperformed the other algorithms, with the ANN performing the best in the last metric.

LSTM was used for predicting the likelihood of failure in Wu, Huang and Sutherland (2020), and it was compared with SVM. The two algorithms performed similarly for predicting that the system was operating as normal, but LSTM outperformed SVM for predicting the 'warning state' and 'failure state'.

Clustering

Clustering algorithms are a class of commonly used unsupervised algorithms. As the name suggests these algorithms attempt to group similar instances in the data set into clusters without knowing their labels. Many of these algorithms suffer from *the curse of dimensionality*, this refers to the fact that the average distance between instances increases as the dimensionality of the data set increases. This leads to difficulties for clustering algorithms because many of then cluster instances based on their distance from each other. One such algorithm is *K-Means*, one of the more popular clustering algorithms, it attempts to cluster the data set into k clusters, where k has to be determined by the user. This algorithm is easy to implement and generally provides good results even on large data sets as long as k is not too large (Carvalho et al. 2019).

In Langone et al. (2015), a novel approach to clustering using a form of SVM called Least Squares SVM with Kernel Spectral Clustering (KSC), was applied in order to distinguish between normal and abnormal operating conditions. The presented method was compared to the K-means algorithm and obtained better results than this more common clustering algorithm. Clustering was also used by Uhlmann et al. (2018), who used the K-means algorithm to cluster sensor data from a 'Selective Laser Melting machine tool'.

Classification and Regression Trees

Another class of common supervised algorithms commonly used for PdM are the *Classification and Regression Trees* (CARTs) algorithms. These algorithms can be used for both regression and classification. They are all based on *Decision Trees* (DTs), this algorithm splits the feature space using one single feature. This results in an algorithm with high interpretability because every decision can be interpreted as a set of boolean logic terms. DT algorithms are however prone to overfitting if allowed to run until each node only has a single data instance, A standard single-DT algorithm is rarely used for PdM, but other CART are common. *Gradient Boosted Tree* (GBT) is an *ensemble* algorithm that combines multiple DTs through a process called *Gradient Boosting*. *Random Forest* (RF) is another ensemble algorithm that combines multiple DTs. This algorithm samples both the data instances and the features of the data sets randomly and trains each DT on these random samples. This results in an algorithm that is very resilient towards overfitting. Both GBT and RF maintain some interpretability even if the decisions are not as easy to follow as with a single DT.

A comparison between DT, RF and GBT for classification using maintenance logs, was conducted by Allah Bukhsh et al. (2019). In this paper, an attempt to predict both, whether maintenance was necessary and the type of the maintenance was made. The GBT performed best for the former problem and RF for the latter. This paper also highlighted the interpretability of tree-based algorithms by demonstrating their ability to generate the feature importance for each algorithm. Another comparison was conducted by Binding, Dykeman and S. Pang (2019). Here a simple logistic regression, RF and a GBT algorithm called Extreme Gradient Boosted Trees, were compared for prediction of failures using real-world operational sensor data. Both RF and GBT outperformed the logistic regression, and each perform the best in different metrics.

A combination of condition-based and event-log approaches using discretized time-series- and event data with RF was used by Naskos et al. (2020), to predict stops in a cold forming press. Both a supervised and an unsupervised approach to prediction was demonstrated, with another algorithm being used to label data based on when failures occurred. Canizo et al. (2017) also used a combination of condition-based and event-log approaches, in an attempt to classify failure types using both alarms and operational data, again using RF.

2.5 Recent trends in PdM

In recent years the industry has been going through what is referred to as the fourth industrial revolution, or *Industry 4.0*. This revolution has been sparked by the growing digitization of many industrial systems, as well as the internet of things. One of the largest trends that has come about with Industry 4.0 is PdM. With the growing amount of digitization via networked sensors, big data, advanced analytics and machine learning, it is becoming possible to perform PdM, especially using data-based models (Bradbury et al. 2018). This has lead to both growing industrial and academic interests in the topic over the past few years.

According to Scully (2019), the number of companies explicitly working within different sectors of PdM has doubled from 2017 to 2019, and estimates that the global PdM market will grow from \$3.3 billion in 2018 to \$23.5 billion in 2024. This report also found more than 180 companies within different industry segments that explicitly state that they offer services related to PdM. It divides the industry into four major segments: Hardware, Connectivity, Storage & Platform and Analytics. The last of which is further divide into nine types depending on which type of analytics they perform, e.g. data visualization, data mining and statistical analysis. This shows that there is already a large market providing services for all the components necessary to perform PdM.

In academia there has also been a growing interest in the PdM field. In the early 2000s the number of publications within the PdM field was around 100 per year, but this number has since grown significantly to more than 500 per year by 2017 (Lughofer and Sayed-Mouchaweh 2019).

Much of the focus on PdM is taking place in the manufacturing industry, but there are many other industries that could potentially benefit from this new trend. One of the industries that has been slower to adapt to changes in maintenance strategies is the marine industry. Shorten (2012) noted that at the time only 17% of ships used any maintenance strategy more advanced than R2F, and that only 12% of these (roughly equivalent to 2% of all) used PdM. This statistic depends on the classification of PdM strategies, and could consist of mostly condition-based approaches, as was discussed in Section 2.3. Allen (2005) found that 71% of component failures in submarines are not age-related, but occur randomly. This shows that even PvM is not a particularly good strategy for maintenance of marine vessels, as the useful life of a component can often not be reliably estimated (Knutsen, Manno and Vartdal 2014). The marine industry could therefore benefit greatly from the adoption of PdM strategies.

In this chapter the concept of alarm management and events were first introduced. Then the three major maintenance strategies were introduced, R2F, PvM and PdM. The different approaches that can used to when performing PdM were briefly described, along with which ML algorithms are commonly used for data-based approaches. Finally PdM's role in Industry 4.0 has been introduced and discussed. The major challenges the field is currently facing, along with how different approaches and algorithms alleviate these has been explained.

One of the industries that has been slow to adapt to this new trend is the marine industry, particularly for maintenance of ships. This will be the building block for the work that will be done in this thesis, as in the next chapters the data sets, coming from multiple vessels will be introduced.

CHAPTER 3

Log events based PdM in Electric Propulsion Systems

Overview and motivation

In this chapter, the definitions of alarms and events introduced in the previous chapter will be applied to the data sets from four different ships. First, a brief introduction to the ships will be given, the sources of data on board the ships will be listed. A slightly different mathematical representation of events, than the one introduced in Chapter 2, will also be defined, as it will be more relevant for these particular data sets. The data set pre-processing process will then be described, using two separate methods: window aggregation and *Random Indexing* (RI). The window aggregation will solve the problems of interpretability and uneven periods between sample, while the RI will represent the events based on the context they commonly appear in. After these methods are presented, they will be combined into one method developed specifically for event-log based data. This pre-processing will allow for the use of the methods that will be presented and developed in Chapters 4 and 5.

3.1 System description and data description

This thesis is based on real world data collected from four *Liquefied Natural Gas Carriers* (LNGCs) equipped with *Electrical Propulsion Systems* (EPSs), these vessels will be referred to as 'vessel 1', 'vessel 2', 'vessel 3' and 'vessel 4' for the rest of the thesis. On board each vessel, the data cover sensor measurements as continuous numerical variables and logged events as structured categorical (nominal) data. The combined data monitor the overall operation across all interconnected equipment such as the *Propulsion Control Unit* (PCU), variable frequency drives, generators and transformers in the propulsion plant, and propulsion motors in addition to the cooling systems, auxiliary units, and protective devices. The data therefore originate from multiple locations, and they are collected in multiple *sources* categorized based on event logging software and their intended use.

Figure 3.1 shows a general overview of the structure of the electrical propulsion plant and propulsion drives. The generators convert fuel to electrical energy and they power all the vessel units, including the propulsion system, through the *High Voltage Main AC SwitchBoard* (HVMACSBD). For reliability and redundancy benefits, the switchboard is divided symmetrically into sections through the even number of generators where the loads are arranged symmetrically through sections of the bus bar. Supply transformers are used to step up/down current/voltage as needed. The frequency-controlled propulsion drives consist mainly of AC to DC power rectifiers connected to a common DC bus and the inverter. At the end, the *Electrical Propulsion Motors* (EPMs) are connected to the propeller through gearboxes.

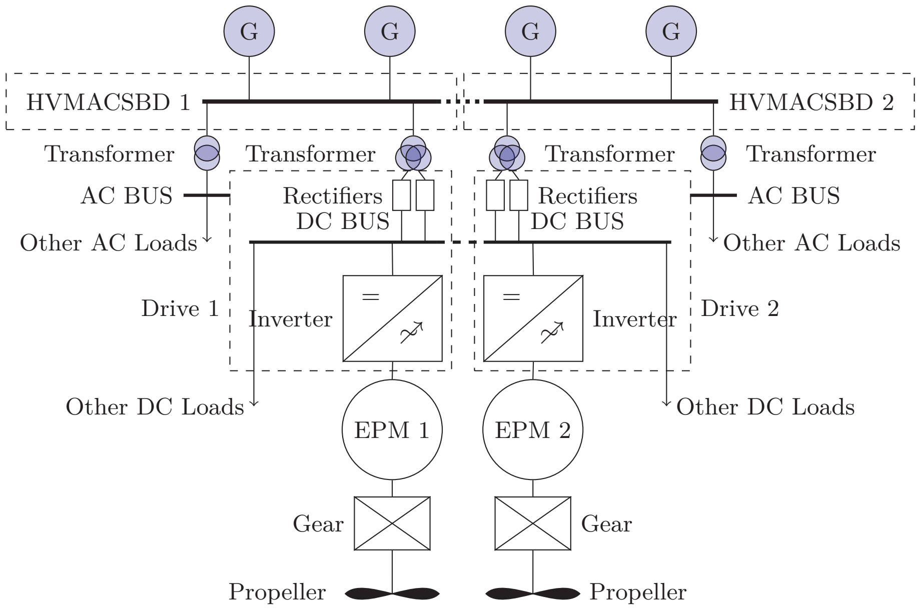


Figure 3.1: Example of an EPS vessel as described in Section 3.1

Because of the vessels' large size and numerous units, data dimensionality easily increases drastically, to the level of thousands of features, especially due to the various events aspects such as types, severity, and levels explained in Section 2.1.

There is approximately 1.5-2 years of data for each vessel, these data are divided into two types: numerical data in the form of continuous signals and event logs, an example of such signal data is the motor speed as shown in Figure 3.2. This thesis will focus on using the rich event data for predicting failures. The raw event data are stored in individual tables for each source with one column for index, one for the timestamp and one for the event message, as illustrated in Table 3.1.

It is worth noting that the events occur irregularly, because the vessels do not run in regular patterns. This is unlike the machinery in some other fields, such as in manufacturing, where the equipment operate continuously.

The concept of an event has previously been defined in Equation (2.1), but that definition is not entirely suitable for these data sets, let us therefore define the i^{th} event as

$$E_i = (t_i, s_i, m_i), 1 < i < N. \tag{3.1}$$

t_i is the timestamp when the i^{th} event was logged, measured in seconds with a particular reference datetime with resolution in milliseconds; s_i is the source on the vessel where the i^{th} event was logged,

$$s_i \in \{\text{Source}^1, \ldots, \text{Source}^k, \ldots, \text{Source}^K\},$$

where ^{k} indicates the k^{th} source, K is the number of sources on the vessel. m_i is the message of the i^{th} event,

$$m_i \in \{\text{Message}^1, \ldots, \text{Message}^m, \ldots, \text{Message}^M\},$$

and ^{m} indicates the m^{th} message, M is the number of different messages per vessel, note that the same message can be logged in multiple different sources e.g. $E_{100} =$ (983462.842, `Drive1_raw_data`, `INU_Stp_Cmd`) and $E_{101} =$ (983462.843, `Drive2_raw_data`, `INU_Stp_Cmd`). N is the total number of events on each vessel ranging from 340 thousand to 1.187 million.

Index	Timestamp	Event message
1	91707658.474	Message A
2	91707658.474	Message B
3	91707747.913	Message C
4	91707747.913	Message D
5	91708001.286	Message B
⋮	⋮	⋮
4757	146774247.429	Message B
4758	146775573.940	Message C
4759	146775573.940	Message D

Table 3.1: Table of event data from one source on a vessel, each row is an event with a timestamp in seconds and an event message

While there are multiple different types of failures that can occur on the vessels, the failures of interest for this thesis are trips in the inverter unit inside the drives. This failure will cause the drives to halt and the vessel will lose propeller control. Because all the electrical appliances on board the vessels are powered by the same generators as the drives, electrical failures in other parts of the system can propagate up through the system and into the engines which can result in the inverter unit tripping, therefore it is crucial to use all the event data. The number of occurrences of the inverter trips are very limited, ranging from 36 failures in 340 thousand events to 14 failures in 580 thousand events.

Unfortunately, while the vessels are all equipped with EPSs, they are not identical. The vessels can have differing number of equipment, software versions, etc. Therefore, while the same methods will be used to model all the vessels, it is not a straight forward process to combine the data from all four vessels to train and test one single model, the vessels will therefore have separate prediction models.

3.2 Pre-processing

As mentioned in Section 3.1, the event data were originally logged in separate tables for each source on the vessels. For the methods that will be introduced

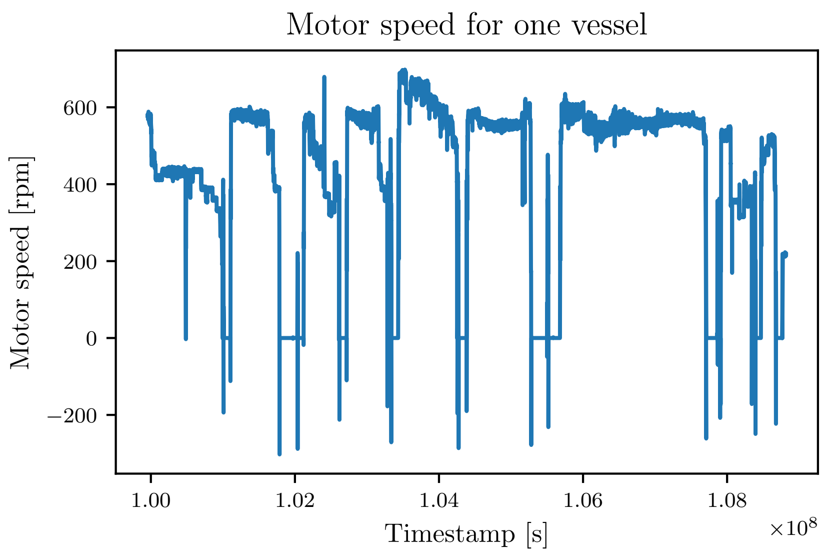


Figure 3.2: Motor speed in rpm for one of the vessels, timestamps offset with a particular reference datetime

in Section 3.3 and Section 3.4, and later developed in Section 3.5, the data sets are easier to work with if they are in chronological order. To distinguish events with the same event message and different sources, the concept of *event types* (Vasquez Capacho et al. 2017) is introduced. An event type e is defined as a unique combination of a source and an event message, where the event message contains information about the unit, severity (warning, serious, fatal), level (low-low, high-high, ...), acknowledgement (unacknowledged, acknowledged), and activation status (active, inactive) (see Section 2.1 for more information about events and alarms),

$$e \in \{\text{Type}^1, \ldots, \text{Type}^y, \ldots, \text{Type}^Y\},$$

where ^{y} indicates the y^{th} event type and Y is the number of event types. The definition of an event that will be used for the rest of this thesis is changed to

$$E_i = (t_i, e_i). \tag{3.2}$$

the source and event message are combined into an event type e_i, the event type of the i^{th} event. An example of some events following this new event definition are E_{100} = (983462.843, {`Drive1_raw_data`, Message A}), where {`Drive1_raw_data`, Message A} is the event type.

Table 3.2 lists general information about each vessel's data set. The table compares multiple properties of each vessel's data set; the number of events and event types, the difference in time between the first and last event logged (the timespan), the number of sources logging events on the ship, and the number of failures (trips in the inverter unit as mentioned in Section 3.1). The timespans of each vessel's data set only differ by a maximum of 2 months, but the amount of events logged varies significantly with 'vessel 3' logging more than 3 times the number of events produced by ‘vessel 1’. The failure to event ratio is defined as $\frac{N_F}{N}$, where N_F is the number of failures and N is the number of events. This failure ratio also differs significantly between vessels; with ‘vessel 1’ experiencing a failure ratio of 1 to $9,431$, while ‘vessel 4’ experiences a ratio of 1 to $41,494$. These failure ratios show a large *imbalance* between the two classes that represent the system status of operation in the data sets.

Vessel	Events	Event types	Timespan	Event sources	Failures
1	339,521	1,737	638 days	16	36
2	443,624	3,232	690 days	36	40
3	1,187,118	3,331	697 days	26	48
4	580,914	2,308	675 days	24	14

Table 3.2: Table summarizing the number of events and event types, time between first and last event, number of sources, and number of failures of interest for each vessel.

Using the event definition in Equation (3.2) all the original data tables (e.g. Table 3.1) from the different sources are combined into one large table, Table 3.3. This new table now contains the event index i, the timestamp t_i and the event type index y as nominal representation of the event type, for each event E_i. The raw data were stored with each event message as a string, this representation is easier for manual analysis by human operators, but is both memory- and computationally inefficient, resulting in the full data set being multiple GB in size. The new data format with a nominal representation of the event type and a separate lookup table is between 9 and 31 MB in size for each vessel, making for both easier storage and operation on the data. The code for this process is found in Chapter 7.

Index	Timestamp	Event type index
0	91666107.283	159
1	91666107.283	160
2	91666107.283	147
3	91666107.283	156
4	91666107.284	155
⋮	⋮	⋮
339518	146775573.940	9
339519	146775574.302	146
339520	146775574.302	154

Table 3.3: Final output table containing one event per row in the format (i, t_i, e_i), timestamps offset with a particular reference datetime

3.3 Aggregating over windows

To train a model using the data sets from the EPS equipped vessels the data must first be transformed in a way that maintains the *event context* of the events. An event context will be defined as a set of events occurring within a certain short time frame of each other, e.g., all the events that occur within a 10 minute window. One way of transforming the data to take into account the event context is the approach proposed in Gutschi et al. (2019), by aggregating over short non-overlapping *Time Windows* (TWs) of a fixed time duration. These TWs represent event contexts and they are then used to create sequences of contexts. This is done by aggregating with various aggregation functions over the TWs using *Rolling Windows* (RWs), as visualized in Figure 3.3. This algorithm can be broken down into two main steps:

1. First the data are split into non-overlapping TWs of a fixed time duration Tw_L. For each TW the number of occurrences of each event type is counted, the TWs are thus represented by the number of times each event type occurred during that TW.

2. To get a sequence of contexts, these TWs are then aggregated over using a RW of size Rw_L. While the TWs merely count the number of occurrences of each event type, the RWs aggregate using various aggregation functions on the event type counts from the TWs.

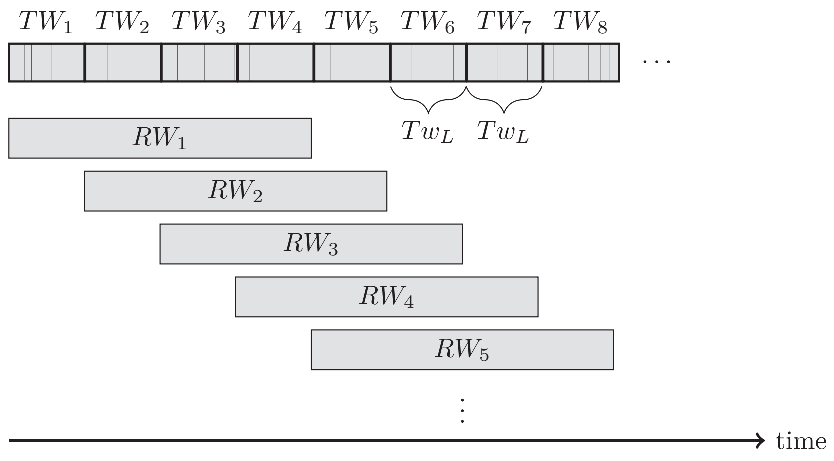


Figure 3.3: An example of window aggregation. The thin, faded lines are timestamps of events, split into TWs of fixed length Tw_L, then aggregated over with RWs of size 4.

For this project three aggregation functions are chosen,

- The minimum is chosen because the absence of a particular event type can explain higher predictive confidence that the RW does not predict a failure.

- The mean is chosen because the number of occurrences of an event type being consistently high over time could be an important predictor of an imminent failure.

- The maximum is chosen because a high number of occurrences of a particular event type could be an indicator for a failure.

The full algorithm is described in Algorithm 1.

Algorithm 1: Aggregating over time windows and rolling windows

```
Data: Event data $E$
Result: Matrix of RW vectors, $RW$
$TR = Rw_L/Tw_L - 1$ // Number of TWs per RW
$N_T = (t_n - t_0)/Tw_L$ // Number of time windows
$N_R = N_T - TR$ // Number of rolling windows
$TW \in \mathbb{R}^{Y \times N_T}$ = Empty time window matrix;
$RW \in \mathbb{R}^{3Y \times N_R}$ = Empty rolling window matrix;
for $tw = 0 : N_T$ do
    foreach $i$ do
        if $tw \cdot Tw_L + t_0 <= t_i < (tw + 1) \cdot Tw_L + t_0$ then
            $y$ = The event type index of event $e_i$;
            $[TW]_{tw,y} += 1$;
for $tw = 0 : N_T - TR$ do
    for $y = 0 : Y$ do
        $[RW]_{tw,3e} = \min([TW]_{tw,y} : [TW]_{tw+TR,y})$;
        $[RW]_{tw,3e+1} = \text{mean}([TW]_{tw,y} : [TW]_{tw+TR,y})$;
        $[RW]_{tw,3e+2} = \max([TW]_{tw,y} : [TW]_{tw+TR,y})$;
```

3.4 Random indexing

Random Indexing (RI) is an incremental method for both representing context based data and dimensionality reduction. The method was originally developed for word space models, but can be generalized to other context based data (Sahlgren 2005). RI is based around representing each unique data point (word, event, etc.) as the context it appears in. Examples of usage for the method are: Failure prediction based on log files (Fronza et al. 2013), automatic text summarization (Chatterjee and Sahoo 2015) and enhancing web proxy caching (Pernabas, Fidele and Vaithinathan 2019).

The RI process consists of two main steps:

1. Choose a vector of length N_{RIL} smaller than the number of event types Y in the data set. Assign each event type an empty *index vector* of length N_{RIL} and sparsely populate it with 1s and -1s in random locations. Additionally, assign each event type an empty *context vector* of the same length.

2. Loop through each event in the data set, every time an event occurs add the index vectors of a preset number of preceding and succeeding events' types to the context vector of the event type. These context vectors are the new representation of each event.

The event data obtained from the EPS equipped vessels share multiple similarities with word based data. The data has a set number of unique words/event types that occur successively in common contexts. The use of co-occurrence matrices, which describe how commonly different words appear next to each other, is common when making word space models. This is because they are both theoretically attractive and experimentally successful (Sahlgren 2005), but they quickly become computationally impractical as the amount of data increases. This is because the size of the co-occurrence matrix scales with the number of unique words in the data set, however the vast majority of words rarely occur (Zipf's law) (Zipf 1949) resulting in a large, but very sparse co-occurrence matrix. This similarly holds true for the data obtained from the EPS equipped vessels where only a handful of event types occur the majority of the time, as illustrated in Figure 3.4. This problem is usually solved by applying some forms of dimensionality reduction like singular value decomposition or principal component analysis (Sahlgren 2005), this however is computationally expensive and must be done every time new data is acquired.

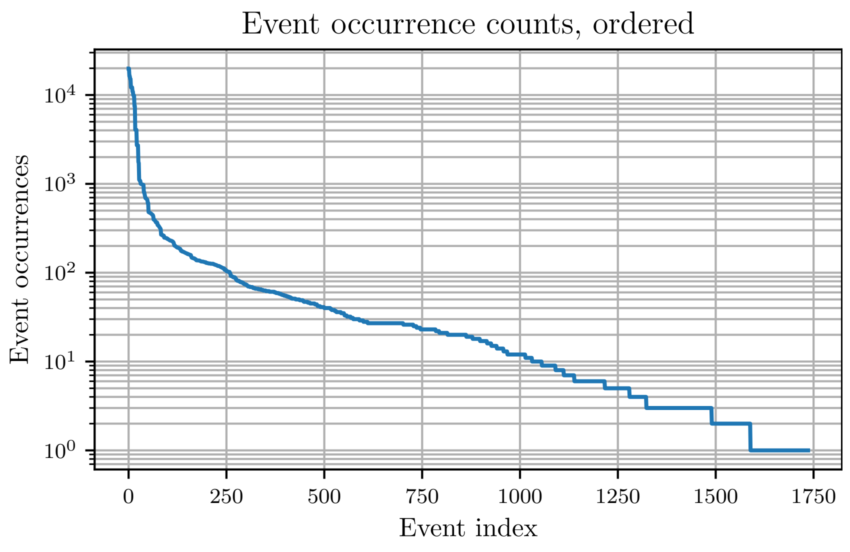


Figure 3.4: Number of times each event type occurred on 'vessel 1' .

RI has a number of advantages over these method. It is inherently incremental and it is therefore possible to look at the context of an event type without analyzing the entire data set. This is especially useful for huge data sets which take a long time to process. This also means that the context vectors can be updated later if more data is acquired, without having to reprocess the original data again, as would be necessary with methods like taking the SVD of the co-occurrence matrix. Because the dimensionality of the context vectors is fixed, RI can also easily be expanded on with new event types whenever they are introduced, without increasing the dimensionality of the data. Also since the dimensionality of the context vectors is lower than the number of event types, there is an implicit dimensionality reduction depending on the selected context vector length N_{RIL}.

The implementation of RI used for this project requires a slight modification from the usual algorithm. The context vectors are usually calculated iteratively with the following equation

$$C_{e_i} = C_{e_i} + \sum_{j=i-V}^{i+V} I_{e_j}, j \neq i, \tag{3.3}$$

for all i. C_{e_i} and I_{e_i} are the context vector and index vector of the event type e_i of the i^{th} event E_i, and V is the number of events before and after event i to take the sum of. While the events from the EPS vessel data are ordered by timestamp, the events come from different sources on the vessel, therefore the timestamps may be slightly shifted at the order of milliseconds which can result in ordering of some events being wrong. There are also occasionally long gaps between events, these can be hours, days or sometimes even weeks long. Because of this, a slightly modified version of the algorithm is developed. The way context vectors are created is changed from taking the sum of the index vector of a certain number of preceding and succeeding events, to taking the sum of the index vectors of the events in a time based window centered around the given event

$$C_{e_i} = C_{e_i} + \sum_{E_j \in W} I_{e_j}, \tag{3.4}$$

for all i. C_{e_i} and I_{e_i} are the context vector and index vector of the event type e_i of the i^{th} event E_i, and W is the set of events that occur within a time frame centered on the timestamp t_i of event E_i. This eliminates two problems, the problem of ordering for events with close timestamps mattering, and the problem of two neighbouring events counting as being in a context even if a long time has passed between them. The algorithm for this modified version of RI can be seen in Algorithm 2. The python code for this process is found in Chapter 7.

Algorithm 2: Modified Random Indexing algorithm for use with time based data

```
Data: Event data $E$
Result: Matrix of context vectors, $C$
$N_{RIL}$ = cv length;
$N$ = Number of events in $E$;
$Y$ = Number of event types in $E$;
$S$ = Number of non-zero values in each index vector, even number;
$W$ = Context window size;
$I$ = Empty index matrix of size $Y \times N_{RIL}$;
$C$ = Empty context matrix of size $Y \times N_{RIL}$;
for $j = 1 : Y$ do
    pos = Generate $S/2$ random integers between 1 and $Y$;
    neg = Generate $S/2$ random integers between 1 and $Y$, excluding
      those in pos;
    $[I]_{j,pos} = 1$;
    $[I]_{j,neg} = -1$;
for $i = 1 : N$ do
    Calculate $C_{e_i}$ iteratively using Equation (3.4).
```

3.5 Overall feature engineering approach

The data for each of the four vessels are pre-processed separately as explained in Section 3.2. Then the window aggregation- and RI methods introduced in Sections 3.3 and 3.4 respectively must combined, such that the window aggregation samples contain the RI features. To accomplish this, a method that incorporates the context vectors from RI into the window aggregation method is developed.

When constructing the TWs in the window aggregation, the context vectors corresponding to the event type of each event inside the TW will be summed to create the TW context for each window. When creating the RWs, the TW contexts are summed with exponentially decaying weights w, with the last TW context in each RW being assigned the largest weight. The modified window aggregation algorithm incorporating RI is shown in Algorithm 3. The python code for this process is found in Chapter 7.

Algorithm 3: Overall approach combining window aggregation and Random Indexing

```
Data: Event data $E$, Matrix of context vectors $C$
Result: Matrix of RW vectors, $RW$
$TR = Rw_L/Tw_L - 1$ // Number of TWs per RW
$N_T = (t_n - t_0)/Tw_L$ // Number of time windows
$N_R = N_T - TR$ // Number of rolling windows
$TW \in \mathbb{R}^{(Y+N_{RIL})\times N_T}$ = Empty time window matrix;
$RW \in \mathbb{R}^{(3Y+N_{RIL})\times N_R}$ = Empty rolling window matrix;
$w$ = Vector of decaying weights;
for $tw = 0 : N_T$ do
    foreach $i$ do
        if $tw \cdot Tw_L + t_0 <= t_i < (tw+1) \cdot Tw_L + t_0$ then
            $y$ = The event type index of event $e_i$;
            $[TW]_{tw,y} += 1$;
            $[TW]_{tw,Y:Y+N_{RIL}} += C_y$;
for $tw = 0 : N_T - TR$ do
    for $y = 0 : Y$ do
        $[RW]_{tw,3e} = \min([TW]_{tw,y} : [TW]_{tw+TR,y})$;
        $[RW]_{tw,3e+1} = \text{mean}([TW]_{tw,y} : [TW]_{tw+TR,y})$;
        $[RW]_{tw,3e+2} = \max([TW]_{tw,y} : [TW]_{tw+TR,y})$;
    $[RW]_{tw,3Y:3Y+N_{RIL}} = w \times ([TW]_{tw,Y:Y+N_{RIL}} :$
      $[TW]_{tw+TR,Y:Y+N_{RIL}})$;
```

Combining the time aggregation and RI as shown in Algorithm 3 results in the samples and features for each vessel, which will be used for training and testing in later chapters. The samples from one of the vessels are presented in Table 3.4, the RWs are the samples, and the features are a combination of the window aggregations and context vectors from the RI. There are three features per event type, each corresponding to one of the aggregation functions discussed in Section 3.3: min, mean and max. This results in $3 \times 1737 = 5211$ features.

In addition to these, there are N_{RIL} features corresponding to the length of the context vector from the RI method.

RW_1	$\min(e_1)_1$	$\text{mean}(e_1)_1$	$\max(e_1)_1$	$\cdots$	$\max(e_{N_e})_1$	$C_{1,1}$	$\cdots$	$C_{N_{RIL},1}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
RW_n	$\min(e_1)_n$	$\text{mean}(e_1)_n$	$\max(e_1)_n$	$\cdots$	$\max(e_{N_e})_n$	$C_{1,n}$	$\cdots$	$C_{N_{RIL},n}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
RW_N	$\min(e_1)_N$	$\text{mean}(e_1)_N$	$\max(e_1)_N$	$\cdots$	$\max(e_{N_e})_N$	$C_{1,N}$	$\cdots$	$C_{N_{RIL},N}$

Table 3.4: Data set resulting from Algorithm 3. N is the number of samples for the vessel, N_e is the number of event types, and N_{RIL} is the length of the context vector.

Now, the vessels equipped with EPSs have been described in detail to provide a better understanding of the vessels the data relevant to this thesis originate from. The size of the raw data sets have been introduced and the mathematical definition of an event has been given. The data set pre-processing has been described and the concept of an event type has been explained along with a re-definition of an event using this new concept. Two methods of data preparation have been introduced; aggregating over windows using TWs and RWs, and RI using index vectors and context vectors. The use of the combined method presented in Algorithm 3 has resulted in data sets with lower dimensionality than the raw data, while the majority of the features remain explainable (the aggregation function features are easily explainable while the context vector features are not.)

There are however still some issues with the data sets that will be important address when discussing which methods to use for analysing the data sets further. The dimensionality of each data set is still large, with 'vessel 1' having more than $5,000$ features (shown in Table 3.4), and the data are still very imbalanced as mentioned in Section 3.1, with the failure ratio being between 1 failure to $9,431$ events and 1 failure to $41,494$ events for the different vessels. The data are also not properly labelled, the timestamps of the failures are known, but it is unknown which of the samples cause each failure, as there is no guarantee that the last sample before a failure is the one that resulted in it. These issues will be further discussed in Chapters 4 and 5.

CHAPTER 4

PdM through Random Forest

Overview and motivation

This thesis will present a purely event-log based approach to predict failures and time to failure. The lack of easily accessible expert knowledge and difficulty of augmenting this into the data and models, will lead to some issues which will be solved in this chapter and Chapter 6.

In this thesis, failure prediction models are developed through extensions to *Random Forest* (RF). To discuss RF, *Classification and Regression Trees* (CARTs) will first be introduced and explained in detail, along with important definitions. These definitions will then be used to explain RF predictions, the training process, their benefits, and introduce important concepts that will be extended on in later in the thesis.

The reasons for choosing RF are to tackle the many difficulties presented when using event-log data for prediction, some of which were presented at the end of the last chapter. These include RFs' resilience to noise features, and their ability to determine feature importance. Towards the end of this chapter these reasons will be explained thoroughly.

4.1 Classification and Regression Trees

Classification and Regression Trees (CARTs) are simple, yet powerful methods for predictive modelling, and as the name suggests they can be used for both regression and classification. These methods are well known for their interpretability, and for being able to handle a mix of both numerical and categorical data. They are recursive algorithms that model the data by dividing the data set into N-dimensional boxes (where N is the number of features) called *nodes*, using binary axis-parallel *splits*. The method starts by making one large node containing the entire training set, then both a feature and a value are chosen to split the data into two new nodes. This process is repeated for the new node, and is referred to as *growing a tree*. The growing of the tree continues until one of several stopping criteria are met, which will be discussed later. After the tree is fully grown, the final obtained nodes are called *leaf nodes*, while the ones that have been split are called *internal nodes*. An example of the regions obtained from growing such a tree is seen in Figure 4.1.

Predicting y_i, corresponding to either the numerical value or class label of a sample X_i using a CART is simple after the tree is grown. Both regression- and

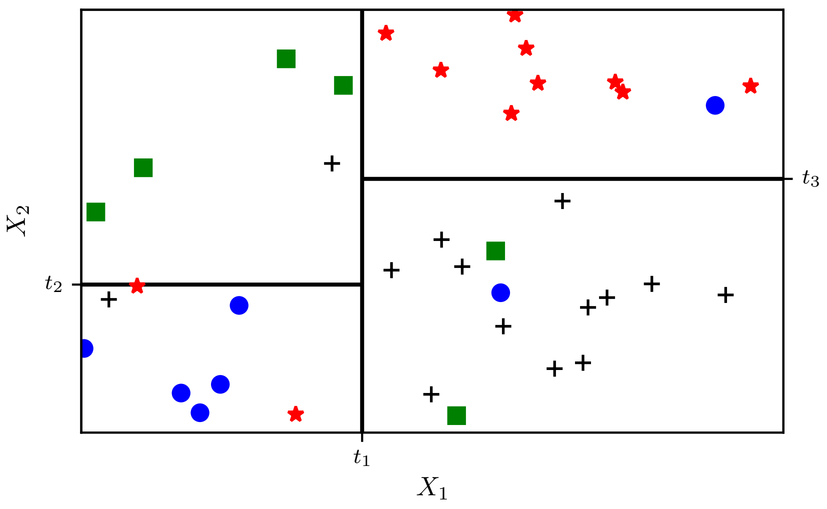


Figure 4.1: A data set with 2 features modelled by a tree with 4 leaf nodes, after performing 3 splits.

classification trees predict in the same way, by assigning a predicted numerical value or class label $\hat{y}_i$ to X_i based on which leaf node X_i belongs to. For regression trees, $\hat{y}_i$ is generally estimated to be the mean of the values y of the other samples X in the same leaf node. For classification trees, $\hat{y}_i$ is usually the majority class y in the leaf node.

To grow a CART, a *cost* function must first be chosen, to later measure the quality of a split. In a regression setting, the cost is often defined as (Murphy 2012):

$$\text{cost}\left(\mathcal{D}\right) = \sum_{i \in \mathcal{D}} \left(y_i - \bar{y}\right)^2 , \tag{4.1}$$

where $\mathcal{D}$ is the set of data pairs $\{X_i, y_i\}$ belonging to the node being split, and $\bar{y} = \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} y_i$.

For classification however, there are multiple cost functions to choose from. Their definitions depend on the *class-conditional probabilities* of the node (Murphy 2012, p. 548)

$$\hat{\pi}_c = \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} \mathbb{I}\left(y_i = c\right) , \tag{4.2}$$

which is the ratio of number of samples belonging to class $c \in \mathcal{Y}$ (where $\mathcal{Y}$ is the set of all possible classes) to the total number of samples across all classes in the set of data pairs $\mathcal{D}$ belonging to the node. Some common cost functions for classification include (Hastie, Tibshirani and Friedman 2009, p. 309):

- Misclassifcation error:

$$\text{cost}\left(\mathcal{D}\right) = 1 - \hat{\pi}_{\hat{y}}, \tag{4.3}$$

where $\hat{y}$ is the $\text{argmax}_c \left(\hat{\pi}_c\right)$

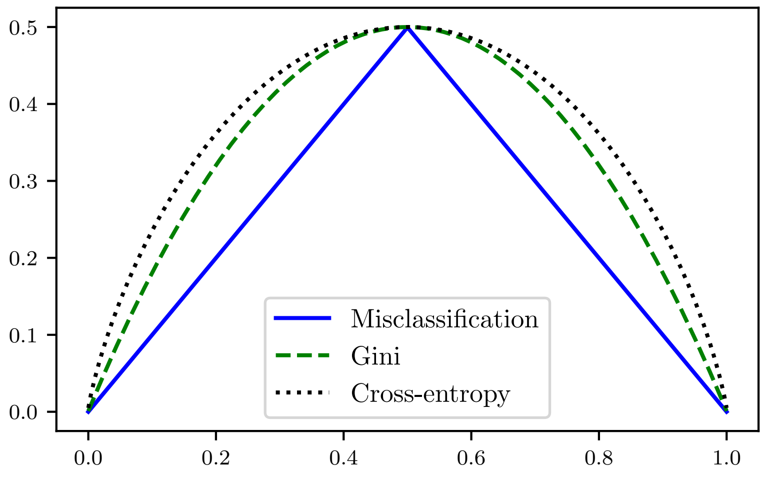


Figure 4.2: Node impurity for two-class classification with the introduced cost functions. The horizontal axis corresponds to the probability of one of the classes. Cross-entropy has been normalized to have a maximum point of 0.5. Based on Figure 9.3 in Hastie, Tibshirani and Friedman (2009, p. 309).

- Gini index:
$$\operatorname{cost}(\mathcal{D}) = \sum_{c=1}^{C} \hat{\pi}_c (1 - \hat{\pi}_c), \tag{4.4}$$
where C is the number of classes.

- Cross-entropy or deviance:
$$\operatorname{cost}(\mathcal{D}) = -\sum_{c=1}^{C} \hat{\pi}_c \log \hat{\pi}_c. \tag{4.5}$$

These cost functions are visualized in Figure 4.2.

All three of these cost functions have their own use cases, but they have a few notable differences between them. Firstly the misclassification error is not differentiable, this can easily be observed in Figure 4.2. The Gini index and cross-entropy put more weight on the class-conditional probability than the misclassification error does, this means that they assign a lower cost to nodes with higher *purity*. The purity of a node is determined by how homogeneous it is, as an example say node A contains 10 samples belonging to class 1, and 5 belonging to class 2, while node B contains 5 samples belonging to class 1, and 0 belonging to class 2. While both node A and B contains 5 more samples from class 1 than from class 2, node A is less homogeneous and therefore has a higher impurity than node B.

After a cost function is chosen the tree can be grown. For every leaf node, three choices have to be made; which feature can be used for the split, which value the feature can be split at, and if this split is worth performing. For numerical features, the feature to split and where to split it are determined by

(Murphy 2012, p. 545):

$$(j^*, t^*) = \arg \min_{j \in \{1,\ldots,N\}} \min_{t \in \mathcal{T}_j} \operatorname{cost}\left(\{X_i, y_i | X_{ij} \leq t\}\right) + \operatorname{cost}\left(\{X_i, y_i | X_{ij} > t\}\right), \tag{4.6}$$

where $t \in \mathcal{T}_j$ is the splitting value in the set of values the j^{th} feature can have, and $j \in \{1, \ldots, N\}$. This expression must be modified slightly for splitting categorical features:

$$(j^*, k^*) = \arg \min_{j \in \{1,\ldots,N\}} \min_{k \in \mathcal{K}_j} \operatorname{cost}\left(\{X_i, y_i | X_{ij} = k\}\right) + \operatorname{cost}\left(\{X_i, y_i | X_{ij} \neq k\}\right), \tag{4.7}$$

where $k \in \mathcal{K}_j$ is a value in the set of values the j^{th} feature can have.

To prevent overfitting, there are usually multiple criteria to stop splitting, if any of these criteria are met, the node in question will not be split further and will therefore remain a leaf node. Common stopping criteria include (Murphy 2012, p. 546):

- Is there a significant enough reduction in cost? This is determined by the *gain*, which is generally defined as:

  $$\Delta \triangleq \operatorname{cost}(\mathcal{D}) - \left(\frac{|\mathcal{D}_L|}{|\mathcal{D}|} \operatorname{cost}(\mathcal{D}_L) + \frac{|\mathcal{D}_R|}{|\mathcal{D}|} \operatorname{cost}(\mathcal{D}_R) \right), \tag{4.8}$$

  where $\mathcal{D}_L$ and $\mathcal{D}_R$ are the data sets of the two newly created nodes after an optimal split, as determined by Equation (4.6) or Equation (4.7).

- Is the tree at its pre-determined maximum depth?

- Do the new data sets $\mathcal{D}_L$ and $\mathcal{D}_R$ have sufficiently low costs, $\operatorname{cost}(\mathcal{D}_L)$ and $\operatorname{cost}(\mathcal{D}_R)$?

- Is the number of samples in the new data sets $\mathcal{D}_L$ and $\mathcal{D}_R$ too small?

A few terms whose use will become apparent in later chapters must be introduced. The *prediction confidence* of a sample in a classification tree is the class-conditional probability of the leaf node containing the sample, $\hat{\pi}_c$ (Leistner, Saffari and Bischof 2010). The *classification confidence* of a sample is defined very similarly (Leistner, Saffari and Bischof 2010):

$$F_c(X) = \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} \mathbb{I}(y_i = c) - \frac{1}{C} = \hat{\pi}_c - \frac{1}{C}, \tag{4.9}$$

where C is the number of classes.

In certain scenarios where the gain from performing a split is low even early on, a CART may not be able to grow properly. A solution to this problem is by loosening the stopping criteria and allow the tree to fully grow. This results in overfitting, to fix this, a process called *pruning* (Murphy 2012, p. 549) can be used. Pruning is the process of combining nodes again after a tree is grown. A tree can thus be fully grown until a single sample remains in each leaf node, and then pruned to combine some of the nodes again. Sub-trees are pruned in the order of lowest increase in error.

CARTs have multiple benefits compared to other methods, such as their interpretability as mentioned earlier. Another major benefit is the ability to easily determine the *feature importance* of a trained model. One way of determining the importance of a feature is (Hastie, Tibshirani and Friedman 2009, p. 368):

$$\mathcal{I}_j^2\left(T\right) = \sum_{\ell=1}^{L-1} \Delta_\ell \mathbb{I}\left(v(\ell) = j\right), \tag{4.10}$$

where j indicates the feature, T is the tree, $L-1$ is the number of internal nodes, ℓ is an internal node, $v(\ell)$ is the feature used for splitting node ℓ, and Δ_ℓ is the estimated gain from splitting node ℓ.

One of the major flaws of CARTs are their instability (Hastie, Tibshirani and Friedman 2009, p. 312). This is because CARTs are inherently greedy algorithms, as they split each node using the feature and value that minimizes the cost at the time of the split. If the training set changes slightly, either by removing or adding some samples, one of the earliest splits may change, resulting in that entire sub-tree changing drastically as well.

A common method that alleviates this problem is *Gradients boosted trees*. Gradient boosted trees is a method that predicts by combining the prediction of multiple trees, and therefore belongs to a group of methods called *ensemble learners*. These methods combine multiple weaker learners (called *base learners*) to give one prediction, often providing better results than a single learner, but at the cost of interpretability.

4.2 Random Forest

Another ensemble learner that can alleviate the instability problems of CART methods is *Random Forest* (RF), first introduced by Breiman (2001). As an ensemble learner, RF combines the predictive power of multiple trees to give one stronger predictions.

Like CARTs, RF can be used for both regression and classification. Predicting using RF consists of aggregating the predictions of the base learners (see Figure 4.3), for regression this commonly is accomplished by taking the average prediction (Hastie, Tibshirani and Friedman 2009, p. 589)

$$\hat{f}_{\text{rf}}^B\left(X\right) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}_b(X), \tag{4.11}$$

where, B is the number of trees in the forest, and $\hat{f}_b(X)$ is the predicted value y of sample X by tree b. For classification trees, one way of aggregating, is by taking a majority vote (Hastie, Tibshirani and Friedman 2009, p. 588)

$$\hat{f}_{\text{rf}}^B\left(X\right) = \textit{majority vote}\left\{\hat{f}_b\left(X\right)\right\}_1^B. \tag{4.12}$$

This is not necessarily the best approach however. Another way of aggregating the predictions is via the classification confidence. The classification confidence of CARTs (Equation (4.9)) can be extended to RF by taking the mean confidence of all the trees (Leistner, Saffari and Bischof 2010):

$$F_c\left(X\right) = \frac{1}{B}\sum_{b=1}^{B}\left(\hat{\pi}_c - \frac{1}{C}\right) = \frac{1}{B}\sum_{b=1}^{B}\left(\hat{\pi}_c\right) - \frac{1}{C}. \tag{4.13}$$

The prediction aggregation for a classification tree can then be calculated as the class with the highest classification confidence (Leistner, Saffari and Bischof 2010):

$$\hat{f}_{\mathrm{rf}}^{B}\left(X\right) = \arg\max_{c \in \mathcal{Y}} F_c\left(X\right). \tag{4.14}$$

This method of aggregating the predictions can be superior in certain scenarios. As an example, imagine a binary classification problem with a forest consisting of only 3 trees, here the classification confidence for a given class would be $F_c(X) \in [-0.5, 0.5]$. The trees each assign a classification confidence for whether a sample X_i belongs to class 1 or class 2, $(0.01, 0.01, -0.5)$ for class 1, and $(-0.01, -0.01, 0.5)$ for class 2. Equation (4.12) would predict sample X_i to belong to class 1, even though the classification confidence for class 1 as calculated using Equation (4.13) would be -16%. While Equation (4.14) would predict class 2, as it has a higher classification confidence of 16%.

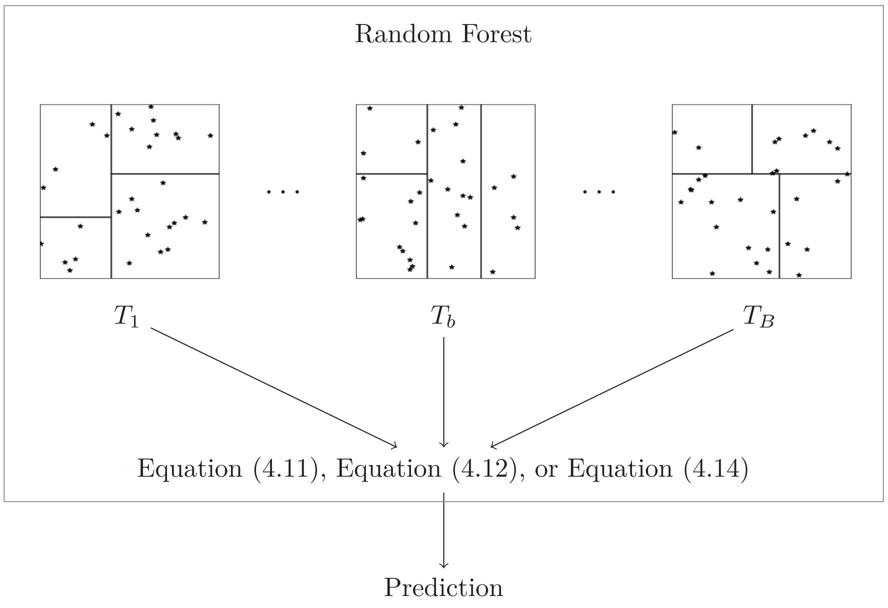


Figure 4.3: Illustration of the way Random Forests predict, by aggregating the predictions of the base tree learners.

If the trees were completely uncorrelated, taking the average would be equivalent to taking the average of B i.i.d. random variables. This would result in a variance of $\frac{1}{B}\sigma^2$ for the prediction of the forest, compared to a variance of σ^2 for each individual tree. Unfortunately this is not the case as the trees are trained on the same data set and are therefore only i.d. (not independent), resulting in a variance of (Hastie, Tibshirani and Friedman 2009, p. 588):

$$\sigma_T = \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2, \tag{4.15}$$

where σ_T is the variance of the whole forest, and ρ is the positive pairwise correlation between the trees. This shows that reducing the correlation between the trees is necessary, which is why RF uses a technique called *bootstrap* *aggregation.* Bootstrap aggregation is a technique that involves fitting models on different subsets of the training data. In RF this is accomplished by using a random subset of the training data to grow each individual tree in the forest. To further decrease the correlation between the trees, a subset of both the samples and the features is used.

This bagging results in RF being very robust against both overfitting and *noise features.* Hastie, Tibshirani and Friedman (2009, p. 596) demonstrated that even with only 6 relevant features, and 100 noise features, RF can still produce satisfactory results.

Another benefit of using RF is its use of *Out-of-Bag* (OOB) samples. Because only a subset of the trees in the forest is used to train on any given sample, the error rate of the forest can be estimated by predicting on each sample using only the trees that did not use that sample for training. This OOB error estimate is almost identical to the N-fold cross-validation error (Hastie, Tibshirani and Friedman 2009, p. 593).

As with CARTs, the feature importance of RF can easily be calculated, as the mean importance of a given feature among all the trees in the forest

$$\mathcal{I}_j^2 = \frac{1}{B}\sum_{b=1}^{B}\mathcal{I}_j^2\left(T_b\right), \tag{4.16}$$

where $\mathcal{I}_j^2\left(T_b\right)$ is the importance of feature j for tree b, as calculated using Equation (4.10). Again it can be observed that this results in reduced variance compared to CART as long as the correlation between the trees is low, because of the inherent variance reduction from taking the mean.

One potentially major issue with RF for some applications is its computation time. Because RF is an ensemble learner, whenever a prediction is made, every tree in the forest needs to predict. Thus if making predictions very quickly is necessary, RF may not be the ideal predictive method, particularly if the number of trees in the forest is high, fortunately, this is not the case for this thesis.

4.3 RF-Based approach for PdM

RF is one of many algorithms commonly used for PdM, as discussed in Section 2.4. This is because of its numerous benefits, such as the ones described in Section 4.2. This includes its generally low variance, resilience to overfitting and noise features, ability to determine feature importance, and its ease of use and implementation compared to more complex algorithms like neural networks. This is why RF will be the algorithm used for prediction in this thesis. The full PdM process that will be used in this thesis can thus be observed in Figure 4.4. The MIL-B-RF algorithm shown in the ‘model’ box will be introduced in Chapter 5.

After the pre-processing of the data in Chapter 3, there is now one data set for each vessel. For each data set, the first $3N_e$ features, where N_e is the number of event types for a given vessel, are the 3 aggregation features for every event type on that vessel, as calculated in Section 3.3. In addition the next N_{RIL} features are from the from the RI algorithm in Section 3.4, representing

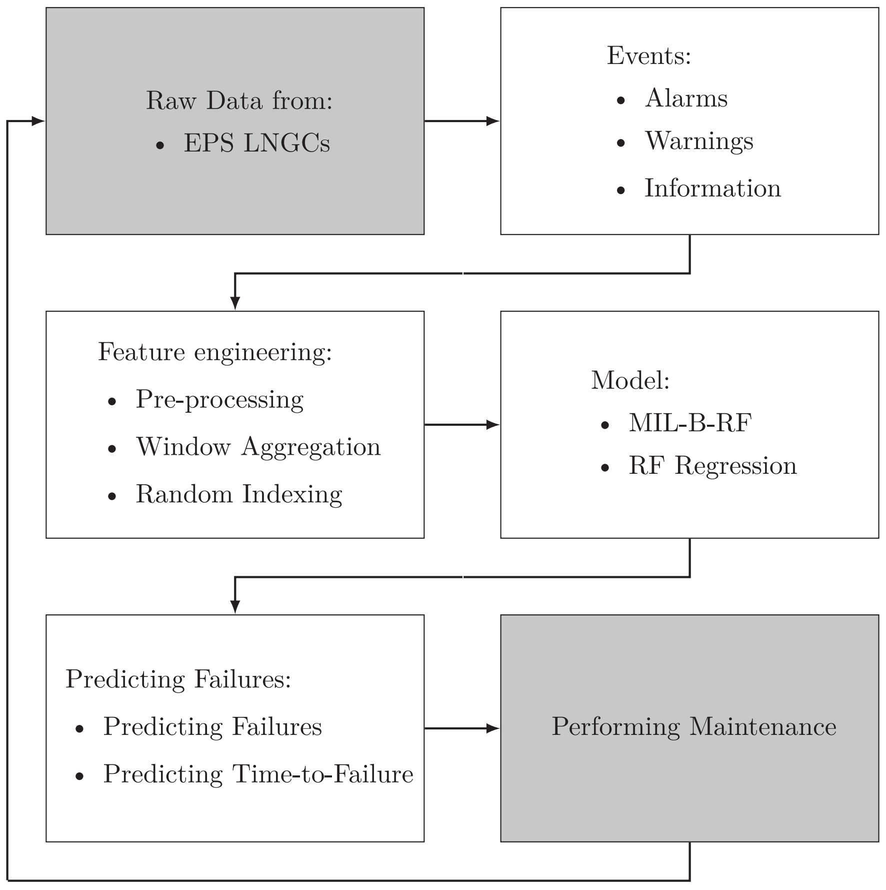


Figure 4.4: Visualization of the full Predictive Maintenance process, as presented in this thesis. Gray boxes are part of the process, but they are outside the scope of this thesis.

the event contexts. Each sample X_i can be expressed as

$$X_i = [X_{1,i}, X_{2,i}, \ldots, X_{3N_e,i}, X_{3N_e+1,i} \ldots, X_{3N_e+N_{RIL},i}] \,, \tag{4.17}$$

and they represent a rolling window of length Rw_L, rolling Tw_L at a time, as presented previously in Table 3.4. As an example, if Rw_L is set to 60 minutes, and Tw_L is set to 10 minutes, then sample X_1 represents the vessel operation during the time period 6:00-7:00, X_2 represents 6:10-7:10, etc.

The main goal of the predictions in this thesis is to perform predictive maintenance, whether maintenance is required or not is a binary classification problem, $y_i \in \{0, 1\}$:

0. A failure in the inverter unit is unlikely to occur within the next B_L days, the vessel therefore does not require maintenance.

1. A failure in the inverter unit is likely to occur within the next B_L days, the vessel therefore requires maintenance.

This will be accomplished by predicting which samples may cause a failure. One major issue however, is that the causes of the past failures are completely unknown, and it is also unknown if it is even possible to find the causes within the data sets at all. Determining the failure cause exactly would require expert knowledge, which is out of the scope for this thesis, and also very difficult to determine by a human for data sets of this size. Because of this, the labels must somehow be estimated. Suppose a naive approach is performed, where every sample within a certain time frame before a failure is labelled 1, and these samples are then used to train a RF model. This does not provide satisfactory results, likely because many of the samples labelled 1 in this way are samples from normal operation and may therefore be similar to some of the samples labelled 0. The resulting confusion matrix from a naive approach on one of the data sets can be observed in Table 4.1. A more intelligent approach to automatically labelling the data sets is clearly necessary.

	0	1
0	13,391	377
1	7,887	1,236

Table 4.1: Confusion matrix from prediction using naive labelling of the data. Predicted labels along the horizontal axis, true labels along the vertical axis. With $N_{RIL} = 100$, $Tw_L = 10$ minutes, and $Rw_L = 60$ minutes. RF model with 100 trees, and all trees fully grown.

The time-to-failure will also be estimated from the samples which have predictive power that can be used to indicate a failure, where $y_i \in \{0, 1, 2, \dots\}$ hours. As this is a regression problem, this will be done using a regression RF. Given the small amount of failures in each data set, this may be challenging to predict accurately.

The large number of features in each data set is one of the main reasons for the choice of RF as the ML algorithm for this problem. It is unknown if all, or only a subset of the features can be used to predict failures in the inverter unit, there is therefore potentially a large amount of noise features, which RF is robust against.

While the step from individual events to RWs has reduced the number of samples, these data sets are still very imbalanced. The data set for each of the vessels contain 30 or less failure samples out of close to 100 thousand samples in total.

As mentioned earlier, RF is one of the most common algorithms used in PdM. Therefore, some examples of previous use cases and methodologies for RF in PdM will be looked at.

In Allah Bukhsh et al. (2019), maintenance logs and condition data from railway switches were used to predict maintenance needs and failure types, in a supervised learning problem. The maintenance data was unbalanced, with 79% of the logs indicating that some maintenance has been performed, while only 21% were from maintenance not being performed. To mitigate the data imbalance, the majority class was under-sampled.

In a smaller problem focusing only on one piece of machinery, Binding, Dykeman and S. Pang (2019) used sensor data from an industrial printing machine to determine either if a failure is likely to occur in the near future (a classification problem), or to determine the RUL (a regression problem). The printer was supposed to report sensor values every minutes, but some values were missing due to failures in the data collection and/or transmission process. This problem was solved through the use of linear interpolation of the sensor data. The data had been collected historically to understand the machine's operational status, and not for the explicit purpose of PdM, this is similar to the data sets presented in this thesis. In this paper, a prediction horizon of 30 minutes was used. The labelling of the data sets came from an event logging system, this was used to label the data depending on whether or not a failure occurs within the next 30 minutes. The features engineered from the sensor data was then used to train a linear regression, RF, and extreme gradient boosted trees.

In an on-line failure prediction approach, Canizo et al. (2017) used RF to predict if a failure is imminent for several wind-turbines, at 10 minute intervals. This prediction was done using 448 different alarm types and 104 operational data parameters. The RF parameters used for the final model were obtained through a grid search using a few selected values for the number of trees in the forest, and for the maximum depth of the trees. In that case, the number of trees did not have a large effect on the prediction ability of the model, but the maximum depth of the trees did have a significant impact.

Using a combination of event-log data and sensor data, Naskos et al. (2020) attempted to predict failures in a cold forming press, and proposed both supervised and unsupervised approaches. The sensor data was used to create artificial events, using a method called Matrix Profile, and the prediction horizon was set between one and several hours. To counteract the problem of imbalanced data, the samples close to each failure were over-sampled. The supervised learning approach uses a method called *Multiple Instance Learning* (MIL), in an attempt to determine which sample leading up to a failure caused it. While the unsupervised approach used a clustering based outlier detection algorithm.

The implementations of RF that will be used in this thesis are the `RandomForestClassifier` (with some slight modification which will be discussed later in Section 5.2) and `RandomForestRegressor` classes in the `scikit-learn` (Pedregosa et al. 2011) library for Python. It is worth noting that the `RandomForestClassifier` implementation aggregates the prediction of the base learners using the classification confidence as described in Equation (4.14), and not via majority vote. The `RandomForestRegressor` simply takes the mean prediction of the base learners as described in Equation (4.11).

As explained in this section, there are still some major problems with the data sets that need to be addressed. There is an extreme imbalance between the failure samples and non-failure samples. It is currently unknown how many of the features obtained in Chapter 3 are noise features that do not contribute to describing failures. Because of the difficulty of augmenting expert knowledge into the data and models, the data are not labelled properly. While the samples that contain failures are known, which samples caused these failures remain unknown. The problems of labelling and of data imbalance will be further discussed in the next chapter.

CHAPTER 5

Multiple Instance Learning through Balanced RF

Overview and motivation

While RF is a method well suited for the objectives of this thesis, it is unable to deal with the two crucial challenges presented at the end of the previous chapter: imbalanced and unlabeled data. Firstly, a solution must be found to the problem of imbalanced data. A more detailed discussion of the problem, as well as solutions related to RF will be discussed in Sections 5.1 and 5.2.

Secondly, labelled data is needed for doing predictions with RF, as it is a supervised method. Section 5.3 contains an in-depth investigation of this challenge, and introduces the concept of weakly supervised learning. A proposed solution to the data-labelling issue will be presented in-depth in Section 5.4.

Finally, in Section 5.5 it will be shown how the proposed algorithms can be combined into a framework that can be used for predictions in Chapter 6. Also the general usefulness of this new combined method for PdM will be discussed.

5.1 Imbalanced Data Classification and Cost Sensitive Approaches

Imbalanced data is a major issue for many classification algorithms, as many of them assume that the data are roughly balanced (Sun, Wong and Kamel 2011). RF is no exception to this (Chen, Liaw and Breiman 2004), as the method attempts to minimize the cost, which is generally defined equally for all classes. This can present some issues, because in many cases, such as in this thesis, the minority class is of most interest, which is often the class corresponding to an abnormal state. As explained back in Section 2.2, this is particularly common in PdM because of the widespread use of PvM strategies. If only the misclassification rate is used to determine the quality of a model for the data sets in this thesis, near 0% misclassification rate could be achieved by simply predicting that every sample belongs to class $y_i = 0$.

The imbalanced data problem is commonly divided into two different categories; *between-class* imbalance, and *within-class* imbalance (He and Ma 2013, p. 16). Between-class imbalance, as the name suggests is the imbalance between the number of samples of different classes, which is a very prevalent problem in the data sets presented in this thesis. The other category, within-class imbalance refers to the fact that samples of the same class may not be similar. While this has not been looked at for the data set presented in this thesis yet, it could be that the failures in the inverter unit are caused by different sequences of events, and the samples may therefore not be entirely similar.

While the obvious solution to imbalanced data sets is to acquire more data, this is usually not feasible. For PdM, acquiring more data would require the system to experience a significant amount of failures, resulting in prohibitively high economic costs. Therefore, working with very imbalanced data sets requires methods that are designed specifically to handle this problem.

One common way of making methods more robust against imbalanced data sets, is to use *weighted* or *cost-sensitive* (Kaur, Pannu and Malhi 2019) methods. These techniques attempt to compensate for the data imbalance by assigning a higher cost to misclassifying samples belonging to the minority class. This is commonly accomplished by assigning a weight to each class, with a higher weight being assigned to the minority class.

In Chen, Liaw and Breiman (2004), two solutions to the imbalanced data problem are presented for RF. One of these methods is a weighted approach to RF, called *Weighted Random Forest* (WRF). Standard RF tends to be biased towards the majority class, which is why WRF assigns a higher cost to misclassifying the minority class, by slightly modifying Equations (4.3) to (4.5). This is accomplished by weighting the classes differently, by assigning the minority class a higher weight. The weights are used at two stages in the WRF algorithm, they are used when calculating the cost of a given split, and in the leaf nodes when performing predictions. An alternative WRF method is to weight the predictions of each individual tree $\hat{f}_b(X)$ in the forest, based on weights determined by performance on a separate test set (Winham, Freimuth and Biernacka 2013).

While WRF does have its benefits, the method also has its downsides. A notable one of these is determining the weights of each class. To prevent overfitting, the weights should ideally be determined using separate test sets, unfortunately this results in the training set becoming even smaller, which can be an issue if there are already very few samples belonging to the minority class. Another issue is that because the minority class is weighted more heavily, more samples of the majority class will likely be misclassified as the minority class.

Still, WRF has been shown to perform well in many imbalanced data scenarios. WRF was used to analyze sinkholes in J. Zhu and Pierskalla (2016). Using a standard RF method a 91.55% accuracy rate was achieved, but only 70.72% of true sinkholes were predicted to be sinkholes (true positive rate). The best results were obtained with the minority class being assigned a weight 4 times higher than the majority class. With WRF using these weights, a lower accuracy of 89.07% was obtained, but the true positive rate increased substantially to 91.09%.

In Winham, Freimuth and Biernacka (2013), the latter of the two mentioned WRF methods was used to model genes using real-world data. Here the predictions of the individual trees were weighted to increase the contribution of the trees that predict the minority class better. WRF performed at least as well as standard RF in all measures, with a slight improvement in prediction performance being observed in certain scenarios.

5.2 Balanced RF

One of the major issues with using RF for learning from imbalanced data, especially when the degree of imbalance becomes large, is that each tree in the forest only trains on a bootstrap sample of the original data. If the number of samples in the minority class is low enough, each tree's bootstrap sample may only contain a few, or even no samples from the minority class. This will result in some trees being unable to predict the minority class well or at all.

One way of solving this is to either oversample the minority class, or to undersample the majority class. The other RF method presented in Chen, Liaw and Breiman (2004), uses undersampling of the majority class and is called *Balanced Random Forest* (BRF). This algorithm can be broken down into three steps for binary classification problems:

1. Draw a random bootstrap sample from the minority class, then randomly draw the same number of samples from the majority class with replacement.
2. One CART in the RF is trained on this sample from the original data set, as described in Section 4.2.
3. Steps 1. and 2. are then repeated until the desired number of trees in the RF are trained.

Both WRF and BRF were shown experimentally to provide superior results to standard RF for imbalanced data sets in Chen, Liaw and Breiman (2004). Unlike WRF however, BRF does not normally require any more parameters to estimate than a normal RF. This makes the model slightly easier to use, and removes the need for a separate test set just to estimate this new parameter, like WRF requires with its weights. Because of the undersampling, BRF is also less computationally complex.

One downside with BRF however, is that depending on the size of the minority class, large parts of the majority class may not be used for training. This can result in loss of information, because a part of the data set is not used at all, resulting in worse prediction of the majority class.

BRF has been used to success in multiple imbalanced data problems. In Kobyliński and Przepiórkowski (2008), BRF was used in a natural language processing classification problem. The paper attempted to classify whether a given Polish sentence was definitional or not. The ratio between positive and negative samples was roughly 20:1, and it was noted that just classifying every sentence as non-definitional would achieve approximately 95% accuracy. The results of the classification show that switching from RF to BRF improved the results significantly. This paper compared its results with a previous paper analyzing the same data set, and an increase in the precision of the minority class from 17% to 21.4% when using BRF instead of RF was observed. A different feature selection strategy was also used compared to the previous paper however.

BRF was used for analysis of gene data in Achawanantakun et al. (2015). The paper compares a method using BRF to a logistic regression based method. When trained on the imbalanced data set, BRF had a slightly lower specificity (3.46% lower) than the regression, but a much higher sensitivity was observed (26.28% higher).

In this thesis BRF will be used for failures prediction. This is mainly because WRF would require the estimation of weights. This makes BRF easier to use, while both methods produce similar results according to Chen, Liaw and Breiman (2004).

The BRF implementation that will be used is the `BalancedRandomForestClassifier` class from the python library `imbalanced-learn` (Lemaître, Nogueira and Aridas 2017), which is an extension to the `RandomForestClassifier` class implemented in the `scikit-learn` (Pedregosa et al. 2011) library.

5.3 Weakly Labelled Data

The methods for training a model usually falls within one of three different common categories, depending on how the data set is labelled. *Supervised* learning is the term used when the label of every sample is known. *Unsupervised* learning is used when none of the samples' labels are known. And *semi-supervised* learning is when the labels of some samples are known, while others are unknown. The data sets presented in this thesis however, do not fit into any of these categories.

Unlabelled data is not an uncommon problem in ML, especially in practical applications when working with real-world data. This problem is often circumvented through the use of unsupervised learning methods, which do not require labelled data to train models, e.g. clustering algorithms. In this thesis however, the goal is to predict failures, which is not feasible using unlabelled clustering, as this only helps in determining which samples are similar. The information present in the data sets must therefore be used somehow. The time of each failure is known, which is valuable information for determining which samples caused a failure. This type of data is referred to as *weakly labelled data*, and the problems relating to it is solved through the use of methods designed for *weakly supervised* learning.

Weakly labelled data refers to data where partial or inaccurate information about the label of each sample is known, but not the exact labels. Note that this is different from the case of semi-supervised learning where some samples are labelled completely, and other not at all. Weakly labelled data is generally split into two categories: *inaccurate* weakly labelled data, and *inexact* weakly labelled data (Zhou 2017).

In an inaccurate supervision problem, each sample has a *true class label probability* for each class. These probabilities are defined as:

$$p_{i,c} = p(y_i = c|X_i). \tag{5.1}$$

An example of true class label probabilities for two samples in a three-class classification problem could be; $(p_{1,1}, p_{1,2}, p_{1,3}) = (0.63, 0.21, 0.16)$ and $(p_{2,1}, p_{2,2}, p_{2,3}) = (0.35, 0.01, 0.64)$. This may be the case in certain scenarios when dealing with noisy data, where there is some uncertainty in what class a sample belongs to.

Inexact supervision problems are slightly different. In these problems, samples belong to groups of arbitrary size, referred to as *bags*, and the bag as a whole has a label, based on whether or not at least 1 sample in the bag belongs to a certain class. An example of two such bags could be; $B^1 = \{X_1, X_2, \ldots, X_i\}$

and $B^2 = \{X_{i+1}, X_{i+2}, \dots, X_k\}$, where each bag B^j is also assigned a label. In a binary classification setting with possible sample labels $y = 0$ and $y = 1$, each bag is assigned one of two possible labels, negative or positive, expressed as B^- and B^+. Three conditions are imposed on each individual sample in the bags,

$$\begin{cases} \sum\limits_{i \text{ s.t. } X_i \in B^-} y_i = 0 & (5.2a) \\ \sum\limits_{i \text{ s.t. } X_i \in B^j} y_i > 0, \forall j \text{ s.t. } B^j \text{ is } B^+ & (5.2b) \\ \sum\limits_{c \in \mathcal{C}} p_{i,c} = 1, \forall i, & (5.2c) \end{cases}$$

where $\mathcal{C}$ is the set of all classes. Negative bags B^- do not contain any samples with a label $y = 1$, and all the samples in these bags therefore have a label $y = 0$. Each positive bag B^+ contains *at least one* sample with a label $y = 1$, but how many, or which ones are unknown. The final condition will become useful in Section 5.4.

An overview of the two different forms of weakly supervised learning, and more common learning methods are shown in Figure 5.1. The question marks indicate that the true label is unknown, while a label followed by a question mark indicates that some information is known about the label, but that the true label is unknown. The boxes grouping the samples in the inexact supervision indicate the bags. Combinations of these learning types are also possible.

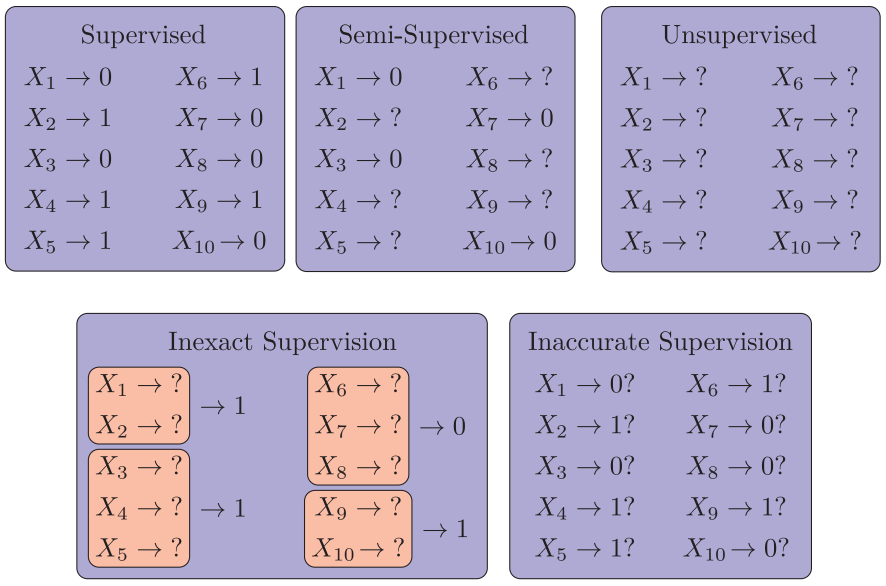

Supervised
$X_1 \to 0$ $X_6 \to 1$
$X_2 \to 1$ $X_7 \to 0$
$X_3 \to 0$ $X_8 \to 0$
$X_4 \to 1$ $X_9 \to 1$
$X_5 \to 1$ $X_{10} \to 0$
Semi-Supervised
$X_1 \to 0$ $X_6 \to ?$
$X_2 \to ?$ $X_7 \to 0$
$X_3 \to 0$ $X_8 \to ?$
$X_4 \to ?$ $X_9 \to ?$
$X_5 \to ?$ $X_{10} \to 0$
Unsupervised
$X_1 \to ?$ $X_6 \to ?$
$X_2 \to ?$ $X_7 \to ?$
$X_3 \to ?$ $X_8 \to ?$
$X_4 \to ?$ $X_9 \to ?$
$X_5 \to ?$ $X_{10} \to ?$
Inexact Supervision
$X_1 \to ?$ $X_2 \to ?$ $\to 1$
$X_3 \to ?$ $X_4 \to ?$ $X_5 \to ?$ $\to 1$
$X_6 \to ?$ $X_7 \to ?$ $X_8 \to ?$ $\to 0$
$X_9 \to ?$ $X_{10} \to ?$ $\to 1$
Inaccurate Supervision
$X_1 \to 0?$ $X_6 \to 1?$
$X_2 \to 1?$ $X_7 \to 0?$
$X_3 \to 0?$ $X_8 \to 0?$
$X_4 \to 1?$ $X_9 \to 1?$
$X_5 \to 1?$ $X_{10} \to 0?$

Figure 5.1: An overview of learning with different forms of supervision. Inspired by Figure 1. in Zhou (2017).

Getting properly labelled data for PdM can be very expensive, and sometimes nearly impossible. To determine the cause of a failure, expert knowledge is usually required. These experts must analyze the data collected from around the time of the failure to determine the cause exactly, which can be both expensive and require a lot of time. If the system is large and complex enough, this may even be an unfeasible task.

Modelling the data sets presented in this thesis can be expressed as an inexact supervision problem. As explained in Section 3.5, each sample represents a time window containing multiple events, and while the samples containing the events that cause the failures are unknown, the samples containing the failures are. If it is assumed that the sample or samples containing the events that caused a specific failure occurred within a certain time period before the failure, then all the samples in this time period can be put into a positive bag B^+ indicating that at least one sample in the bag caused the failure. This can be done for each failure in the data set, and all the samples that do not belong to any of these bags can be put into a negative bag B^-, indicating that no samples in the bag caused a failure, as it is known that no failures resulted from these samples.

5.4 Multiple Instance Learning

One way of uncovering labels for samples in a weakly supervised learning scenario, is through the use of an iterative learning approach known as *Multiple Instance Learning* (MIL). This approach uses the described bags for inexact supervision to assign labels to each individual sample in the bags. MIL is an iterative approach that attempts to solve an optimization problem that consists of both labelling the data and training a model with good prediction power. During the iterations, models are trained and used to update the probability of every sample belonging to each class. This process is repeated until the final labels and a trained model are obtained. This process can also be extended to non-binary classification, where the bag label indicates that at least one sample has the same label as the bag, while the other samples in the bag can have any label.

From here on, a superscript ^{j} will indicate the j^{th} bag, and a subscript $_i$ will indicate the i^{th} sample in a bag. Let the j^{th} bag be denoted as $B^j, j = 1, \ldots, n$, with corresponding label $y^j \in \mathcal{Y} = \{0, 1\}$. Bags containing only samples with a label of $\hat{y}_i^j = 0$ are assigned a bag label of $y^j = 0$ and are referred to as negative bags B^-, while bags containing at least one sample with a label of $\hat{y}_i^j = 1$ are assigned a bag label of $y^j = 1$, and are referred to as positive bags B^+. Each bag contains n^j samples $\{X_1^j, X_2^j, \ldots, X_i^j, \ldots, X_{n^j}^j\}$ with corresponding, but unknown labels $\{\hat{y}_1^j, \hat{y}_2^j, \ldots, \hat{y}_i^j, \ldots, \hat{y}_{n^j}^j\}$. MIL aims to optimize both the loss function of the model, and the sample labels. The optimization problem can be expressed as (Leistner, Saffari and Bischof 2010):

$$\begin{aligned}
(\{\hat{y}\}, \mathcal{G}) = \arg \min_{\{\hat{y}\}, \mathcal{G}(\cdot)} \mathcal{L}(F_c(X)) \\
\text{s.t.} \forall y^j \neq 0, \sum_{i=0}^{n^j} \mathbb{I}(y^j = \arg\max_{c \in \mathcal{Y}} F_c(X_i^j)) \geq 1, \\
\forall y^j = 0, \sum_{i=0}^{n^j} \mathbb{I}(y^j \neq \arg\max_{c \in \mathcal{Y}} F_c(X_i^j)) = 0
\end{aligned} \tag{5.3}$$

where

$$\mathcal{L}(F_c(X)) = \sum_{j=1}^{n}\sum_{i=0}^{n^j} \ell(F_c(X_i^j)), \tag{5.4}$$

$\mathcal{G}$ is the model, and $F_c(X_i^j)$ is the classification confidence of the model for sample X_i^j belonging to class c, this is defined in Equation (4.13) for RF. ℓ is a loss function, one such function that can be used is the negative *classification margin*. The margin of a sample is defined as (Xia, Q. Zhu and Wei 2015):

$$\mathcal{M}_{\mathcal{G}}(X_i^j, \hat{y}_i^j) = F_{\hat{y}_i^j}(X_i^j) - \max_{\substack{c \in \mathcal{Y} \\ c \neq \hat{y}_i^j}} F_c(X_i^j), \tag{5.5}$$

where $\mathcal{M}_{\mathcal{G}}$ is the margin of model $\mathcal{G}$ for one sample X_i^j with label $\hat{y}_i^j$. The margin is 1 if the model has 100% confidence in the right label, and -1 if it has 100% confidence in the wrong label. The margin is positive as long as the model is predicting the correct label. The conditions in Equation (5.3) simply ensure that the conditions of Equations (5.2a) and (5.2b) hold. This optimization problem attempts to solve for the optimal parameters of the model while also uncovering the true labels of the data, which is a non-convex optimization problem. Because of this, a technique called *Deterministic Annealing* (DA) is generally employed to solve the problem. The label of each sample is either 0 or 1, this is therefore also an integer optimization problem, a solution to this is to instead look at the true class label probabilities of each sample, Equation (5.1).

The settings discussed so far have been for general multi-class problems, this thesis focuses primarily on binary classification, therefore this will be the focus from here on. To calculate the true class label probability of each sample using DA, an entropy term is added to transform the problem into a convex optimization problem. To find the true class label probabilities one must therefore calculate (Leistner, Saffari and Bischof 2010),

$$p_{i,c}^{j*} = \arg\min_{p \in \mathcal{P}} E_p(\mathcal{L}) - T\mathcal{H}(p), \tag{5.6}$$

where $\mathcal{P}$ is a set of probability distribution, $\mathcal{F}(\hat{y})$ is the objective function described in Equation (5.3), T is the *temperature* which is used to regulate the effect of the entropy term $\mathcal{H}$. This temperature decreases each iteration t and is generally determined using a *cooling function*, $T = C(t)$. This ensures that the optimization problem approaches the original problem as the number of iterations t increased. The new loss function in Equation (5.6) can be re-formulated as (Leistner, Saffari and Bischof 2010):

$$\mathcal{L}_{DA} = \sum_{j=1}^{n}\sum_{i=1}^{n^j}\sum_{c=0}^{1} p_{i,c}^j \ell(c) - T\sum_{j=1}^{n}\sum_{i=1}^{n^j}\sum_{c=0}^{1} p_{i,c}^j log(p_{i,c}^j). \tag{5.7}$$

By inserting the negative margin $-\mathcal{M}_{\mathcal{G}}$ for the loss ℓ, and then taking the derivative of Equation (5.7) with respect to p and setting it equal to 0, the optimal choice of $p_{i,c}^j$ is obtained (Xia, Q. Zhu and Wei 2015):

$$p_{i,c}^{j*} = \exp\left(\mathcal{M}_{\mathcal{G}}(X_i^j, c) - T\right)/T, \tag{5.8}$$

Algorithm 4: Multiple Instance Learning, inspired by Algorithms 1 and 2 in Xia, Q. Zhu and Wei (2015).

```
**Data:** Bags $B^j$ containing samples $X_i^j$, and bag labels $y^j$
**Result:** Sample labels $\hat{y}_i^j$ and trained model $\mathcal{G}(X_i^j)$
$t = 0$;
$T = C(t)$ A cooling function must be chosen, for example $C(t) = e^{-t}$;
$c$ is some pre-determined minimum temperature, used as a stopping
  criteria;
Set all $\hat{y}_i^j$ to the label $y^j$ of the bag $B^j$ containing $X_i^j$;
Train model $\mathcal{G}(X_i^j)$ on samples and labels $(X_i^j, \hat{y}_i^j)$;
**while** $T > c$ **do**
    $t = t + 1$;
    $T = C(t)$;
    **for** $j = 1 : n$ **do**
        **for** $i = 1 : n^j$ **do**
            $p_{i,c}^j$ is calculated using Equation (5.9);
            $\hat{y}_i^j$ chosen randomly with probability $p_{i,c}^j, \forall c$;
        $\hat{y}_i^j$ is adjusted to ensure at least one sample $X_i^j$ in each positive
          bag $B^+$ has a label of $\hat{y}_i^j = 1$ using Equation (5.10)
    Train a new model using the samples and adjusted labels $(X_i^j, \hat{y}_i^j)$
```

This is then used to assign a probability of sample X_i^j belonging to class c, taking into account Equation (5.2b):

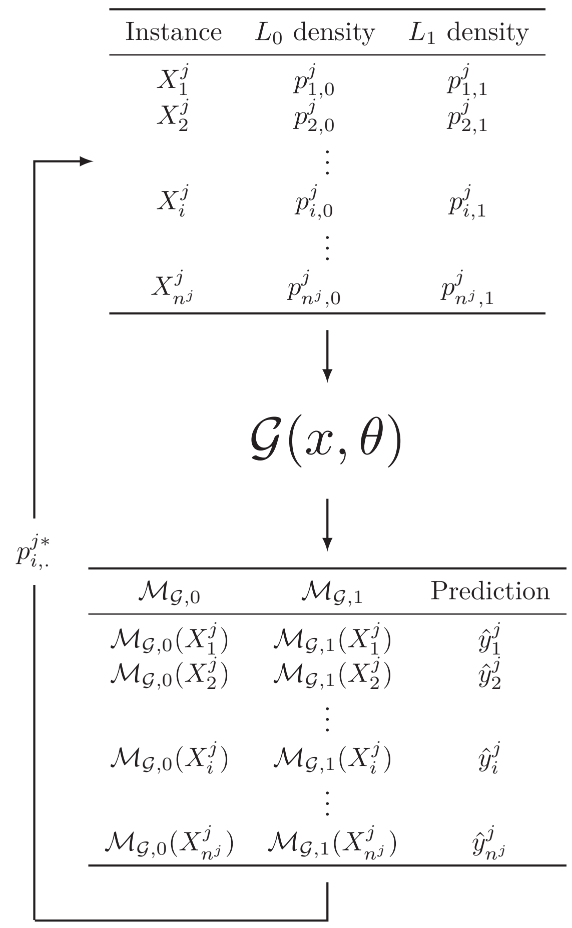
$$p_{i,c}^j = \begin{cases} p_{i,c}^{j*}/\Psi_i^j & \text{if } y^j = 1 \\ 1 & \text{if } y^j = 0, c = 0 \\ 0 & \text{if } y^j = 0, c = 1 \end{cases} \tag{5.9}$$

where $\Psi_i^j = \sum_{c=0}^{1} p_{i,c}^j *$ is simply the normalization factor, which is required to satisfy the third condition in Equation (5.2c). Then, a new label for each sample is assigned randomly, with a probability $p_{i,0}^j$ for label $\hat{y}_i^j = 0$ and $p_{i,1}^j$ for label $\hat{y}_i^j = 1$. The sample with the highest true class label probability p_{i,y^j}^j of having a label equal to the bag label y^j, must be labelled as that class if no other samples in the bag are:

$$\forall j : \sum_{i=1}^{n^j} \mathbb{I}(\hat{y}_i^j = y^j) = 0 \Rightarrow \hat{y}_{i^*}^j = y^j, i^* = \arg\max_i p_{i,y^j}^j . \tag{5.10}$$

This is to satisfy the conditions in Equations (5.2a) and (5.2b).

Each sample X_i^j has been assigned a new label $\hat{y}_i^j$, and a new model is trained using the updated labels. This process of updating the sample labels and a new model being trained is then repeated for the desired number of iterations, or until the temperature $T = C(t)$ reaches a pre-set minimum temperature. The full algorithm as we have implemented it is described in Algorithm 4. The full MIL process for a single bag is also visualized in Figure 5.2. The algorithm forms a loop, which is the iterations the algorithm goes through.



Instance	L_0 density	L_1 density
X_1^j	$p_{1,0}^j$	$p_{1,1}^j$
X_2^j	$p_{2,0}^j$	$p_{2,1}^j$
	$\vdots$	
X_i^j	$p_{i,0}^j$	$p_{i,1}^j$
	$\vdots$	
$X_{n^j}^j$	$p_{n^j,0}^j$	$p_{n^j,1}^j$

$\mathcal{M}_{\mathcal{G},0}$	$\mathcal{M}_{\mathcal{G},1}$	Prediction
$\mathcal{M}_{\mathcal{G},0}(X_1^j)$	$\mathcal{M}_{\mathcal{G},1}(X_1^j)$	$\hat{y}_1^j$
$\mathcal{M}_{\mathcal{G},0}(X_2^j)$	$\mathcal{M}_{\mathcal{G},1}(X_2^j)$	$\hat{y}_2^j$
	$\vdots$	
$\mathcal{M}_{\mathcal{G},0}(X_i^j)$	$\mathcal{M}_{\mathcal{G},1}(X_i^j)$	$\hat{y}_i^j$
	$\vdots$	
$\mathcal{M}_{\mathcal{G},0}(X_{n^j}^j)$	$\mathcal{M}_{\mathcal{G},1}(X_{n^j}^j)$	$\hat{y}_{n^j}^j$

Figure 5.2: The iterative MIL process of a single bag visualized. A model is trained on the samples X_i^j from all bags, the margin $\mathcal{M}_{\mathcal{G}}$ of this model $\mathcal{G}$ is used to update the true class label probability $p_{i,.}^j$ of each sample. These updated probabilities are then used to update the sample labels $\hat{y}_i^j$.

MIL has been used or mentioned occasionally for use in PdM problems, generally in relation to event-log based approaches. Naskos et al. (2020) performed PdM using event-log data, using a more simple version of MIL. In this paper a saturation function was used to label a certain number of samples, closest in time, to the failure as causing the failure without any optimization. Here, the samples were assigned weights, not probabilities, this may be useful in certain cases where it is known that the cause of a failure is always observed right before the failure, but this is not always the case.

MIL was used in a similar PdM problem to the one presented in this thesis in Sipos et al. (2014), which looked at event logs from some mining equipment. Here the samples were placed into bags, with each bag having a binary label. The same assumptions as in this thesis are made, where it is assumed that within a certain time window before a failure, at least one sample caused the failure. Unlike the work in this thesis however, this paper only aimed to label the bags themselves, not the individual samples in each bag.

With some slight modifications, MIL can also be used for inaccurate supervision problems. Instead of bagging the samples, they are simply labelled using their already known true class label probabilities.

5.5 Novel Approach: Multiple Instance Learning through Balanced Random Forest

Here, we will propose a new method that combines MIL and BRF, this new method will be referred to as *Multiple Instance Learning through Balanced Random Forest* (MIL-B-RF). The name of this method simply refers to the use of BRF as the model that is trained during the iterations and returned at the end of the MIL. This new method attempts to solve both the problems of imbalanced data, and weakly labelled data, which standard RF struggles with. It is worth noting that with this method the imbalanceness of the data changes depending on how the MIL labels the samples. To our knowledge, a combination like MIL-B-RF has not been suggested before.

As discussed back in Section 2.2, both weakly labelled (referred to as unlabelled at the time) and imbalanced data are common problems in the PdM field. The weakly labelled data problem can potentially be solved by having access to expert knowledge, but this may become prohibitively expensive, or unfeasible if the system is too complex. Data imbalance will always be present in PdM problems, because the up time of any system will always be higher than the down time, resulting in very few failures in comparison to standard operational data. MIL-B-RF has potential to be useful in other PdM problems as well.

A simplified view of the what the nodes in a single trained tree in the MIL-B-RF algorithm may look like, is shown in Figure 5.3. In reality however, the model would consist of an entire forest of trees. The data sets presented in this thesis have thousands of features, not just two. They have much more extreme imbalanceness than presented in the figure, with class imbalance rate in the range of 1000s. The bags are also much larger, containing 100s of samples.

The solution to the joint challenge of imbalance and weakly labelled data is obtained by combining solutions to the two separate challenges. The novel method MIL-B-RF combines the solution to the imbalance, BRF, with the solution to the weakly labelled data problem, MIL. This method will be used for the predictions in Chapter 6, on the data sets presented back in Section 3.5.

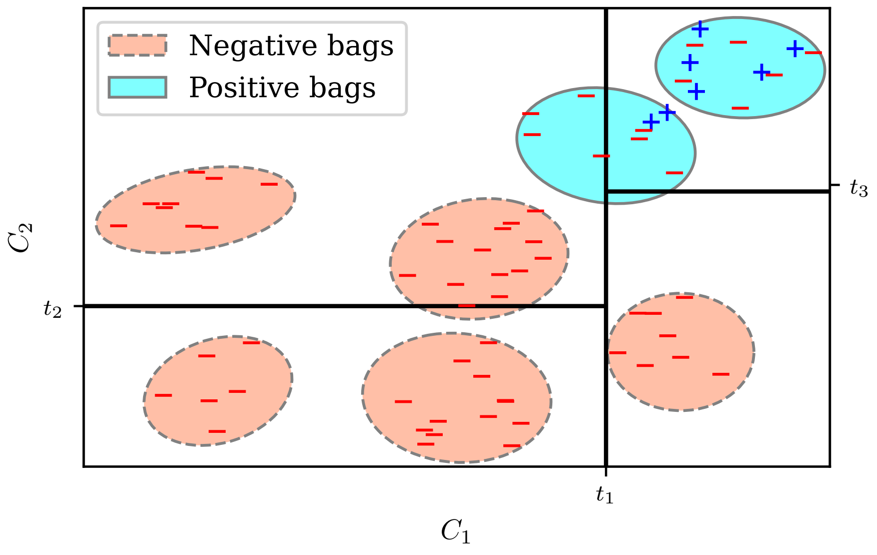


Figure 5.3: A demonstration of the results of MIL-B-RF for imbalanced, weakly (inexactly) labelled simulated data with only two features and a single tree, for simplicity. The ellipses are separate bags, each containing its own set of samples. Red, dashed bags are negative, and blue, solid bags are positive. While the true labels of the sample are unknown, they are shown here for illustration purposes.

CHAPTER 6

Results, discussion and analysis

Overview and motivation

In this chapter, the methods introduced in Chapters 3 to 5, will be used to analyze the data sets introduced back in Chapter 3. In Section 6.1 the pre-processing of the data sets will be explained, along with the parameter values that were used for the window aggregation (Section 3.3) and RI (see Section 3.4). After that, in Section 6.2, the hyperparameters that will used for the model training using MIL-B-RF (see Section 5.5) will be explained along with why the different values were chosen. With the data sets pre-processed and the hyperparameters chosen, Section 6.3 will present and discuss the results obtained using the MIL-B-RF method. Finally, in Section 6.4 the obtained results will be compared with both the results obtained from using the same methods on subsets of the features, and with the results from the most related papers that studied similar problems.

6.1 Training and testing data

Four data sets were collected and prepared, for training and testing the developed PdM methods, each from a different vessel, these vessels are referred to as 'vessel 1', 'vessel 2', 'vessel 3' and 'vessel 4', in this thesis. The raw data sets consist of individual events, with the majority being informational or warnings, as seen in Table 6.1. In this table, the number of events in each vessel's data sets with an event message containing the word 'Alarm' is counted. It is worth noting that there may be alarm events that do not contain the word 'Alarm' in their event message. In this table it can be observed that less than 1% of the events from any given vessel are alarms, with vessel 1 experiencing the highest alarm to event ratio, and vessel 4 the lowest. More information about the raw data sets is listed in Table 3.2, in Section 3.2.

These raw events were then used to create the samples that were used for modelling. Each event type was represented as the context they commonly appear in using *Random Indexing* (RI), as described in Section 3.4. This algorithm requires some parameter values which have to be chosen. The RI was calculated using index- and context vectors of length $N_{RIL} = 100$, and an index vector sparsity of $S_{RI} = 10\%$. These values were chosen based on the original proposal of the algorithm (Kanerva, Kristoferson and Holst 2000), where the $N_{RIL} = 1,800$ was selected for a dictionary of size 60,000. Through

Vessel	Event counts	Alarm Events counts
1	339,521	3,372
2	443,624	2,994
3	1,187,118	5,015
4	580,914	1,289

Table 6.1: Proportion of alarms in the log event data.

some preliminary testing by increasing N_{RIL} from 100 to 500 and then 1,000, no noticeable difference in prediction ability was observed, each individual RI feature simply had a lower importance when the number of features was increased. The context vectors were then created using a window of size $CTw_L = 10$ sec, to determine the event type context of each event type. This was chosen as a reasonable value to represent a 'now' on the vessels. As the original algorithm was developed for representing words used in sentences, it determines the context based on distance, and not time. There is therefore not any previous works to base this number on.

Then the event counts and RI representations were aggregated over *Time Windows* (TWs) and *Rolling Windows* (RWs) to get the samples that will be used for model training. For the window aggregation, the length of each TW was set to $Tw_L = 10$ min. The TWs were then used to construct the RWs, which had a length of $Rw_L = 6 \times Tw_L = 1$ hour. These values were chosen, as they seemed like reasonable amounts of time to represent what is currently occurring on a vessel. The used aggregation functions were the minimum, the mean, and the maximum. The motivation behind these specific functions, and the window aggregation in general was explained in Section 3.3. The combination of the window aggregation and RI was explained in Section 3.5.

All these initial parameter values are summarized in Table 6.2.

Name	Parameter	Value
Index vector length	N_{RIL}	100
Index vector sparsity	S_{RI}	10%
Context vector window length	CTw_L	10 sec
Time window length	Tw_L	10 min
Rolling window length	Rw_L	1 hour
Aggregation functions	-	min, mean, max

Table 6.2: Initial reasonable parameter values, for pre-processing the data sets

At the end of Section 2.2, a couple of major challenges for PdM were presented. The two most prevalent for the data sets from the vessels, were the difficulty and cost of incorporating expert knowledge into the data sets and the models, and the data imbalance.

The raw data are extremely imbalanced, as shown in Table 3.2, back in Section 3.2. This was somewhat mitigated by the window aggregation, which resulted in a large reduction in number of samples, and only a small reduction in the number of failure samples for most vessels, as seen in Table 6.3. While this was not the main reason for the use of the window aggregation, it is an additional benefit of using the method. All the vessels have raw data sets that span roughly the same amount of time, which is the reason they have roughly the same amount of samples, but the vessels logged varying amounts of events in this time frame. The number of samples containing failures is lower than the total number of failure events, because multiple failures can occur within the same TW. This should not be an issue, because if multiple failures occur in quick succession, it is reasonable to assume that the failures had the same cause, perhaps because the system was restarted before the problem was solved properly, and the failure quickly occurred again.

Vessel	Events	Failure events	Samples	Failure samples
1	339,521	36	91,562	24
2	443,624	40	99,142	30
3	1,187,118	48	100,141	17
4	580,914	14	96,946	12

Table 6.3: The number of events and failure events in each vessel's raw data sets, compared to the total number of samples, and number of positive samples after the window aggregation.

The work of analysing and determining the cause of every failure, and how they differ from standard operation is difficult and a costly, time consuming process, and as such is not feasible. The challenge to augment the expert knowledge, into the machine learning models and data, will therefore be solved with MIL. The imbalance should also be reduced, as multiple samples in each bag are likely to be labelled as causing a failure. This MIL, combined with the BRF into MIL-B-RF, drastically reduces the effects of the imbalance present in the raw data sets.

One important note about the data sets, is that they were not collected with the specific intent of doing PdM. These events were logged with event messages intended to be read and interpreted by humans, such that operators can determine what is happening on the vessels. This means that the data may contain many correlated, and redundant events, which is one of the reasons why RI is used.

6.2 Hyperparameter analysis

In addition to all the parameters mentioned in the previous section, that were used in the pre-processing of the data sets, there are also multiple hyperparameters that were used for the training of the model itself. For the BRF model; the number of trees, the stopping criteria, and the number of features each tree used, all had to be determined. The MIL algorithm also required a cooling function $C(t)$, a stopping temperature or maximum number of iterations, and a bag size. The initial values for these hyperparameters are presented in Table 6.4.

The maximum tree depth and the number of features per tree is set to the suggested starting values for classification as described in Hastie, Tibshirani and Friedman (2009, p. 592). Here it is suggested that $\sqrt{N_F}$ is used, where $N_F = 3N_e + N_{RIL}$ is the number of features in the data set, and that the trees

Hyperparameter name	Value
Number of trees	100
Maximum depth	Grow until every node is pure
Number of features per tree	$\sqrt{3N_e + N_{RIL}}$
Cooling function	$e^{-1/5t}$
Stopping temperature	0.1
Bag length	3 days

Table 6.4: Initial hyperparameter values for training the model.

are grown until all nodes contain only a single sample. In Hastie, Tibshirani and Friedman (2009, p. 591) it can also be observed that the performance of the RF stopped improving at around 100-200 trees. Leistner, Saffari and Bischof (2010) proposed the use of a cooling function of the form $e^{-C \cdot t}$, where C is a constant that determines how fast the MIL algorithm converges toward the final estimated labels. The exponentially decaying function slowly reduces the contribution of the added entropy, term as shown in Equations (5.6) and (5.7). The stopping temperature was chosen fairly arbitrarily, because it was easily modifiable by observing when the convergence was reached.

In Figure 6.1 the total number of samples in the training set estimated to have a label of $\hat{y} = 1$ is shown as a function of the number of iterations. It can be observed that the results converge quickly, and that the stopping temperature can probably be increased, as only about half the shown iterations are required. This corresponds to a stopping temperature of 0.4 for 6 iterations, or 0.35 for 7 iterations. Even after convergence is reached the number of samples labelled $\hat{y} = 1$ fluctuates slightly, this is because of how the labels are assigned randomly using the probability of the sample belonging to the two classes according to Equation (5.9). In Leistner, Saffari and Bischof (2010) a cooling function of $e^{-\frac{1}{2}t}$ was used, which converges in even fewer iterations, usually in about 3 iterations from some preliminary testing.

To use MIL as described in Sections 5.4 and 5.5, a bag size had to be chosen as well. This bag size was based on how long before a failure its cause could reasonably have occurred, and was set to $B_L = 3$ days. This value was chosen as it should be a reasonable amount of time to detect and perform the maintenance required to avoid potential failures. Bags of this size results in 432 samples in each positive MIL bag. If one positive bag would overlap with another, the second bag is shortened to ensure that each sample only appears in one bag. The samples outside the positive bags were simply treated as belonging to one large negative bag, as them being in one or multiple negative bags does not matter for the algorithm.

Another important measure that had to be chosen is how to determine if a failure is likely to occur based on the predictions of the model. Whether a single sample predicting that a failure will occur should be enough to determine that a failure is likely, or if multiple positive predictions within a certain time can be used. The first of these two options was chosen. The second approach can be easily implemented given sufficient data including sufficient failures to tune the extra parameters.

The size of the data sets make them slightly inconvenient to work with.

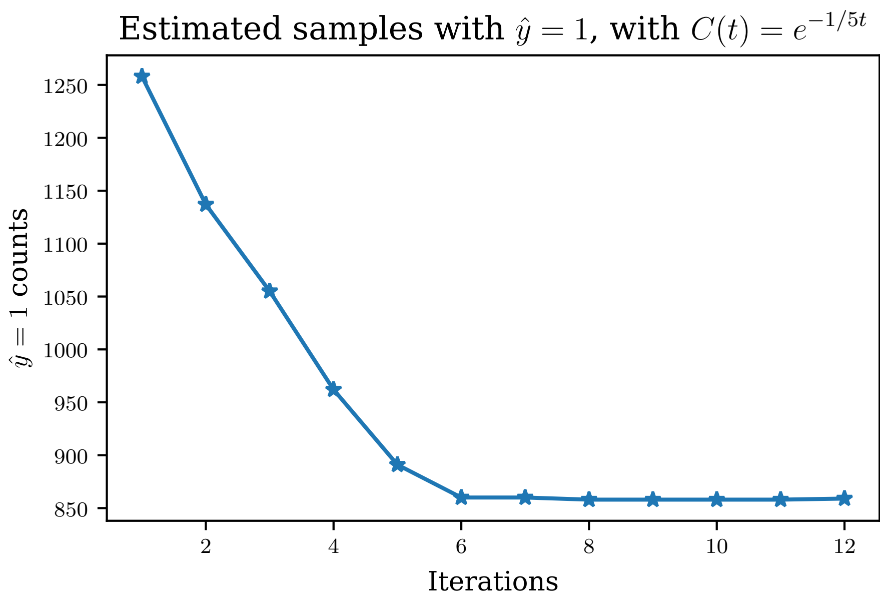


Figure 6.1: The number of samples in the training set estimated to have a label $\hat{y} = 1$ during the MIL iterations while training the models of vessel 1.

While the sparsity of the data allows it be stored in a more efficient format (using the 'Sparse' data-types in Pandas (Reback et al. 2020) (McKinney 2010)) for long term storage, thus taking up only 70-75MB per data set, the data have to be uncompressed to be used for training and testing. This results in roughly 5GB of memory being used when training a model per data set.

6.3 PdM results

Even though MIL was only used to estimate the labels of the training set, the testing set was also bagged in a similar fashion, with the 3 day window before a failure being referred to as a bag. These bags are not used in any way while predicting, but they are instead used to determine if any failures are predicted in the 3 day windows before a failure.

This project covers a total of four vessels, to avoid repetitions, and while all the methods presented were used for all four of them, some results will only be shown for one vessel, since all results and figures for all failures in all bags cannot be properly shown in a limited space. The resulting confusion matrices will be shown for all four vessels.

For simplicity and easier visualization, the data sets were split into training and testing sets based on timestamps instead of randomly, manually ensuring that there are roughly the same amount of failures in both sets. This also made it easier to analyse the time frames before and after the bags to look for patterns. This would be difficult to do if the bag-level data were split randomly, or via other methods such as cross-validation. Information about the resulting training and testing sets for each vessel is found in Table 6.5. While the training and testing sets for some of the vessels may look very imbalanced, particularly for vessel 3, this is because of the time duration between the failures. For example, vessel 3's test set has 13 failures in total, but 2 failures occur on one day, and 11 failures occur on another day, so in reality the set only contains 2 clusters of failures.

Vessel	Train span	Testing span	Train set failures	Test set failures
1	420 days	217 days	6	18
2	591 days	99 days	14	16
3	554 days	143 days	4	13
4	436 days	238 days	5	7

Table 6.5: The time spans of the training and testing sets for each vessel, and the number of failures in each of these sets.

The estimated labels and the bags of the training set after the model training using the MIL-B-RF algorithm, are observed in Figure 6.2 for the whole training set, and a zoomed in version in Figure 6.3. Here the rectangles represent the bags with a length of 3 days, and the points indicate the labels of the individual samples. During the training process of vessel 1 the MIL-B-RF algorithm was reliably able to label at least one sample X_i^j in each bag B^j as $\hat{y}_i^j = 1$ without resorting to the use of Equation (5.10), except for in one bag. During the training process, bag B^4 was occasionally reported to not contain any positively labelled samples X_i^4, the sample with the highest true class label probability $p_{i,1}^4$ of having a true label of 1 in the bag, was therefore corrected to have that label in order to satisfy Equation (5.2b) as performed by Equation (5.10).

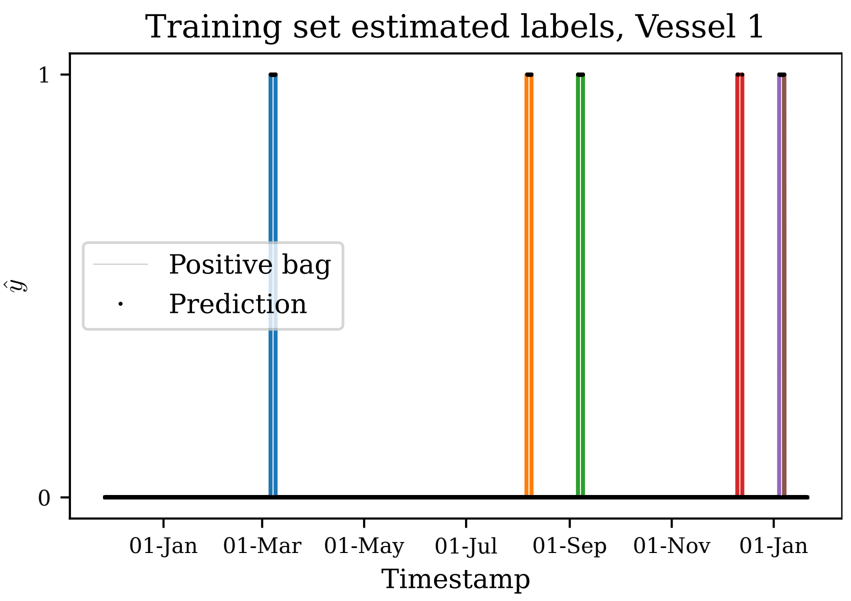


Figure 6.2: The labelled training data for vessel 1, after the MIL algorithm, with bags shown.

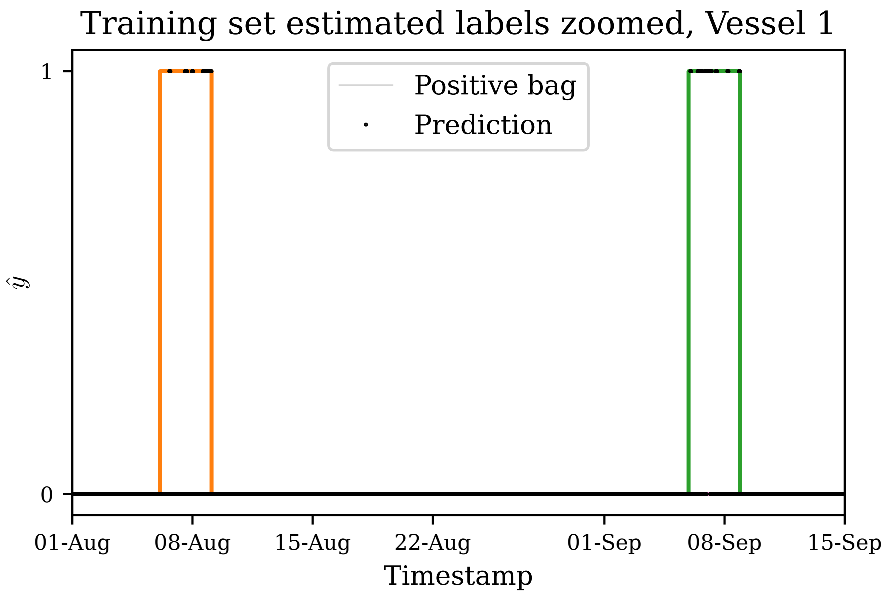


Figure 6.3: A zoomed plot of the training data for vessel 1, after the MIL algorithm, with bags shown

The classification confidence of each sample across the entirety of vessel 1's training set is visualized in Figure 6.4, a zoomed in plot showing the first bag in Figure 6.5, and another zoomed in plot showing a large cluster of bags over about a week in Figure 6.6. In Figure 6.5 it can be observed that there is a large number of samples with very high classification confidence a couple of days before the bag around 04-Feb. These samples could be an indication that the bag size can be expanded to be more than 3 days in length. However, because the data sets are weakly labelled, it is impossible to know if a failure was ever close to occurring. Large number of high classification confidence samples within a short time frame outside a bag could also be an indication that a failure was going to occur, but that the problem was solved by the crew on board in time. This is not reflected in the data sets, and is therefore purely hypothetical, but it presents an additional challenge when working with the data sets.

A similar scenario can also be observed in Figure 6.6 around 20-Apr, where a cluster of samples with very high classification confidence appear. These samples are however located further away from a failure at almost 1 week before. A single sample with very high classification confidence can also be observed a couple of days before the early April bag.

Because the data are weakly labelled, and the true labels are therefore unknown, it makes little sense to look at a standard confusion matrix for the testing set. We propose to instead look at the number of samples X_i with a predicted label $\hat{y}_i = 0$, outside the bags as the true negative, and those with a predicted label $\hat{y} = 1$ as the false positive. Doing the equivalent within the positive bags would not be appropriate, as not every sample within the positive bags has a true label of 1. The false negative were therefore instead determined

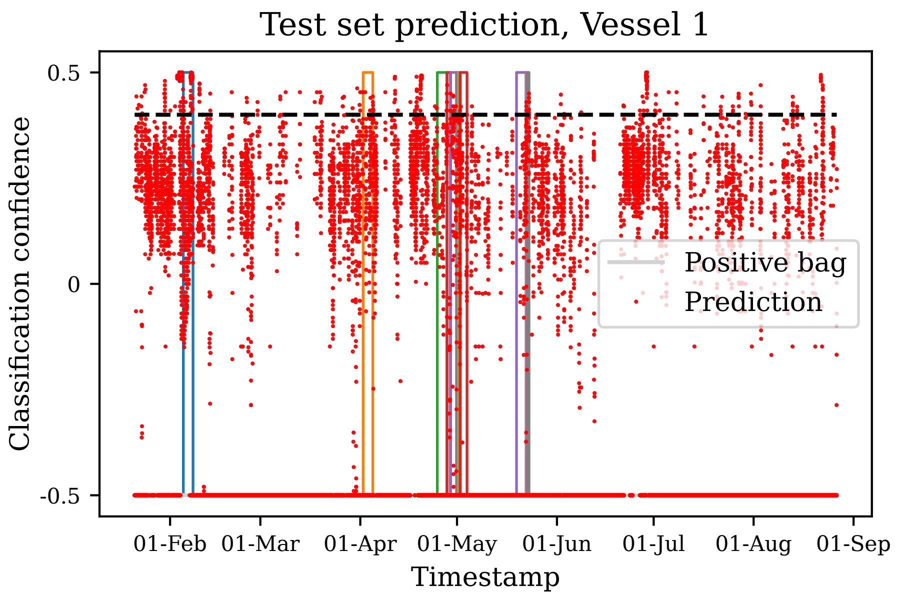


Figure 6.4: Classification confidence for samples in the entire test set of vessel 1. The rectangles indicate the positive bags.

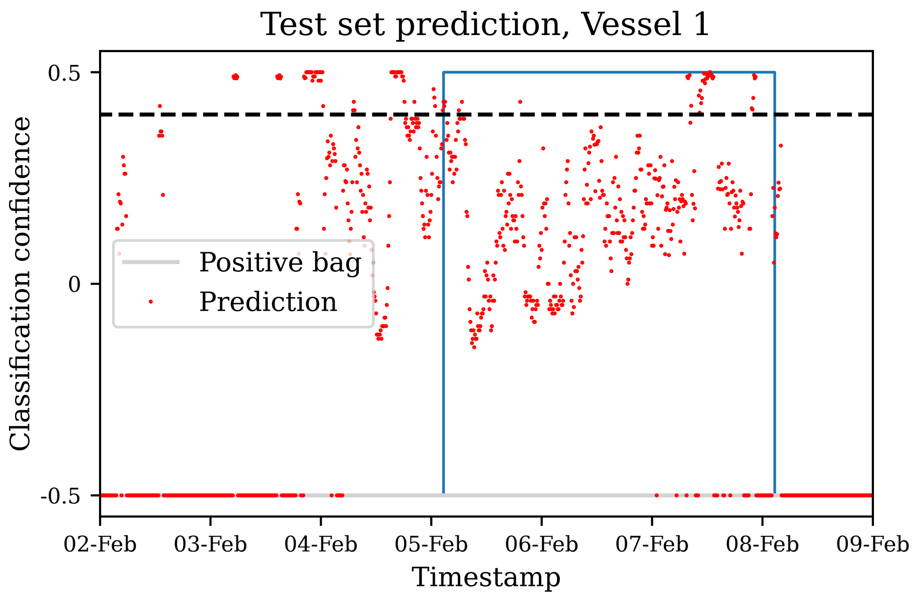


Figure 6.5: Classification confidence for samples in the test set of vessel 1, zoomed in on the first bag. The rectangles indicate the positive bag.

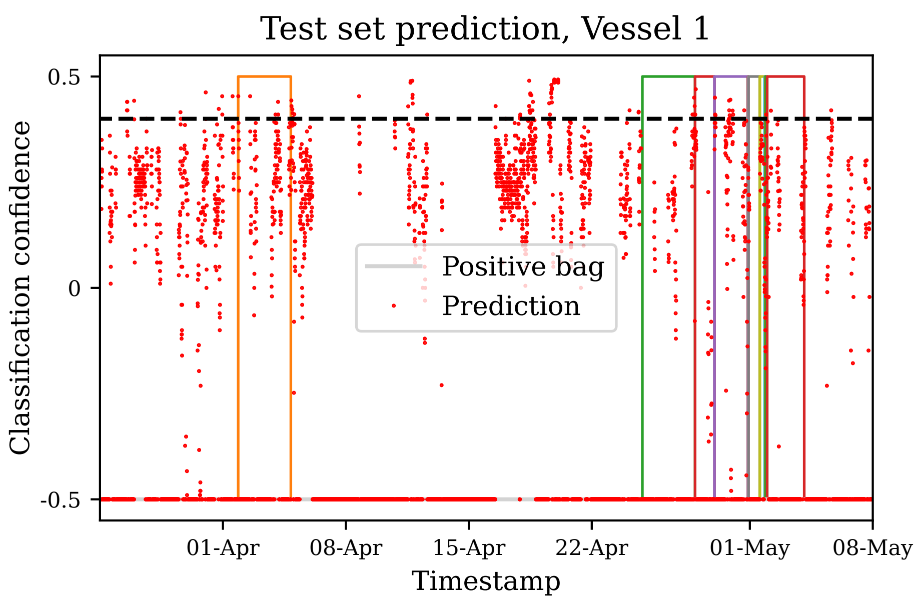


Figure 6.6: Classification confidence for samples in the test set of vessel 1, zoomed in on a cluster of bags. The rectangles indicate the positive bags.

by the number of positive bags that do not contain any X_i with a predicted label $\hat{y}_i = 1$, equivalently the true positive was the number of bags containing at least one sample X_i with a predicted label of $\hat{y}_i = 1$. The confusion matrices resulting from this process for all the vessels, using a classification confidence of 0.4 as the cutoff point are seen in Tables 6.6 to 6.9 on Pages 55 to 56.

The results seem good, and are fairly similar for all the vessels, but vessel 3, and 4 have more false positives. This is likely related to the fact that these vessels have the lowest amount of distinct failures, as both of them only have failures occurring on 2 or 3 different days in both their training and test sets.

	0	1
0	28,333	281
1	1	17

Table 6.6: Raw confusion matrix for the test set of vessel 1. Predicted labels in the vertical direction, and true labels in the horizontal direction.

	0	1
0	12,603	227
1	0	16

Table 6.7: Raw confusion matrix for the test set of vessel 2. Predicted labels in the vertical direction, and true labels in the horizontal direction.

	0	1
0	19,144	553
1	0	13

Table 6.8: Raw confusion matrix for the test set of vessel 3. Predicted labels in the vertical direction, and true labels in the horizontal direction.

	0	1
0	30,912	2,081
1	0	7

Table 6.9: Raw confusion matrix for the test set of vessel 4. Predicted labels in the vertical direction, and true labels in the horizontal direction.

The feature importance from the model trained on vessel 1 can be observed in Figure 6.7. The metric used is the Gini importance, which is calculated from the total reduction in the Gini cost (see Equation (4.4)) resulting from the splits using that feature. From this figure, the RI features at the end can easily be distinguished from the aggregation features, these can be observed more easily in Figure 6.8. While the RI features appear to be the largest contributors to the predictive power of the model, there are a few aggregation features that are clearly more significant than the rest. When analyzing the 25 features with highest importance as seen in Table 6.10, it can be observed that there are no features resulting from the min aggregation function. The mean and the maximum aggregation features also appear to have similar importance in multiple cases. Both of these observations may indicate that the minimum and either the mean or the maximum aggregations can be removed to vastly reduce the number of features. This will be investigated in the next section.

max(e_{878})	mean(e_{878})	mean(e_{877})	mean(e_{1191})	mean(e_{824})
C_{71}	C_{49}	C_4	C_{11}	max(e_{877})
C_{26}	C_5	C_{75}	C_{81}	max(e_{809})
C_{43}	C_{93}	mean(e_{809})	C_{57}	C_{29}
C_{17}	C_{83}	C_{40}	C_{88}	C_{89}

Table 6.10: The 25 most important features for vessel 1, ordered by importance. Left to right, then top to bottom

An attempt was also made at training a model to predict the time-to-failure. This model was trained with a standard random forest regression (using `RandomForestRegressor` from 'scikit-learn' (Pedregosa et al. 2011)), as it was only trained on the samples with a positive estimated label in the training set. Because of the large reduction in samples for this application, the total number of features is significantly larger than the number of samples, therefore, only the 50 most important features according to the failure prediction model were used to train the time-to-failure model. To evaluate the performance of this model, only the samples in the training set predicted to have a label of $\hat{y} = 1$ and are

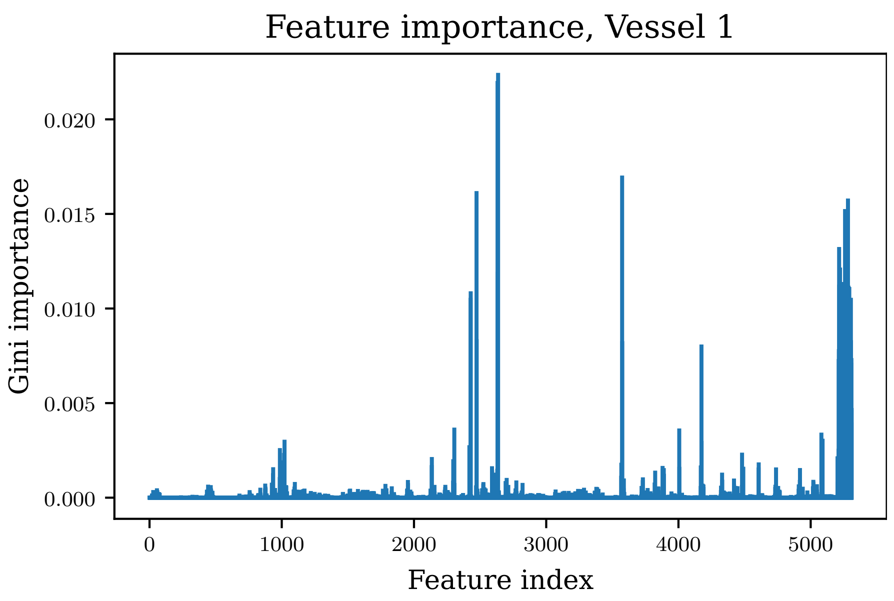


Figure 6.7: Feature importance for vessel 1.

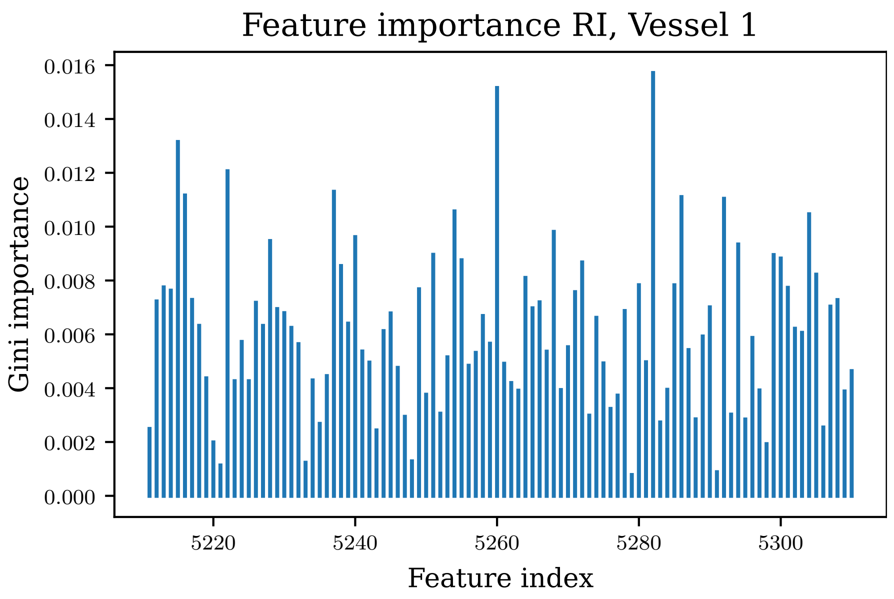


Figure 6.8: Feature importance of the RI features for vessel 1.

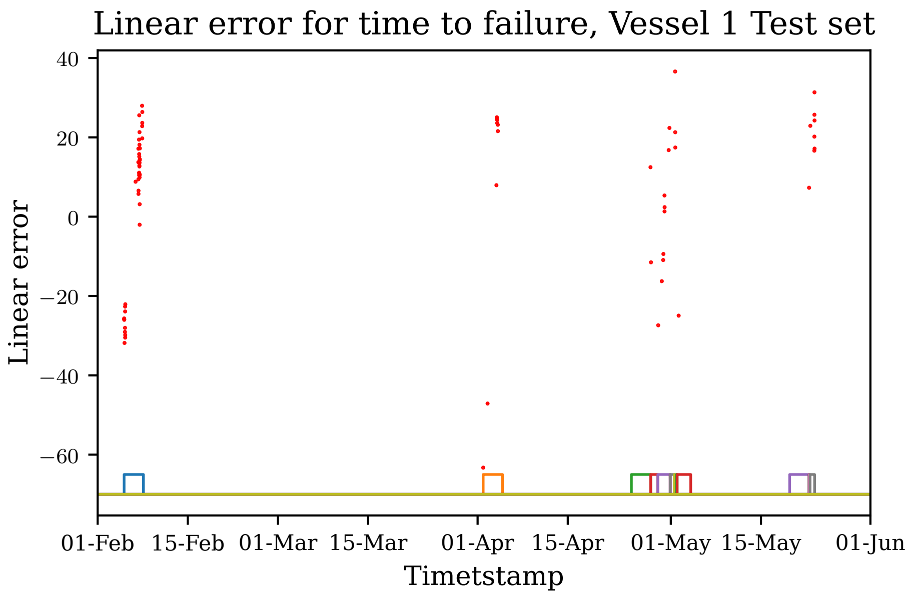


Figure 6.9: Time-to-failure prediction on the positively predicted samples from the test set of vessel 1. The height of the bags in the plot are lowered to allow for easier observation of the samples.

located within a bag are used. This is because using the samples outside the bags would result in large errors at resulting from the miss-prediction of the failure prediction model, and not the time-to-failure model. The time-to-failure prediction error in the resulting predictions on vessel 1's test set are presented in Figure 6.9. Here the timestamps of the samples are plotted against the difference between the predicted time-to-failure and the true time-to-failure. The model is clearly predicting that failures will occur later than they do the majority of the time, which may be the result of insufficient number of samples for training. Some outliers that predict the failure occurring more than 40 hours before it does can also be observed at the start of April.

6.4 Discussion and comparison

To see the benefits of using both the window aggregation functions and the RI features to train one model, a separate model was trained on each subset of features. In Table 6.10, it was also observed that only aggregation features that used the maximum or the mean as the aggregation function appeared among the most important features, and that for multiple event types, the mean and the maximum seem to have roughly equal importance. This may be an indication that the minimum and either the mean or the maximum could be removed without significantly impacting the results. A model was therefore also trained on the subset of features consisting only of the aggregation features using the maximum as the aggregation function. The model trained on the full set of features will be referred to as M_0, the model trained on the subset consisting only of the window aggregation features M_1, the one trained only on the RI features M_2, and the one trained only on the maximum aggregation features M_3.

In Table 6.11 the resulting confusion matrices obtained through these four models trained on each subset of features from vessel 1 are presented. It can be observed that all the models trained on the subsets of features (M_1, M_2, M_3) have a lower true positive rate than the model trained on the full data set (M_0), but some also have a lower false positive rate. M_1 has the lowest true positive rate, while keeping roughly the same false positive rate as M_0. M_2 also has a lower true positive rate than M_0, but has a lower false positive rate as well. It can also be seen that the results from M_3 are not significantly different from those obtained from M_1, indicating that the mean and the minimum may be redundant aggregation functions. While the M_2 does have better predictive ability than the M_1 and M_3, they are not as explainable. To properly perform PdM, explainable features are required such that the cause of a potential failure is understood and preventive measures can be taken. Combining these sets of features provides a mix of the predictive power of the RI features and the explainability of the window aggregation features.

M_0		M_1		M_2		M_3	
28,364	281	28,364	250	28,444	170	28,319	295
1	17	6	12	5	13	5	13

Table 6.11: Comparison of the confusion matrices of the models trained using different subsets of features for the data set of vessel 1.

There are multiple other papers that have attempted to solve similar problems, or used similar methods to solve other problems. Here, some of the results from these papers will be compared to the results presented in this thesis. These comparisons are also summarized in Section 6.4.

Two papers that used MIL for PdM are Sipos et al. (2014) (Ref. 1) and Naskos et al. (2020) (Ref. 2). Ref. 1 solved a similar problem to the one presented in this thesis, using MIL and SVM on real-world data from medical equipment to perform PdM. The only major difference begin that Ref. 1 attempted to classify the bags themselves, and not the individual samples in the data set. Ref. 2 used RF, but only a simplified version of MIL, the samples were assigned labels based on the distance from the failure without using any form of iterative optimization. A mix between real-world and simulated data from some manufacturing equipment was used in Ref. 2. In Ref. 1, two different data sets were analyzed, and an PM-AUC (PdM-based AUC (Area Under Cuve)) (a slightly modified version of AUC) of 0.319 was observed for one data set, and 0.730 for the other. While in Ref. 2, F_1-scores ranging from 0.48 to 0.61 for the different data sets were achieved.

It is worth noting that Ref. 1 used a different approach for handling the issue of multiple failures in short succession. After a failure was observed, a certain window of time was deemed to be an ‘infected period’ with none of the data in this period being used for training or testing. This technique could perhaps have been used in this thesis as well, but it would have lead to a reduction in the number of available failures for training and testing, which was not desirable.

Reference	Application domain	ML method	Performance
This thesis	PdM: Vessels	MIL-B-RF on event-log data	FPR[a]: 1.0-6.3%, TPR[b]: 94.4-100%
Ref. 1	PdM: Medical equipment	MIL combined with SVM[c] on event-log data	PM-AUC scores of 0.319-0.730
Ref. 2	PdM: Manufacturing equipment	Simplified[d] MIL with RF on event-log data	F_1-scores between 0.48-0.61
Ref. 3	NLP[e]: Polish sentences	BRF on 20:1 imbalanced data	F_1-score of 32.6%
Ref. 4	PdM: Computer system	RI with weighted SVM	Balance between 0.36 and 0.94

[a]False Positive Rate
[b]True Positive Rate
[c]Support Vector Machines, see Section 2.4
[d]Labels were simply assigned to samples based on distance from failure
[e]Natural Language Processing

Table 6.12: Comparison of results between different papers using similar methods as the ones presented in this thesis.

A couple of other papers that are less closely related to the work presented in this thesis, but still used some of the same methods. The natural language processing paper Kobyliński and Przepiórkowski (2008) (Ref. 3), which attempted to classify Polish sentences in a real-world data set using BRF. As well as Fronza et al. (2013) (Ref. 4) which used RI along with weighted SVM to predict failures in a computer system using log files. The data set presented in Ref. 3 was labelled, and the imbalance of 20:1 was small compared to the imbalance of the problem presented in this thesis. An F_1-score of 32.6% was achieved with a precision of 21.4% and recall of 69.0%. Ref. 4 tested the RI and SVM approach on a real data set consisting of 6 different applications with very varying results. True negative rates varied between 51% and 98% and a true positive rate between 17% and 93%, resulting in a balance between 36% and 94%.

While the results presented in this thesis are promising, with a low *False Positive Rate* (FPR) and a high *True Positive Rate* (TPR), there are still a large number of false positives compared to the number of true positives. Because of the failure, miss-classifying just 1% of the samples outside the bags while correctly classifying almost every failure still results in only 5.7% of the positive predictions for vessel 1 being true positives. This is simply too low for the model to be properly used for PdM as only about 1 out of every 18 predicted samples lead to a failure.

CHAPTER 7

Conclusions and future works

Conclusion

This thesis presented a machine learning framework based on event data for *Predictive Maintenance* (PdM) on vessels equipped with *Electrical Propulsion Systems* (EPSs). Multiple different algorithms were combined into one method (*Multiple Instance Learning through Balanced Random Forest* (MIL-B-RF)) designed specifically to handle some of the common problems encountered in the PdM field.

To start off, a general introduction to alarm systems and the maintenance field were given, before the focus was moved to PdM specifically. Here the scope of the thesis was laid out, and the challenges currently facing the field were presented along with the approaches and algorithms that can be used to solve them.

The data sets were then presented along with background information about the ships that they originate from. The most significant challenges with working on the data sets were also presented. In particular, these were; the lack of expert knowledge resulting in weakly labelled data, extreme data imbalance because failures were rarely observed, and interpretability to ensure that preventive measures can be taken. Using only the event-logs, it was observed that the results were promising, being roughly in-line with the results observed in papers solving similar problems and using similar techniques.

Two different methods were combined for the pre-processing of the data, window aggregation and *Random Indexing* (RI). The window aggregation provided very explainable features, while RI provided good prediction ability by taking into account the context of the event types. When comparing the results of the models using different subsets of features at the end of the thesis, it was observed that RI provided superior prediction power compared to the window aggregation, but the window aggregation does still provide more explainable features.

Random Forest (RF) was the model that was used for failure predictions. This model handled the data sets well because of its resilience to overfitting and noise features, and for its decent interpretability. The presented results were good considering the very large amount of features (more than 10,000 for one data set), with many of window aggregation features in particular potentially being noise features.

To handle the data imbalance prevalent in this, and many other PdM problems, *Balanced Random Forest* (BRF) was used. This method undersamples the majority class to artificially reduce the between-class imbalance in the data sets while training. This appears to have worked well, as the models do predict the minority class, which would likely not be the case for a standard RF model, as near 100% accuracy could be achieved by simply predicting every sample as not causing a failure.

Training a supervised model on weakly labelled data required the true labels of the samples to be uncovered, *Multiple Instance Learning* (MIL) was the algorithm chosen to accomplish this. The MIL algorithm used in this thesis iteratively solves an optimization problem by assigning labels to each samples based on their true class label probability, while simultaneously training a model with the best predictive power. It is difficult to say whether the MIL labelled the data well, as the true labels are unknown, but the presented results would indicate that it was a success.

Finally, comparisons were conducted between the results obtained by using the methods presented in this thesis on different subsets of the final data sets, and with the results presented in other papers solving similar problems. Despite the observed results being good, with a high true positive rate and a low false positive rate, they are not accurate enough to be implemented in a real world PdM strategy in EPS equipped vessels, because of the data imbalance.

Future works

While the work presented in this thesis achieved promising results, there are multiple other methods that could potentially be investigated in future works.

In addition to the event-log data used throughout the thesis, the continuous signal data from various components on board the vessels can be investigated. If this signal data were to be used in combination with the event-log data to create a hybrid model, better results may be possible to achieve. Attempting this would however come with several challenges. A method that allows for the combination of continuous and categorical data must be used. Also, many of the alarms in the event data occur because a measured condition exceeds a given threshold, this results in a high correlation between some of the events and the signal data which must be taken into account.

If a sufficient amount of high quality data was collected, a way of performing hyperparameter selection could be investigated. The hyperparameters presented in this thesis were not optimized, because this would require a separate validation set or another process such as cross-validation. Neither of these options were desirable, as a validation set for each vessel would have resulted in many of the training, testing and validation sets only containing one or two failures or clusters of failures, which would be insufficient for training a model. Using cross-validation would have been a more appropriate approach, but was avoided because it would have made it difficult to perform some interesting analysis that required the data to remain sequential, such as analyzing the failure predictions that occur shortly before the start of the bags to determine whether or not the bag size could be increased.

While RF is a good method for handling data sets with potentially large amounts of redundant features, it is not the only model that is popular within the PdM field. *Support Vector Machine* (SVM) is another popular model for PdM, as its use of kernels also allows it to perform well in high-dimensional settings.

The use of an SVM model does however result in many more hyperparameters, making proper hyperparameter optimization more important.

In this thesis, the used MIL approach re-trained the full forest during each iteration of the algorithm. A different approach that could perhaps produce superior results would be the iterative re-training of the individual trees in the forest. This would however be a lot more difficult to implement because of the python libraries that were used in this thesis, as they do not provide a simple way of retraining individual trees in a forest. This could be investigated as a reasonable approach if another implementation is used, or if the algorithm is implemented from scratch.

Because a combination of BRF and MIL was used, the number of samples used for training in each iteration of the MIL algorithm was not consistent. This is because the BRF algorithm undersamples the majority class to reduce the imbalance during training. However, MIL adjusts the number of samples in the minority class between iterations. This results in the number of bootstrap samples drawn for the training of the BRF model also changing with each iteration. A solution to this problem could potentially be investigated in future works.

The time-to-failure model could be investigated further in future works. While time-to-failure predictions were made, the results obtained were not too promising. A cluster model could potentially also be investigated. If a good clustering process is found, different failure modes could perhaps be discovered, this may however prove difficult with the small amount of failure data available.

Bibliography

Achawanantakun, R. et al. (Aug. 2015). 'LncRNA-ID: Long non-coding RNA IDentification using balanced random forests'. In: *Bioinformatics* vol. 31, no. 24, pp. 3897–3905.

Allah Bukhsh, Z. et al. (2019). 'Predictive maintenance using tree-based classification techniques: A case of railway switches'. In: *Transportation Research Part C: Emerging Technologies* vol. 101, pp. 35–54.

Allen, T. M. (2005). 'U . S . Navy Analysis of Submarine Maintenance Data and the Development of Age and Reliability Profiles ABSTRACT'. In:

Binding, A., Dykeman, N. and Pang, S. (2019). 'Machine Learning Predictive Maintenance on Data in the Wild'. In: *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, pp. 507–512.

Bradbury, S. et al. (Oct. 2018). *Digitally enabled reliability: Beyond predictive maintenance*. Tech. rep. McKinsey & Company.

Breiman, L. (Oct. 2001). 'Random Forests'. In: *Machine Learning* vol. 45, no. 1, pp. 5–32.

Canizo, M. et al. (2017). 'Real-time predictive maintenance for wind turbines using Big Data frameworks'. In: *2017 IEEE International Conference on Prognostics and Health Management (ICPHM)*, pp. 70–77.

Carvalho, T. P. et al. (2019). 'A systematic literature review of machine learning methods applied to predictive maintenance'. In: *Computers & Industrial Engineering* vol. 137, p. 106024.

Chatterjee, N. and Sahoo, P. K. (2015). 'Random Indexing and Modified Random Indexing based approach for extractive text summarization'. In: *Computer Speech & Language* vol. 29, no. 1, pp. 32–44.

Chen, C., Liaw, A. and Breiman, L. (July 2004). 'Using Random Forest to Learn Imbalanced Data'. In: *University of California, Berkeley.*

Çınar, Z. M. et al. (2020). 'Machine Learning in Predictive Maintenance towards Sustainable Smart Manufacturing in Industry 4.0'. In: *Sustainability* vol. 12, no. 19.

DeltaV (Nov. 2016). *OPC Alarms and Events Overview.* Emerson Process Management.

European Southern Observatory (ESO) (Feb. 2007). *User Manual of the Central Control Software (CCS).*

Fronza, I. et al. (2013). 'Failure prediction based on log files using Random Indexing and Support Vector Machines'. In: *Journal of Systems and Software* vol. 86, no. 1, pp. 2–11.

Goel, P., Datta, A. and Mannan, M. S. (2017). ‘Industrial alarm systems: Challenges and opportunities’. In: *Journal of Loss Prevention in the Process Industries* vol. 50, pp. 23–36.

Goel, P., Pistikopoulos, E. et al. (2019). ‘A data-driven alarm and event management framework’. In: *Journal of Loss Prevention in the Process Industries* vol. 62, p. 103959.

Gutschi, C. et al. (2019). ‘Log-based predictive maintenance in discrete parts manufacturing’. In: *Procedia CIRP* vol. 79. 12th CIRP Conference on Intelligent Computation in Manufacturing Engineering, 18-20 July 2018, Gulf of Naples, Italy, pp. 528–533.

Hastie, T., Tibshirani, R. and Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* New York, NY: Springer New York.

He, H. and Ma, Y. (2013). *Imbalanced Learning: Foundations, Algorithms, and Applications.* 1st. Wiley-IEEE Press.

Janssens, O. et al. (2016). ‘Convolutional Neural Network Based Fault Detection for Rotating Machinery’. In: *Journal of Sound and Vibration* vol. 377, pp. 331–345.

Kanerva, P., Kristoferson, J. and Holst, A. (2000). ‘Random Indexing of Text Samples for Latent Semantic Analysis’. In: *Proceedings of the Annual Meeting of the Cognitive Science Society.* Vol. 22.

Kaur, H., Pannu, H. S. and Malhi, A. K. (Aug. 2019). ‘A Systematic Review on Imbalanced Data Challenges in Machine Learning: Applications and Solutions’. In: *ACM Comput. Surv.* vol. 52, no. 4.

Knutsen, K. E., Manno, G. and Vartdal, B. (Jan. 2014). *Beyond condition monitoring in the maritime industry.* Tech. rep.

Kobyliński, Ł. and Przepiórkowski, A. (2008). ‘Definition Extraction with Balanced Random Forests’. In: *Advances in Natural Language Processing.* Ed. by Nordström, B. and Ranta, A. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 237–247.

Langone, R. et al. (2015). ‘LS-SVM based spectral clustering and regression for predicting maintenance of industrial machines’. In: *Engineering Applications of Artificial Intelligence* vol. 37, pp. 268–278.

Leistner, C., Saffari, A. and Bischof, H. (Aug. 2010). ‘MIForests: Multiple-Instance Learning with Randomized Trees’. In: pp. 29–42.

Leite, D. (2019). ‘Comparison of Genetic and Incremental Learning Methods for Neural Network-Based Electrical Machine Fault Detection’. In: *Predictive Maintenance in Dynamic Systems: Advanced Methods, Decision Support Tools and Real-World Applications.* Ed. by Lughofer, E. and Sayed-Mouchaweh, M. Cham: Springer International Publishing, pp. 231–268.

Lemaître, G., Nogueira, F. and Aridas, C. K. (2017). ‘Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning’. In: *Journal of Machine Learning Research* vol. 18, no. 17, pp. 1–5.

Lughofer, E. and Sayed-Mouchaweh, M. (2019). ‘Prologue: Predictive Maintenance in Dynamic Systems’. In: *Predictive Maintenance in Dynamic Systems: Advanced Methods, Decision Support Tools and Real-World Applications.* Ed. by Lughofer, E. and Sayed-Mouchaweh, M. Cham: Springer International Publishing, pp. 1–23.

Maciel, L. and Ballini, R. (2019). ‘Fuzzy Rule-Based Modeling for Interval-Valued Data: An Application to High and Low Stock Prices Forecasting’. In:

Predictive Maintenance in Dynamic Systems: Advanced Methods, Decision Support Tools and Real-World Applications. Ed. by Lughofer, E. and Sayed-Mouchaweh, M. Cham: Springer International Publishing, pp. 403–424.

Mathew, J., Luo, M. and Pang, C. K. (2017). 'Regression kernel for prognostics with support vector machines'. In: *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1–5.

McKinney, W. (2010). 'Data Structures for Statistical Computing in Python'. In: *Proceedings of the 9th Python in Science Conference*. Ed. by Walt, S. van der and Millman, J., pp. 56–61.

Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press.

Naskos, A. et al. (2020). 'Event-Based Predictive Maintenance on Top of Sensor Data in a Real Industry 4.0 Case Study'. In: *Machine Learning and Knowledge Discovery in Databases*. Ed. by Cellier, P. and Driessens, K. Cham: Springer International Publishing, pp. 345–356.

Pedregosa, F. et al. (2011). 'Scikit-learn: Machine Learning in Python'. In: *Journal of Machine Learning Research* vol. 12, pp. 2825–2830.

Peng, Y., Dong, M. and Zuo, M. J. (Sept. 2010). 'Current status of machine prognostics in condition-based maintenance: a review'. In: *The International Journal of Advanced Manufacturing Technology* vol. 50, no. 1, pp. 297–313.

Pernabas, J. B., Fidele, S. F. and Vaithinathan, K. K. (2019). 'Enhancing Greedy Web Proxy caching using Weighted Random Indexing based Data Mining Classifier'. In: *Egyptian Informatics Journal* vol. 20, no. 2, pp. 117–130.

Praveenkumar, T. et al. (2014). 'Fault Diagnosis of Automobile Gearbox Based on Machine Learning Techniques'. In: *Procedia Engineering* vol. 97. "12th Global Congress on Manufacturing and Management" GCMM - 2014, pp. 2092–2098.

Ran, Y. et al. (2019). *A Survey of Predictive Maintenance: Systems, Purposes and Approaches*.

Reback, J. et al. (Mar. 2020). *pandas-dev/pandas: Pandas 1.0.3*. Version v1.0.3.

Reising, D. V. C., Downs, J. L. and Bayn, D. (2004). 'Human Performance Models for Response to Alarm Notifications in the Process Industries: An Industrial Case Study'. In: *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* vol. 48, no. 10, pp. 1189–1193.

Sahlgren, M. (2005). 'An Introduction to Random Indexing'. In: *Methods and applications of semantic indexing workshop at the 7th international conference on terminology and knowledge engineering*.

Sakib, N. and Wuest, T. (2018). 'Challenges and Opportunities of Condition-based Predictive Maintenance: A Review'. In: *Procedia CIRP* vol. 78. 6th CIRP Global Web Conference – Envisaging the future manufacturing, design, technologies and systems in innovation era (CIRPe 2018), pp. 267–272.

Scalabrini Sampaio, G. et al. (2019). 'Prediction of Motor Failure Time Using An Artificial Neural Network'. In: *Sensors* vol. 19, no. 19.

Schmidt, B. and Wang, L. (June 2015). 'Predictive Maintenance: Literature Review and Future Trends'. In: *Proceedings of the 25th International Conference on Flexible Automation and Intelligent Manufacturing*. Vol. 1.

Scully, P. (2019). *Predictive Maintenance Companies Landscape 2019*. URL: https://iot-analytics.com/predictive-maintenance-companies-landscape-2019/ (visited on 27/01/2021).

Shorten, D. (2012). 'Marine Machinery Condition Monitoring. Why has the shipping industry been slow to adopt'. In: *TECHNICAL INVESTIGATIONS DEPARTMENT (ed.). Lloyd's Register EMEA, United Kingdom.*

Silva, W. and Capretz, M. (2019). 'Assets Predictive Maintenance Using Convolutional Neural Networks'. In: *2019 20th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pp. 59–66.

Sipos, R. et al. (2014). 'Log-based predictive maintenance'. In: *Proceedings of the 20th ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1867–1876.

Sun, Y., Wong, A. and Kamel, M. S. (Nov. 2011). 'Classification of imbalanced data: a review'. In: *International Journal of Pattern Recognition and Artificial Intelligence* vol. 23.

Susto, G. A. et al. (2015). 'Machine Learning for Predictive Maintenance: A Multiple Classifier Approach'. In: *IEEE Transactions on Industrial Informatics* vol. 11, no. 3, pp. 812–820.

Tinga, T. and Loendersloot, R. (2019). 'Physical Model-Based Prognostics and Health Monitoring to Enable Predictive Maintenance'. In: *Predictive Maintenance in Dynamic Systems: Advanced Methods, Decision Support Tools and Real-World Applications*. Ed. by Lughofer, E. and Sayed-Mouchaweh, M. Cham: Springer International Publishing, pp. 313–353.

Uhlmann, E. et al. (2018). 'Cluster identification of sensor data for predictive maintenance in a Selective Laser Melting machine tool'. In: *Procedia Manufacturing* vol. 24. 4th International Conference on System-Integrated Intelligence: Intelligent, Flexible and Connected Systems in Products and Production, pp. 60–65.

Vasquez Capacho, J. et al. (2017). 'Alarm management via temporal pattern learning'. In: *Engineering Applications of Artificial Intelligence* vol. 65, pp. 506–516.

Wang, J. et al. (2017). 'Predictive maintenance based on event-log analysis: A case study'. In: *IBM Journal of Research and Development* vol. 61, no. 1, 11:121–11:132.

Winham, S. J., Freimuth, R. R. and Biernacka, J. M. (2013). 'A weighted random forests approach to improve predictive performance'. In: *Statistical Analysis and Data Mining: The ASA Data Science Journal* vol. 6, no. 6, pp. 496–505.

Wu, H., Huang, A. and Sutherland, J. W. (2020). 'Avoiding Environmental Consequences of Equipment Failure via an LSTM-Based Model for Predictive Maintenance'. In: *Procedia Manufacturing* vol. 43. Sustainable Manufacturing - Hand in Hand to Sustainability on Globe: Proceedings of the 17th Global Conference on Sustainable Manufacturing, pp. 666–673.

Xia, Y., Zhu, Q. and Wei, W. (2015). 'Weakly Supervised Random Forest for Multi-Label Image Clustering and Segmentation'. In: *Proceedings of the 5th ACM on International Conference on Multimedia Retrieval.*

Zhou, Z.-H. (Aug. 2017). 'A brief introduction to weakly supervised learning'. In: *National Science Review* vol. 5, no. 1, pp. 44–53.

Zhu, J. and Pierskalla, W. P. (2016). 'Applying a weighted random forests method to extract karst sinkholes from LiDAR data'. In: *Journal of Hydrology* vol. 533, pp. 343–352.

Zipf, G. K. (1949). *Human Behavior And The Principle Of Least Effort.* Addison Wesley Press, Inc.

Appendices

Code

GenerateTimelines.py

```python
import scipy.io
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import dates

def create_df(vessel, data_type=None, files=None, sort=True):
    path = f'..\\DATA_V~1\\Vessel{vessel}'
    dframes = []
    srcs = []
    vals = []
    vals_start = []
    counter = 0
    if files == None:
        files = os.listdir(path)
    for file in files:
        print("Current: " + path + "\\" + file + "
        ↪   ", end="\r")
        data = scipy.io.loadmat(path + '\\' + file,
        ↪   struct_as_record=False)['data'][0][0]

        if data_type != None:
            try:
                if data.type[0] != data_type:
                    #print(data.type)
                    continue
            except IndexError:
                print('File empty: ' + path + '\\' + file)
                continue

        srcs.append(file[:-4]) # Slice off .mat from file name
        timestamps = data.timestamp[:, 0]
        N = len(timestamps)
        if data.type[0] == 'string':
            values = data.value[:, 0]-1+len(vals) # Index from
             ↪  0
            vals_start.append(len(vals))
            for value in data.Cats:
                vals.append(value[0][0])
        else:
            values = data.value.reshape(1, -1)[0]

        try:
            drive = int(file[3])
        except ValueError:
            drive = int(file[5])

        dframes.append(pd.DataFrame({'time': timestamps,
        ↪   'drive': drive, 'src': np.ones(N,
        ↪   dtype=np.uint8)*counter, 'type': values}))
        counter += 1

    df = pd.concat(dframes, ignore_index=True)
    if sort:
        df = df.sort_values(by=['time'])

    return df, srcs, vals, vals_start

def save_files(names, df, srcs, vals, vals_start):
    df.to_pickle(names[0])
    np.save(names[1], np.array(srcs))
    np.save(names[2], np.array(vals))
    np.save(names[3], np.array(vals_start))

def create_event_timelines():
    for vessel in range(1, 5):
        names = [f'Data\\Vessel{vessel}_timeline.pkl',
                 f'Data\\Vessel{vessel}_srcs',
                 f'Data\\Vessel{vessel}_vals',
                 f'Data\\Vessel{vessel}_vals_start']
        save_files(names, *create_df(vessel, 'string'))
    print("Done!
    ↪   ")

def read_mat(file):
    mat = scipy.io.loadmat(file,
    ↪   struct_as_record=False)['data'][0][0]
    temp = np.zeros((mat.timestamp.shape[0], 2))
    temp[:, 0] = mat.timestamp.reshape((1, -1))[0]
    temp[:, 1] = mat.value.reshape((1, -1))[0]
    mat_df = pd.DataFrame(temp, columns=("Timestamp",
    ↪   "Value"))
    mat_df["Timestamp"] = pd.to_datetime(mat_df["Timestamp"],
    ↪   unit="s")
```

```python
    return mat_df

def read_data(vessel):
    df = pd.read_pickle(f'Data\\Vessel{vessel}_timeline.pkl')
    ↪  .reset_index().drop('index',
    ↪  'columns')
    srcs = np.load(f'Data\\Vessel{vessel}_srcs.npy')
    vals = np.load(f'Data\\Vessel{vessel}_vals.npy')
    vals_start =
    ↪  np.load(f'Data\\Vessel{vessel}_vals_start.npy')
    return df, srcs, vals, vals_start
```

RandomIndexing.py

```python
import GenerateTimelines
import pandas as pd
import numpy as np
from pathlib import Path
from scipy import sparse
from scipy.stats import randint

def create_random_indexing(vessel, tw_size=10, length=100,
↪   density=0.1, random_state=0, vessel=1):
    pd.set_option('float_format', '{:f}'.format)
    df = pd.read_pickle(f"Vessel{vessel}_timeline.pkl")
    vals = np.arange(df["type"].max()+1)

    # vals.shape[0]x100 sparse matrix with 10% density,
    ↪  non-zero values 1 and 2, 2 changed to -1
    rv = randint(1, 3)
    index_vectors = sparse.random(vals.shape[0], length,
    ↪  format="csr", dtype=np.float32, density=0.1,
    ↪  random_state=0, data_rvs=rv.rvs).astype(np.int8)
    index_vectors[index_vectors == 2] = -1

    # Check that no index_vector has 0 non-zero values
    temp = sparse.csc_matrix(np.zeros((index_vectors.shape[0],
    ↪  1)))
    for i in range(index_vectors.shape[1]):
        temp += np.abs(index_vectors[:, i])
    assert np.amin(temp.A) != 0

    # Ensure all index_vectors are unique,
    ↪  http://www.ryanhmckenna
    ↪  .com/2017/01/efficiently-remove-duplicate-rows-from
    ↪  .html
    x = np.random.rand(index_vectors.shape[1])
    assert np.unique(index_vectors.dot(x)).shape[0] ==
    ↪  vals.shape[0]
    while np.unique(index_vectors.dot(x)).shape[0] < vals.shape[0]:
        _, unique_idx = np.unique(index_vectors.dot(x), return_index=True)[1]
        for i in range(vals.shape[0]):
            if i in unique_idx:
                print(i)
                index_vectors[i] = sparse.random(1, length, format="csr", dtype=np.float32, density=0.1, random_state=0, data_rvs=rv.rvs).astype(np.int8)[0]

    assert np.unique(index_vectors.dot(x)).shape[0] == vals.shape[0]

    # Each context vector is the sum of every context an event is found in, context is tw_size/2 sec before and after
    # Context vectors are dense
    context_vectors = np.zeros((vals.shape[0], length), dtype=np.int32)
    for i in range(1, len(df)-1):
        tw_start = df['time'][i] - tw_size/2
        tw_end = df['time'][i] + tw_size/2
        window = df[(df['time'] >= tw_start) & (df['time'] < tw_end)]
        for index, val in window['type'].items():
            context_vectors[df["type"][i]] += index_vectors[df["type"][val]]
        print(f"Progress: {(i-1)/(len(df)-2)*100:.2f}%", end="\r")

    sparse.save_npz(f"Vessel{vessel}_index_vectors.npz", index_vectors)
    np.save(f"Vessel{vessel}_context_vectors.npy", context_vectors)

    return context_vectors

def read_context_vectors(vessel):
    return np.load(f"Vessel{vessel}_context_vectors.npy")
```

RollingWindows.py

```python
import GenerateTimelines
import RandomIndexing
import pandas as pd
import numpy as np
import itertools
from pathlib import Path
from scipy import sparse

def rolling_windows(df, vessel, tw_size=60*10, rw_sizes=60*60):
    if vessel == 1:
        trip1 = 20
        trip2 = 156
    elif vessel == 2:
        trip1 = 65
        trip2 = 264
    elif vessel == 3:
        trip1 = 147
        trip2 = 147 # Only 1 trip type for vessel 3
    elif vessel == 4:
        trip1 = 16
        trip2 = 40

    context_vectors = RandomIndexing.read_context_vectors(vessel)
    tw_start = df['time'].iloc[0] - df['time'].iloc[0] % tw_size
    tw_num = np.ceil((df['time'].iloc[-1]-tw_start)/tw_size).astype(int)
    tw = np.zeros((tw_num, len(vals)))
    tw_context = np.zeros((tw_num, context_vectors.shape[1]))
    tw_counter = 0
    tw_end_times = np.zeros(tw_num)
    tw_labels = np.zeros(tw_num)

    while tw_start < df['time'].iloc[-1]:
        tw_end = tw_start + tw_size
        tw_end_times[tw_counter] = tw_end
        # Start from end of last window, stop after first end
        window = df[(df['time'] >= tw_start) & (df['time'] < tw_end)]

        window_counts = window['type'].value_counts()
        for val, count in window_counts.items():
            tw[tw_counter][val] = count
            tw_context[tw_counter] += context_vectors[val]*count
            # Label tw BEFORE the tw containing "INU Tripped" as 1
            if val == trip1 or val == trip2 and tw_counter > 0:
                tw_labels[tw_counter-1] = 1

        # Print progress
        progress = (tw_start-df['time'].iloc[0])/(df['time'].iloc[-1]-df['time'].iloc[0])*100
        print(f"Progress: {progress:.2f}%", end="\r")

        tw_start = tw_end
        tw_counter += 1


    rw_sizes = np.array(rw_size)//tw_size
    tw_length = tw.shape[0]
    num_vars = tw.shape[1]
    num_context_vars = tw_context.shape[1]
    aggregations = ["min", "mean", "max"]
    num_aggregations = len(aggregations)
    s = np.zeros((len(rw_sizes), tw_length-rw_sizes.max(), num_vars, num_aggregations))
    s_context = np.zeros((len(rw_sizes), tw_length-rw_sizes.max(), num_context_vars))

    # Loop over all rolling windows sizes
    for i in range(rw_sizes.shape[0]):
        weights = np.logspace(0, 1, rw_sizes[i])
        weights = weights/np.sum(weights)

        # Loop over all time_windows
        for j in range(rw_sizes.max(), tw_length):
            window = tw[(j-rw_sizes[i]+1):(j+1)]
            s_j_min = window.min(axis=0)
            s_j_mean = window.mean(axis=0)
            s_j_max = window.max(axis=0)
            s[i, j-rw_sizes.max()] = np.stack((s_j_min, s_j_mean, s_j_max), axis=1)

            window_context = tw_context[(j-rw_sizes[i]+1):(j+1)]
            w_sum = np.dot(window_context.reshape(-1, rw_sizes[i]), weights)
            s_context[i, j-rw_sizes.max()] = w_sum

            # Print progress
            print(f"Progress: {(i + j/len(tw))/len(rw_sizes)*100:.1f}%", end="\r")

    val_names = [f"val_{a:d}_{b}" for a, b in itertools.product(range(num_vars), aggregations)]
    ctx_names = [f"ctx_{i:d}" for i in range(num_context_vars)]
    var_names = val_names + ctx_names + ["label"]
    cols = pd.Index(var_names, name="vars")
    rows = pd.Index(pd.to_datetime(tw_end_times[rw_sizes.max():], unit='s'), name="end_times")
    s_dfs = []

    for i in range(rw_sizes.shape[0]):
        s_combined = np.concatenate((s[i].reshape(s.shape[1],
        ↪   s.shape[2]*s.shape[3]), s_context[i],
        ↪   tw_labels.reshape(-1, 1)[rw_sizes.max():]),
        ↪   axis=1)
        s_df = pd.DataFrame(s_combined, columns=cols,
        ↪   index=rows, dtype=pd.SparseDtype("float32",
        ↪   0.0)).astype({"label": 'bool'})
        s_df.to_pickle(f"Vessel_{vessel}_rw_size_{rw_⌟
        ↪   sizes[i]*tw_size//3600}h.pkl")
        s_dfs.append(s_df)

    return s_dfs
```

MILBRF.py

```
import numpy as np
from sklearn.ensemble import RandomForestRegressor
from imblearn.ensemble import BalancedRandomForestClassifier
from pathlib import Path
import matplotlib.pyplot as plt
import pandas as pd
import matplotlib
from numpy import sqrt
from matplotlib.dates import DateFormatter

class MILBRF:
    def __init__(self, data, labels, tw_size=60*10,
    ↪   bag_size=60*60*24*3):
        self.train_data = data[:'20XX-01-20']
        self.train_labels = labels[:'20XX-01-20']
        self.test_data = data['20XX-01-21':]
        self.test_labels = labels['20XX-01-21':]
        self.tw_size = tw_size
        self.bag_size = bag_size
        self.bags, self.num_bags =
        ↪   self.create_bags(self.train_data,
        ↪   self.train_labels)
        self.positive_bag_indecies = np.where(self.bags !=
        ↪   0)[0]

        self.model = BalancedRandomForestClassifier(
            n_estimators=100,
            max_features='auto',
            class_weight={False : 1, True : 1},
            n_jobs=4)
        self.model.fit(self.train_data, self.bags != 0)

        self.ttf_model =
        ↪ RandomForestRegressor(n_estimators=300, n_jobs=4)

    def create_bags(self, data, labels):
        bags = np.zeros_like(labels, dtype=np.int)

        bag = 1
        for idx in np.where(labels)[0]:
            bags[idx] = bag
            bag += 1
            i = 1
            while idx-i > 0 and data.index[idx-i] >
            ↪ data.index[idx] -
            ↪ pd.Timedelta(self.bag_size*10**9):
                if bags[idx-i] != 0: break
                if idx-i < 0: break
                bags[idx-i] = bags[idx]
                i += 1


        reduce = 0
        for i in range(bag):
            bag_indecies = np.where(bags == i)[0]
            if bag_indecies.shape[0] == 0:
                reduce += 1
            else:
                bags[bag_indecies] -= reduce

        return bags, bag-reduce

    def margin(self):
        probas = self.model.predict_proba(self.train_data)
        margin = 2*probas-1
        return margin

    def temperature(self, t, C=1/5):
        return np.exp(-t*C)

    def p(self, T):
        p = np.exp((self.margin()-T)/T)
        return p/p.sum(1)[:, None]

    def algorithm1(self, p):
        positive_bag_indecies = np.where(self.bags != 0)[0]
        new_labels = np.zeros_like(self.train_labels)
        for i in self.positive_bag_indecies:
            new_labels[i] = np.random.choice([0, 1], 1,
            ↪ p=p[i])

```

```python
        for bag in range(1, self.num_bags):
            bag_indecies = np.where(self.bags == bag)[0]
            if new_labels[bag_indecies].max() == 0:
                print(f"No positive labels in bag {bag}")
                idx = p[bag_indecies][:, 1].argmax()
                new_labels[idx] = 1

        print(f"Positive labels: {new_labels.sum()}")
        return new_labels

    def algorithm2(self, end = 0.1):
        t = 0
        T = self.temperature(t)
        while T > end:
            print(f"Temperature: {T}")
            p = self.p(T)
            new_labels = self.algorithm1(p)
            self.model = BalancedRandomForestClassifier(
                n_estimators=100,
                max_features='auto',
                class_weight={False : 1, True : 1},
                n_jobs=4)
            self.model.fit(self.train_data, new_labels)
            t += 1
            T = self.temperature(t)
        self.train_time_to_failure(new_labels)
        return new_labels

    def train_time_to_failure(self, labels, num_features=50):
        train_label_indecies = np.where(self.train_labels)[0]
        label_indecies = np.where(labels)[0]
        ttf_labels = np.zeros(label_indecies.shape[0])

        for i in range(len(label_indecies)):
            train_label_index = train_label_indecies[np.where(train_label_indecies >= label_indecies[i])[0][0]]
            ttf_labels[i] = (self.train_data.index[train_label_index].timestamp() - self.train_data.index[label_indecies[i]].timestamp())/3600 # In hours

        self.important_features = self.train_data.iloc[:, self.model.feature_importances_.argsort()[-num_features:][::-1]].columns
        self.ttf_model.fit(self.train_data[self.important_features][labels][1:], ttf_labels[1:])
        return ttf_labels.astype(int)
```

```

if __name__ == "__main__":
    vessel = 1

    rw = pd.read_pickle(f"Vessel_{vessel}_rw_size_1h.pkl")
    ↪   .sparse.to_dense()
    tw_size = 60*10 # 10 minutes
    bag_size = 60*60*24*3 # 3 days

    model = MILBRF(rw.iloc[:, :-1], rw.iloc[:, -1],
    ↪   bag_size=bag_size)
    new_labels = model.algorithm2()
    pred_proba = model.model.predict_proba(model.test_data)[:,
    ↪   1]
    pred = pred_proba > 0.9
```