**DTU Management**
Department of Technology Management and Economics

# Data-driven predictive maintenance planning for maritime industry

Master Thesis

**Data-driven predictive maintenance planning for maritime industry**

Master Thesis
February, 2023

By
Angelos Daglaroglou


Cover photo: Vibeke Hempler, 2012
Published by: DTU, Department of Technology Management and Economics, Akademivej, Building 358, 2800 Kgs. Lyngby Denmark
www.man.dtu.dk
ISSN: [0000-0000] (electronic version)
ISBN: [000-00-0000-000-0] (electronic version)

ISSN: [0000-0000] (printed version)
ISBN: [000-00-0000-000-0] (printed version)

## Approval

This thesis has been prepared over six months at the Section for Business Analytics, Department of Technology Management and Economics, at the Technical University of Denmark, DTU, in partial fulfilment for the degree Master of Science in Engineering, MSc Eng.

It is assumed that the reader has a basic knowledge in the areas of statistics.

Angelos Daglaroglou - s202915

Dagla

.....

*Signature*

13/02/2023

.....

*Date*

## Abstract

Data-driven prediction of maintenance planning can have a twofold benefit. Both for the company that provides spare parts for the maintenance and for the customer that has a maintenance scheduled. In maritime industry in particular, since the customer's vessels operate worldwide, scheduling is of even greater importance. When a vessel is under scheduled maintenance, the parts have to be at the right place at the right time. This means that when the order for the parts is made, the parts need to be on the shelf, ready for delivery. Predicting the customer's demands, assists in inventory management and reduces the risk of stockouts, from the spare part provider's perspective. At the same time, it ensures that the vessel will have the spare parts delivered on time minimizing the risk for the vessel to stay off-hire, from the vessel-owner's perspective.

Predicting spare parts demand with classic statistical methods is very challenging due to non-normal characteristics that spare part demand possesses. In this study, seven machine learning classification models were tested and evaluated, including a Random Forest, an XGBoost classifier and a Neural Network. Those three seem to have promising potential. These methods were improved by optimizing the thresholds, balancing the weights of the classes and by tuning the hyperparameters.

The Random Forest is the model that achieved the best score, with the Neural Network, however, being a very good alternative.

## Acknowledgements

**Guido Cantelmo**, Assistant professor, DTU
For the guidance, close supervision of the thesis and the useful ideas

**Filipe Rodrigues**, Associate Professor, DTU
For the general direction of the project and his helpful feedback

**Henrik Bruun**, Manager in Customer intelligence & digitalization, MAN Energy Solutions
For the initial idea and assistance whenever it was needed

**Alexander Bartmann**, Sales manager, MAN Energy Solutions SE
For providing the data and useful insight on the domain

**Friends and family**
Especially Anastasia, for her support during the whole time of working on the project

# Contents

# 1 Introduction

With an increased pressure on marine traffic (regulations, more goods being transported, increasing fuel prices, green transition), higher effectivity is needed. The increased availability of data, allows companies to use it, analyze it and make predictions based on it. The advantages of data-driven analysis are known and used in maritime industry for more than a decade [1]. However, the ship management companies, as well as the ship owners, are still not in complete understanding of all the potential that the use of data can offer in maintenance prediction of their vessels. The main reason is that spare parts costs for the vessels are seemingly low compared to the rest of the vessels expenses, like crew salaries and fuel costs. Optimization in operating costs of the latter areas seem to the ship managers a more sensible thing to do.

However, this is not always the case, since delays in maintenance might cause loss of revenue. *MAN ES PrimeServ* is a company that provides spare parts for engines' maintenance for the global maritime industry. The procedure that is followed after a vessel's maintenance is scheduled, might be time consuming. A vessel's engine that is under scheduled repair needs spare parts that are ordered from companies, like MAN. When the order is made, the company often needs to source the parts, receive them for quality control and then send them to the vessel's location. This process might take time, leading to delays. Having delays means that the ship gets off-hire or risks breakdown, which leads to money loss for the customers. In order to serve customers more efficiently, the customers are to a large degree expecting fast and efficient service. A way to meet these demands is for MAN ES PrimeServ to predict the need for spare parts for maintenance (*overhaul*). Predicting the maintenance schedule of the customers will allow MAN to be prepared for the order intake for an upcoming period. This way the parts will already be in the company's warehouse ready for shipping, therefore minimizing the risk of delays.

This study's goal is to use data from customers, that together with the data that MAN has already available will serve the following purposes:

1. Budgeting for the customers: By being able to budget (i.e., for 1 year), larger orders can be placed.
2. Just in time: Ensuring that the right parts are at the right place, the right time.
3. Proactivity: By being able to predict customer needs, to serve customers more proactively.

MAN offers spare parts into bulks, called *Maintenance Kits (MKs)* (see section 4). When a vessel is in need of a scheduled maintenance, spare parts do not need to be ordered individually but they are rather ordered by one bundle that includes all the needed parts for the specific job. Therefore, the ship engineers change all the parts at the same time, no matter the amount of their wear. The kits include the same type and amount of parts.

The study's goal is predicting the jobs that can be served by maintenance kits and standard spare parts for the main engine of the vessel. Seven *Machine Learning models*, algorithms that can learn and adapt on data patterns, will be tested and evaluated in their default state. Then they will be compared, to provide an indication of their capability to predicting the spare parts' need for the customers' vessels for the upcoming year. Afterwards, the ones that seem to have the most potential will be tuned and potentially

improved through the methods of dimensionality reduction, class weighting and hyper-parameter tuning. The final goal will be to create a process that would be applicable to larger-scale fleets and therefore, bigger datasets.

The structure of the thesis, after this introduction, is the following: *Section 2* is the starting point, where the problem is set in a framework. The problem will be categorized and the various contributions from previous studies will be reviewed. *Section 3* describes the methods that are used in this case. They are selected based on previous studies described in the literature review and the nature of the problem. In *Section 4*, an overview of the actual data is given. After the initial analysis, the pre-processing of the data is described - the way the data is transformed to be used in a machine learning model. *Section 5* evaluates and compares the results of the tested models. *Section 6* is a critical discussion of the results and the methodology followed, while *section 7* is the conclusion and the final wrap-up of the study.

# 2 Literature review

Seeing the problem in its root, it is a warehousing management problem. In order for MAN to be able to serve customers, sufficient stock of spare parts is needed. Stock needs warehouse space to be stored. Warehouses and material handling systems are main elements in the goods flow and build the connection between the producers and the customers [2]. Warehouses have however, in the past, been constantly referred to as cost centres and rarely adding value [3]. Storage room costs, while goods stored do not produce any revenue. Warehousing management is a problem that companies have been trying to optimize. The pressure remains on managers to increase productivity and accuracy, reduce cost and inventory whilst improving customer service.

Companies attempt to achieve high-volume production and distribution using minimal inventories throughout the logistic chain that are to be delivered within short response times. For this reason techniques that improve *warehouse management* and *inventory management* are developed [4]. Warehouse management refers to the day-to-day operations of the warehouse, while the inventory management is the process of ordering, storing, using and selling a company's inventory. In a market that changes constantly though, optimizing the inventory can only offer limited value, so it is considered not to be enough.

A company that wants to be proactive does not only needs to optimize the current situation faced, but to also predict the forthcoming period. One way to achieve this, is by approaching it as an order intake problem. Forecasting techniques for order intake, like time series, have been used [5], but they have limitations. Spare parts forecasting exhibits non-normal characteristics [6]. Demand that has infrequent occurrences, low average volumes or highly variable volumes is characterized as non-normal, so it is challenging to approach it using forecasting techniques [7].

Another way for a company to be proactive, is predicting the maintenance schedule of the customers. Since the 1950s, when the first scientific approach to maintenance management emerged [8], maintenance activities are planned, not performed only when failure occurred. Having large amounts of available information on past maintenance, modern models can be used to predict upcoming maintenance. For this exact reason mining techniques have been developed to handle large amount of historical data [9] and cluster, classify, or predict, based on the available information in the data.

This type of data, as said, has non-normal characteristics. Machine learning models however, have the ability to read and understand this types of patterns. They have been tested in sporadic demand data (when the quantity demanded is zero for several periods) against traditional statistical methods, which were built for this purpose [10], and outperformed them. Also, they seem to perform well in cases of imbalanced data, when one class constitutes one small minority of the data [11]. These patterns are common in predictive maintenance problems. This is why this type of models have been tested in transportation maintenance scheduling in railways [12] and aviation [10].

While the maintenance prediction is a problem that is encountered often in literature, this study will approach it as a *binary classification* problem, where the question to be answered is if a maintenance will occur in the next period or not. This simplifies the process, which means more accurate, faster, and easier to interpret results. The models tested, as well as the methods to improve them, are the ones most commonly used in the industry and, after evaluated, they will be comprehensively compared throughout the study.

# 3 Methodology

In this section, the models, the metrics and model improvement methods will be described. Having a clear understanding of the challenges to be encountered, the dataset has initially been analyzed to find if characteristics commonly detected in maintenance prediction problems are indeed found in the dataset. The outcome of the preliminary analysis (section 4) indicates sparsity of the data, as well as imbalance in classes. The models, methods as well as metrics used in this study (their results given in section 5) are chosen, taking into consideration that it is a binary classification problem and that the data contains these traits.

## 3.1 Models

The models were chosen with two factors in mind. Some are strong classifiers that are widely used in industry to solve complicated prediction problems (e.g. XGBoost, Artificial Neural Networks). Others, are weaker learners that for some problems perform equally well with some of the stronger ones but are faster and tuned much easier than the former. Of course, in industry the availability of time is always a big factor, so this should be taken under consideration when comparing the results.

The algorithms for all the models have one process always identical. They are first *trained* and then *tested*. This happens by splitting the data into train set and test set. The train set is almost always the biggest of the two (usually around 70-75%) and is used so that the model can learn the pattern that the data follows; what values of the features (input columns) give which output (the prediction target). After the training, the model is tested on the test set. It reads the features, makes predictions and compares the predictions of the outputs to the actual outputs. The more that are identical, the better the model performs.

### 3.1.1 Logistic Regression

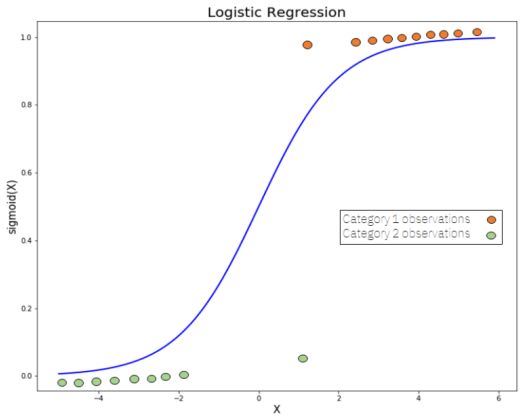



Figure 3.1: Logistic Regression on categorical data

Logistic regression is the simplest model and has been widely used in the past [13]. Nowadays, with more computational power available, it is easy to train more complicated models, that give much more accurate predictions at, practically, no extra computational cost. Therefore, it is usually used as a benchmark to evaluate the stronger classifiers.

Binary logistic regression (since the problem is a binary classification problem) is the counterpart of the linear regression, but in a classification context. It classifies the predictions in two categories (yes/no, 1/0, true/false). The linear regression assumes a linear connection between the input and the output. In simple terms, it is the line drawn in Euclidean space with equation:

$$ax + by = c. \tag{3.1}$$

The logistic regression however, is bound between $0$ and $1$ as shown in figure 3.1. This is achieved by applying the nonlinear logistic transformation:

$$\frac{1}{1 + e^{\chi}}. \tag{3.2}$$

The logistic regression uses the Maximum Likelihood Estimation (MLE) for parameter estimation, which is a probability. Every data point is assigned one corresponding probability. If this probability is above 0.5, the output is classified as 1, else it is classified as 0.

### 3.1.2 Bernoulli Naive Bayes

Naive Bayes classifiers are a group of algorithms that are simple but efficient. They assume that there is no relation among the features of a class. Even though this assumption is very poor, these classifiers tend to perform very well under this assumption [14]. Furthermore, in some cases, they seem to outperform more capable classifiers when the size of the data is small [15], as is the situation in this study. Bernoulli Naive Bayes belongs to this group of algorithms.

The Naive Bayes models are based on Bayes theorem:

$$P\left(y \mid X\right) = \frac{P\left(X \mid y\right) P(y)}{P(X)}, \tag{3.3}$$

on which the 'naive' assumption (that all the features are mutually independent) is applied. Finally, the decision formula used in Bernoulli Naive Bayes is the following:

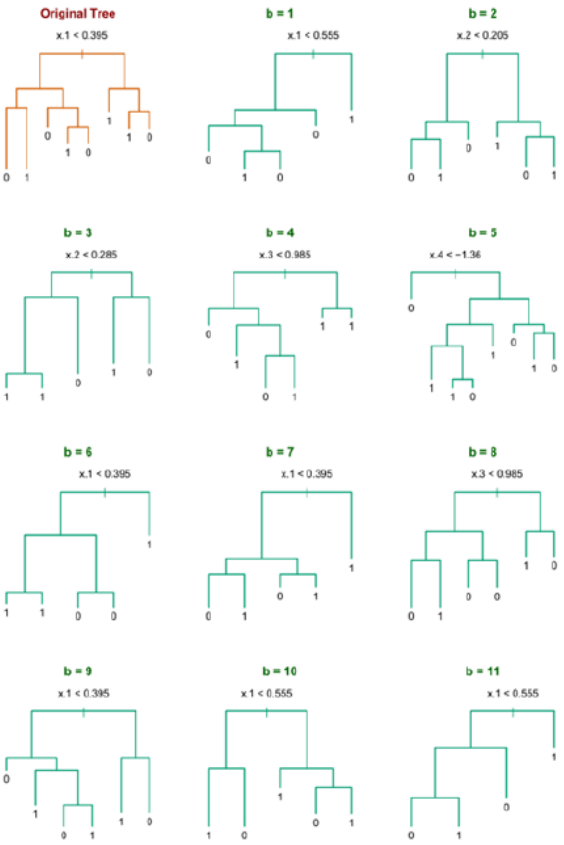
$$P\left(x_i \mid y\right) = P(i \mid y)x_i + \left(1 - P(i \mid y)\right)\left(1 - x_i\right) \tag{3.4}$$

According to this formula, the input feature needs to be binary and follow the Bernoulli distribution, so any continuous variables should be excluded when trining this model.

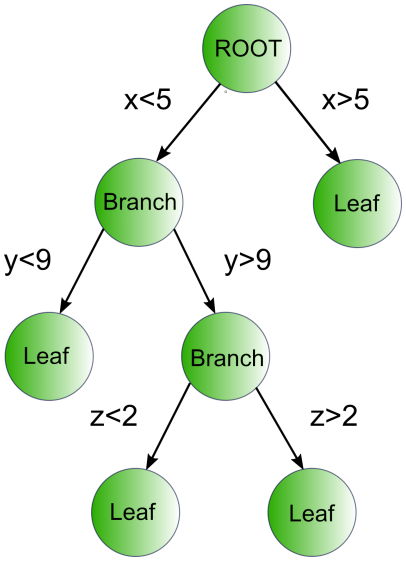
### 3.1.3 Random Forest

Random forest is one of the algorithms that utilizes decision trees for classification (or regression, depending on the problem). Decision trees are simple, fast but weak learners. The way that a tree works is by splitting the data into dense regions and sparse regions. Each tree consists of nodes as seen in figure 3.2b; in each one of which the data is split in the most efficient way. The feature as well as the value on which the split will be made upon is chosen. The choice is evaluated by a metric (gini index, entropy or information gain) and then the split is made. Then, the same process is repeated on the newly created branches and split again until the data is sufficiently homogeneous.

The Random forest is based upon an ensemble of decision trees. The main concept is to train many weak learners, *bag* them (figure 3.2a) and make a final decision according to majority vote of those learners. *Bagging* is done when the algorithm constructs a number of trees and averages the resulting predictions. Since each decision tree has low bias and high variance, averaging them aims to reduce variance. Random forest improves the concept of bagging trees by *decorrelating* the trees and reduce variance further. Trees are very sensitive to changes of the dataset. This means that by changing the data on which




(a) Bagging trees




(b) Example of a decision tree

they are trained the result will differ each time. Random forest takes advantage of this by allowing each individual tree to randomly sample from the dataset with replacement. This way, two things are achieved; (1) the trees are uncorrelated and (2) the samples taken differ significantly from tree to tree since the available data remains the same.

Another important characteristic of the Random Forest is the feature availability. A decision tree can make a split based upon any feature available in the dataset. In random forest, there are fewer available features for each tree to choose from, therefore even lower correlation among the trees. This low correlation is, eventually, really important because it leads to a classifier that is very difficult to overfit, and therefore easy to tune and train. *Overfitting* is the not wanted situation that occurs when the model is trained to follow too closely the data it was trained on. When this happens, the predictions will follow the pattern that was learned too strictly and the model will not be able to generalize when unseen data is given.

### 3.1.4 AdaBoost

Adaptive Boosting (AdaBoost), like Random forest classifier, utilizes an ensemble of decision trees. However, instead of bagging, it uses *boosting* on a set of weak classifiers (small trees), which are grown in an adaptive manner (hence, they are not independent), to create better *stumps* (decision tree with one split) and to remove bias. Except the first classifier, every consequent classifier is grown from the previous one.

Initially, a stump is grown. After this first stump is grown, its performance is evaluated by calculating the error and the misclassified observations are given a higher weight.

After the weights are updated, the next stump is grown, but it will get a weighted input that emphasises on the misclassificatons and will adapt, to include more on them in the correct class. The same steps are followed to each iteration and the new weights are added to the ones of the previous tree. The final estimator is a weighted average of all the stumps, which together form one strong model with boosting.

While every decision tree of the algorithm has big bias, boosting decreases the bias, since the weights of the observations are not equal. Also, since it is an ensemble of trees, it reduces the variance. Finally, even though it is an old algorithm, it is still a very popular choice for binary classification problems [16].

### 3.1.5 Gradient Boosting Classifier

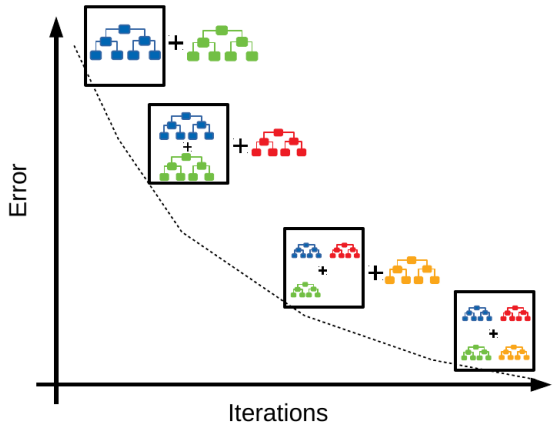



Figure 3.3: Gradient Boosting algorithm's iterative process

Gradient Boosting is an algorithm that, as AdaBoost, uses boosting technique [17] to create a strong model by combining many weak classifiers. The main difference is that instead of adjusting the old classifiers, they are kept intact and new ones are created.

The algorithm starts from one naive leaf, which, in a binary problem, calculates the logarithm of the ratio of each class and then converts it to a probability, just like a decision tree does. From this probability, the *residual* (a pseudo-residual) between the observed and the predicted value is calculated and then used to create the next tree. These residuals become the values of the leaves. However, since the tree will not be fully grown, one leaf might contain multiple values of pseudo-residuals, thing that goes against the way a tree works. Therefore, the leaf is transformed by using the sum of the residuals and the previous probabilities (log-odds). Finally, the old tree is added to the new one multiplied by a learning rate. The learning rate is a small number (the default value is 0.1), which shows the contribution of each new tree to the previous.

As seen, during those steps, new trees are created, by selecting new features. For this, one new term is introduced, the *loss function*. The objective of Gradient Boosting classifiers is to minimize the loss, meaning the difference between the actual class value of the input sample and the predicted class value.

### 3.1.6 XGBoost

The final model used in this project that is based on decision trees is XGBoost (EXtreme Boosting) and works similarly to the Gradient Boosting. Initially, a leaf node is created from the initial predictions and then a pseudo-residual is calculated for each sample. Next, a decision tree is built based on these residuals and it is added to the previous one.

The difference is however in the efficiency that the algorithm does it, by introducing techniques that improve the learning process. Each newly created tree is split by measuring the node's Gain (similar to the log-odds of Gradient Boosting), which is dependant on its leaf nodes' *similarity scores*. Similarity score is the analogous of log-odds of the Gradient boosting, but for XGBoost. This score introduces the $\lambda$ regularization parameter, which limits the algorithm's ability to learn. Boosting techniques are prone to overfitting this is why it is important to limit the model's ability to learn. If the new tree is fully grown, there is a chance that it will learn the training set too well and will have limited capability to generalize and predict the output of another set. This problem can be limited by 'pruning' the tree. Increasing the $\lambda$ parameter, can decrease the gain of each leaf and the branch's ability to learn. XGBoost also introduces the $\gamma$ parameter which does the same, but in the opposite way. If a node exceeds a given threshold of gain, then its leaves are removed from the model and limit the model's chances of overfitting. Finally, like in Gradient boosting there is the learning rate, in XGBoost there is the *eta* parameter which dictates the newly created tree's contribution to the previous.

All the above show that there is a trade-off between how slow will the model be allowed to learn and the training time. The faster the learning capability of the model, the faster it will train, but more prone to overfitting it will be. Allowing less learning rate, means that more estimators will be needed, thus more training time. This is why, especially in XGBoost, tuning the model's parameters and reading its behaviour in those changes is an important part of the training process that should not be overlooked.

### 3.1.7 Artificial Neural Network

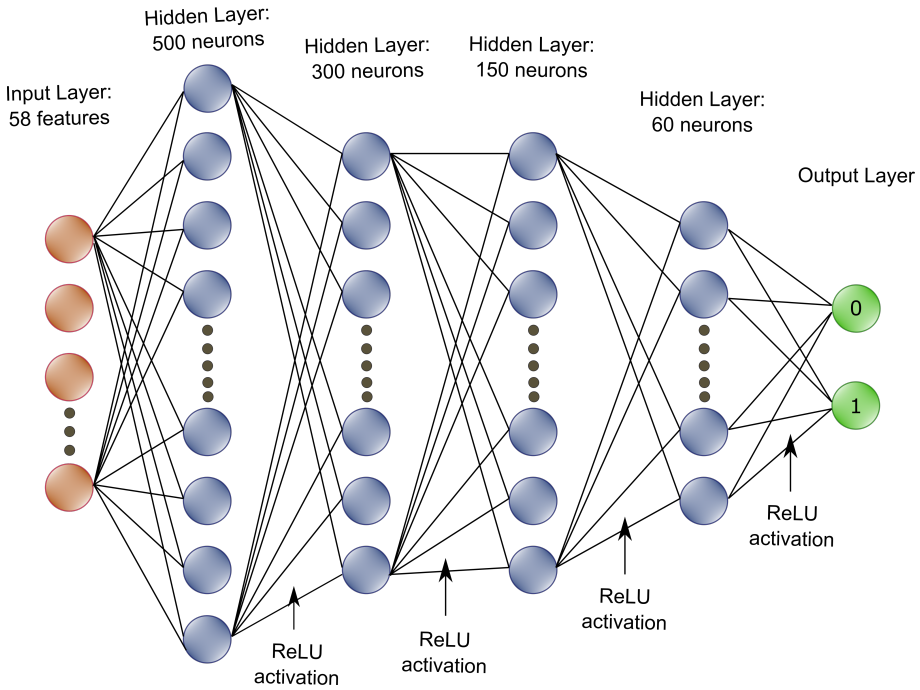



Figure 3.4: Structure of the Deep Neural Network used in this study

The final model that is used in this study is a Feed-Forward Artificial Neural Network. Artificial Neural networks (ANNs) are flexible and strong models that have a built that tries to imitate the human brain neurons and are able to fit the most complicated problems. They consist of one *input layer*, some *hidden layers* that process the data and an *output layer* that gives the results. Each layer consists of multiple *neurons* that operate as weights to the values (figure 3.4).

The input layer is the first layer that takes the features of the dataset in and therefore consists of as many neurons as features available. The output of this layer becomes the input of the first hidden layer. The neurons of the hidden layer process the data, change their weights and transfer the output to the next layer. This continues until the output layer, which gives the final results. The layers are *fully connected*, which means that every output of every neuron becomes input for every neuron of the next layer. Each time a neuron processes a new value, its weight transforms to fit the importance of the new information to the actual result and then it is passed through to all the neurons of the next layer. The importance of this trait should be noted, as the biggest advantage of a neural network, in comparison to the other models. This weight adjusting can, up to a degree, replace the feature selection process (see section 3.3.2), since the variables that are irrelevant will not affect the weights and the model will be more robust to that noise.

As in the previous models, the data is split into a train and a test set. One portion of the train set however, is used as *validation set* during the training of the model. Training of neural networks occurs in sessions, called *epochs*. One epoch is one full cycle of training all the data. After each epoch the weights of the neural network have been adjusted and predictions have been made. These predictions are evaluated by the model on the validation set, so there is an indication of the performance. This information then, is used in the next epoch, when the model is trained again and the weights are then re-adjusted.

Since a neural network is such a complicated and capable predictor, it tends to overfit. There are techniques that reduce this risk, like *dropout* and *early stopping*. With dropout a portion of the layer's outputs are hidden randomly, so the observations become noisy and therefore more difficult to be followed too closely. Early stopping allows the training process to stop automatically when no improvement is noticed after a specific number of epochs. This way, the training can be set to last many epochs, but early stopping will stop the process before that number is reached, if there is no improvement for for a number of epochs, and therefore before it starts overfitting.

However, still much care is needed when tuning its hyperparameters, reading and explaining the results and choosing the correct components. The most essential components used in every neural network are the activation function, the loss function and the optimizer.

**Activation function**

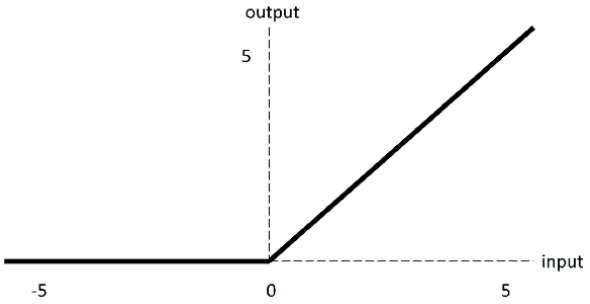



Figure 3.5: Rectified Linear Unit (ReLU) function

The transformations that are applied on each neuron, are linear (not much different than a logistic regression); this is why adding an activation function between the layers is needed. The simple but very efficient ReLU (Rectified Linear Unit) activation function (figure 3.5) introduces non-linearity to the model. The way it works is by simply allowing (or not)

information passing from one neuron to the next. If the output of a neuron is negative, information does not pass to the next, but if it is positive it passes intact. Main advantages of this function are:

- it is more computationally efficient, since it only activates the neurons with positive input.
- Activates intensely the neurons for a big (compared other activation functions like tanh and sigmoid) interval.

These are also the reasons why it is the most commonly used activation function in neural networks.

**Loss function**
As in the previous classifiers, neural networks need an objective. This is minimizing the *loss function* - the deviation of the prediction away from the ground truth label. Binary cross entropy is used for binary classification problems. It compares the predicted class to the actual class and penalizes the misclassified predictions' probabilities based on the distance of the actual value. The final loss formula as seen below, in simple terms, is the average of the logarithmic transformation of these probabilities.

$$\text{Log loss } = \frac{1}{N}\sum_{i=1}^{N} - (y_i{}^* \log(p_i) + (1 - y_i) * \log(1 - p_i)) \tag{3.5}$$

**Optimizer**
The optimizer is an algorithm used to change the attributes of the neural network such as weights and learning rate in order to minimize the loss function. Minimizing the loss function is a stochastic process, so the purpose of this algorithm is to find a value close to the minimum, efficiently. Adam (Adaptive Moment Estimation) optimizer combines the strengths of other optimizers. It approaches very fast the global minimum, but it also reduces learning rate when close to the minimum, so that it does not miss it. In addition to that, the algorithm has faster running time, low memory requirements, and requires less tuning than any other optimization algorithm [18].

## 3.2 Methods for classification model evaluation

After training the models, as said in the previous section, they are tested against the test set. During the testing the predicted output is compared to the actual output of the test set. This section is about the methods that are used to illustrate the results of this comparison: the confusion matrix, the metrics for classification and the ROC and Precision-Recall curves.

### 3.2.1 Confusion matrix

In a classification problem the most intuitive metric to be used is the confusion matrix. It is essentially a table built to show what is the true class of an observation and what class is it predicted to be, by the model. Based on that, the observations are grouped, summed and compiled in a square table, that consists of a number of columns and rows equal to the number of classes.

Its simplest form is met in a binary classification problem, a table that consists of two columns and two rows. The labels commonly used for each type of predictions are:

- **True Positives (TP)**: Observations correctly classified as positive
- **True Negatives (TN)**: Observations correctly classified as negative
- **False Positives (FP)**: Observations falsely classified as positive
- **False Negatives (FN)**: Observations falsely classified as negative

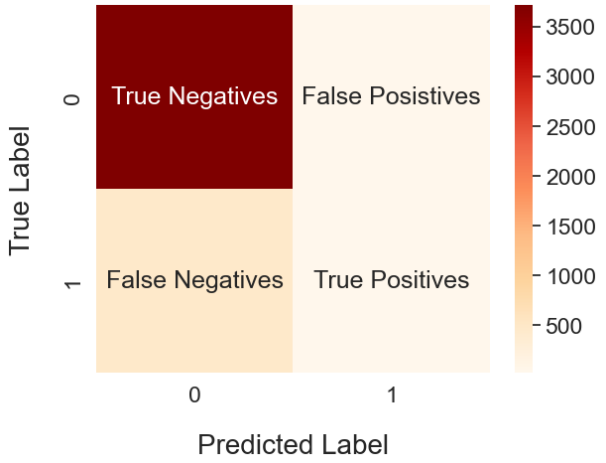



Figure 3.6: Example of confusion matrix for binary classification

In sklearn, the library of algorithms used in this project, the columns represent the predicted values, while the rows represent the actual ones. Simply put, the main diagonal (top left to bottom right) are the correct predictions and the rest are the wrong.

### 3.2.2 Metrics

Understanding of the confusion matrix labels, is needed for the metrics that the machine learning models give as output. These scores range from 0 to 1 and are used to evaluate compare the models' performance. The closer to 1, the better.

The first and most straightforward metric is accuracy. It can be calculated as the sum of the correct predictions divided by the sum of the total observations:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{FP} + \text{FN} + \text{TP} + \text{TN}} \tag{3.6}$$

It is the most intuitive and commonly used metric for a classification problem. However, it can be misleading. In an imbalanced classes problem a high accuracy can be achieved even with a no skill predictor. For example, if the output has 10 true values and 90 false. A model that predicts always false will have a 0.9 accuracy but, in reality, it will be predicting nothing meaningful.

Usually, in an imbalanced dataset the focus is on detecting the observations that belong to the *minority class*. The class that is most rarely met in the dataset. Examples of these type of problem might range from medical (e.g. disease diagnosis [19]) to fraud detection, in which there are few true cases, but it is important to be predicted correctly.

There are four metrics focusing on true labels, simple to calculate, given the values from the confusion matrix:

1. **Precision**: Observations of the positive class predicted correctly.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{3.7}$$

2. **Recall**: Observations that were predicted as positive and were actually positive.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3.8}$$

3. **F1 score**: The harmonic mean between precision and recall (also seen as f-measure).

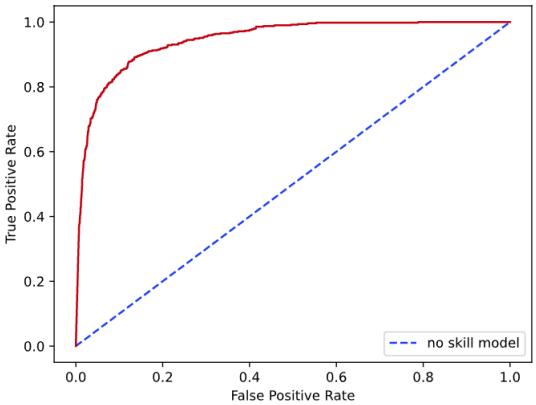
$$\boldsymbol{F}\text{ - measure} = \frac{2^{*}\text{ Recall*Precision}}{\text{Recall} + \text{Precision}} \tag{3.9}$$

4. **Youden's J statistic**: Observations that were predicted as positive and were actually positive (recall) added with Observations that were predicted as negative and were actually negative (True negative rate).

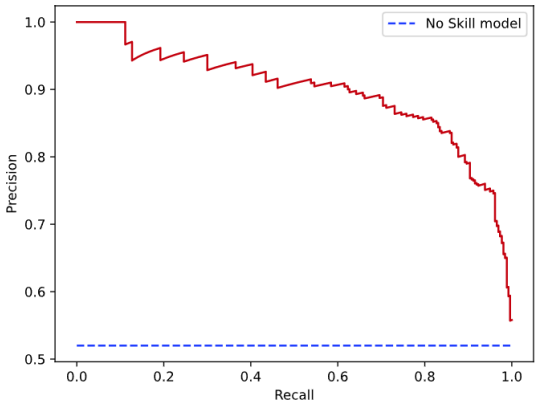
$$\text{Youden's J statistic} = \frac{\text{TP}}{\text{TP} + \text{FN}} + \frac{\text{TN}}{\text{TN} + \text{FP}} - 1 \tag{3.10}$$

It is "generally not to be recommended to only use one statistic" [20] when comparing models. Comparing two models, one with high accuracy and one with high recall is difficult. F1 score and J score combine these two metrics into one and they are a safe way to measure how well a model performs. However, since Youden's statistic also rewards the negative class, F1 score is better suited in evaluation of the results of class imbalance problems.

### 3.2.3 ROC and Precision-Recall curves




(a) ROC curve of a model compared to a no skill classifier




(b) Precision-Recall curve of a model compared to a no skill classifier

Finally, the most comprehensive methods of evaluation are the ROC curve (Receiver Operating Characteristic Curve) and the precision-recall curve. These graphs show how the models perform in a more analytical way.

Classification models, as said previously, return probabilities. In their default state, if the predicted probability of an observation is above 0.5, the observation is classified as

positive, else it is classified as negative. This happens because default threshold of a model is 0.5. These curves show how the classifiers perform in all the thresholds.

Specifically the ROC curve is the graph that for every threshold, plots the True Positive Rate (defined the same as the recall, seen in the previous section) in one axis and on the other the False Positive Rate defined as:

$$\text{False Positive rate } = \frac{\text{FP}}{\text{FP} + \text{TN}} \tag{3.11}$$

What is expected to see, is a plot that starts from point (0,0) and ends at (1,1). In a classifier the performs well (as in figure 3.7a), this curve is desired to travel close to point (0,1). This means, that there is some point (one threshold) that this model gives high True Positive value and low False Positive rate. At this point there are many actual true observations classified as true and only few negative actual observations classified (falsely) as true. In this type of plot, a naive classifier will have a plot travelling close to the diagonal from point (0,0) to (1,1), meaning that in every case it predicts randomly almost half observations correctly positive and half incorrectly.

Again however, when evaluating models, it is not recommended to always trust one metric or graph. Predictors in imbalanced datasets tend to demonstrate good performance when visualised in a ROC curve. This is why a precision-recall curve might give a better overview (figure 3.7b). The idea is similar to the previous one. One axis plots precision and the other the recall. Desired outcome is a plot travelling from point (1,[positive class ratio]) to (0,1) and passes closely to (1,1) in between. That point is when precision and recall take values close to their maximum. A naive classifier however (one that always predicts the positive class), will have a plot that travels almost horizontally, having a constant precision that approaches the ratio of the positive class.

Comparing two curves might sometimes be challenging. *AUC (Area Under Curve)*, can be calculated to give a score to a model across all thresholds. This score can be used to compare two binary classification problems directly. The AUC is the area between the x-axis and the curve and is determined simply by calculating the integral of the curve.

## 3.3 Improving the models

Machine learning models are widely used in industry and in research. Off the shelf solutions are available and make their application easier. The models available in digital libraries are built in a way, so that they can fit in many situations. This is a good perk indeed, but it does not mean that they can give an optimal solution to each individual problem. For this reason, there are techniques used to improve, to tune and make the models fit in the problem at hand.

### 3.3.1 Threshold movement

Machine learning models are predicting, as said previously, class probabilities. The decision for translating the predicted probability to class label, is controlled by the model's threshold. When it is above 0.5 it is given a class label of 1, else 0. This default value, does not always work optimally for various reasons, for example when the class distribution of the data is very skewed.

The simplest, yet very efficient method to improve a model's performance is by adjusting this decision threshold. This solution is proven to benefit very much the predictors when dealing with imbalanced data [21], since they tend to overpredict the majority class.

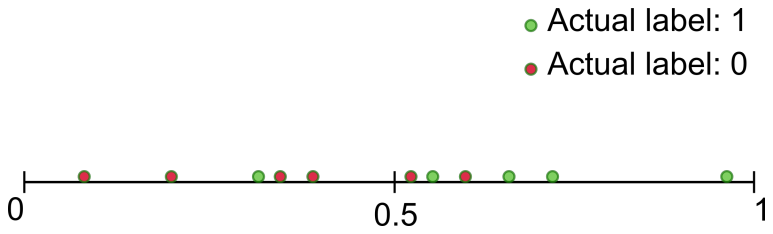



Figure 3.8: Predicted labels on threshold 0.5


Figure 3.9: Predicted labels on threshold 0.6

An example of how can moving the threshold improve a model is illustrated in figures 3.8 and 3.9. The figures show the actual classes of the observations; green for class 1, red for class 0. The predictions that the models do are based on the threshold. Every observation on the left of the threshold is predicted as class 0, while everything on the right as class 1. The axis represents the predicted probability that every observation is given, from lowest (left) to highest (right).

In figure 3.8 the threshold is at 0.5. Everything on the left is predicted as 0, meaning there are 4 True Negative and 1 False Negative value. Everything on the right is predicted as 1, meaning 3 True Positive and 3 False Positive values. Calculating the accuracy as defined in section 3.2.2, the result is 0.73.

Moving the threshold to a higher value, means that fewer predictions are allowed to be accepted as 1; only those that have higher probability. This movement leads some observations previously predicted as positive, now being predicted as negative. Figure 3.9 illustrates how the predictions of the model are affected. Again, following the same logic as above, it is seen that there are 7 True Negative, 1 False Positive and 3 True Positive predictions. This change of the threshold increased the accuracy in this example from 0.73 to 0.82.

Of course depending on the problem there are different things to consider. First of all this is a trade-off. Allowing more classes to be predicted as one label, means that some classes of the other label are also misclassified. Also, which metric needs to be maximized. In this example, accuracy indeed increased, but the f1 score for class 1 was reduced. If trying to get a good accuracy is the goal, then this movement helped. If the problem focuses on predicting the True labels, such a movement would not be advisable.

### 3.3.2 Curse of dimensionality - Dimentionality reduction

The features or variables of a dataset, (or, simply put, the columns) are referred to as dimensions. When a dataset consists of many features, machine learning models might find it difficult to make good predictions. This is what in machine learning is called *curse of dimensionality*. If it could be visualized in a geometrical space, a dataset's features would represent the dimensions of the space, while the rows would be the datapoints. This would lead to a sparse space, where the samples would be non-representative. In such cases, overfitting would be very probable for the models, since they would memorize the train set, instead of learning patterns. Even more, in the extreme scenario that the data contains more columns than rows, the features would not be linearly independent and even some models would not work at all.

Other than that, more features might mean more *noise* - more irrelevant information that a model is built upon and follows. The model will predict a pattern that contains useless information. Next, with more columns, the problem becomes more complex and more difficult for a model to learn. This results in higher tuning effort needed from the data scientist's side and more training time for the model.

In a problem with many categorical variables, like the one described in this study, it is expected to have increased number of features. Every distinct value that a categorical variable takes, results to increase of the number of columns inputted in the model. When having to do with hundreds of vessels, this number is expected to rise to a great number.

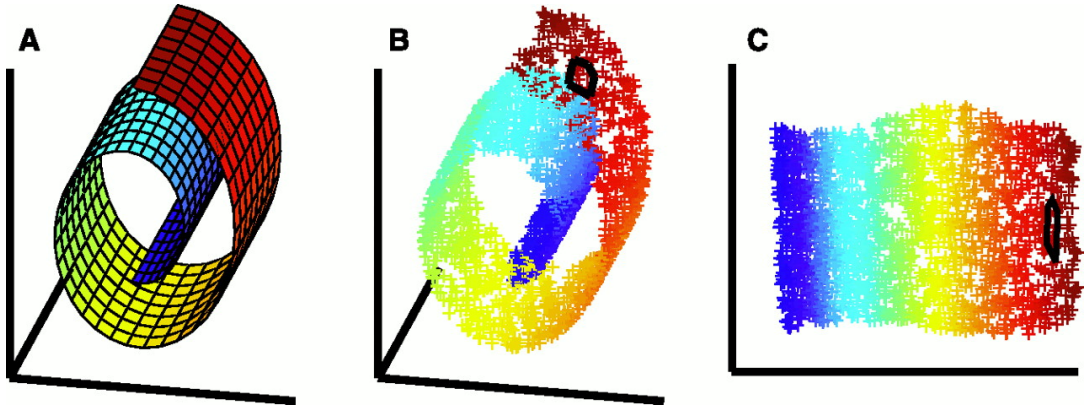



Figure 3.10: Visual illustration of dimensionality reduction with LLE

Dimensionality reduction refers to techniques that reduce the numbers of features. This can be achieved by using methods of different nature. The first and most obvious is *feature selection*. Selecting which variables are more important and more relevant to the prediction, is important. This way noise and complexity is reduced and the model can produce safer results. Second technique is by doing *matrix decomposition*. Essentially, this way the dataset is split into multiple smaller parts, ideally without reducing its variance. Finally, manifold learning is used to project the data on a space with fewer dimensions.

In this study a manifold learning technique will be used; Locally Linear Embeddings (LLE). LLE is an unsupervised learning method (does not need train and test set) that seeks a lower-dimensional projection of the data which preserves distances within local neighborhoods. LLE attempts to discover nonlinear structure in high dimensional data by exploiting the local symmetries of linear reconstructions [22].

The algorithm of LLE first finds a number (provided to the model) of the nearest neighbors. Next, approximates each data vector as weighted linear combination of the nearest neighbors. Finally, it uses these weights to reconstruct a lower dimension vectors in which each data point is again described as linear combination of its neighbors. This three-step process is visualized in figure 3.10.

The benefits of using LLE is that in contrast to other dimensionality reduction techniques, it describes each datapoint in relation to its neighbors. This means that a possible feature correlation is transferred to the lower dimensional data. Also, since LLE tends to accumulate sparse matrices, it is more efficient than the other algorithms in terms of computational space and time.

In this study, and after some tries have been made, only the variables that describe the vessel will be embedded (vessel names and cylinders), while the rest will stay intact. Since

maintenance kit's number stays the same and the years along which the data is trained, will hardly be increased, there is no need to include them in this process. The selected columns will be reduced to just four variables and the decision will be made based on the 30 closest neighbours.

### 3.3.3 Class weighting

Applying class weights in classes is another way to deal with imbalanced data. This is a method applied either to multiclass or binary classification problems. As stated previously, in imbalanced data, especially in binary classification, the target is usually to detect the minority class. Detecting if a patience is positive to some disease, fraud detection or spam mail detection are some examples. In those types of problems the minority class might be as low as 1%, but it is of high importance to predict them. Of course threshold movement might be of help, but this is usually not enough.

Applying weights to the classes is a way to penalize the misclassification of the minority class. In their default state, machine learning algorithms give equal weights to classes. By applying higher weights to the minority class (or lower to the majority class) the model will increase its bias towards the class with the highest weight.

The most obvious way to do it is by increasing the minority class weight correspondingly to its ratio to the majority class. So, if the minority class constitutes the 10% of the dataset, giving a weight 9 times higher that the majority class' weight, the bias of the model will get more balanced.

What should be taken into consideration though is that increasing the bias towards one class, it will mean that there will be misclassifications for the other class too. So, again, it is a trade-off that should be taken into consideration when calibrating the models.

### 3.3.4 Hyperparameter tuning

Machine learning models have *parameters*, as well as *hyperparameters*. Parameters are tuned by the model when fitting the data. Example of a parameter is the weights of the neurons in a Neural Network. Hyperparameters on the other hand, are tuned manually. Similarly to the threshold default value, the hyperparameters are set on values that make the training quick and competent for a wide range of problems. However, these values need to be tuned, in order to fit the individual dataset.

Table 3.1: Example of hyperparameter values to be tested for a Random Forest

| Hyperparameter | Value 1 | Value 2 | Value 3 |
|---|---|---|---|
| n_estimators | 500 | 1000 | 1500 |
| max_depth | None | 10 | 20 |
| min_samples_split | 2 | 4 | 6 |

There is usually no way to know the correct values of the hyperparameters, so they have to be tested and compared. The way it is done, is by trying all the possible combinations for the values of the hyperparameters given. GridSearchCV is a tool that handles this work. For example, if the values of table 3.1 are given, 27 different combinations will be tested. GridSearchCV will use these combinations to train and fit 27 different models, compare their performance and return the best-performing model.

This process is done with *cross-validation*. The train set is split into into distinct sets, called folds. One of these folds is used as validation set, similar to what happens in the Neural Network. The models using the given hyperparameters will be validated against the validation test and a score will be returned. Then the train set will be trained again and

tested with another fold. The model again, with the same values of the hyperparameters will be validated and a score will be returned. This process is repeated as many times as the number of the folds. Then, new values will be given to the hyperparameters and they will be validated again, as the previous values, for all the folds. The hyperparameter values that returned the best score overall, can be selected as the best to use for the model.

Even though this procedure is not actually manual, adding more values or more hyperparameters to be tested, increases the training time exponentially. The challenge is up to the practitioner to know the hyperparameters that affect the model the most and the intervals that the values should range.

# 4 Preliminary data analysis

*MAN Energy Solutions* designs and produces engines. A big part of the company's services is the after-sales. Every engine, especially the ones used in ships, is designed with a big lifespan. However, specific parts are designed to wear in order to protect other, more important parts of the engine. These parts need to be changed after the engine has run a specific amount of time, so it is crucial that the company is able to serve customers using MAN engines efficiently and effectively. In order to simplify the process, MAN has proposed the solution of maintenance kits (MKs) - bundles of items designed for a specific maintenance type of one cylinder of the engine. Some parts of the engine's cylinders tend to wear almost simultaneously, for example, cylinder cover parts. The cylinder cover is the top part of the cylinder and its purpose is to offer complete sealing, when extreme pressures are applied. Its wearable parts can be for example various O-rings and packing rings (circular-shaped gaskets used to seal gaps). When the cover is maintained, all its wearable items are worn up to a degree. Of course, some less than others, but since maintaining a cylinder is time consuming, it is to the engineer's benefit to change all the parts associated with this job. A maintenance kit is designed to contain all the spare parts needed for the corresponding overhaul.

The dataset that is used in this study contains maintenance information of the fleet of one of MAN's most loyal customers. Each overhaul occurred in the past can be assigned to maintenance kits (or in a few cases, just standard spare parts) used by the vessel at that time. A machine learning algorithm can be trained using this information and then predict the maintenance kits' demand for this customer in a consequent period of time. Knowing the demand for these kits, makes it trivial to calculate the amount and type of spare parts needed for this period as shown in figure 4.1. It is notable that since engines differentiate in size and power, so do the parts. However, the amount of spare parts remains the same regardless the type of the engine.


Figure 4.1: Implementation of maintenance kits in after-sales

## 4.1 Dataset description

Before starting to work on the data it is always meaningful to have an overview first. This way it can be clear from the beginning what type of problem it is, how can it be approached and furthermore to see if it contains bad data. Especially in machine learning, some observations might not be wrong, but still might prove confusing for the models and making them difficult to train.

The dataset consists of 7 variables, mostly categorical, and 3383 observations, including inspections, cleaning and calibrations (see table A.1). Since these are jobs that do not

have to do with purchasing of parts, they need to be excluded. Also, including years that have too few observations or MKs that occur really rarely should be avoided. This happens, because if there is not enough information, it might confuse the training process. Figure 4.2, makes it easier to anticipate the unavailability of the data in some cases. For example, Alpha lubricator is only seen in 2 cases in the dataset, while it is known that these are among the most common overhauls. Knowing this, trying to predict their demand, would not have good results. For that reason, these observations are removed and will not be predicted

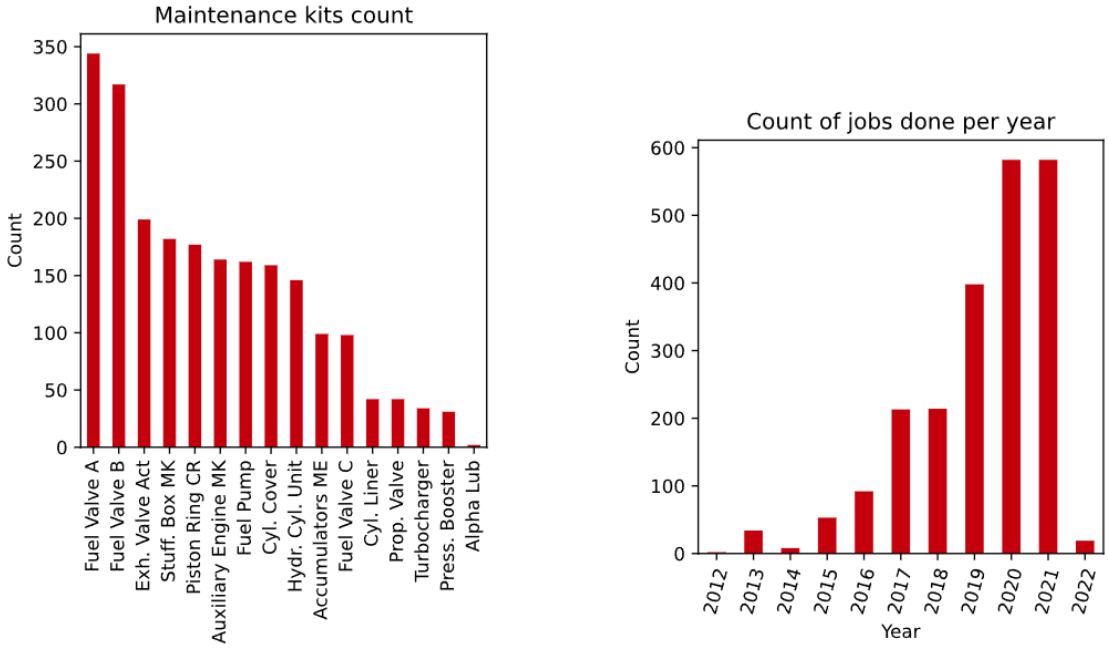



Figure 4.2: Count of overhauls by type (Left) and count of total overhauls per year (Right)

Another important step for data validation is checking for outliers in the continuous variables There might be occasions that some values are extremely high or low. This can happen either by wrong import of the data (some extra digits typed when typing) or because there are indeed extreme values, for example an old vessel that has run for many more hours than the rest. This is the situation in the box plot in figure 4.3. There are three observations between 300.000 and 360.000 that need to be validated. According to the other observations of this vessel (which show values close to 40.000 running hours), it seems that they are mistakenly imported figures. In this case, they are deleted and not used in the models.

Having cleared the data, it should now be known what is the target of the predictions. There are 14 overhauls for each of the 28 of the customer's vessels' main engines to be predicted A.3. The engines contain from 6 to 9 cylinders. What needs to be predicted is which (if any) overhauls have been made in each cylinder in each year.

## 4.2 Data pre-processing

After the dataset has been analyzed and any potential weaknesses have been found, it can be pre-processed in a way that it can be used in model training.

As said, the dataset contains all the overhauls that the vessels have been through. The

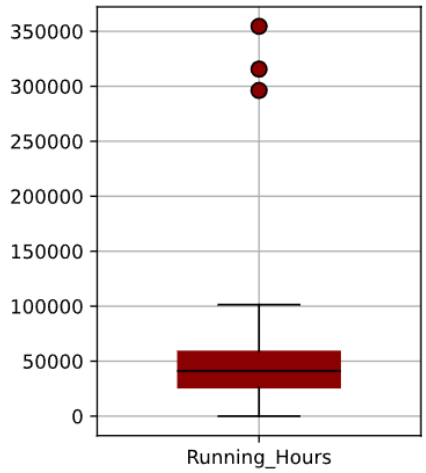



Figure 4.3: Boxplot of vessels' running hours

objective of the study is to predict the amount and the type of overhauls that will be needed in the next year by the vessels. At this point an assumption needs to be made. Spare parts are not changed more often than 8000 hours (approximately 333 days, if the vessel runs without stop). Since a vessel needs to stay docked when arriving to a port for some period, to load and unload goods, it is safe to assume that the life of a spare part will extend to more than a year after installing it. This is the assumption that allows the problem to be shaped as binary classification.

Having made the previous assumption it is inductive that a vessel will need each type of maintenance up to once per year per cylinder (since each MK is designed for one cylinder's overhaul). Since the dataset contains information about the cylinder that the overhaul is taking place on, the problem can be shaped into a binary classification problem, with each observation referring to one overhaul, of one cylinder of one vessel on a specific year (see appendix Table A.1). The column named done is the output and, since all the overhauls in the data have already been done, takes value 1 for all the observations.

In order to have binary classification, the dataset needs to be expanded with all the possible combinations of overhauls, that have not occurred. This means that new observations for each vessel, for each cylinder and each year are added. The only exception in this process is the Fuel Valve MKs. Each one of them is applied on one of the three injectors of the cylinder. In order to be consistent, each job in an injector is predicted as a different job (Fuel Valve A, Fuel Valve B, Fuel Valve C as shown in figure 4.2 ), even though the MK is the same in these cases. The standard output for a binary classification then can be used; an '1' for n overhaul that was carried out and a '0' for the observations that were added manually, as seen in appendix A.2.

Equally important to having a clear overview of the input variables, is having a look to the output - how the predictions are distributed. As seen in figure 4.4 the classes are not evenly divided. One class, referred to as *majority class*, occurs much more often than the other (minority class). Machine learning models are based on problems with balanced output classes [23], but there are methods that can adapt the models in such a problem [11] as seen in paragraph 3.3.

In order to finalize the data preparation, the data needs to be formatted in a manner that is readable by machine learning algorithms. The models are able to read numbers in order

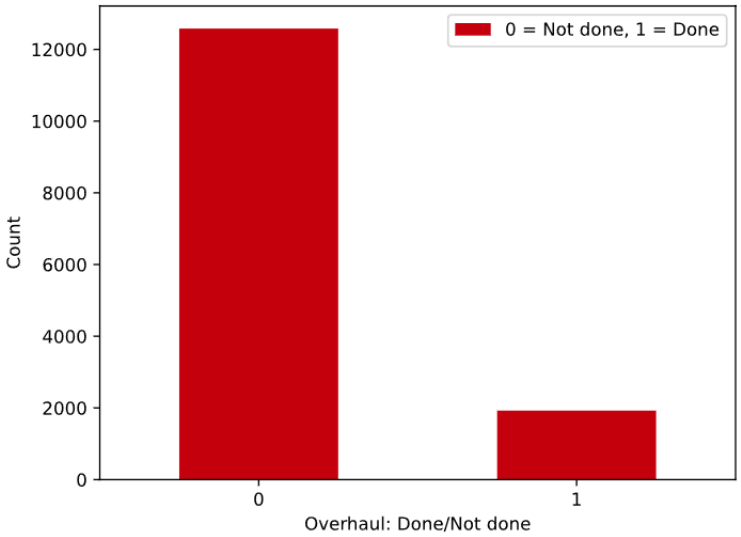



Figure 4.4: Summary of the output classes

to make predictions. Categorical variables either in nominal or numerical form, cannot be used as they are. The data that is used contains nominal variables (appendix A.2) that needs to be processed. Other than that, numerical data imply some kind of hierarchy as well as an arithmetic relation. For example, 6 is interpreted by the algorithms as twice as high as 3. This is why also arithmetic variables have to be treated the same way as the other categorical. For example a cylinder number, does not indicate some kind of order or relation to the others; the numbers assigned to them represent their names, as a way to distinguish them. Not formatting these types of variables will not cause errors to the models (as opposed to those of nominal types), but will produce wrong results. 1-out-of-K encoding creates one binary column for each value of the categorical attributes. Each one of these newly created columns take the name of the corresponding value. All the observations of a specific value in a categorical feature in the starting dataset, are assigned to a value equal to 1 in the created column with the same name and a value of zero to the rest.

Having many categorical variables with many values, means that the final form of the dataset will have many columns, that are mostly consisted by zeros (sparse data). It seems that machine learning algorithms can handle that type of problems [24], but it should be taken under consideration throughout the case study.

# 5 Results

Table 5.1: Methods implemented in each section

| | Optimal threshold | Dimensionality reduction | Weight balancing | Hyperparameter tuning |
|---|---|---|---|---|
| Models in secton 5.1 | | | | |
| Models in secton 5.2 | X | | | |
| Models in secton 5.3 | X | X | | |
| Models in secton 5.4 | X | | X | |
| Models in secton 5.5 | X | | X | X |

In this chapter the results are demonstrated after each training session. First of all, the models will be tested with their default hyperparameter values (except some basic modifications) and see how well they perform in this type of problem. This is something that helps in distinguishing which models might have the potential to achieve a good result. In the next section, the best models will be tested on how they perform on compressed data (embeddings). Next (section 5.4), the best default models, together with the ones that performed well in the embeddings, will be improved with the method of weight balancing. Finally (section 5.5), the models hyperparametrs will be tuned. After each one of those steps, the models will be compared. The purpose is to demonstrate how each method affects the training and if the models are improved by using any of the techniques shown in section 3.3. Finally, the best model will be proposed, taking into consideration parameters like performance, complexity and explainability.

## 5.1 Initial models

As said, the models have been trained with their default hyperparameter settings. Only changes occurred, are some typical increase in the number of estimators to 500 on models based on decision trees, where this is applied.

For the neural network, a dropout of 0.3 is used, meaning that in each layer 30% of the neurons' outputs are hidden. Early stopping is applied if no improvement in *validation loss* is noticed. The validation loss is a metric similar to the loss of a model, but this is calculated after each epoch. If validation loss has stopped decreasing for 25 epochs, the models stops the training.

Figure 5.1 illustrates the confusion matrices of the models on the test set. The test set consists of 491 true labels and 3.738 false labels. As expected, the linear models seem that they overpredict the majority class. The predictions for the true label were less than 40 out of 490 (Recall < 0.1). The two simpler boosting techniques appear to also have problems to classify the true label correctly, with recall much less than 0.5. Finally, the rest of the models, appear to have found some patterns that the data follows and have managed to recognize quite many of the true values. It is worth noticing, that the random forest, even though it achieved much less recall than XGBoost and the Neural Network, it only misclassified a very small portion of the negative values of the test set (precision $\approx 0.83$).

As observed, confusion matrices can give a good, intuitive first impression about the models' performance. However, as mentioned in section 3.2.2 and also seen here, a model

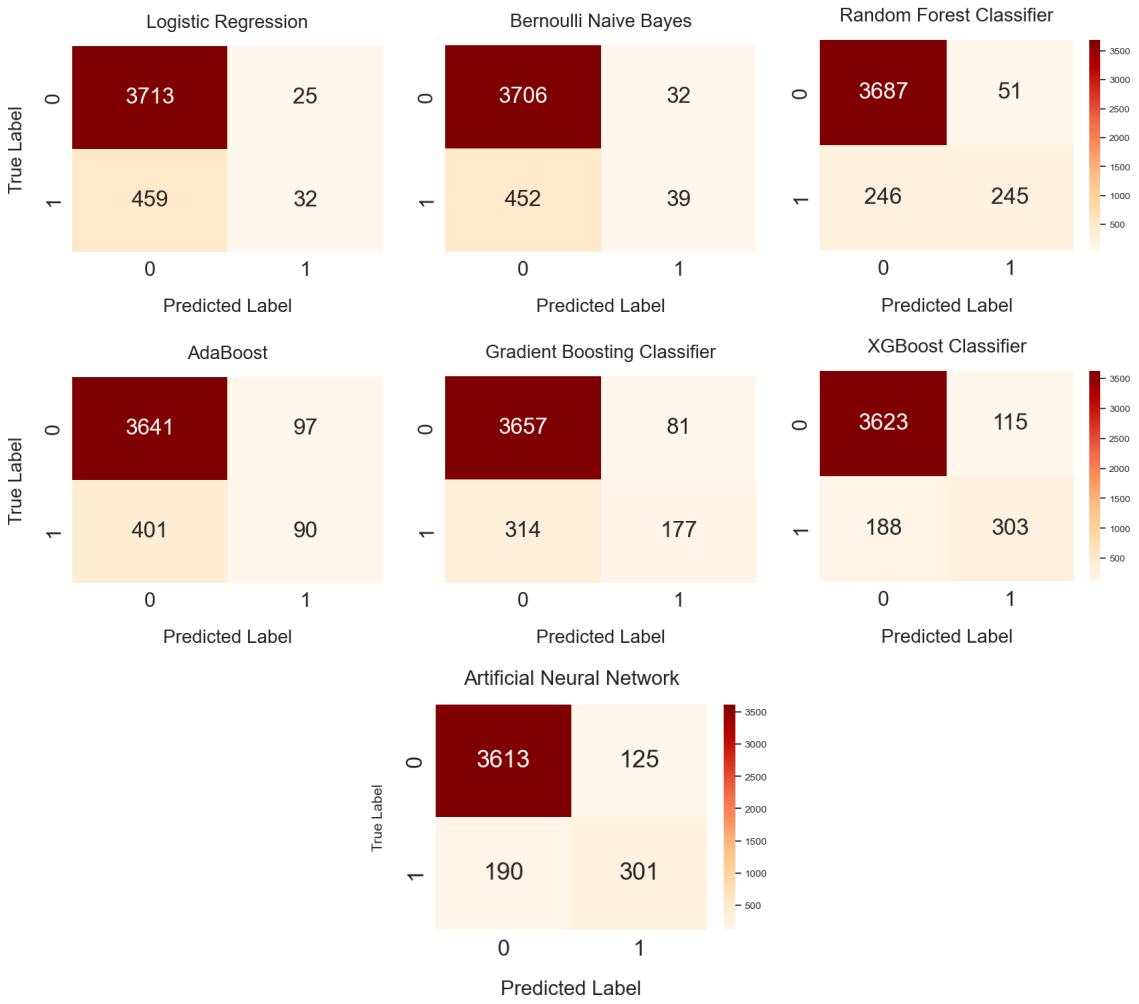



Figure 5.1: Confusion matrices of initial models

should not be evaluated using just one metric. Indeed, the random forest performs much differently than the other two having a high precision and quite low recall, contrary to the other two. It would be interesting to see the f1 score which is derived by those scores.

The comparison of the f1 scores, as seen in table 5.2, shows that the Random Forest performs worse than the XGBoost and the Neural Network, but not as badly as the recall would indicate. The difference is just 0.04 compared to the highest performing model, which is a minor gap.

## 5.2 Threshold movement

The first step gave a picture of the potential of the models. However, the default threshold of 0.5 does not work for all the models equally. Next thing to consider before the final evaluation of the models is to see how they perform at each separate threshold. ROC and Precision-Recall curves in figures 5.2 and 5.3 visualize it comprehensively.

What needs to be noticed in these figures, is not the individual values per se, but how the models perform in the complete range of the thresholds. It seems, as indicated from table 5.2, that the Random Forest, the XGBoost and the Neural Network perform similarly. Their curves are close to each other and quite separated from the rest. What is unexpected though is the low performance of AdaBoost classifier. A boosting technique that should

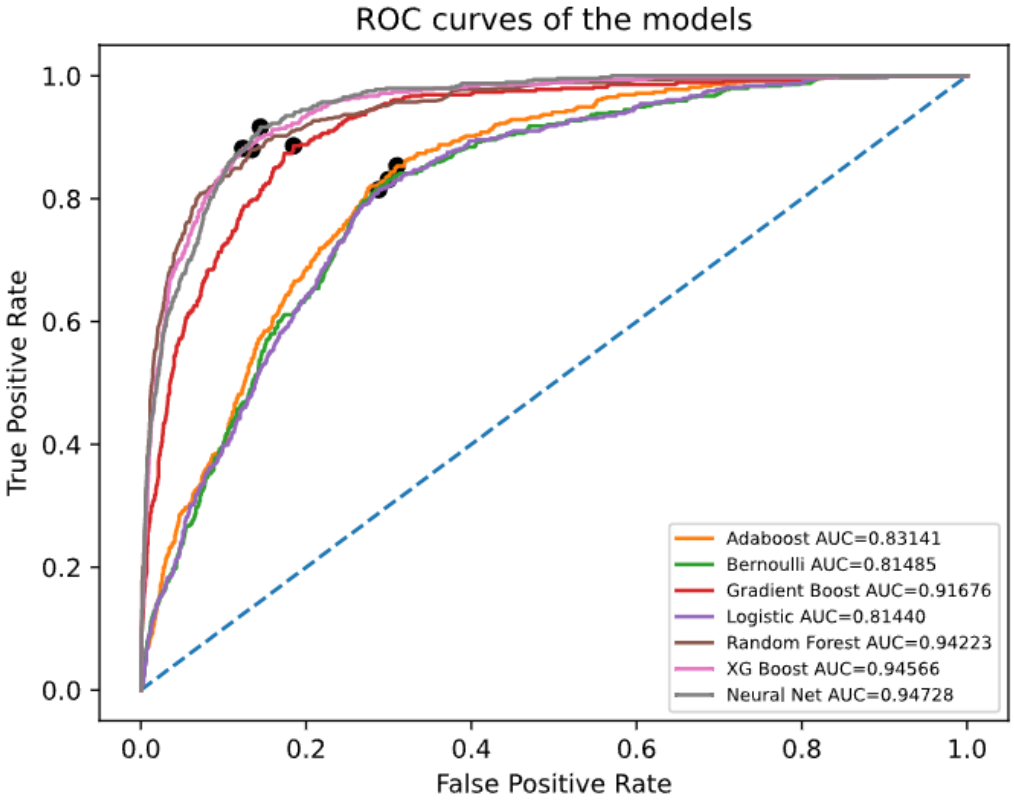



Figure 5.2: ROC curves of the initial models

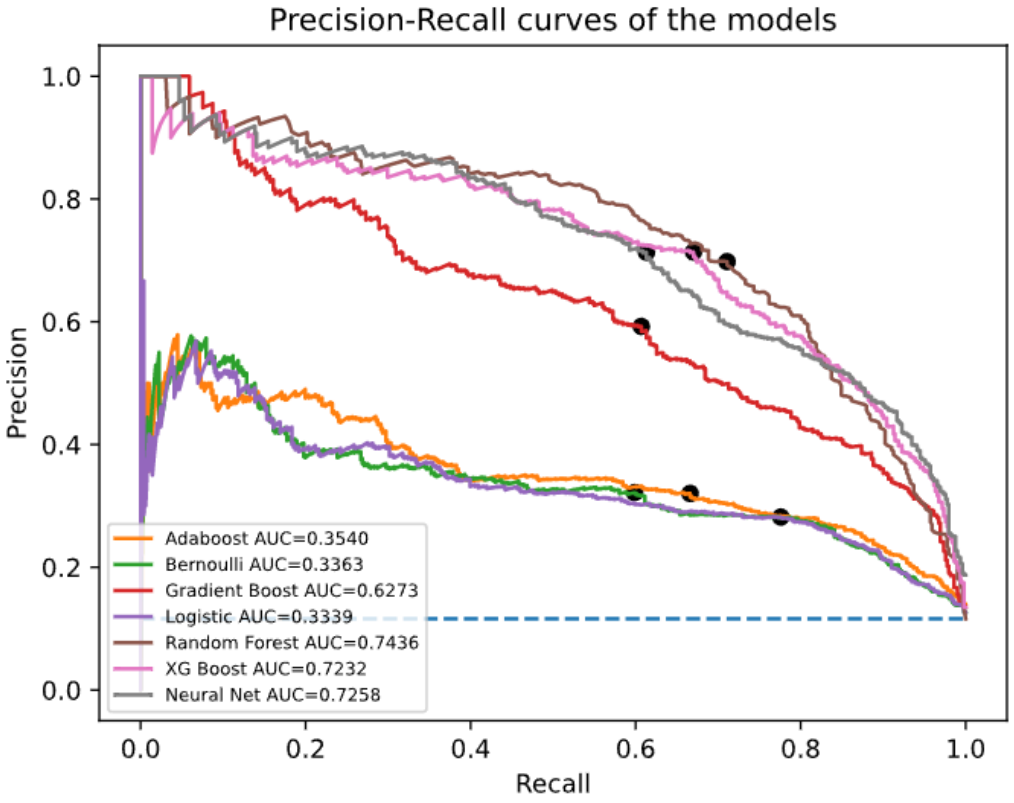



Figure 5.3: Precision-Recall curves of the initial models

Table 5.2: Table of the tested models and their corresponding scores

| Classifier | Precision | Recall | F1 Score |
|---|---|---|---|
| Logistic Regression | 0.561 | 0.065 | 0.117 |
| Bernoulli Naive Bayes | 0.549 | 0.079 | 0.139 |
| Random Forest | 0.828 | 0.499 | 0.623 |
| AdaBoost | 0.481 | 0.183 | 0.265 |
| Gradient Boosting | 0.686 | 0.360 | 0.473 |
| XGBoost | 0.725 | 0.617 | 0.667 |
| Neural Network | 0.707 | 0.613 | 0.656 |

fit well in a binary classification problem, should at the least be distinguished from the two weakest classifiers (Bernoulli classifier and Logistic Regression).

The second thing that these graphs contain is the best threshold given according to the highest J-score (shown in figure 5.2) and highest F1-score (figure 5.3), marked as black dots, also explicitly seen in table 5.3. The value of the threshold is found based on the performance of the test set. This means that, these thresholds are the ones that return the highest score, when evaluated to the test set. The optimal thresholds are located on the *elbow* of the curve (where the curve bends the most). The optimal thresholds calculated here, are the ones that maximize the given scores, but they are not the definite best. Depending on what the decision maker wants to achieve, the threshold can be decreased (and include more actual True values on the predictions) or increased and be more sure about the decisions predicted as true. What should be considered is that there will be the trade-off, of how many negative values will also be misclassified while doing that, as seen in 3.3.1.

Finally, the last metric seen in these plots are the AUCs. This area is also a good way to compare models, since it shows their behaviour throughout all the values of the thresholds. As expected, the best performing models have bigger AUCs.

Table 5.3: Optimal thresholds for the models, according to their J-scores and f1-scores

| Classifier | Threshold for highest J-score | Threshold for highest F1 score |
|---|---|---|
| Logistic Regression | 0.119 | 0.135 |
| Bernoulli Naive Bayes | 0.112 | 0.196 |
| Random Forest | 0.164 | 0.336 |
| AdaBoost | 0.499 | 0.499 |
| Gradient Boosting | 0.134 | 0.317 |
| XGBoost | 0.089 | 0.441 |
| Neural Network | 0.095 | 0.504 |

Seeing the results above, it makes sense to only work towards the improvement on the best three models, since it seems improbable that the four others might surpass the former in performance.

**Confusion matrices for different thresholds**

Next it is worth looking and comparing how the confusion matrices change when adjusting the thresholds. Figure 5.4 contains the confusion matrices for the thresholds that give the best J-score and f1-score.

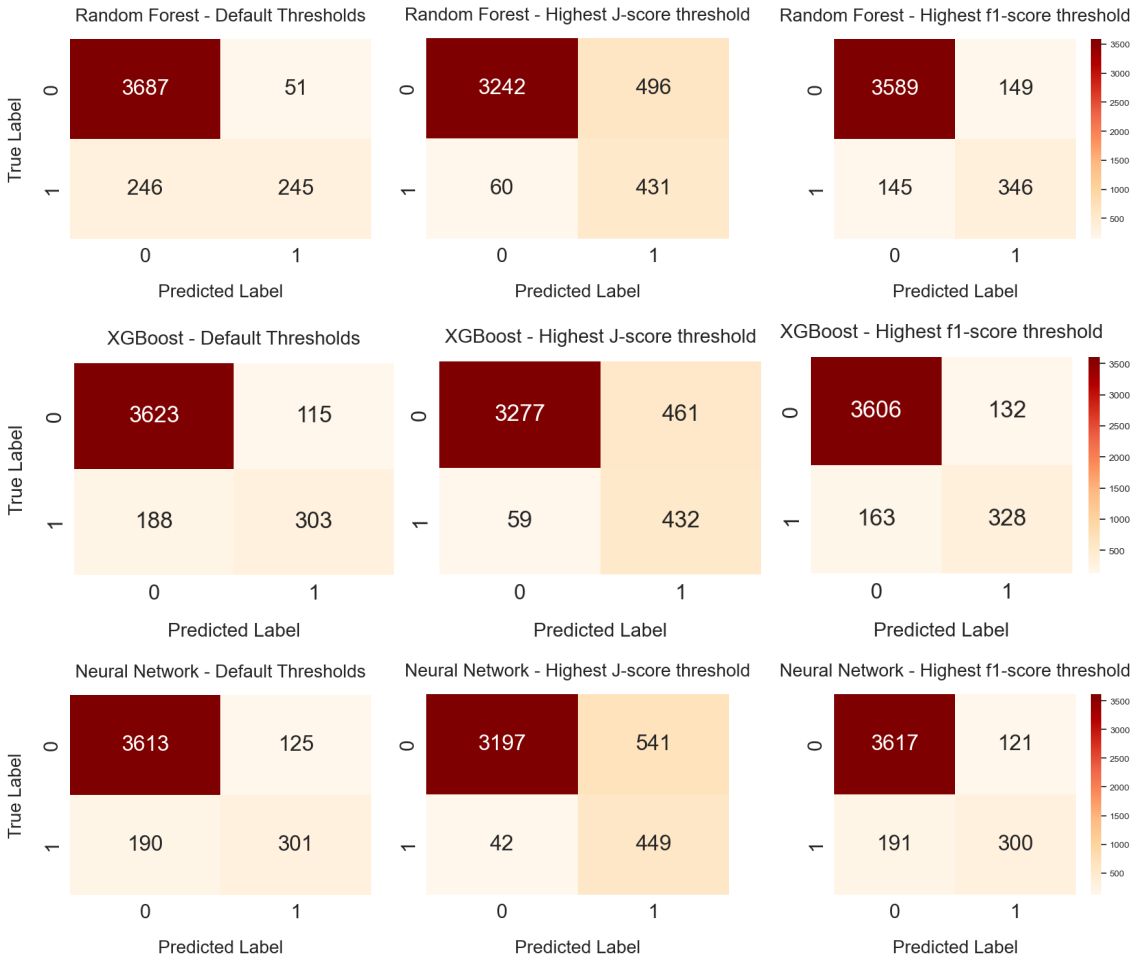



Figure 5.4: Comparison of confusion matrices with default (left), highest J-score (middle) and highest f1-score (right) values of the threshold for the three best performing models

What can be observed in these matrices is how the metric that is used, affects the predicted labels of the observations. The J-score focuses on predicting as many True Positives with the trade-off of reduced Precision (more False Positives allowed), while the f1 score sacrifices the Recall for better confidence on the True Positives. At the same time, J-score moves the threshold towards the lowest end of the threshold interval (table 5.3). Moving the threshold too much introduces external bias to the model, something that should not be preferred, if there is another option available. For that reason, f1 score will be used for the consequent tests when evaluating the models.

## 5.3 Dimensionality reduction - Locally linear embeddings

The purpose of this study's models is to not only work for this dataset, but to also build a model that can generalize well enough, when handling a bigger dataset (bigger fleet/more vessels). A dataset with more variables might be too complex to be trained, so it is of interest to see how the models perform after the dataset has been embedded, something that will be taken into consideration when proposing the best model.

The XGBoost and the neural network seem to be massively affected by the dimensionality reduction. The decrease in their performance is considerably big for both of them. The Random forest however is not affected as much. It has a performance of 0.637 f1-score (Table 5.4) at threshold 0.380, which is not much lower than how it performs on the initial

Table 5.4: Comparison of models' performance on initial data and embedded data

| Classifier | Best f1-score on initial data | Best f1-score on embedded data |
|---|---|---|
| Random Forest | 0.704 | 0.637 |
| XGBoost | 0.691 | 0.487 |
| Neural Network | 0.659 | 0.379 |

data. Indeed, the Precision-Recall curve (figure 5.5) indicates that the behaviour of the Random Forest trained in embedded data throughout all the of the thresholds is quite close to the best performing models. This reduction, is of course expected since compressing the data leads to loss of information. In this case, though, the decrease in performance can be accepted, taking into consideration that it can be proven to be beneficial in a case with bigger dataset with more features, as described before.

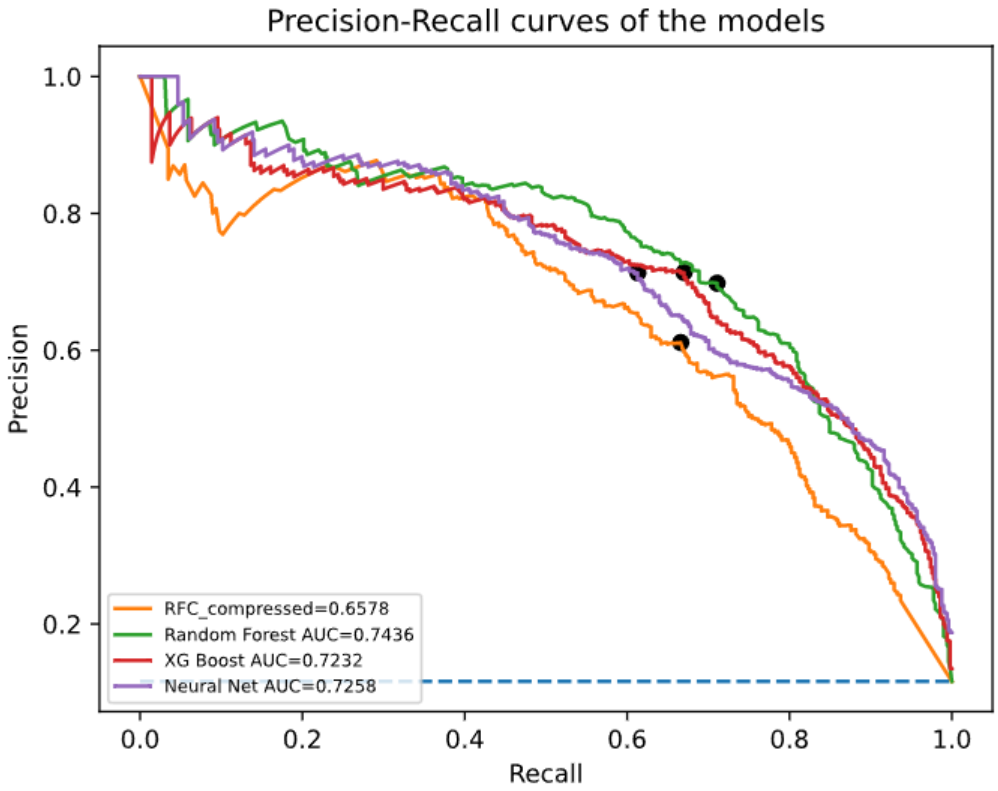



Figure 5.5: Comparison of Random Forest (LLE) Precision-Recall to the initial models

## 5.4 Weight balancing

Since it is decided which models seem that have the most potential, next thing to consider is how the models can be improved and achieve better results. For the next sections, the Random Forest (with and without LLE), the XGBoost and the Neural Network will be tested. First thing to be tested is if introducing weight balancing is improving the results. The models seem that they favour the prediction of the majority class and it seems that giving a better reward on predicting the positive labels, might prove helpful. The ratio to be used is from the train set, so there will not be data leakage. The train set consists of 11.233 negative labels and 1.452 positive, a ratio of 89% to 11% and these ratios are the values that are used as weights.

Table 5.5: Comparison of models' score before and after applying weights to the labels

| Classifier | F1 score w/o balanced weights | F1-score with balanced weights |
|---|---|---|
| Random Forest | 0.704 | 0.716 |
| XGBoost | 0.691 | 0.694 |
| Neural Network | 0.659 | 0.683 |
| Random Forest (LLE) | 0.637 | 0.638 |

Table 5.5 indicates minor improvement to all the models, with the neural network being more affected by this change, with 2.5% increase of the score. Tuning the weights further, did not seem to much improve the models consistently, so the best way to proceed in the last part is by applying the standard ratios.

## 5.5 Hyperparameter tuning

Final, but highly important improvement step, is hyperparameter tuning. So far the default values of the hyperparameters have been used, almost exclusively. However, a model might not perform well in a given problem in its default state. The actual values that would improve the models cannot be estimated by the practitioner. There are some hyperparameters that seemingly affect the model the most, so these will be tuned given the values shown in tables 5.6, 5.7, 5.8 and 5.9:

Table 5.6: Hyperparameter values tested on the Random Forest

| Hyperparameter | Value I | Value II | Value III |
|---|---|---|---|
| n_estimators | 500 | **1000** | 1200 |
| max_features | sqrt | log2 | **None** |
| criterion | gini | **entropy** | |

Table 5.7: Hyperparameter values tested on the XGBoost

| Hyperparameter | Value I | Value II | Value III |
|---|---|---|---|
| n_estimators | 500 | **1000** | 1500 |
| learning_rate | 0.001 | **0.1** | 0.3 |
| subsample | 0.5 | 0.75 | **1** |
| max_depth | 3 | 6 | **9** |

The number of estimators in decision tree models is the most important hyperparameter, which dictates how many decision trees will be used to train the model. Usually, more estimators means better results and more training time. Maximum features (Tables 5.6 and 5.9) is how many features will be available when a decision tree makes a split. This number is usually restricted, in order to decorrelate the trees (section 3.1.3), to the square root or the logarithm of the total features. If set to 'none' then all features are available in each branch. Subsample and max depth are seen in XGBoost (Table 5.7). The first prevents overfitting by sampling the data before growing a tree, the second is the maximum depth (number of splits) allowed to the decision trees. The hyperparameters optimized in the Neural Network (Table 5.8) are the batch size, while the other two belong to the optimizer of the model. Learning rate is similar to the ones met in Random Forest while beta1 is the decay of the learning rate (learning rate decreases, as the training of a Neural Network progresses).

Table 5.8: Hyperparameter values tested on the Neural Network

| Hyperparameter | Value I | Value II | Value III |
|---|---|---|---|
| batch_size | 32 | 64 | **128** |
| optimizer_beta_1 | **0.5** | 0.8 | 0.9 |
| optimizer_learning_rate | **0.001** | 0.01 | 0.3 |

Table 5.9: Hyperparameter values tested on the Random Forest with embedded data

| Hyperparameter | Value I | Value II | Value III |
|---|---|---|---|
| n_estimators | 500 | 1000 | **1200** |
| max_features | sqrt | log2 | **None** |
| criterion | gini | **entropy** | |

All the combinations of the values shown on the tables have been tested and cross-validated within the training set and the values in bold font are the ones that gave the best scores. Finding the optimal value for a hyperparameter is challenging. What is desired here is trapping the optimal value in an interval as small as possible. For example, the batch size found in Neural Network is on the upper end of the tested interval. What would be needed to be done, is repeating the process on larger values (e.g. [128, 256, 320]). Due to how computationally expensive is tuning this process, higher batch size was tested manually and 256 was selected as optimal value.

In other cases, the hyperparameters took the highest (or lowest) possible values, or values too extreme to be meaningful to move the interval lower or higher. An example is the beta1 value in the neural network. Knowing its function and reading the results of the cross-validation, testing an even lower value, would not have a good effect on our model. One consideration is that, due to the stochastic nature of the algorithms, tuning the model using new values for the other hyperparameters, would return a beta1 value higher than this.

Afterwards, these values were used to train the models anew, as normal, and tested on the test set. As before, the next thing to do is compare the new models with the previous and see if there is any improvement:

Table 5.10: Comparison of previous models' score with tuned models

| Classifier | F1 score w/o balanced weights | F1 score with balanced weights | F1 score after tuning |
|---|---|---|---|
| Random Forest | 0.704 | 0.716 | 0.746 |
| XGBoost | 0.691 | 0.694 | 0.712 |
| Neural Network | 0.659 | 0.683 | 0.727 |
| Random Forest (LLE) | 0.637 | 0.638 | 0.630 |

The results show that the Random Forest and the Neural Network were quite positively affected. More than 4% of improvement is noticed on both of their f1 scores reaching 0.746 and 0.727 correspondingly. Also, the Random Forest with the LLE decreased by a small amount (less than 1% ) which can be considered statistical error. Taking a look at the confusion matrices (figure 5.6), one can intuitively see that the models, especially the Random Forest, give quite better results. Having a very small False Positive Rate (about

0.05) it seems that the predictions will very rarely indicate a positive label, that is also not actually True, while being able to identify quite a lot actual True values.

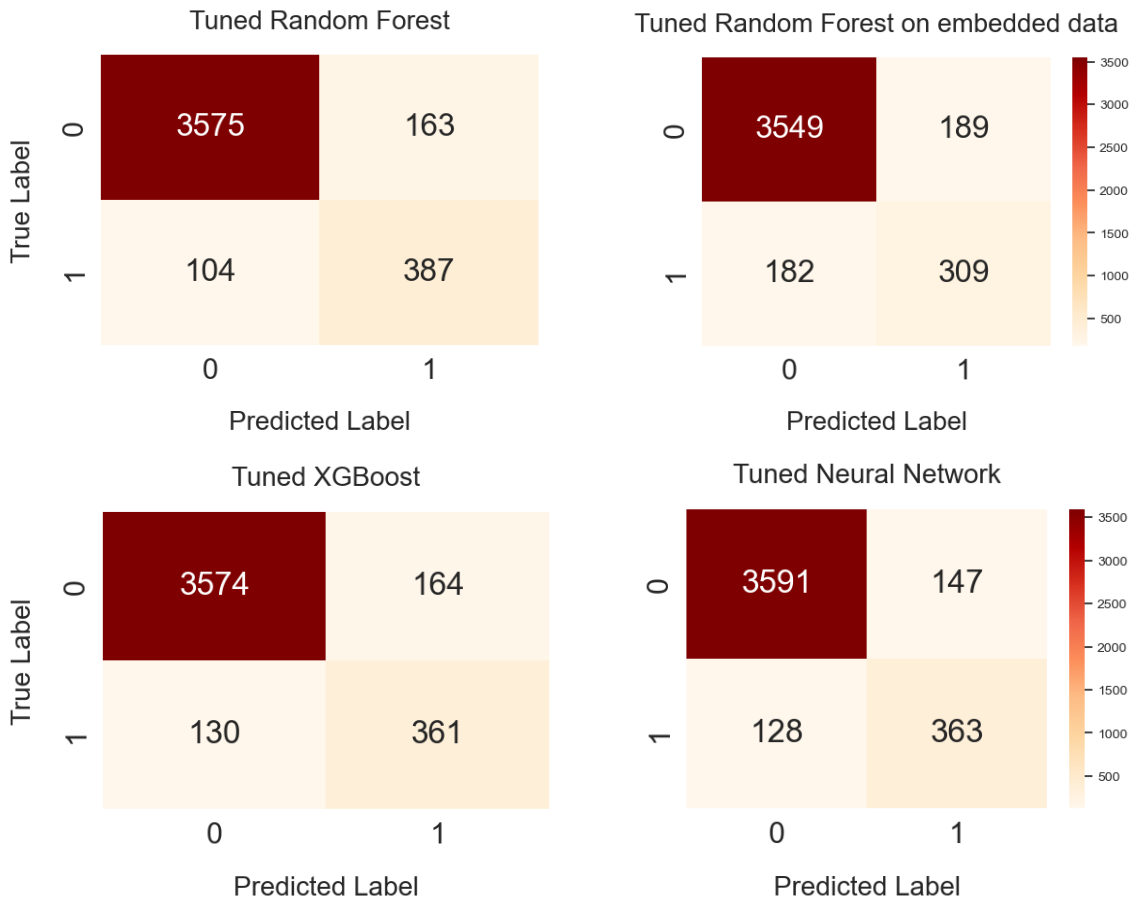



Figure 5.6: Confusion matrices of the models after hyperparameter tuning

## 5.6 Proposed model

After seeing the results, it seems that the Random Forest performs a little better than the rest of the models. It would be interesting to see how the Neural Network and XGBoost perform after tuning them further, since their final performance is not close to the Random Forest's. This, however, is something that adds to the complexity of the model, the training time and the effort to build.

Based on the criteria set in the beginning of this section, it seems that the Random Forest is the model best suited for the prediction of the vessels' maintenance. Even though the potential to improve is not large, it performs well, it is simple to tune and fast to train. On top of that, being able to adapt in an embedded dataset makes it flexible enough for other datasets. It is a model that can be used, at least initially, to offer some insight of what to be expected in the upcoming year from the specific customer.

# 6 Discussion

In this chapter a robustness check using the best model will be made, by testing it on years 2021 and 2022, separately. Then, this outcome will be discussed together with the final results of the previous section, the study's limitations, potential and future work that can be done based on those results.

## 6.1 Robustness check

This section will make use of the Random Forest, since it was the best model overall, and check how robust it is to market changes. The machine learning models used in this study, work under the strong assumption that the data is evenly distributed. Before splitting the data in test and train set, it is shuffled. That shuffle, even though it gives randomness to the data, makes the output distribution quite even. Meaning that both test set and train set have almost equal ratio of true labels compared to the whole set. In a real life scenario, this might not be correct. As seen in figure 4.2, the jobs done each year increase remarkably. It is worth investigating how the models' behaviour changes. In the next section, the proposed model will be used to predict year 2021 first and year 2022 afterwards. Then their performance will be compared.

The data used for the next part will simulate a situation where the models predict the next year (2021 and 2022, correspondingly) and therefore there is availability of data only for the years prior to that. For the needs of this section, the data will be expanded with the scheduled jobs for year 2022. The data is taken from early 2022, so it is a safe assumption to say that the planned overhauls for 2022 will actually be carried out in this year.

Figure 6.1 shows the jobs done (or planned to be done) for the years from 2016 until 2022. Worth noticing is that the distribution is becoming more stable in the last 3 years, compared with the previous period.

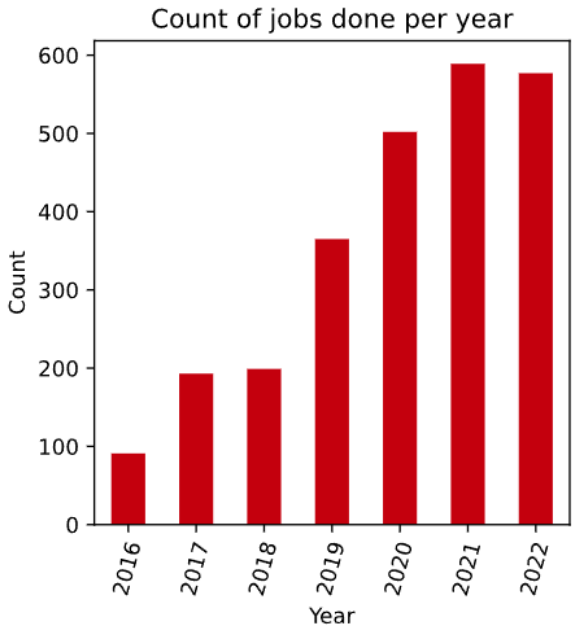



Figure 6.1: Jobs per year, including scheduled maintenance for 2022

### 6.1.1 Predict years 2021 and 2022

The process of training the model is the exact same as the one followed in section 5. The model is trained on weights given by the training data, the hyperparameters are tuned and the results are given using the optimal thresholds.

There are two noticeable differences in this process, however. First, the train and test set will not be split using the method used in section 4. For this training session, the test set will be the complete year that is the target of predictions, while the data will be trained on the years before that.

Second difference is that, since data from next year is not available, the optimal threshold will be calculated by cross-validation in the training data, process similar to the one described in section 3.3.4. The thresholds and the f1 scores achieved from cross-validation are described at table 6.1

Table 6.1: Comparison of Random Forest scores in cross-validation

| | Cross-validation threshold | Cross-validation f1-score |
|---|---|---|
| Model predicting 2021 | 0.343 | 0.672 |
| Model predicting 2022 | 0.354 | 0.739 |

After the training, the results are given as predicted probabilities (float numbers). The optimal threshold is used then to translate the probabilities to predictions (1 and 0). These predictions are merged to the test set giving a result, as shown in appendix A.3

### 6.1.2 Evaluate models

The models seem to perform similarly to each other and to the models that were trained in data with equal distribution, when seeing the cross-validation scores. When comparing the predictions with the actual values however (reminder: data that would normally not be available), the performance gives a different picture. The f1 score when predicting 2021 is almost 0.15 and the performance when predicting 2022 is about 0.35.

These two scores show how important the assumption of evenly distributed data is for these machine learning models and how sensitive they are to excessive market changes. One thing is that much lower scores were achieved compared to the models trained on evenly distributed data. The second thing though, is how much better predictions the model gave for 2022 when the data was of a little higher quality.

## 6.2 Limitations and potential

The initial goal of this work is to create a procedure that involves predicting the scheduled maintenance that all the company's main customers will need in the next year, assign these predictions to maintenance kits and thereby providing a full list of all the spare parts that will be needed for MAN to have it ready in stock.

The results showed that the potential is good. The data that has been used is on only 28 of the customer's vessels and contained information on just the previous overhaul done on each cylinder.

The way the problem was shaped is also very beneficial. A binary classification problem is a comparably simple task for machine learning algorithms and, moreover, with easily interpretable results.

One limitation of the process is that the company will need historical data from the customers. The main key to shape this problem as a binary classification is having the information on the jobs done, on a cylinder level; not just vessel or engine. This information cannot be pulled from data that MAN has available, except from the occasions that the repairs are supervised by an MAN senior engineer. These cases have not been used in this study, so the quality of this data has to be evaluated separately, if it can be used.

A second limitation, as explained in the previous section, is how sensitive are the models to big market changes. This means that this method can be used mainly on loyal (or at least, frequent) MAN's customers. In the specific case, it can be achieved by excluding years that have too low amount of overhauls (like 2016 and 2017) or adding older maintenance data for the same vessels.

## 6.3 Future work

The study shows how machine learning can be used for predicting maintenance schedule and proposes a way to approach the problem. There are weaknesses, as described above, that need to be handled, in the future.

To start with, the class imbalance characteristic of the problem can be handled by *undersampling*, *oversampling* or a combination of these two. Using undersampling techniques, the samples taken from the majority class are reduced in quantity and with oversampling samples are generated from the current observations. There are algorithms developed for these purposes and can prove useful for such a problem [25] and benefit the decision tree algorithms.

Next, training a neural network is a challenging process. In this study, a simple dense feedforward neural network was used. By using advanced methods, more complicated neural networks can be built which will be able to generalize better and give results more robust to changes. It is expected that a Neural Network can achieve much better results, given that there is enough time to build and train.

Finally, hyperparameter tuning, is computationally demanding and time consuming. If there is availability of time they can be tuned further and more precisely. It is necessary to continue tuning the hyperparameters when the GridSearch (section 5.5) results are on the extremes of the intervals. It can also be beneficial, to continue looking for an optimal value, looking in smaller intervals, even when the value is within the limits of the search.

These are methods that can build better-performing and more robust to market changes, models. It would be interesting to be applied on data with more features and more observations and see how they perform and what results can be achieved.

# 7 Conclusion

This study case demonstrates a method that can be used to predict the maintenance schedule of a MAN's customer's vessels. The end goal is to be able to predict accurately the overhauls needed for the next year. This will be beneficial for MAN and the customer on many levels. MAN PrimeServ will be able to plan ahead and have available the inventory needed for the next year. This will also aid in a consultatory sense. Knowing all the maintenance has the vessels been through, MAN will be able to inform the customer about the upcoming year. In this aspect, this is how the customer is also benefited. Knowing one year in advance the overhauls needed, budgeting for the maintenance will be much easier.

For this project a dataset of 3.384 observations has been used, expanded for the needs of the models to 16.914. 7 machine learning models have been trained and compared, with 3 of them performing sufficiently well. These models have been tried on embedded data, 1 of those seemed to have potential and this model was also included, together with the others, for improvement. These 4 models have been improved through the methods of threshold movement, weight balancing and hyperparameter tuning. The final results showed that the Random Forest should be, at this stage, proposed for use. The reasons are:

- Net performance: Better f1 score than the other models
- Flexibility: Performing decently in embedded data
- Complexity: Easy to build and to tune

In its current state, this study provides a methodology to be followed on this or on an expanded dataset. It can also be used as a tool that gives indication for warehouse needs and can help in inventory management. In general, the study shows that MAN can incorporate machine learning into its business to predict the maintenance schedule of particular customers.

# Bibliography


[1] Leonard Heilig, Silvia Schwarze, and Stefan Voss. “An analysis of digital transformation in the history and future of modern ports”. In: *Proceedings of the 50th Hawaii International Conference on System Sciences (2017)* (2017). DOI: 10.24251/hicss.2017.160.

[2] Michael Ten Hompel and Thorsten Schmidt. *Warehouse management*. Springer, 2008.

[3] Gwynne Richards. *Warehouse management: a complete guide to improving efficiency and minimizing costs in the modern warehouse*. Kogan Page Publishers, 2017.

[4] Jeroen P Van den Berg and Willem HM Zijm. “Models for warehouse management: Classification and examples”. In: *International journal of production economics* 59.1-3 (1999), pp. 519–528.

[5] Peter Kacmary et al. “Creation of Annual Order Forecast for the Production of Beverage Cans—The Case Study”. In: *Sustainability* 13 (July 2021), p. 8524. DOI: 10.3390/su13158524.

[6] Seongmin Moon. “Predicting the Performance of Forecasting Strategies for Naval Spare Parts Demand: A Machine Learning Approach”. In: *Management Science and Financial Engineering* 19 (May 2013). DOI: 10.7737/MSFE.2013.19.1.001.

[7] Wenbin Wang and Aris A Syntetos. “Spare parts demand: Linking forecasting to equipment maintenance”. In: *Transportation Research Part E: Logistics and Transportation Review* 47.6 (2011), pp. 1194–1209.

[8] Catarina Teixeira, Isabel Lopes, and Manuel Figueiredo. “Classification methodology for spare parts management combining maintenance and logistics perspectives”. In: *Journal of Management Analytics* 5.2 (2018), pp. 116–135. DOI: 10.1080/23270012.2018.1436989.

[9] Carol Romanowski and Rakesh Nagi. “Analyzing Maintenance Data Using Data Mining Methods”. In: Nov. 2001, pp. 235–254. ISBN: 978-1-4419-5205-9. DOI: 10.1007/978-1-4757-4911-3_10.

[10] Merve Şahin, Recep Kızılaslan, and Ömer F. Demirel. “Forecasting Aviation Spare Parts Demand Using Croston Based Methods and Artificial Neural Networks”. In: *Journal of Economic and Social Research* 15.2 (2013). DOI: https://www.academia.edu/38072308/Forecasting_Aviation_Spare_Parts_Demand_Using_Croston_Based_Methods_and_Artificial_Neural_Networks.

[11] Chao Chen and Leo Breiman. “Using Random Forest to Learn Imbalanced Data”. In: *University of California, Berkeley* (Jan. 2004).

[12] Pedro Gerum, Ayça Altay, and Melike Baykal-Gürsoy. “Data-driven predictive maintenance scheduling policies for railways”. In: *Transportation Research Part C: Emerging Technologies* 107 (Oct. 2019), pp. 137–154. DOI: 10.1016/j.trc.2019.07.020.

[13] Jennie Pearce and Simon Ferrier. “Evaluating the predictive performance of habitat models developed using logistic regression”. In: *Ecological modelling* 133.3 (2000), pp. 225–245.

[14] Irina Rish. “An Empirical Study of the Naïve Bayes Classifier”. In: *IJCAI 2001 Work Empir Methods Artif Intell* 3 (Jan. 2001).

[15] Pedro M. Domingos and Michael J. Pazzani. “On the Optimality of the Simple Bayesian Classifier under Zero-One Loss”. In: *Machine Learning* 29 (1997), pp. 103–130.

[16] Yoav Freund and Robert E. Schapire. “A desicion-theoretic generalization of on-line learning and an application to boosting”. In: *Lecture Notes in Computer Science* (1995), pp. 23–37. DOI: 10.1007/3-540-59119-2_166.

[17] Jerome H. Friedman. “Greedy function approximation: A gradient boosting machine.” In: *The Annals of Statistics* 29.5 (2001). DOI: 10.1214/aos/1013203451.

[18] John Duchi, Elad Hazan, and Yoram Singer. “Adaptive subgradient methods for online learning and stochastic optimization.” In: *Journal of machine learning research* 12.7 (2011).

[19] Mohammed Khalilia, Sounak Chakraborty, and Mihail Popescu. “Predicting disease risks from highly imbalanced data using Random Forest”. In: *BMC Medical Informatics and Decision Making* 11.1 (2011). DOI: 10.1186/1472-6947-11-51.

[20] Brian S Everitt and Anders Skrondal. “The Cambridge dictionary of statistics”. In: (2010).

[21] Carmen Esposito et al. “GHOST: adjusting the decision threshold to handle imbalanced data in machine learning”. In: *Journal of Chemical Information and Modeling* 61.6 (2021), pp. 2623–2640.

[22] Lawrence Saul and Sam Roweis. “An introduction to locally linear embedding”. In: *Journal of Machine Learning Research* 7 (Jan. 2001).

[23] Ming Zheng et al. “A Method for Analyzing the Performance Impact of Imbalanced Binary Data on Machine Learning Models”. In: *Axioms* 11 (Nov. 2022), p. 607. DOI: 10.3390/axioms11110607.

[24] Bhuvana Adur Kannan et al. “Forecasting Spare Parts Sporadic Demand Using Traditional Methods and Machine Learning - a Comparative Study”. In: *SMU Data Science Review*. 9th ser. 3.2 (2020). DOI: https://scholar.smu.edu/datasciencereview/vol3/iss2/9/.

[25] Mayuri S Shelke, Prashant R Deshmukh, and Vijaya K Shandilya. “A review on imbalanced data handling using undersampling and oversampling technique”. In: *Int. J. Recent Trends Eng. Res* 3.4 (2017), pp. 444–449.

# A Appendix

Table A.1: Types of variables in the initial data

| Variable name | Variable Type I | Variable Type II |
|---|---|---|
| Vessel Name | Categorical | Nominal |
| Equipment Name | Categorical | Nominal |
| Equipment Code | Categorical | Nominal |
| Job Title | Categorical | Nominal |
| Frequency | Categorical | Ordinal |
| Job Type | Categorical | Nominal |
| Last Done Date | Date | N/A |
| Last Done Hours | Continuous | Ratio |

Table A.2: Types of variables in the final form

| Variable name | Variable Type I | Variable Type II |
|---|---|---|
| Vessel Name | Categorical | Nominal |
| Running Hours | Continuous | Ratio |
| Maintenance kit | Categorical | Nominal |
| Cylinder Number | Categorical | Nominal |
| Maintenance year | Categorical | Ordinal |
| Done | Categorical | Binary |

Figure A.1: Example of the observations of the dataset

| | VESSEL_NAME | Running_Hours | Maintenance_Kit | Cylinder_Number | maintainance_year | done |
|---|---|---|---|---|---|---|
| **1** | Aqua Bonanza | 68693.0 | Fuel Valve B | 2 | 2020 | 1 |
| **5** | Aqua Bonanza | 67792.0 | Fuel Valve B | 3 | 2020 | 1 |
| **7** | Aqua Bonanza | 67792.0 | Fuel Valve B | 4 | 2020 | 1 |
| **11** | Aqua Bonanza | 70121.0 | Fuel Valve B | 5 | 2021 | 1 |
| **14** | Aqua Bonanza | 68693.0 | Fuel Valve B | 6 | 2020 | 1 |

Figure A.2: Part of the dataset used in training

| | VESSEL_NAME | Running_Hours | Maintenance_Kit | Cylinder_Number | maintainance_year | done |
|---|---|---|---|---|---|---|
| **7598** | Hyde | 57267.727273 | Cyl. Cover | 6 | 2018 | 1 |
| **7599** | Hyde | 57267.727273 | Exh. Valve Act | 6 | 2018 | 0 |
| **7600** | Hyde | 57267.727273 | Fuel Valve (16.000 hrs) A | 6 | 2018 | 0 |
| **7601** | Hyde | 57267.727273 | Fuel Pump | 6 | 2018 | 0 |
| **7602** | Hyde | 57267.727273 | Piston Ring CR | 6 | 2018 | 1 |
| **7603** | Hyde | 57267.727273 | Stuff. Box MK | 6 | 2018 | 1 |
| **7604** | Hyde | 57267.727273 | Turbocharger | 6 | 2018 | 0 |
| **7605** | Hyde | 57267.727273 | Cyl. Liner | 6 | 2018 | 1 |

Table A.3

| List of overhauls |
|---|
| Fuel valve A |
| Fuel valve B |
| Fuel valve C |
| Cylinder Cover |
| Exhaust Valve |
| Fuel Pump |
| Piston Ring |
| Stuffing Box |
| Turbocharger |
| Cylinder Liner |
| Hydraulic Cylinder Unit |
| Pressure Booster |
| Proportional Valve |
| Accumulators |

Figure A.3: Example of predictions for 2022

| | VESSEL_NAME | Running_Hours | Maintenance_Kit | Cylinder_Number | maintainance_year | prediction |
|---|---|---|---|---|---|---|
| **84** | Aqua Bonanza | 76504.045455 | Fuel Valve (16.000 hrs) B | 1 | 2022 | 0 |
| **85** | Aqua Bonanza | 76504.045455 | Cyl. Cover | 1 | 2022 | 0 |
| **86** | Aqua Bonanza | 76504.045455 | Exh. Valve Act | 1 | 2022 | 0 |
| **87** | Aqua Bonanza | 76504.045455 | Fuel Valve (16.000 hrs) A | 1 | 2022 | 1 |
| **88** | Aqua Bonanza | 76504.045455 | Fuel Pump | 1 | 2022 | 1 |
| **89** | Aqua Bonanza | 76504.045455 | Piston Ring CR | 1 | 2022 | 0 |
| **90** | Aqua Bonanza | 76504.045455 | Stuff. Box MK | 1 | 2022 | 0 |
| **91** | Aqua Bonanza | 76504.045455 | Turbocharger | 1 | 2022 | 0 |
| **92** | Aqua Bonanza | 76504.045455 | Cyl. Liner | 1 | 2022 | 0 |
| **93** | Aqua Bonanza | 76504.045455 | Hydr. Cyl. Unit | 1 | 2022 | 0 |
| **94** | Aqua Bonanza | 76504.045455 | Press. Booster | 1 | 2022 | 0 |

Technical
University of
Denmark

Akademivej, Building 358
2800 Kgs. Lyngby
Tlf. 4525 1700

www.man.dtu.dk