NTNU

Norwegian University of Science and Technology

Project Thesis

Explainable Condition Monitoring for Wind Turbines

NTNU students:
Stian Ismar
Morten Olsen Osvik

Supervised by:
Bjørn Haugen
Odd Erik Gundersen

December 16, 2019

Abstract

Wind turbines are subjected to extreme and varying loads, which could lead to faults. It is important to develop methods that can explain the root cause of these faults, to prevent them or provide maintenance before the fault evolves. Applying data aggregated by wind turbines to explainable AI (XAI) methods, can help to diagnose the state of a wind turbine and provide explanations. These explanations can benefit wind turbine operators by providing explanations next to the diagnosis.

In this thesis, we present explanations created with a decision tree (DT) model and a framework called LIME. The models are based on two years of data from a Supervisory Control and Data Acquisition (SCADA) system from 25 wind turbines at Bessaker, Norway. The DT model choice is motivated by relevant findings in a literary survey, as well as the interpretable nature of shallow decision trees. The LIME framework is a wrapper for explaining non-human interpretable models and is selected due to its ability to provide explanations to any complex machine learning model. The LIME framework provides explanations from a support vector machines model (SVMs). Relevant research papers suggest that SVMs handle SCADA data well.

The models (DT and SVMs) were able to diagnose approximately 60 % of all states correctly. A shallow DT provided consistent interpretable explanations, but failed to generalise well. We experienced the LIME explanations for the SVMs to be inconsistent. Despite LIME's challenges, our work demonstrates how similar explainable frameworks could be applied to complex real world data.

Results provided suggests that the wind turbine parameters and status log is a limiting factor in the project, due to its lack of relevance. Our research implies that relevant SCADA data is essential for providing valuable insight into the understanding of the diagnosis through XAI. Moreover, alternative explainable frameworks should be explored in further research on this topic.

Preface

This project thesis is a part of the M. Sc. Engineering and ICT degree at the Norwegian University of Science and Technology (NTNU). The thesis has been conducted in collaboration with the Norwegian power company TrønderEnergi.

Data from wind turbines situated at Bessaker, Norway have been analysed throughout this thesis. Theory and application of chosen methods for creating explainable fault diagnosis algorithms are thoroughly described, and is valuable for the upcoming master thesis in the autumn of 2020.

Trondheim, December 2019

Morten Olsen Osvik

Stian Ismar

Acknowledgements

We would like to express our gratitude towards our supervisors Bjørn Haugen and Odd Erik Gundersen for their excellent guidance throughout this project thesis. We would also like to thank TrønderEnergi and their AI-team for valuable input and data access.

Abbreviations

AI	=	Artificial Intelligence
CART	=	Classification And Regression Trees
CM	=	Condition Monitoring
CV	=	Cross Validation
DARPA	=	Defense Advanced Research Projects Agency
DT	=	Decision Tree
GDPR	=	General Data Protection Regulation
LIME	=	Local Interpretable Model-agnostic Explanations
ML	=	Machine Learning
NN	=	Neural Network
RF	=	Random Forest
RUS	=	Random Under Sampling
SCADA	=	Supervisory Control and Data Acquisition
SMOTE	=	Synthetic Minority Over-Sampling Technique
SVM	=	Support Vector Machines
WT	=	Wind Turbine
XAI	=	Explainable Artificial Intelligence

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Topic and Context

With new wind parks being established both on-shore and off-shore as renewable energy resources, research on the topic is beneficial in terms of minimising cost and maximising performance. The International Renewable Energy Agency (IRENA) (2012) estimated that the operation and maintenance of wind turbines (WTs) account for 20% to 25% of the levelised cost of energy (LCOE: the cost of the power produced). Even though technological advancements have greatly reduced the downtime for wind turbines, faults still occur- some of which take weeks to repair (Hahn and Durstewitz, 2005).

Wind turbines are subjected to extreme and varying loads and are estimated to have a lifetime of about 20 years (Hahn and Durstewitz, 2005). Condition monitoring (CM) is a maintenance strategy used for continuously monitoring components of wind turbines, to repair or replace faulty components ahead of time. The strategy is not fully applied in the wind turbine sector (Leahy et al., 2018). Advanced CM systems can be used on turbine components by utilising vibration sensors. However, research has shown that existing systems on WTs like Supervisory Control and Data Acquisition (SCADA) systems can provide valuable data for monitoring components (Leahy et al., 2018).

Machine learning (ML), a sub-field within artificial intelligence (AI), has shown to be promising as an approach to CM using data from SCADA systems from WTs. Previous research shows that ML methods are applicable for detecting pitch faults (Godwin and Matthews, 2013), detecting, classifying and predicting a variety of faults using SVMs (Leahy et al., 2018), and detecting and explaining vibration faults (Abdallah et al., 2018). Other commonly reported WT faults include faults to the yaw system, the gearbox, electrical system faults, and generator faults (Hahn and Durstewitz, 2005).

Although ML methods have shown to be good at diagnosing many different faults from SCADA data, the methods often lack the ability to *explain* the reasoning behind their decisions. Another sub-field within AI called Explainable Artificial Intelligence (XAI) focuses on just this task.

Operators and technicians acting on the bases of ML models need to be able to understand the logic behind the decisions to use them correctly and efficiently (Siekmann and Wahlster, 2019). Ribeiro et al. (2016) state that users will not benefit from the decision of a model if they do not understand it. Also, in many cases, ML decisions cannot be acted upon based on blind faith as the consequences of a bad decision could be dreadful or costly, e.g. when predicting a medical diagnosis or when deciding whether a gearbox is in a *healthy* condition or not.

1.2 Focus and Scope

The data to be examined in this paper is streamed from a SCADA system of 25 wind turbines located at Bessaker, Norway. Access to the system is provided by the Norwegian power company TrønderEnergi.

The first task of this paper is to download and pre-process data from a SCADA system. Further, the data will be used to train three types of ML models, including a decision tree, random forest, and support vector machines. These models will be able to diagnose faults on wind turbines. Then, the models will be used to provide explanations behind the decisions. The decision tree will be *interpretable* by humans just as it is, whilst the support vector machine requires an external method to explain its decisions called LIME.

The main objective of this thesis is to research how explainable/interpretable ML models can be used with SCADA data to:

1. Extract the root cause of selected wind turbine faults.
2. Provide explanations behind fault diagnosis decisions, so that human operators can trust and rely on the models.

Having presented the main scope, it is worth noting that optimising the ML models is outside of the scope of the thesis.

Evaluating the explainable models will be an important part of the thesis.

Chapter 2

Background

2.1 Maintenance Approaches

Maintenance approaches are often categorised into three main groups; **reactive**, **preventive** and **predictive** (Tchakoua et al., 2014).

2.1.1 Reactive and Preventive Maintenance

Reactive maintenance- also known as run-to-failure, performs maintenance when the component is broken. It has some major drawbacks. The cost of repair or replacement is usually very high and it is more likely to damage other components. In addition, a loss in revenue occurs whilst the component is out of order (Tchakoua et al., 2014).

With preventive maintenance, the maintenance is executed regularly based on statistical behaviour and human knowledge. While it usually prevents costly repair jobs associated with a complete component failure, it often results in performing unnecessary maintenance.

2.1.2 Predictive Maintenance

Predictive maintenance's goal is to minimise downtime and maximise productivity by performing maintenance at the optimal time. It relies on data from condition monitoring systems and applies data-driven models. By continuously observing the system, it aims to predict exactly when components will fail. Figure 2.1 shows the cost associated with each approach as well as the desired goal of predictive maintenance (Tchakoua et al., 2014).

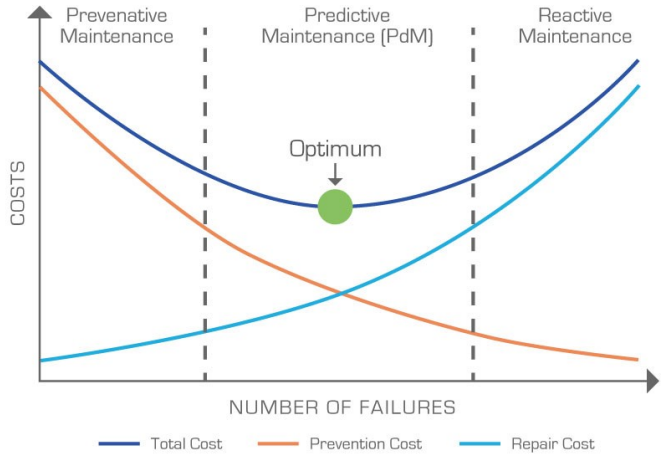


Figure 2.1: Cost associated with the three maintenance approaches.

2.1.3 Condition Monitoring

Condition monitoring is an important tool within predictive maintenance. It involves observing components to identify their conditions and changes in operation that can indicate development of a fault or failure. It often relies on analysis of specific measurements such as temperature measurements, vibration data, strain measurements, etc. (García Márquez et al., 2012).

2.1.4 Fault Detection, Fault Diagnosis and Fault Prediction

The literature distinguishes between three types of CM applications (Stetco et al., 2019):

- **Fault detection** is a binary analysis; determining whether the system is in a faulty state or not.
- **Fault diagnosis** separates the different fault types and aims to classify which is present.
- **Fault prediction** analyses data to find a pattern leading up to a fault, and aims to predict if and when a fault will happen in the future.

This thesis concerns **fault diagnostics**. Thereby, when the word *prediction* is used in this thesis, it relates to *machine learning predictions* or decisions.

2.2 Wind Turbines

The wind turbines analysed in this thesis are 25 2,3 MW Enercon wind turbines located at Bessaker, Norway. The wind farm has produced power since 2008 and

is owned and operated by TrønderEnergi. Bessaker wind farm produces 175 GWH yearly and supplies energy to about 9000 households[1].

2.2.1 What is SCADA Data?

The Wind farm controller, SCADA (Supervisory Control And Data Acquisition), provides operational data measured on wind turbines. This data can be used in condition monitoring. Throughout this thesis, "SCADA data" will be used to refer to the data being provided and used by the SCADA control system.

The data used in this study covers a time period of two years from October 2017 to October 2019 and consists of two different data sets from the SCADA system: *Operational data* and *Status Log*.

2.2.2 SCADA Operational Data

The SCADA system continuously monitors different operational parameters in the wind turbines. These are parameters such as wind speed, power outputs, currents and voltages for various components, and temperatures of e.g. generators. An illustration is shown in Table 2.1.

Timestamp	Active Power	Wind Speed	Rotor Speed	Ambient Temp	Spinner Temp	...
25/10/2017 00:00:00	1422	16.9	19.9	11	24	...
25/10/2017 00:00:30	1403	16.2	19.4	11	24	...
25/10/2017 00:01:00	1446	20.7	21.1	11	24	...

Table 2.1: Example of SCADA operational data

2.2.3 SCADA System Status Log

All wind turbines have a status log with information about their state. An example is shown in Table 2.2. There are three main status categories: "OK" - for normal operating condition, "Warning" - when the turbine is running but something is odd, and "Alarm" - when the turbine is not running.

Status Category	Description	TimeStamp
OK	Turbine Running	25/10/2017 07:12:59
Warning	Underperforming	26/10/2017 10:22:01
Alarm	Lack of Wind	26/10/2017 13:04:15
Alarm	Generator heating- (Manual)	27/10/2017 08:44:20

Table 2.2: Simple example of SCADA status log

[1] https://tronderenergi.no/produksjon/kraftverk/bessakerfjellet

2.3 Machine Learning

Machine learning (ML) is a collection of methods that can detect patterns in data with little human intervention, and use these patterns to understand new, unseen data (Murphy, 2012).

2.3.1 Supervised and Unsupervised Learning

ML is mostly divided into two types, **supervised** and **unsupervised** learning (Murphy, 2012).

Supervised learning approaches aim to learn a mapping function from a sample x to a **target variable** y. Consider a data set consisting of N samples or observations on the form $D = \{(x_i, y_i)_{i=1}^N\}$, where x is the input and y the *known* target variable (output). D can then be used to learn the mapping function. Each data sample x_i consists of S **features** — sometimes referred to as **attributes** or **parameters**. These features have numbered values for the N observations. The target variable can be a continuous number e.g. representing the *Remaining Useful Life (RUL)* of a component, or a discrete, categorical value like *Fault1*. An approximation function or **model** $\hat{f}$ is the result from training on the data D. $\hat{f}$ is then capable of predicting target values for new samples $\hat{f}(x_0) = y_0$ (James et al., 2013).

Common supervised machine learning algorithms or models include tree based models (Breiman et al., 1984), support vector machines (James et al., 2013), K-nearest neighbours (James et al., 2013), and neural networks (Mitchell, 1997).

The unsupervised learning approach differs from supervised learning because the target variable is unknown. The data only consists of inputs, $D = \{(x_i)_{i=1}^N\}$ (Murphy, 2012). The goal with unsupervised learning is to discover interesting patterns in the data. *Clustering* is an unsupervised learning technique where the data samples x_i are grouped together (James et al., 2013).

The machine learning methods presented in this thesis will all be supervised. Hence, supervised ML methods will be the focus from this point on.

2.3.2 Regression and Classification Problems

The target variable y_i in supervised problems can either be of a quantitative or categorical value (James et al., 2013). Quantitative values are numerical, e.g. the pricing of a house. Problems that predict quantitative target values are called regression problems. Categorical values have a discrete number of possible **classes** (James et al., 2013). Classification problems have categorical values as the target variable. The classes could either be **binary** or **multi-class** (James et al., 2013). An example of a binary classification problem could be the presence of a tumour in an image with the classes *Tumour*, *No tumour*.

This thesis deals with a multi-class, classification problem.

2.3.3 Machine Learning Process

The process of machine learning can be described in the following four steps (Stetco et al., 2019).

1. Data acquisition and preprocessing

Data acquisition and pre-processing includes fetching data, data analysis, removal of noise, filling in missing values, and re-sampling.

2. Feature selection and extraction

In a data set, sample x_i has the set of features $x_i = (x_{i1}, x_{i2}, ..., x_{ip})^T$. Some ML methods handle a high dimensional data set well (many features/parameters), whilst others do not. Therefore, it is sometimes an important step to filter irrelevant or redundant parameters from a data set using feature **selection** methods. Feature **extracting** is another way of reducing the dimensionality of a data set, by creating new features based on the existing ones. Principal Component Analysis is one common feature extraction method written about by James et al. (2013).

3. Model selection and training

In this step, a suitable ML model is chosen for the problem. In machine learning and statistics, there is not one single model which performs better than all other for all data sets (James et al., 2013). This concept is known as the *no free lunch theorem* (James et al., 2013). Finding and selecting the best method in machine learning problems is therefore one of the most challenging tasks (James et al., 2013).

4. Evaluating model performance

In order to evaluate a model, the data set is often randomly split into two parts- a training set and test set (Kohavi, 1995). The training set commonly consists of $2/3$ of the data samples, while the test set contains the remaining $1/3$ samples. After having selected a machine learning model, e.g. a decision tree, the training set is used to train the model. The test set is then applied to the model to evaluate its performance. The test set is used to compare the model's decisions with the actual known target values (since the target y is known). This splitting method is known as the hold-out method.

After a model $\hat{f}$ is fitted to the training set, the performance of the model can be **evaluated** on the test set. Now, different machine learning models, e.g. decision trees, SVMs, neural networks can be compared side by side. The best performing model can be selected for further use.

The reason why a model $\hat{f}$ is tested on a test set is to evaluate how it performs on unseen data. That is, data it has not been trained on, in which the engineer already knows the target outcome y. Thus, the test data is applied to the model since it resembles unseen data.

A standard approach in ML for evaluating performance is called **cross-validation**, and is considered a better method than using the simple hold-out method described above. Cross-validation is also used for tuning ML models in a step called **validation**. Validating and evaluating a model using cross-validation is described in section 2.3.5.

2.3.4 Machine Learning Models For Classification

In this section, relevant supervised learning models applied in this thesis will be presented.

Decision Tree

A decision tree is a non-parametric, tree structured machine learning method for regression and classification problems. A trained decision tree is able to classify new samples by traversing the tree nodes sequentially. An example of a decision tree can be seen in Figure 2.2. This tree predicts whether a medical patient has a certain diagnosis.

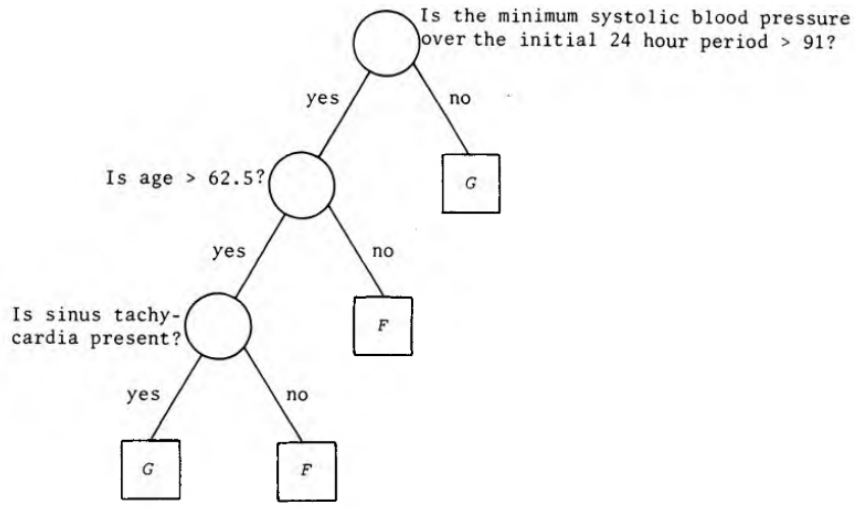


Figure 2.2: Example of a simple learned decision tree from (Breiman et al., 1984)

A decision tree is built from a single node that contains the complete data set. The starting node is then split recursively, often into two branches by selecting

an attribute that best classifies the samples. This attribute is selected based on a splitting criterion.

After a decision tree is built, it can classify unseen samples by comparing the feature values of that sample to the thresholds that were decided when the model was fitted to the training data. Decision trees classify samples by following the branches of the tree from the top (root node) to the bottom (leaf nodes). Each internal node corresponds to a feature in the training data and has a certain threshold that is compared to the sample's attribute value. The final leaf node the tree "ends up on" is the chosen class for the sample.

Decision trees are usually built or learned in a top-down fashion, where an attribute is selected to "split" on in each node. Common algorithms for decision tree learning include ID3, C4.5 and CART (Rokach and Maimon, 2005). These methods differ in how and what order the features in the data set are split on. The data is split such that every data sample is grouped in a subset in a leaf node (Molnar, 2019).

Rokach and Maimon (2005) stated that an impurity based criteria is a common way of selecting the attribute to split on. An often seen impurity based splitting criterion is information gain, which uses the entropy measure. (Rokach and Maimon, 2005). Entropy can be viewed as a measure of impurity for a data set at a node.

The procedure of selecting the next feature to split on using information gain will now be presented. In a set of data with N samples and where p_i is the number of samples of class i, the following ratio can be introduced for every class at a certain node T (Abdallah et al., 2018).

$$p_i(T) = \frac{x_i}{N} \tag{2.1}$$

Nodes with e.g. $p_0 = 1$ have complete purity, meaning only samples of one class exist in that node.

With these ratios for a data set, the impurity index I_d can be calculated with the information entropy *H(T)*

$$H(T) = \sum_{n=i}^{N} -p_i \log_2 p_i \tag{2.2}$$

where N is all possible classes and T is the set of samples at a given node.

With the information entropy H(T), one can calculate the information gain I_g for each attribute. Essentially, information gain is the entropy *before* a split subtracted with the entropy *after* the split. It is desirable to select the attribute to split on which results in the greatest information gain. Therefore, I_g is calculated

for all attributes in the set of samples in node T. (Abdallah et al., 2018). Information gain is given by (Abdallah et al., 2018).

$$I_g(T, M) = H(T) - \sum_{m \varepsilon Values(M)} \frac{\mid T_m \mid}{\mid T \mid} I_d(T_m) \tag{2.3}$$

T is here also the set of samples at a given node. M is one of the possible attributes the node may be split on. $Values(m)$ is the set of possible values for attribute M. T_m is the subset of T for which attribute M has value m. e.g. If the splitting attribute M is equal to *Rainy* with two possible values *yes* and *no*, the samples in T will be split into two subsets $Rainy_{yes}$ and $Rainy_{no}$. $\frac{|T_m|}{|T|}$ is the fraction of samples that belong to T_m.

Mitchell (1997) stated in his book *Machine Learning* that information gain can be viewed as a measure of the effectiveness of an attribute in classifying the training data (Mitchell, 1997).

CART is an example of a tree-building algorithm proposed by Breiman et al. (1984) (Breiman et al., 1984). The method constructs a binary tree, where each node has exactly two outgoing edges (Breiman et al., 1984). Each node is split on the twoing criteria (Breiman et al., 1984).

Decision tree models are to a large extent interpretable to humans, especially because the resulting path down the tree can be used as a simplified rule set, which increases the interpretability to humans (Mitchell, 1997).

The complexity of a decision tree is usually measured based on either the total number of nodes, the number of leaves (terminal nodes), tree depth or the number of attributes used (Rokach and Maimon, 2005).

Mitchell (1997) stated that decision tree models are appropriate to use on data containing errors or with some missing attribute values. Therefore, decision trees could be suitable for real word sensor data like SCADA data (Mitchell, 1997).

Random Forest

Breiman (2001) stated that a collection of decision tree classifiers often have better accuracy than a single tree classifier. Breiman (2001) introduced a classifier named random forest which consists of an ensemble of decision trees. When classifying a sample, all the individual tree classifiers vote to determine the final class label as illustrated in Figure 2.3.

The Random Forest method used in this thesis uses bagging (Bootstrap Aggregating) for training the trees in the random forest model. Bagging, introduced by Breiman (1996), is a method for building ensemble classifiers e.g. trees. Each

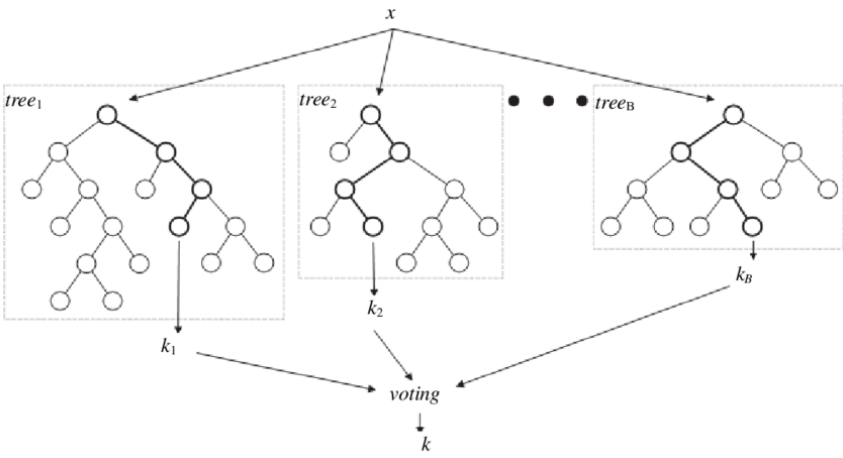


Figure 2.3: Three trees in a random forest classification problem. The trees collectively vote on the final class prediction. Source: Breiman (2001).

classifier is built using a random subset of the samples in the training set (Breiman, 1996). The random subset of samples are made without replacement, meaning that the same sample may appear in multiple subsets (Breiman, 1996).

From a random forest classifier it is possible to extract the importance of each variable or feature (Breiman et al., 1984). Breiman et al. (1984) defined this **variable importance** as the total decrease in node impurity averaged over all trees of the ensemble (Breiman et al., 1984). The Python library scikit-learn by Pedregosa et al. (2011) also used this definition when implementing the feature importance attribute for their random forest module, *sklearn.ensemble.RandomForestClassifier* (Pedregosa et al., 2011).

Support Vector Machines

Another machine learning method for classification is support vector machines (SVMs). SVMs are often thought of as one of the best performing "out of the box" classifiers (James et al., 2013), and is commonly applied with WT data (Stetco et al., 2019).

SVMs are based on the notion that a hyperplane divides the data into two classes. A new observation X can be classified by checking which side of the decision boundary it falls into, illustrated in Figure 2.4. The most fitting hyperplane is the one that best separates the class samples. This is called the *optimal separating hyperplane*. The simplest form of a SVM creates a linear decision boundary. This boundary can either *completely* separate the data samples of different classes, or account for some *slack* (Figure 2.4). SVMs can also be applied even though the data

is not linearly separable. Non-linear SVMs are used for such problems. Furthermore, SVMs are applicable for data with more than two classes (James et al., 2013).

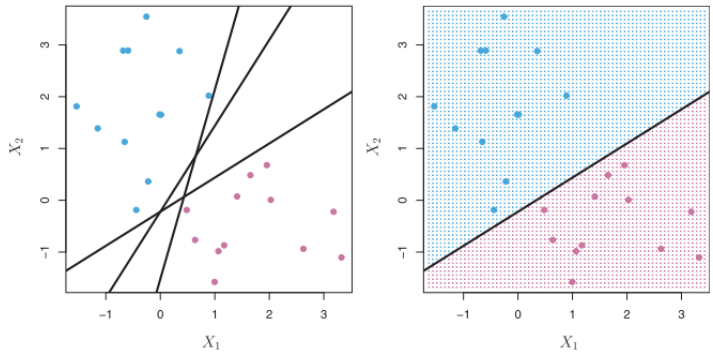


Figure 2.4: Left: Infinite number of lines separating the 2D space. **Right:** Decision boundary for training set consisting of two classes, coloured blue and red. Source: (James et al., 2013)

To understand how a hyperplane is calculated and used for classification, we will look at the hyperplane equations. In two dimensions (p=2), meaning that there are two features/variables in the data, the hyperplane is a line and can be defined as

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0 \tag{2.4}$$

for parameters β_0, β_1, and β_2. In three dimensions (p=3), the hyperplane is a plane, which is harder to visualise. If observation $X = (X_1, X_2, ..., X_p)^T$ lies on the hyperplane, it satisfies the equation

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + .. + \beta_p X_p = 0 \tag{2.5}$$

However, if point X does *not* satisfy 2.5, then it satisfies either

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + .. + \beta_p X_p < 0 \tag{2.6}$$

or

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + .. + \beta_p X_p > 0 \tag{2.7}$$

We will first consider the case where all observations are separable by a hyperplane (Figure 2.4). To explain how equation 2.6 and equation 2.7 are used to classify samples, consider a data set $\mathbf{X}$ with n data samples in p-dimensional space. If there exists a line in two-dimensional space which perfectly separates all the training samples of class 1 and -1, one can classify new samples. This is seen in Figure 2.4. In the same figure, there are two possible classes, i.e. a binary classification problem.

The separating hyperplane defines a classification model *f*. f is then able to predict a new observation x by examining the sign of $f(x) = \beta_0 + \beta_1 X_1 + \beta_1 X_2 + ... + \beta_p X_p$ (James et al., 2013). The further away the new sample is to the separating line, the more certain one can be that the sample is classified correctly (James et al., 2013).

The hyperplane is constructed based on the training data with known targets y_i. The optimal separating hyperplane is the hyperplane which has the furthers minimum distance to the training observations. Thus, the selected hyperplane only depends on a small set of the observations in the training set. This is seen in Figure 2.5, where one red sample and 2 blue samples define the *margins* next to the decision boundary.

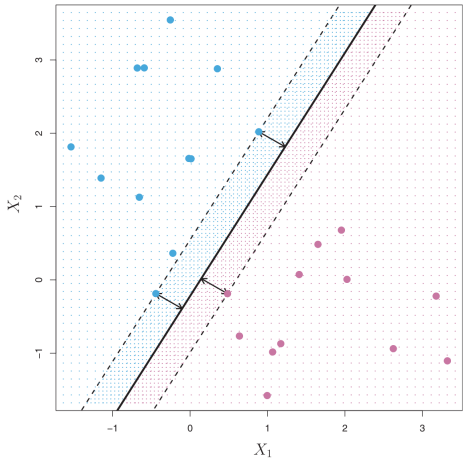


Figure 2.5: The margins on either side of the separating hyperplane. The margin distance is the furthest minimum distance to the training observations. Source: (James et al., 2013)

Finding the optimal hyperplane is a constrained optimisation problem which we will not go into in this thesis. The steps are walked through in this document [2] at *web.mit.edu*.

The non-separable case occurs when the training observations are not perfectly separable as in Figure 2.5. To handle this, a tuning parameter is introduced. This parameter allows some observations to be placed on the *incorrect* side of the margin, or even on the incorrect side of the hyperplane (James et al., 2013). In turn, this gives the SVMs greater robustness and increased performance when classifying most of the training samples (James et al., 2013). See Figure 2.6 for an illustrative example.

[2] http://web.mit.edu/6.034/wwwbob/svm-notes-long-08.pdf

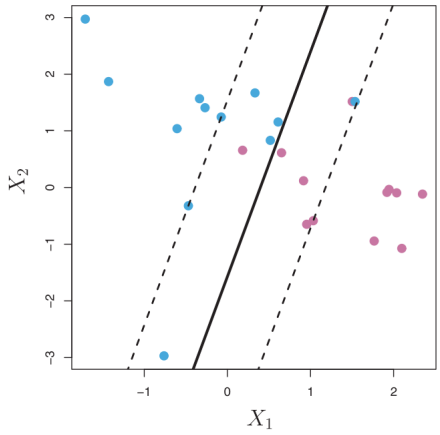


Figure 2.6: SVM allowing samples to be miss-classified in the margins. Source: (James et al., 2013).

When dealing with high dimensional data (many features), it is common to encounter situations where the data is not linearly separable (James et al., 2013) Figure 2.7. For these cases, non linear SVMs can be used. This is done by using a kernel function which maps the data into a higher dimensional feature space to make it possible to perform the linear separation (James et al., 2013).

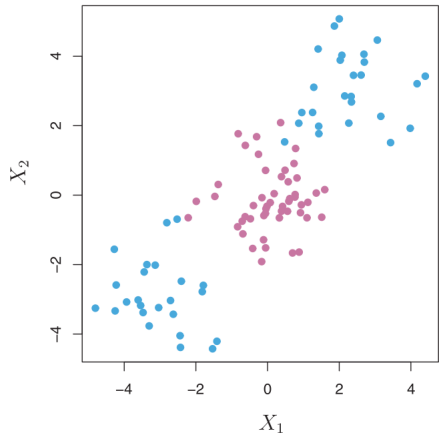


Figure 2.7: An example of a non-linearly separable data set. Source: (James et al., 2013)

We have briefly showed how SVMs are trained and how they classify new samples when there are two possible classes. However, SVMs can be extended to a more general case, to handle an arbitrary number of classes K. A common technique is called *one-versus-one*. The one-versus-one technique was used when doing experi-

ments with SVMs in this thesis.

A one-versus-one approach builds $\binom{K}{2}$ SVMs and uses each of these SVMs when classifying a new sample. The final classification is the one assigned most frequently by the SVMs. For example, if three classes existed in a data set $[Blue, Red, Green]$, $\binom{3}{2} = 3$ SVMs would be constructed, i.e. one for comparing the classes *Green* to *Blue*, *Green* to *Red*, and *Blue* to *red*.

2.3.5 Machine Learning Performance Metrics

In the machine learning process section, 2.3.3, it was mentioned that performance measures are used in both the validation and performance measuring steps. The metrics are important because they allow us to compare the same model type (with different hyper-parameters) and different models with each other. Thus, one is able to select the optimal machine learning model for a problem. In this section, some common performance measures are introduced.

Confusion Matrix

Evaluating the performance of a classification model is often done with a confusion matrix as seen in Figure 2.8 and 2.9 (Chawla et al., 2002). In confusion matrices, one can see the actual class of the samples and which class the machine learning model predicted the sample as. A confusion matrix for a binary class problem such as *fault*/*no fault* is seen in Figure 2.8. The abbreviations are explained in Table 2.3. Confusion matrices can also be extended to fit multi-class problems, as seen in Figure 2.9. The figure specifically considers the class c_k. Confusion matrices can also be normalised, which is relevant for imbalanced data sets (Pedregosa et al., 2011). Each row (ground truth) of a normalised CM sums to 1.

	Predicted Negative	Predicted Positive
Actual Negative	TN	FP
Actual Positive	FN	TP

Figure 2.8: Confusion matrix for a two-class problem

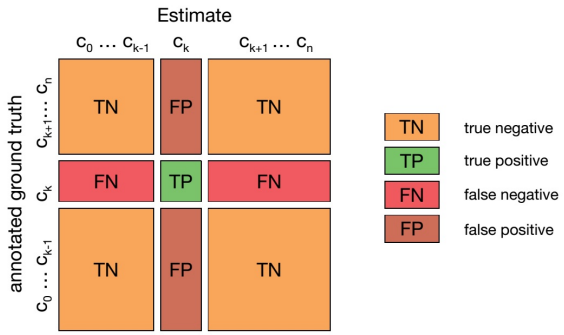


Figure 2.9: Confusion matrix for a multi-class problem. Source: (Krüger, 2018)

Abbreviation	Meaning
TN	Number of negative samples classified correctly
TP	Number of positive examples correctly classified
FP	Number of negatives incorrectly classified as positives
FN	Number of positive samples incorrectly classified as positive

Table 2.3: Abbreviations for the confusion matrix.

Accuracy, Recall and Precision

With the abbreviations in Table 2.3 in mind, model accuracy can be introduced. *Accuracy = (TP + TN)/(TP + FP + TN + FN)*. The error rate is *1 - Accuracy*. These two metrics are reasonable to use when working with balanced data sets, meaning that there is an equal proportion of all classes represented in a data set. However, models trained on imbalanced data sets will have a higher number of correct predictions for the majority class than the minority, if the data has two classes. This means that the accuracy score will produce a less credible result for the model (Chawla et al., 2002). For example, if one has a test data set with 100 samples of the class *healthy*, and 20 samples of class *faulty*, a classifier which always classifies samples as *healthy* will achieve a 80 % accuracy score. Thus, accuracy is not a good metric when evaluating an imbalanced data set.

Recall and precision are two other metrics. They are better suited for evaluating models with imbalanced data sets (Chawla et al., 2002).

$$Recall = \frac{TP}{TP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

Recall answers the question– of all the samples of class X, how many samples were classified by the model as class X? Precision is a measure of how many samples the model classified as class X that actually were of class X (Dumais et al., 1998).

A high number of false negatives leads to a low recall. In the context of fault diagnosis, Leahy et al. (2018) pointed out that a high number of false negatives is associated with **actual faults that are not caught**. A high number of false positives results in many **false alarms** being sent. Many false positive classified samples point to a low precision. Having a high recall is considered more important than having a high precision in condition monitoring, because false alarms are preferred over uncaught errors (Leahy et al., 2018).

Recall and precision can be used to calculate the F1 score, which is the harmonic mean of the metrics.

$$F1 = \frac{2TP}{2TP + FP + FN}$$

Cross Validation

As mentioned in section 2.3.3, data sets are often split into training and testing sets (Figure 2.10). The training set can be used for training a model, and the test set for measuring the performance of the model.

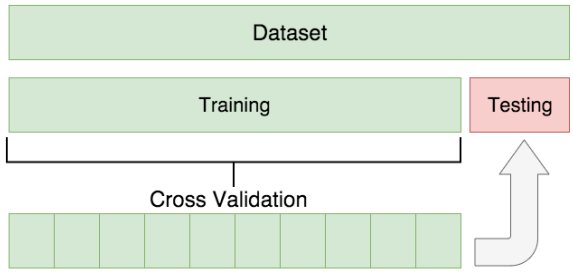


Figure 2.10: Train/test split and k-fold cross validation

A common procedure to evaluate a model is k-fold cross validation (CV) (James et al., 2013). This technique partitions the data set into k equally sized partitions. In k iterations, each partition is used as the test set exactly once, while a model is trained on the k-i other partitions (James et al., 2013). Breiman et al. (1984) presents the steps for the cross-validation technique. The pseudo code for k-fold cross-validation is presented in algorithm 1. Note that the pseudo code is written using the accuracy metric described in section 2.3.5. It returns the average accuracy across all k partitions. Any of the presented measures can be used with CV, or a combination of these.

Algorithm 1 K-Fold Cross-validation

Input: Data set D, integer k
Output: $avg\_accuracy$ (Average accuracy for model)

```
1:  function kFoldCrossValidation(k, D)
2:      accuracy[] ← Instantiate empty array of length k
3:      Randomly partition D into k disjoint subsets of equal size P_i...P_k
4:      for i ← 1 to k do
5:          T_i ← (D − P_i)
6:          train model on P_i
7:          test model on T_i
8:          accuracy[i] ← accuracy on Ti
9:      end for
10:     avg_accuracy ← Average(accuracy[])
11:     return avg_accuracy
12: end function
```

Simply splitting the data into one training and one test part could mean that a poor representation of the data is used for testing, which could yield an inaccurate performance score. Cross validation is used to evaluate a model with as little bias as possible.

Cross validation is also used for tuning a model's input parameters or hyper-parameters (Stetco et al., 2019). These are parameters set by the engineer when the model is created, e.g. the depth d of a decision tree. Tuning parameters like the depth of a tree is important for the performance. If a tree is too deep, it is prone to over fitting, meaning that the DT has good performance on training data, but not on new, unseen data (Miller, 2019). If a DT is too shallow, it under fits the data and fails to learn the underlying patterns (Mitchell, 1997).

Double cross validation is a technique in which a single model's hyper-parameters is tuned and then tested. It is similar to the cross validation technique described above. The data is split into training and testing. The training data is then validated using cross validation for the same model with different hyper-parameters. The best performing parameters is chosen for the model. The test set is then applied to the validated model to *evaluate* the performance, also using cross-validation.

Kohavi (1995) stated that a 10-fold stratified cross validation is the best method for validating and evaluating real world data. The folds in stratified cross validation are stratified so that they contain approximately the same proportions of data samples in each fold as in the original data set (Kohavi, 1995). This is important when evaluating models on imbalanced data, which will be more discussed in the following section.

2.4 Imbalanced Data

Leahy et al. (2018) stated that data imbalance is often an issue when working with real world data such as SCADA data. Chawla et al. (2002) states that a data set is imbalanced if the classes are not approximately equally represented (Chawla et al., 2002). E.g. each class label in a balanced data set of three classes should therefore have 33% of the total data samples, or rows in the data matrix.

2.4.1 Handling Imbalanced Data

There exists a lot of research on the topic of handling imbalanced data sets for both binary and multi-labelled classification problems (Tahir et al., 2012). The most common methods involve under sampling or over sampling the original data set (Tahir et al., 2012). The two methods aim to balance out the class populations by adding more samples of the minority class or removing samples from the majority class. In this section, three ways of handling imbalanced data will be presented: **random undersampling, over sampling using SMOTE**, and **introducing class weights**.

Random Under Sampling

The first technique is random under sampling (RUS). RUS is an intuitive way of removing samples from the population of a majority class. This can be done by randomly selecting samples from the majority class and deleting them from the data set (Tahir et al., 2012). Tahir et al. (2012) also stated that removing samples at random could lower the performance of a classification model trained on the under sampled data, because important data could potentially be lost when removing data points sampling.

Over Sampling with SMOTE

The second technique is an over sampling technique called synthetic minority oversampling technique (SMOTE), introduced by Chawla et al. (2002). Instead of over sampling by duplicating samples of the minority class with replacement, SMOTE creates "synthetic" samples (Chawla et al., 2002). This technique forces the decision region of the least represented sample class to become more general. More general means that a model is able to correctly classify samples beyond the training data (Mitchell, 1997).

The Synthetic Minority oversampling technique takes three input parameters: the minority class samples T, how many percent the class population should increase N, and the number of neighbours k that should be used when creating synthetic examples. (Chawla et al., 2002). The pseudo code can be studied in Appedix D.1.

SMOTE works by generating a desired number of new samples to even out class imbalance. If a minority class has 200 samples, and the user would want to increase

the samples size to 600, SMOTE will create three new samples for every single one of the 200 samples in the data set.

To summarise the pseudo code for SMOTE; SMOTE starts by (1) iterating over the minority samples. (2) For every sample s, one other sample from the k nearest samples to s is selected at random. (3) with this randomly selected sample, the difference between the feature vectors are calculated. The difference vector is then multiplied by a value from 0 and 1. This vector is now added to sample s's feature vector. This "new" sample is added to an array of synthetic samples. Point (2) and (3) are done again for the same sample s, depending on the desired percentage increase N.

Introducing Class Weights

The third technique for handling imbalanced data sets is by using class weights. The Python library Scikit-learn has an input parameter for machine learning models called class_weight (Pedregosa et al., 2011). By setting this parameter to *balanced*, the class weights are adjusted inversely proportional to class frequencies in the input data (Pedregosa et al., 2011). For a data set with three possible classes, the class weights are calculated and returned as a 1-dimensional array with class weights for each class in the data set. The equation for calculating the class weights is shown in equation 2.8.

$$ClassWeights = \frac{N}{N_{classes} \cdot occurences} \tag{2.8}$$

Where N is the total number of samples in the data, $N_{classes}$ is the number of classes represented in the data set.*occurrences* is a 1-dimensional array of the amount of occurrences of each class. The indices in the array correspond to the classes. E.g. a data set with three classes *0,1 and 2* represented will have the occurrences array looking like [21,130,109]. Class 1 is then occurring 130 times in the data set. In this example, the class weight for class 0 will be the highest of the three.

The class weights are used to set a penalty parameter C_i shown in equation 2.9 for each class (Pedregosa et al., 2011). The penalty parameters determine how much the model will be penalised for miss-classifying the samples.

$$C_i = C \cdot Classweight_i \tag{2.9}$$

C is a constant set to determine the penalty by the model being trained. As seen from equation 2.8 and 2.9, if all classes are equally represented in the data set, the penalty parameter is equal for all classes.

2.5 Explainable Artificial Intelligence

DARPA (Defense Advanced Research Projects Agency) defines explainable artificial intelligence as a way for a learned model to explain its decision to a human user (Siekmann and Wahlster, 2019).

AI and machine learning methods have experienced a huge success recently due to their growing performance. This success is mainly driven by the development of SVMs, ensemble methods such as Random Forest, reinforcement learning, probabilistic graph models, and deep learning neural nets (Siekmann and Wahlster, 2019). These methods construct models in their internal representation that enable them to make complex decisions. Although their performance is remarkable, these methods are opaque and provide very little insight as to how they arrive at their conclusion.

Siekmann and Wahlster (2019) presents a relationship between models' performance and their level of explainability. It is shown in Figure 2.11. As seen in the figure, complex models such as deep learning neural nets have an impressive performance but poor explainability. Simpler models on the other hand, such as DTs, typically perform worse but are associated with a higher explainability.

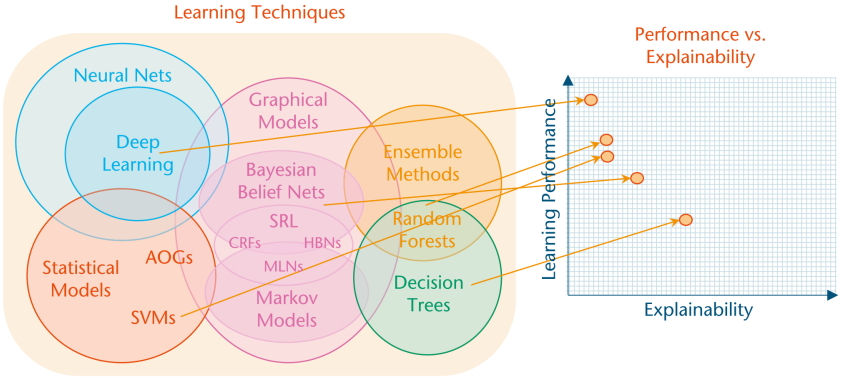


Figure 2.11: Learing performance VS explainability for different learning techniques. Source: Siekmann and Wahlster (2019)

2.5.1 Why Explainable AI?

Machine learning methods are capable of obtaining an impressive performance, however, for some samples they perform poorly. These have to be combated and pruned to improve the model (Siekmann and Wahlster, 2019). It raises the question; how to distinguish bad from good predictions, and how are they to be corrected without understanding the model's reasoning? Explainable AI aims to tackle this

problem.

Trust in Models

DARPA's team argues that understanding the predictions is essential when applying them in the real world. From this thesis perspective, this applies to maintenance technicians and wind turbine operators. They need to know when to trust the prediction and when not to, or it could be costly.

GDRP

The European Union's General Data Protection Regulation (GDPR) went into effect in 2018, protecting consumers data regarding how their data is used. These new rights increase the need for explainable AI and drive further development.

For instance, by article 14; companies using automated decision making are required to provide meaningful information about the logic involved[3]. Article 22 also regards automated decision making, like applying ML techniques to handle loan applications in banks. It states that a consumer has the right not to have a decision solely be based on an automated process, and has the right to express their view and contest the decision[4]. This is a challenge for traditional machine learning technologies and pushes the industry into developing explanations for their decisions.

2.5.2 Explainable Methods

Molnar (2019) describes two ways of making machine learning models interpretable. The first one is to use **interpretable models**, the other is called **model agnostics methods**.

Interpretable Methods

Interpretable models are to some extent understandable to humans and include methods such as decision trees and linear models. In the case of linear regression, the weights of the trained function are visible and often interpretable to humans. For decision trees, the tree split is provided in a rule-based way. This makes it easy for humans to understand why a final prediction is made, by traversing down the tree.

Model Agnostic Methods

Model agnostic methods are often paired with "black box" machine learning models like Neural Nets (NNs) and SVMs. As mentioned in the introduction of section 2.5, these ML models are not interpretable to humans, i.e. they can not be interpreted

[3] https://analyticsindiamag.com/8-explainable-ai-frameworks-driving-a-new-paradigm-for-transparency-in-ai/

[4] https://www.fico.com/blogs/gdpr-and-other-regulations-demand-explainable-ai

by looking at the trained weights or traversing a tree.

Model agnostic methods are usually wrapped around the trained ML model and produce an explanation by examining the feature input sent into the model and observing the output. They do this without accessing the model internals; thus, they can be applied on any machine learning method.

LIME

Ribeiro et al. (2016) presented a way of performing model agnostic explanations called LIME (Local Interpretable Model-agnostic Explanations). The overall goal of LIME is to generate local explanations that are easily interpreted by humans for any machine learning method (Ribeiro et al., 2016). Ribeiro et al. (2016) claims that an explanation must at least be locally faithful to be meaningful. This means that it must correspond to how the model behaves when predicting instances in the vicinity of the instance being predicted.

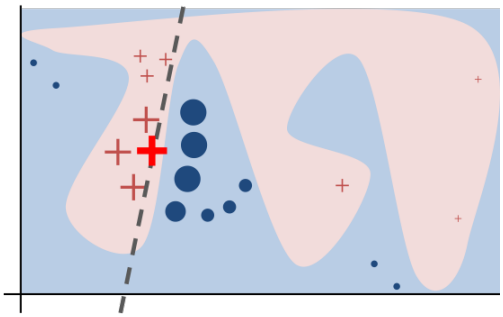


Figure 2.12: Example of locally simple model behaviour. The bold, red cross is the new data instance. Permuted data is shown as blue dots and red crosses. The size of the permuted data points increases the closer they are to the new data point. Source: (Ribeiro et al., 2016)

Ribeiro et al. (2016) argue that even though a model may be very complex, it is much simpler locally, as illustrated in Figure 2.12. Ribeiro et al. (2016) state that it is possible to generate an explanation by approximating the complex model locally using an interpretable model. As such, LIME attempts to understand the underlying model by perturbing an instance, thereby creating a new data set, and observing the predictions on this set. How LIME perturbs the instance depends on the use case, whether it is image recognition, text classification or tabular data; meaning classification/regression based on numerical or categorical features. The aim of this thesis is to classify faults based on numerical sensor inputs, and thus LIME will be further explained with that in mind.

According to Molnar (2019), the samples making up the perturbed data set are taken from the training data's mass center for each feature and not around the instance. The distance between the instance and the new data points is calculated and used to capture its proximity to the original instance. The perturbed data set is fed through the underlying model and predictions are made. LIME examines the top m features best describing the complex model and creates a new data set containing the perturbed data with these features and their predictions. The number of m is chosen by the user.

Next, LIME fits an interpretable model, e.g. decision tree, linear regression, to the new data set, which is weighted by the distance calculated earlier to capture local behaviour. This is illustrated in Figure 2.12 by the data points' size. Finally, an explanation is extracted from the interpretable model by a feature selection method, e.g. looking at the weights of a linear model or the splits in a decision tree.

The following feature selection methods for linear models are implemented in LIME; *forward selection, highest weights, lasso path* and *auto*. Forward selection is when each feature is added iteratively and the most decisive features are kept. It is a computationally costly method when the number of features is high. The highest weight method selects the features with the highest product of weight times original data point. The lasso path uses lasso regression for variable selection. Auto applies forward selection when $m < 5$ and is set to highest weights otherwise.

Mathematically, the explanation for a given instance x is generated by the following expression:

$$explanation(x) = \arg\min_{g \in G} \mathcal{L}(f, g, \pi_x) + \Omega(g) \tag{2.10}$$

Variable	Description
f	Original predictor
g	Expalnation model
x	Instance explained
π_x	Proximity measure
$\mathcal{L}(f, g, \pi_x)$	Locality-aware loss
$\Omega(g)$	Model complexity

Table 2.4: Equation variable description

An explanation is produced by minimising what Ribeiro et al. (2016) calls the locality-aware loss, $\mathcal{L}(f, g, \pi_x)$ and the complexity of the interpretable model, $\Omega(g)$. The locality-aware loss measures how close the explanation model, g, is to the prediction of the model, f, taking the proximity of the data points to the instance into account. The complexity of the interpretable model is for example the number of features in a linear model or the depth of a decision tree. As the model complexity

is determined by the user's selection of m, LIME only minimises $\mathcal{L}(f, g, \pi_x)$ in practice.

Lime Example

Ribeiro et al. (2016) created an example of a LIME implementation showing its benefits and what it is able to uncover called the *Husky or Wolf Classifier*. It was intended as a bad classifier, trained on a data set where all pictures of wolfs had a snowy background and all the huskies were on grass. Rather than distinguishing between husky or wolf, the model learned that wolfs had snow in the background and huskies were on grass. This was obviously incorrect but by solely looking at the performance measures, the model appeared to perform well. However, by inspecting LIME's explanation (Figure 2.13) it became clear that the model was unreliable. Being able to asses the features used by a model is of great importance in other areas as well. Root cause analysis for faults is another example of an application.

(a) Husky classified as wolf (b) Explanation

Figure 2.13: Original image and explanation of a bad model's prediction of *Husky or Wolf*. Source: (Ribeiro et al., 2016)

2.5.3 Evaluating Explanations

In traditional ML, there exists acknowledged evaluation techniques and performance measures such as accuracy, precision, and recall (2.3.5). However, there is no universal evaluation technique for explainable AI yet. So, how do we evaluate explanations and new explainable AI approaches?

DARPA has formulated an explanation evaluation framework (Siekmann and Wahlster, 2019). It includes five main categories which measure the **explanation effectiveness**, and is seen in Table 2.5.

Category/measure	Description
Explanation goodness	Features of explanations assessed against criteria for explanation goodness
Explanation satisfaction	User's subjective rating of explanation completeness, usefulness, accuracy, and satisfaction
Mental model understanding	User's understanding of the system and the ability to predict the system's decisions/behavior in new situations
User task performance	Success of the user performing the tasks for which the system is designed to support
Appropriate Trust and Reliance	User's ability to know when to, and when not to, trust the system's recommendations and decisions

Table 2.5: Explanation Effectiveness Questionnaire. Source: (Hoffman et al., 2018b)

Hoffman et al. (2018b) present a checklist for evaluating the *explanation goodness*, and a survey for evaluating the *Explanation satisfaction* for an explainable model. They conclude that the explanation goodness checklist is intended to be used by researchers to evaluate XAI systems. The checklist is seen in Table 2.6. In contrast, the explanation satisfaction survey is intended for users or research participants that have worked with the XAI system being explained. This survey is placed in Appendix section C.

Another way of differentiating between the two evaluation criteria *explanation goodness* and *explanation satisfaction* is to describe goodness as an **a priori** judgement, and the satisfaction as an **a posteriori** judgement (Hoffman et al., 2018b). A priori in this context refers to knowledge about a XAI system *before* it is used in context, and a posteriori refers to knowledge after the system is used in context. A context in this sense could be after after an explainable AI system is deployed to e.g. a wind farm.

Explanation Goodness Checklist	Yes	No
The explanation helps me understand how the model works		
The explanation of how the model works is satisfying		
The explanation of the model sufficiently detailed		
The explanation of how the model works is sufficiently complete		
The explanation is actionable, that is, it helps me know how to use the model		
The explanation lets me know how accurate or reliable the model is		
The explanation lets me know how trustworthy the model is		

Table 2.6: Yes/No checklist for evaluating Explanation Goodness. Source: (Hoffman et al., 2018b)

User Satisfaction - In Depth

In addition to the *user satisfaction* survey placed in appendix C, Miller (2019) presents some other interesting findings regarding explanations for machine learning models. In short, the four findings Miller (2019) thinks are important for user satisfaction in explainable AI are:

- **Explanations should be contrastive.** Humans do not want a complete explanation for a prediction. Instead, they want the explanation to include differences to another sample prediction.
- **Explanations should be short.** Each explanation should have only 1 to 3 reasons.
- **Probabilities are not relevant.** Referring to probabilities when conveying explanations are not as compelling to humans as referring to causes.
- **Explanations should be suitable for the application audience.** Will the explanations for a CM system be used by a domain expert or non-technical person?

Chapter 3

Literary survey

3.1 SCADA Data and Machine learning

Leahy et al. (2018) trained SVMs with SCADA data from wind turbines to detect, diagnose, and predict faults (Leahy et al., 2018). SVMs were built using different techniques to determine three levels of fault classification. The first level was fault detection. Here, the aim was to classify fault/no-fault. The second level was fault diagnostics, where specific faults were detected and diagnosed. The three faults they decided to classify in the diagnostic part were generator heating faults, excitation faults, and feeding faults. The third and most complex level was fault prediction. The goal of the fault prediction was to classify and diagnose faulty points ahead of time. Generator heating faults and excitation faults showed most promising results in the diagnostic part, so Leahy et al. (2018) also selected these for predictions ahead of time.

In the fault diagnostic part, Leahy et al. (2018) wanted to classify faults on "unseen" data points. That is for instance a data point recorded at a specific time t with the classification *Generator Heating Fault*. This level of classification is the one that is focused on in this paper, since the purpose is to explore *why* a data point was given a certain classification.

3.1.1 Imbalanced Data

Leahy et al. (2018) stated that the data set was imbalanced, meaning that the majority of the data samples were "healthy" samples. To fix this issue, a range of methods were applied to the data. The methods included applying class weights, under sampling, over sampling, and using ensemble learners. For fault detection, the best balanced performance on the SVM model was seen when using class weights. For this method, a recall of .83 was achieved. The precision was at 0.04.

In the fault diagnosis part, the random under sampling method (RUS) gave the best overall performance. With RUS, 89% of generator heating faults were caught (recall of 0.89) with a precision of 0.73. 97% of excitation faults were caught with a precision score of 0.04. The model captured 67% of the feeding faults. The precision score for the feeding faults was at 0.22.

Low precision was common in the models trained by Leahy et al. (2018). However, having a high recall is advantageous because there are higher risks affiliated with missing a fault as opposed to a false alarm (Leahy et al., 2018).

Random under sampling could be a promising sampling method for handling data imbalance when performing fault diagnosis, as seen for the results above. This finding is relevant for our own research.

Chawla et al. (2002) concluded their literature study by saying that under sampling the majority class improves recognition of the minority class, compared to oversampling the minority class (Chawla et al., 2002). In the same paper, the SMOTE technique is presented as an alternative approach to oversampling the minority class with replacement. SMOTE is described in section 2.4.1.

3.1.2 Explainable Models with SCADA Data

Abdallah et al. (2018) explored how decision trees can be applied to data streamed from SCADA from 48 wind turbines, in order to explain the sequence of events that led to a vibration fault. This sequence of events is also called root cause analysis. In the paper, root cause analysis was done by examining the sequence of events that led to a fault classification in the trees. The fault they were examining was *excessive vibration*. Furthermore, Abdallah et al. (2018) proposed that these sequences could be used by engineers to simulate scenarios and replicate results in order to better understand the faults. Decision trees were chosen in their research because they tend to be easier to implement and interpret than other data-driven methods.

Abdallah et al. (2018) did not perform any feature extraction or feature selection methods to reduce the dimensions of the SCADA data. Instead, they manually selected the features *mean temperature of gear bearing*, *active power*, *generator RPM* and *wind speed*. The target variable was denoted *TurbineState*, which could be either *NoFault* or *Vibr*. Using this data, Abdallah et al. (2018) constructed an ensemble of trees using bagging. With a bag size of more than 10, the misclassification rate of the validation data was less than 1%.

Leahy et al. (2018) performed classification in three levels, fault detection, fault diagnosis and fault prediction. In relation to this three-levelled approach, it is clear that Abdallah et al. (2018) did the fault diagnosis part with one fault- i.e. the *Vibr* fault. However, Abdallah et al. (2018) built a tree model with greater

interpretability than the SVM model by (Leahy et al., 2018).

Mitchell (1997) stated that decision tree models are appropriate to use on data containing errors or with some missing attribute values. Therefore, decision trees could be suitable for real word sensor data like SCADA data (Mitchell, 1997).

3.2 Summary and our Research

To summarise the literature, Leahy et al. (2018) and Abdallah et al. (2018) wrote highly relevant papers for this thesis' topic. The objective of this thesis is to extend their research on fault diagnosis with SCADA data, and provide explanations for classifications using XAI methods.

A decision tree will be built as the first interpretable model, also by using an ensemble of trees like Abdallah et al. (2018) did. SVMs have shown to handle high dimensional SCADA data well (Leahy et al., 2018) and will be applied using a model agnostic method.

As stated in the introduction, section 1, the aim of this thesis is to use XAI to perform root cause analysis and provide interpretable explanations for human users.

Chapter 4

Method

In this chapter, the available data will be described in 4.1. Section 4.1.3 will illustrate how a correlation algorithm was created to detect patterns in the error log. In 4.2, the data pre-processing and labelling is described. The labelled data is then used to build the chosen ML models, shown in section 4.3. 4.4 contains technical information of how the models were built and a link to the code online.

4.1 Available Data

4.1.1 SCADA Operational Data

From the operational data from the SCADA system described in section 2.2.2, 34 parameters were selected for each turbine. These are what have been referred to as variables or features in the context of machine learning, in section 2.3. The parameters are listed in Table 4.1.

A correlation analysis of the 34 parameters was carried out, and is placed in Appendix A.1. The plot visualises any existing linear relationship between the parameters in a matrix.

Operational data for the 25 wind turbines was downloaded from the SCADA system through a graphical user interface called BazeField. Data from 25-10-2017 13:20:00 to 25-10-2019 00:00:00 was used.

After the predictor variables were chosen, inspection of the operational data showed that some of the variables were sampled at different rates. See Table 4.2. Variables like *WindSpeed* were sampled at a higher frequency than e.g. temperature variables. The temperature measurements were being updated irregularly when a different value was measured.

Chosen Parameters	
Average blade angle across A B C	NacelleDirection
ActivePower	WindDirection
Spinner-Temperature	WindVane
Nacelle-Temperature	WindSpeed
Ambient-Temperature	RotorSpeed
Log-T-Raw-TowerTemperature	Log-T-Raw-ExcitationHeatSink
Log-T-Raw-FrontBearingTemperature	Log-T-Raw-Rectifier1HeatSink
Log-T-Raw-RearBearingTemperature	Log-T-Raw-Rectifier2HeatSink
Log-T-Raw-BladeATemperature	Log-T-Raw-BladeAPitchHeatSink
Log-T-Raw-BladeBTemperature	Log-T-Raw-BladeBPitchHeatSink
Log-T-Raw-BladeCTemperature	Log-T-Raw-BladeCPitchHeatSink
Log-T-Raw-Rotor1Temperature	Log-T-Raw-BladeAPitchControlBox
Log-T-Raw-Rotor2Temperature	Log-T-Raw-BladeBPitchControlBox
Log-T-Raw-Stator2Temperature	Log-T-Raw-BladeCPitchControlBox
Log-T-Raw-Stator1Temperature	Log-T-Raw-TransformerTemperature
Log-T-Raw-NacelleAmbientTemperature	Log-T-Raw-RectifierCabinetTemperature
Log-T-Raw-NacelleControlCabinetTemperature	Log-T-Raw-ControlCabinetTemperature

Table 4.1: Chosen Parameters

To fill in the missing values (NaN), the first index in each column where a non-NaN value was found was returned.

TimeStamp	**WindVane (Average)**	**WindSpeed (Average)**	**RotorSpeed (Average)**	**NacelleDirection (Average)**	**ActivePower (Average)**	**...**
25-10-2017 13:20:00	-9.1	6.4	13.679999	163.0	382.0	...
25-10-2017 13:20:30	NaN	NaN	NaN	NaN	NaN	...
25-10-2017 13:21:00	NaN	NaN	NaN	NaN	NaN	...
25-10-2017 13:21:30	NaN	6.4	13.679999	NaN	361.0	...
25-10-2017 13:22:00	NaN	NaN	NaN	NaN	NaN	...

Table 4.3: Data set with first row filled.

After the first non-NaN value was located in each column, it was copied to the first row of that column. The operational data was now similar to that of Table 4.3. With the first row filled with values, the Python library Pandas by McKinney et al. (2010) was applied to fill the missing values. Specifically, the *pandas.DataFrame.fillna* module was used to get the result in Table 4.4.

TimeStamp	WindVane (Average)	WindSpeed (Average)	RotorSpeed (Average)	NacelleDirection (Average)	ActivePower (Average)	...
25-10-2017 13:20:00	NaN	NaN	NaN	NaN	382.0	...
25-10-2017 13:20:30	NaN	NaN	NaN	NaN	NaN	...
25-10-2017 13:21:00	NaN	NaN	NaN	NaN	NaN	...
25-10-2017 13:21:30	NaN	6.4	13.679999	NaN	361.0	...
25-10-2017 13:22:00	NaN	NaN	NaN	NaN	NaN	...

Table 4.2: Unprocessed data set

TimeStamp	WindVane (Average)	WindSpeed (Average)	RotorSpeed (Average)	NacelleDirection (Average)	ActivePower (Average)	...
25-10-2017 13:20:00	-9.1	6.4	13.679999	163.0	382.0	...
25-10-2017 13:20:30	-9.1	6.4	13.679999	163.0	382.0	...
25-10-2017 13:21:00	-9.1	6.4	13.679999	163.0	382.0	...
25-10-2017 13:21:30	-9.1	6.4	13.679999	163.0	361.0	...
25-10-2017 13:22:00	-9.1	6.4	13.679999	163.0	361.0	...

Table 4.4: Filled data set.

The process of filling the missing values was performed on data from all 25 wind turbines.

4.1.2 SCADA System Status Log

The status logs for all 25 wind turbines were also download from BazeField. The starting time for these "errors" were, similarly to the operational data, set to 25-10-2017 13:20:00. This log consisted of time series data and looked like Table 4.5. Each row in the matrix is a new error with an Alarm code and a starting and stopping time.

Turbine	Alarm	Description	Start	End	Duration
BESS-WTG01	Status_9_Substatus_8	9:8 Generator heating (Manual)	01-10-2019 09:27:00	01-10-2019 15:33:30	06:05:41
BESS-WTG01	Status_62_Substatus_30	62:30 Feeding fault	01-10-2019 15:33:00	01-10-2019 15:34:00	00:00:01
BESS-WTG01	Status_62_Substatus_30	62:30 Feeding fault	02-07-2018 08:37:00	02-07-2018 08:37:30	00:00:01
BESS-WTG01	Status_62_Substatus_30	62:30 Feeding fault	07-01-2018 15:55:00	07-01-2018 15:55:30	00:00:01
BESS-WTG01	Status_9_Substatus_8	9:8 Generator heating (Manual)	08-11-2017 14:47:00	08-11-2017 21:09:00	06:21:13

Table 4.5: First 5 rows of the status log for WT 1 at Bessaker.

The aim of this paper was to classify and explain wind turbine faults. It is possible to use supervised machine learning methods by extracting faults from the SCADA system alarm log, described in section 2.2.3. Only faults are of interest, thus "OK" and "Warning" states were filtered out.

The states labelled "Alarm" were not only turbine faults but rather states where the turbine was unable to produce power. E.g. "Code: 2:1: Lack of wind" and "Code: 8:0: Maintenance" were obviously not turbine faults. Other "Alarm"-states were not as obvious and required domain knowledge to determine if they were actual faults.

To figure out which states were faults and which were not, a research trip to Bessaker wind farm was carried out to gather domain knowledge. Wind turbine operator Hans Kjetil Stein at Bessaker, and wind turbine expert Leif Kyhl Johansen at TrønderEnergi provided valuable knowledge. E.g. "Code 9:8: Generator Heating" may sound like a fault where the generator is overheating but turns out to be a state where the generator is heated to evaporate moisture in order to prevent electrical failures.

The most common alarm faults are listed and counted in Appendix B.1.

4.1.3 Analysis of the Status Log

An alarm sequence analysis was performed to find out if there existed a relation between the alarms. The analysis was performed by counting how many times an alarm sequence appeared on each turbine during a time period of 12 minutes. This means that there is a maximum of 12 minutes between the first alarm ends and the last alarm starts in the sequence. The duration was chosen to allow electrical and mechanical component failures to occur. The wind turbine may be in operating mode in between alarms or go directly to another alarm.

4.2 Combining Status Log and Operational Data

The operational data was combined with the status log, in order to get a data set which could be used for supervised machine learning. From section 2.3, the data needed to be on the form $D = \{(x_i, y_i)_{i=1}^{N}\}$, where N is the number of "rows" in the data set. From there, ML methods can be used to approximate $\hat{f}$, which can classify new data samples. In this case, the x_i is the operational data with 34 features for each i data sample, and y_i is the corresponding status for that sample.

To be able to combine operational data with the status log, the data had to be in the same time format. The desired time format was *DD-MM-YYYY hh-mm-ss*. The column **TimeStamp** in the operational data (Table 4.4) was already on the correct format. However, the columns **Start** and **End** in Table 4.5 had to be converted to this format.

4.2.1 Selecting Errors

Selected Error	Status Code
Feeding fault (Diff. P-set/P-actual)	62:7
Feeding fault (Feeding safety circuit faulty)	62:30
Generator heating (Manual)	9:8

Table 4.6: The three selected faults to diagnose and explain.

The three errors selected are shown in table 4.6. The fourth class in the data set was the *other* class, which consisted of all remaining data samples in the data set that were not of the three fault classes. These faults were selected for different reasons.

During a research trip to Bessaker, it was discovered that the reason why feeding faults occur was not exactly pinpointed. Feeding faults occur when there are faults in the power feeder cable of the turbine (Leahy et al., 2018). It was considered interesting to examine whether applying machine learning and explainable AI to these faults would be able to produce new, valuable knowledge or correlations. There are various kinds of feeding faults. The two selected are the most frequent in the data set.

As mentioned earlier, *Generator heating (Manual)* is not an actual turbine fault and is not as interesting from a fault perspective. It was selected because it occurs frequently in the data set and serves as a benchmark and an indication for what is possible given many samples.

4.2.2 Labelling Data

To obtain a supervised classification problem, all the operational data needed to be labelled, which means to have a target value *y*. The operational data was sampled every 30 seconds. As mentioned above, there was a start and end time for each error in the status log. To match the 30 seconds sampling rate of the operational data, all start and end times in each row of the status log was modified. All *Start* values were set to DD-MM-YYYY hh:mm:00. The *End* column was rounded up to the nearest 30 seconds, shown in Table 4.5. For instance, an alarm starting at 01-10-2019 09:27:32 would be set to 01-10-2019 09:27:00. An alarm ending at 01-10-2019 15:33:21 would be rounded up to 01-10-2019 15:33:30.

A new column was added to the operational data set called **Status**. All values in this new column were set to *Other*. Having the times on the same format, it was possible to iterate the status log and label all the data points in the operational data that fell into the closed interval *[Start, End]* for a particular error. Now, the rows that were not updated with a new label still had the class label *Other*.

All the 25 matrices with operational data and corresponding status were merged into one single matrix.

4.2.3 Resampling Data Set

In order to balance the data set, random under sampling, 2.4.1, was carried out on the majority classes. The majority classes were *Other* and *Generator heating (Manual)*. These two classes were under sampled to 10,000 samples. This was accomplished by using the Pandas library function *pandas.DataFrame.sample* in Python. For randomised operation, a random state integer is often selected, so that experiments can be reproduced. For this thesis, a random state of 12 was chosen. This parameter was passed to the mentioned sampling function.

4.3 Building Models

The ML models were trained on the combined data from section 4.2. The models that were trained in this thesis were decision trees (2.3.4), random forest (2.3.4) and SVMs (2.3.4). When training the models, the hyper-parameters were not optimised.

4.3.1 Decision Tree and Random Forest

To be able to build a decision tree, feature selection was performed in order to reduce the feature space from 34 variables to 5. The selection was performed using the feature importances attribute from a random forest (2.3.4). A Random Forests was built in Python using the *sklearn.ensemble. RandomForestClassifier* module (Pedregosa et al., 2011). The RF was trained using data which was under sampled and over sampled using SMOTE. With the random forest module, it is possible

to extract the feature importances-attribute from the random forest model. The feature importances parameter is described in background section 2.3.4.

With the Random Forest model built and evaluated, the top *five* important features were extracted as a list. The data set was then filtered on these five features and used to build the decision tree using the scikit-learn module *sklearn.tree.DecisionTreeClassifier*. The tree was assigned a maximum depth of 3.

4.3.2 SVM and LIME

SVMs (2.3.4) were chosen as the underlying "black box" ML algorithm to build explanations for with the LIME framework. In order to train the SVM model, the under sampled data was scaled. This was done using MinMax module from Scikit-learn called *sklearn.preprocessing.MinMaxScaler*. (Pedregosa et al., 2011).

A polynomial kernel function was selected in this thesis (Ben-Hur et al., 2008) with the *gamma* parameter set to *auto* .

Two different SVM models were constructed on the data set. The SVMs performance was evaluated using stratified cross validation on two different versions of the under sampled data set described in section 4.2.3:

1. Using balanced class weights (2.4.1)
2. Using SMOTE (2.4.1)

The SVM with the highest f1-score was then used with LIME in order to produce explanations for the classifications.

The LIME framework requires the underlying model to output probabilities for each target class. This was applied to the SVM models by setting the *probability* parameter to *true*. The target class names were mapped to integers for better visualisation in LIME. The mapping is shown in Table 4.7.

Status Description	**Integer**
Other	0
Generator heating - (Manual)	1
Feeding fault - (Diff. P-set/P-actual)	2
Feeding fault - (Feeding safety circuit faulty)	3

Table 4.7: Conversion of status descriptions to integers.

The training data, target names, the SVM model and several instances from the test set were fed into the LIME framework to produce explanations. The parameters were kept to default; distance metric was set to *euclidean*, the number of samples generated was set to *5000* and feature selection method was set to *auto*, described in background section 2.5.2. It was considered likely that the SVM model depended

on multiple features when classifying. To have too few features in the explainer was therefore not sensible and the number of features m, was set to 8.

4.3.3 Validating and Evaluating Models

In order to assess the performance of the decision tree, random forest and SVM, stratified cross validation (2.3.5) was applied (2.3.5). This was done using the scikit-learn module *StratifiedKFold* by Pedregosa et al. (2011). In essence, recall, precision and f1-score was computed for every fold for each class. In the end, the average recall, precision and f1-score was returned for each class on the whole data set.

Stratified cross validation was used for evaluating models on the complete data set, and the validation step of selecting optimal hyper-parameters was not performed. This is addressed in discussion 6.2.1.

4.4 Hardware and Technical Implementation

The programming language Python v3.7.5 is used for all data processing and ML. The code is written in Jupyter Notebook, an open-source web application that allows you to mix markdown sections with code. This results in code that can be easily reviewed. All the code written for this thesis is in the following repository: `https://github.com/stianismar/project-thesis/`

The library used for data processing was mainly Pandas v0.25.3. Scikit-learn v0.21.3 was used for implementing the machine learning methods in this thesis. LIME v0.1.1.36 was used in the implementation.

With regards to hardware, the models were trained on a MacBook Pro with 3.1 GHz Intel Core i5 processor and 8 GB RAM.

Chapter 5

Results and Analysis

In this chapter, the results from the alarm correlation-algorithm, data pre-processing, and models will be presented and analysed. In 5.1, the results from the status log analysis are shown. Section 5.2 presents the final labelled data set. The final section, 5.3 describes the trained models, and compare how they performed when diagnosing faults. The next chapter will analyse and compare the explanations the models produced.

5.1 Analysis of the Status Log

A list of the 45 most most frequent alarm sequences is found in Appendix B.3 and B.4. The result shows that many alarms are related, although, most alarms concern turbine stops for whatever reason, and not specifically due to a turbine fault, leading to uninteresting relations. E.g. the two most frequent sequences, the first one being *Cable twisted (Right (2-3 turns))* $\rightarrow$ *Lack of wind (Wind speed too low)*, and the second one being *Lack of wind* $\rightarrow$ *Lack of wind.* A section containing the 20 most frequent sequences but without the two just mentioned is shown in Figure 5.1.

Some alarm relations are sensible, e.g. *Maintenance* $\rightarrow$ *Generator heating.* When a wind turbine is paused for a long period of time, moisture often gathers in the system and the generator is put in heating mode to prevent electrical failures as mentioned in section 4.1.2. The same applies to *Lack of wind* $\rightarrow$ *Generator heating.* Others require further domain knowledge to understand, e.g. *Anemometer Interface - (Fault 1 ultra sonic sensor)* followed by *Fault wind measurement - (Constant wind direction).*

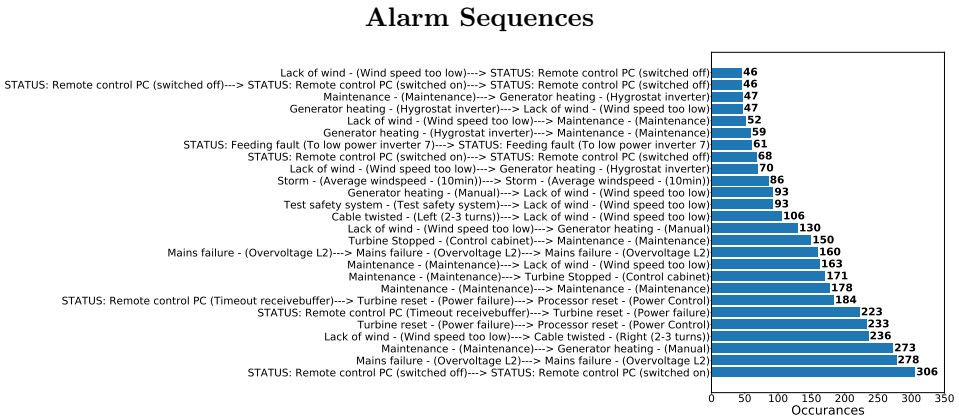


Figure 5.1: Frequently repeated alarm sequences counted for all wind turbines at Bessaker.

In addition to the analysis of the alarm sequence above, the distribution of chosen alarms was also analysed, shown in Figure 5.2. The plot shows how the three selected alarms over the 25 wind turbines were distributed across the 25 wind turbines at Bessaker. It is interesting to note that wind turbine with id 21 has a remarkably high occurrence of *Feeding fault - (Diff. P-set/P-actual)*.

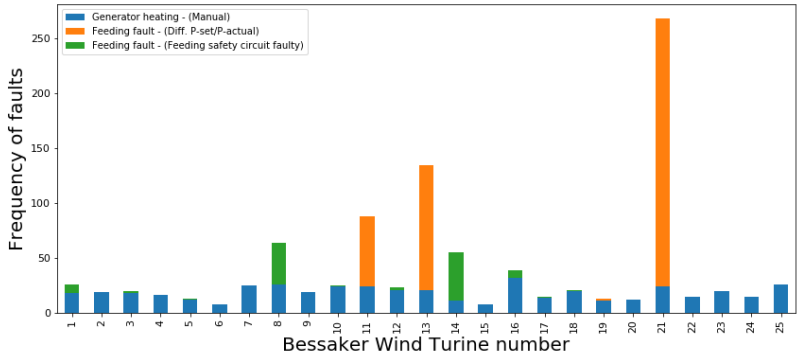


Figure 5.2: Plot showing how the 3 faults are distributed all 25 wind turbines (1-25 on the x-axis).

5.2 Combining Status Data and Operational Data

5.2.1 Feature Selection

The 34 features that were downloaded through the SCADA system were used when training all models, except for the DT. The dimensionality of the feature space could

have been made smaller by removing some features. Insight from domain experts at Bessaker wind farm could have aided in the removal of less relevant parameters, or through methods such as Principal Components Analysis (PCA) (Godwin and Matthews, 2013). PCA is a method for selecting features that best describe the data (James et al., 2013). PCA was not carried out, because it merges parameters which would complicate the explanation process.

The correlation plot, found in Appendix A.1, showed that a lot of the parameters were correlated linearly. However, none of the features were removed, because we did not want to loose any information which could be interesting in the explainable models.

The boxplots for all parameters is also placed in Appendix (A.2). This shows the distribution of the data in each feature, and could be used to discover sensor faults, e.g. if a sensor measures physically impossible values.

5.2.2 Labelling Data

The amount of faults in the data set after filtering the relevant errors is shown in Table 5.1. One can see from the table that the data is highly imbalanced.

Status Code	Description	Amount	Percentage
-	Other	51 966 851	99.553%
9:8	Generator heating - (Manual)	228 668	0.438%
62:7	Feeding fault - (Diff. P-set/P-actual)	4 098	0.008%
62:30	Feeding fault - (Feeding safety circuit faulty)	383	0.001%

Table 5.1: Unprocessed dataset

5.2.3 Resampling Data Set

Resampling Results

After running RUS and SMOTE, the four class populations were all equal to **10,000** samples.

As introduced in the 2.4.1 section, Random Under Sampling is an intuitive way of removing samples from the majority class. However, like Tahir et al. (2012) shed light on, this has to be done with care. Removing samples at random could mean that crucial data points would be removed, causing the model to classify poorly.

For this thesis, no consideration was taken in regards to loss in generalisation because of removed data points. The main scope of this thesis was producing results that were explainable to human users, not achieving optimal classification perfor-

mance.

The population of the minority class, *Feeding fault- (Feeding safety circuit faulty)* grew from 383 to 10.000 using smote. This means that 9617 artificial samples were made, which could affect the models ability to classify the majority samples (Chawla et al., 2002). This is analysed in section 5.3.3 below.

5.3 Building Models

In this section, the results for RF, DT and SVMs will be presented. Furthermore, the performance metrics for each of the selected faults will be compared. Lastly, the results of the LIME method will be discussed.

Since the focus of this thesis was to explore ways of creating interpretable machine learning models on data from wind turbines, little time was spent on optimising the models. Hence, the performance metrics were likely to have been better had the models been tuned during training.

5.3.1 Decision Tree and Random Forest

As described in the method section, a random forest model was first trained in order to get the feature importances to build a decision tree.

Random Forest
The first model that was built was the random forest classifier. Using the feature importances function described in section 2.3.4, the most important features are listed in Table 5.3. The result was a RF classifier with the performance results in Table 5.2.

	Feeding fault (Diff. P-set/P-actual)	**Feeding fault (Feeding safety circuit faulty)**	**Generator heating (Manual)**	**Other**
Precision	0.986769	0.892609	0.987341	0.972982
Recall	0.999268	0.963765	0.973300	0.978600
F1-score	0.989240	0.905605	0.984493	0.974092

Table 5.2: RF performance scores.

Predictor variable
ActivePower
Log-T-Raw-TransformerTemperature
Log-T-Raw-Rectifier1HeatSink
Log-T-Raw-Rotor1Temperature
Log-T-Raw-Stator1Temperature
Log-T-Raw-Stator2Temperature

Table 5.3: Table with the selected variables from RF.

Decision Tree

The decision tree of depth 3 can be inspected below in Figure 5.3. The *value* variable in the root nodes give an understanding of how the samples are grouped in each node, i.e. how the tree generalises based on the selected features.

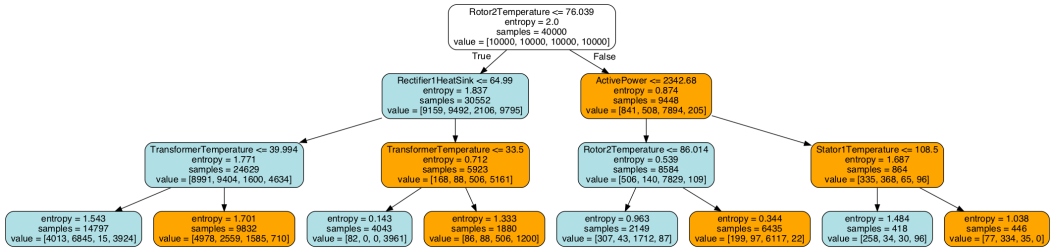


Figure 5.3: The decision tree built with features from the Random Forest model.

The confusion matrix for the decision tree with depth equals to three is shown in Figure 5.4.

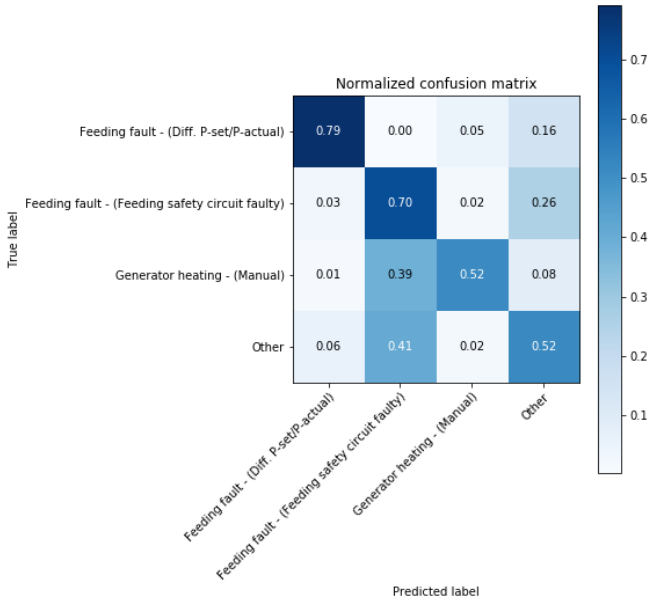


Figure 5.4: Confusion matrix for the decision tree with depth equals to 3.

The performance metrics for the decision tree are shown in Table 5.4.

	Feeding fault (Diff. P-set/P-actual)	**Feeding fault (Feeding safety circuit faulty)**	**Generator heating (Manual)**	**Other**
Precision	0.826383	0.032481	0.930589	0.767562
Recall	0.790388	0.699865	0.516000	0.518500
F1-score	0.818766	0.040135	0.801653	0.700171

Table 5.4: DT performance scores.

As described in section 4.3.1, the Random Forest classifier was trained on the complete data set, in order to extract the feature importances. The top five highest scoring features were selected to build the decision tree.

By examining the decision tree, it is possible to see how the samples are classified. The tree can be translated to a set of rules by traversing it. For instance, one rule derived from the decision tree in Figure 5.3 is seen in equation 5.1.

$$
\begin{aligned}
Rotor2Temperature \leq 76.039 \rightarrow \\
Rectifier1HeatSink \leq 64.99 \rightarrow \\
TransformatTemperatures > 39.994 \rightarrow \\
Class = FeedingFault - (Diff.P - set/P - actual)
\end{aligned}
\tag{5.1}
$$

Strobl et al. (2007) pointed out that the feature importance measure from random forest is *not* reliable in problems where predictor variables vary in scale and the number of distinct values. Having to scale the data could be inconvenient, especially since we wanted to retain the possibility of extracting the individual tree-node decisions from the RF. Scaling was not performed before the RF was trained in this thesis. Another way of measuring feature importance and selecting features from a data set is discussed in further work, chapter 8.

The maximum depth of the decision tree was selected to be three. This was to ensure that it would be interpretable by humans. The depth of the tree would normally be carefully selected by validating the model performance.

5.3.2 SVM

Before the SVM was trained, the data was scaled using MinMaxScaler from scikit-learn. The results are in Table 5.5. The scaling was important to do because SVMs are known to be sensitive to the way features are scaled (Ben-Hur et al., 2008).

As stated in the method section 4.3.2, two SMVs were trained on the same data set, where one used the balanced class weights-technique, whilst the other used the SMOTE method. The method achieving the best f1-score was the one applying the SMOTE technique. Thus, this version of the SVMs was selected to use with LIME.

WindSpeed	**RotorSpeed**	**NacelleDirection**	**ActivePower**	**Spinner-Temp**	...
0.084699	0.347789	0.225627	0.017062	0.684211	...
0.139344	0.519966	0.353760	0.000000	0.578947	...
0.125683	0.376556	0.228412	0.029130	0.592105	...
0.295082	0.856591	0.306407	0.732002	0.578947	...
0.030055	0.027909	0.871866	0.000000	0.565789	...

Table 5.5: Scaled data set

SVM and Balanced Class Weights

The confusion matrix for SVM with balanced class weights is shown in Figure 5.5.

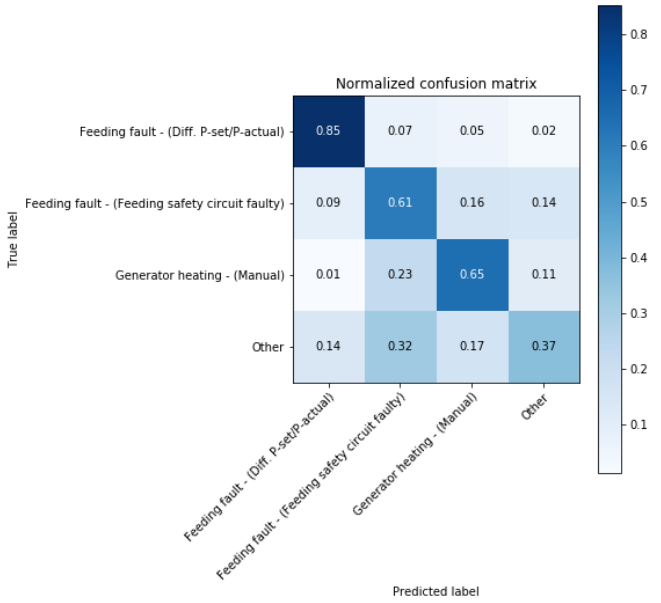


Figure 5.5: SVM with class weights balanced.

The performance results are shown in Table 5.6 underneath.

	Feeding fault (Diff. P-set/P-actual)	**Feeding fault (Feeding safety circuit faulty)**	**Generator heating (Manual)**	**Other**
Precision	0.691034	0.038415	0.767164	0.749424
Recall	0.851642	0.607962	0.649500	0.367600
F1-score	0.718011	0.047271	0.740316	0.620419

Table 5.6: SVM and balanced weights performance scores.

SVM and SMOTE

The confusion matrix for SVM trained on data over sampled with SMOTE is shown in Figure 5.6.

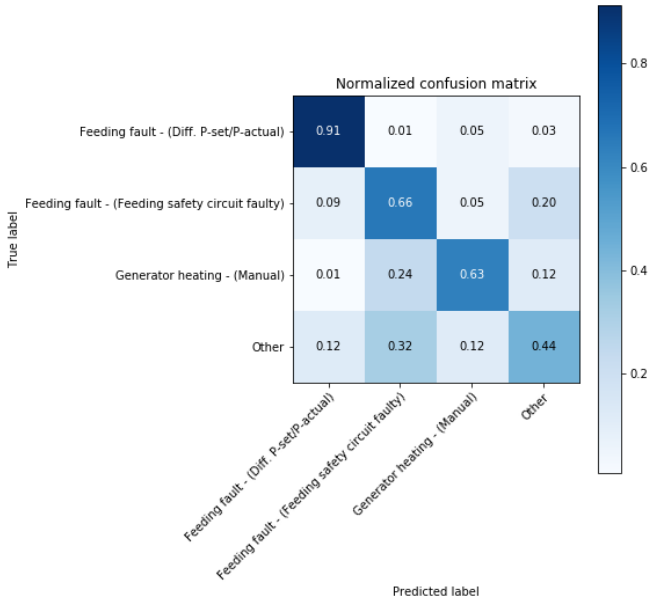


Figure 5.6: CM for SVM built with data over sampled with SMOTE.

The performance results are shown in Table 5.7 underneath.

	Feeding fault (Diff.P-set/P-actual)	**Feeding fault (Feeding safety circuit faulty)**	**Generator heating (Manual)**	**Other**
Precision	0.732200	0.042663	0.816632	0.765072
Recall	0.911426	0.657625	0.629800	0.440600
F1-score	0.762098	0.052476	0.770853	0.666713

Table 5.7: Performance measures for an SVM trained on data over sampled with SMOTE.

5.3.3 Comparison of the performance metrics for the faults

Figure 5.7, 5.8, 5.9, and 5.10 underneath show how the decision tree model, the random forest, and the SVMs performed when classifying samples from the data set. This section is structured so that the faults are examined one by one. The performance metrics are a result of the k-fold cross validation 2.3.5. It is important to state that SMOTE was applied on all models on the two classes with the fewest

samples- *Feeding fault- (Diff. P-set/P-actual)* and *Feeding fault - (Feeding safety circuit faulty).*

To recap, precision is a measure of how many samples that were classified as class X, and actually *were* of class X. Recall is a measure which answers the following question- out of all the samples of class X, how many samples were classified as class X? Low recall for a fault means that other classes were classified instead, which can be dangerous and costly in the context of component failures. A low precision leads to many false alarms for a certain fault. The f1-score is the harmonic mean of the recall and precision.

Feeding fault- (Diff. P-set/P-actual)

Figure 5.7: Performance metrics for *Feeding fault- (Diff. P-set/P-actual)*

The *Feeding fault- (Diff. P-set/P-actual)* had initially 4098 samples in the original data set before if was over sampled using SMOTE. The DT and RF performed similarly across all three metrics (precision, recall, f1-score) for the *Feeding fault- (Diff. P-set/P-actual).* The SVM however, had a lower recall than precision, which means that the SVM does not detect all *Feeding fault- (Diff. P-set/P-actual).*

The random forest outperformed the two other methods for this fault. One of the reasons for this is that the RF is an ensemble method, which has the advantage of using many classifiers. It is interesting to see how the decision tree classified the samples, based on the features importances from the RF. The Decision Tree had the overall lowest F1-score for this fault.

Feeding fault - (Feeding safety circuit faulty)

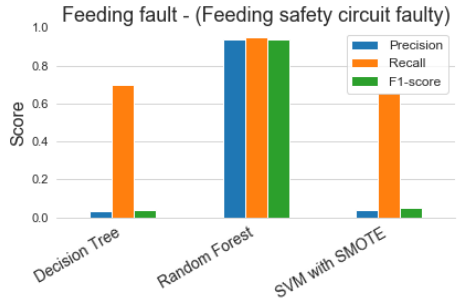


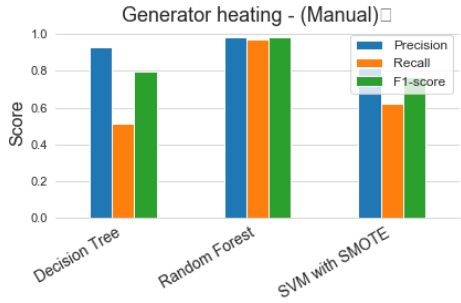
Figure 5.8: Performance metrics for *Feeding fault - (Feeding safety circuit faulty)*

The *Feeding fault - (Feeding safety circuit faulty)* was the most difficult fault to classify, with 383 data points in the original data set. This sample set was increased from 383 to 10,000 using SMOTE, a 2500% increase. This could, as mentioned, have caused the performance to decrease on the classes with more samples (Chawla et al., 2002).

The random forest performed best for this fault as well. The decision tree model had the lowest f1-score, with recall equals to 0.7 and a precision of 0.032. The low precision can be explained from the confusion matrix in Figure 5.4, where a lot of the classified *Feeding fault - (Feeding safety circuit faulty)* actually were *Generator heating- (Manual)* and *Other*.

The SVM with SMOTE also had a low precision. From the confusion matrix for the SVM in Figure 5.6, it is also clear that a lot of the classified *Feeding fault - (Feeding safety circuit faulty)* were actually of the class *Generator heating- Manual* and *Other*. The recall with SVM was at 0.658 as shown in Table 5.7.

Generator heating- (Manual)



.

Figure 5.9: Performance metrics for *Generator Heating- (Manual)*

10,000 of the samples were of the class *Generator heating- (Manual)*. The Decision Tree and SVM with SMOTE had a poorer recall for the *Generator heating- (Manual)* fault. Specifically, for the Decision Tree model, 39% of the faults that were actually *Generator heating- (Manual)* faults were classified as *Feeding fault - (Feeding safety circuit faulty)*.

The SVMs classified 24% of the Generator Heating faults as *Feeding fault - (Feeding safety circuit faulty)* and 12% of them as *Other*. For this fault, the decision tree achieved a higher f1-score than the SVMs with SMOTE.

Other

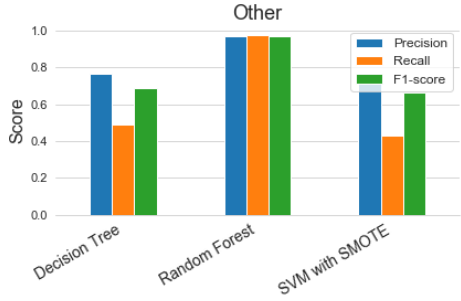


Figure 5.10: Performance metrics for *Other*

The *Other* class also had 10,000 samples. The performance of the Random Forest was almost at 1 for all metrics. The decision tree had the lowest f1-score, at 0.7. The recall for the SVM was at 0.44, meaning that a lot of the samples that were

indeed *other* were classified as other classes. 20% were classified as *Feeding fault - (Feeding safety circuit faulty)*, seen in the confusion matrix in Figure 5.6.

5.3.4 LIME

One of LIME's explanations is shown in Figure 5.11. Keep in mind that the target variables are mapped to integers for better easier visualisation (see Table 4.7).

The left image shows the SVM model's classification probability. The middle image shows which features support the predicted class and which do not. It also shows why these features were selected by showing their weights from the linear model and their boundary value. This image is repeated in Figure 5.12 with better visibility. The exact values of the weights are listed in Table 5.8. The right image shows the most important features and their values as they are inserted into the model.

As deduced from LIME, the SVM model correctly classifies this data instance to be a *Feeding fault - (Diff. P-set/P-actual)* with a probability of 95%. LIME scores *Log-T-Raw-Rotor2Temperature* as the most important feature with the highest weight of 0.8975. This is because its value = 0.41 is above the threshold value = 0.38, created by LIME's model. The other features can be inspected similarly.

LIME enables the user to evaluate if the underlying model makes sensible generalisations. It is possible to question if each feature should play an important part in the root cause analysis. E.g. should the feature; *Average blade angle across A B C* be a valuable contributor to detect this kind of fault? To assess this correctly, the user should have a broad understanding of WTs.

The values are scaled using the MinMaxScaler which transforms the values to the range: 0 to 1. This enhances SVM's performance but lowers the readability of the explanation. To be mores sensible to the user, the values should be mapped back to their original values with the correct unit of measure.

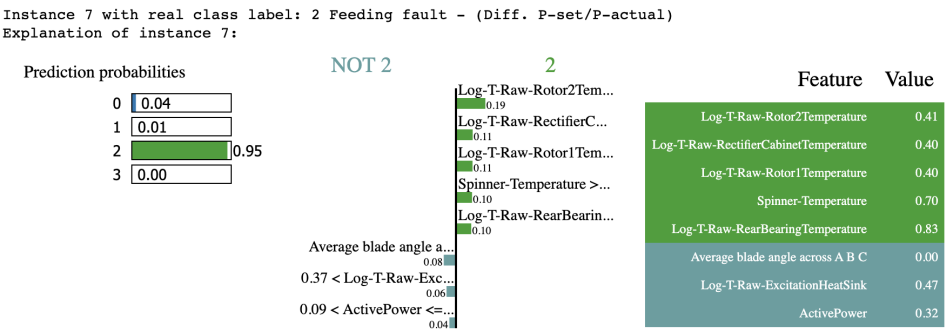


Figure 5.11: LIME explanation for a correct *Feeding fault - (Diff. P-set/P-actual)* SVM classification.

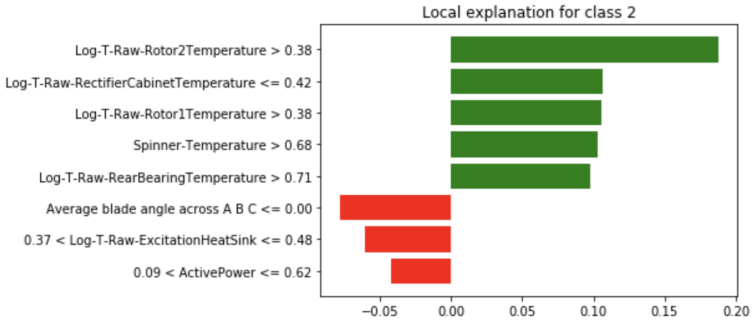


Figure 5.12: LIME feature plot 2

Feature	Weight
Log-T-Raw-Rotor2Temperature	0.1875
Log-T-Raw-RectifierCabinetTemperature	0.1065
Log-T-Raw-Rotor1Temperature	0.1057
Spinner-Temperature	0.1027
Log-T-Raw-RearBearingTemperature	0.0975
Average blade angle across A B C	-0.0779
Log-T-Raw-ExcitationHeatSink	-0.0603
ActivePower	-0.0422

Table 5.8: Feature weights.

Another example of a LIME explanation with a different data instance is shown in Figure 5.13. As we can observe, the SVM model incorrectly classifies it as a *Feeding fault - (Diff. P-set/P-actual)* with a probability of 89% when it actually is labelled as *Other*. We are able to inspect the most important features and compare them to the correctly classified example above. In this case, the most important feature is *Log T-Raw-TransformerTemperature*. Perhaps a WT operator could be

able to detect if this is a mistake or not, and determine if the other most important features selected are sensible.

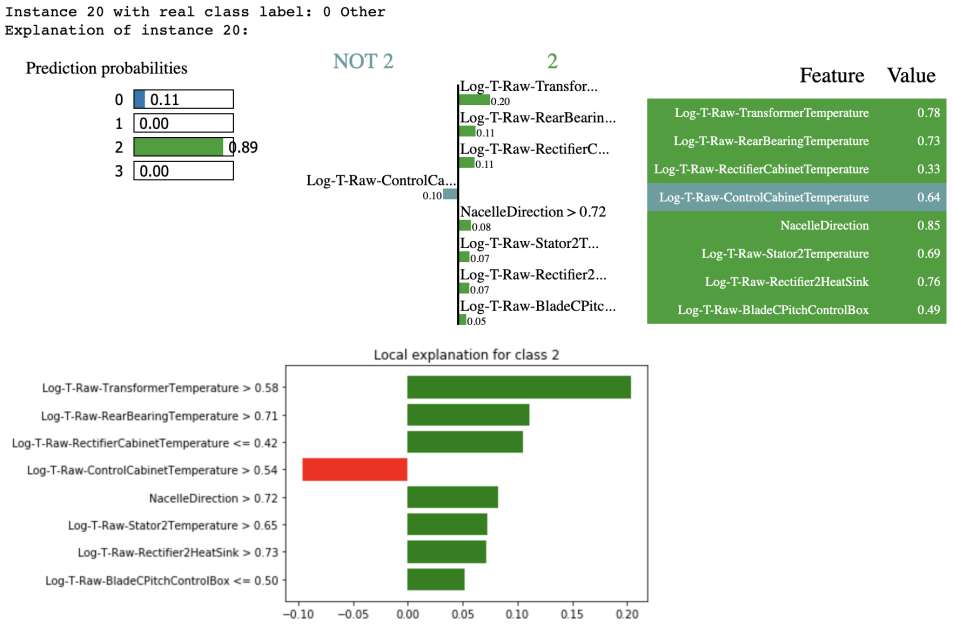


Figure 5.13: LIME explanation for an incorrect *Feeding fault - (Diff. P-set/P-actual)* SVM classification.

A third example is shown in Figure 5.14. When the classifier is uncertain, as in this example, manual inspection could be useful. The SVM model predicts *Generator heating - (Manual)* when actually it belongs to the *Other* class. A manual read through could perhaps correct the misclassification.

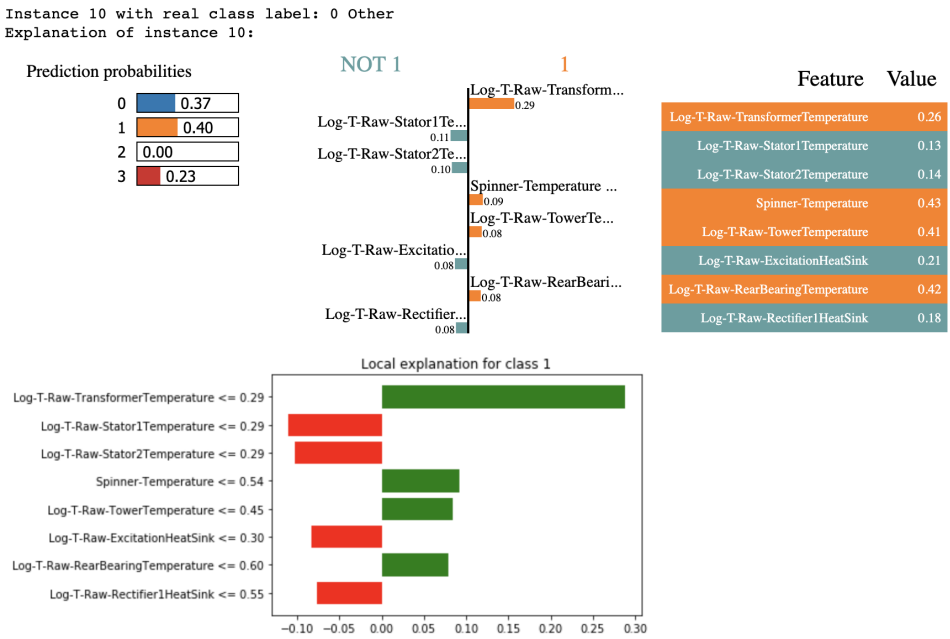


Figure 5.14: LIME explanation for an incorrect *Feeding fault - (Diff. P-set/P-actual)* SVM classification.

One of LIME's drawbacks is illustrated in figure 5.15. With the same data instance as in the first example, and using the same SVM model with exactly the same parameters, a different explanation is generated. This is further discussed in section 6.2.3.

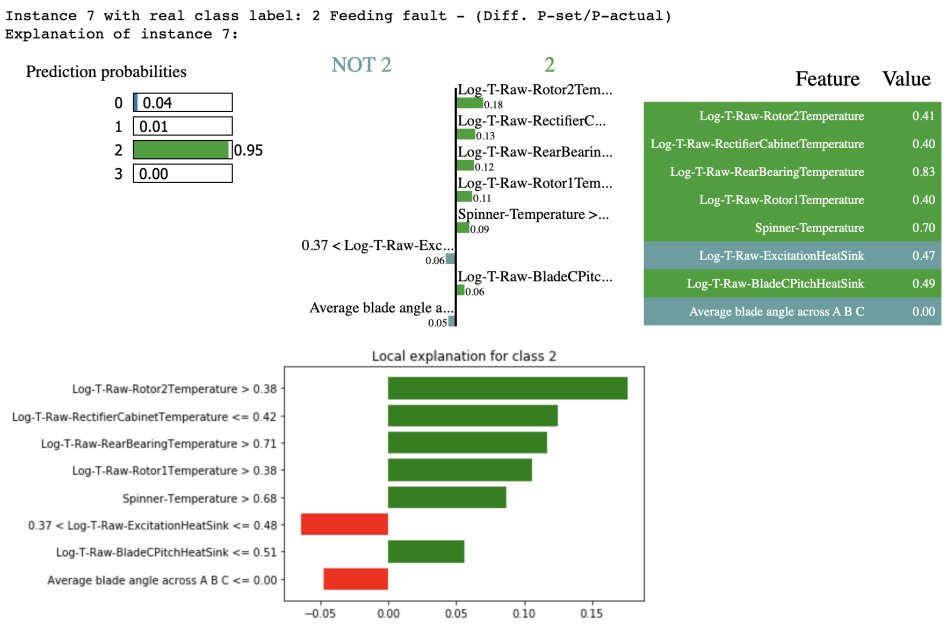


Figure 5.15: A different LIME explanation for a correct *Feeding fault - (Diff. P-set/P-actual)* SVM classification.

Other examples of explanations are included in Appendix A.3.

Chapter 6

Discussion

In this chapter, the results are further analysed. In 6.1, the status log sequence algorithm is discussed in brief. The two explainable models are discussed and evaluated using a framework for XAI evaluation in 6.2. In section 6.3, the potential of using the two models on the SCADA data set is examined. The relevance of the SCADA data and error log used in this project is discussed in section 6.4.

6.1 Analysis of the Status Log

The alarm sequence analysis carried out in section 5.1 was performed to see if any interesting relationships could be discovered. Some of the sequences mentioned, like the heating of a turbine after it has gone through maintenance, are sensible. However, as is discussed in the end of this section, the alarm log contains mostly entries that are *not* considered faults. Thus, the algorithm could be more valuable for both root cause analysis and WT operators when the alarms in the status log are of higher relevance.

6.2 Explainable AI Models

Whether an explanation for a ML algorithm decision is sufficient or not depends a lot on the user. As stated by Hoffman et al. (2018a): *"The fact that a computer system is interpretable does not mean that it is human understandable; the formal interpretation has explanatory value only to computer scientists."* The splits in a decision tree or the weights that LIME uses to extract its features, may not make as much sense to users who are not computer scientists. Consequently, the explainable methods applied in this thesis might not be as explainable to wind turbine operators.

The next section will discuss and compare the explainable models that were created.

6.2.1 A Note on the Performance of the Machine Learning Models

The trained models were tested using cross validation. However, as stated in the background section in 2.3.5, cross validation is often used for validating a model's performance when selecting the optimal hyper-parameters. This involves optimising the performance of a single model-type, such as a decision tree model. The example of a hyper-parameter used in the background section was the depth of decision tree. In this thesis, optimising specific models was not a part of the main scope, and limited time was allocated to doing this. Thus, the results for recall and precision could most likely have been better in section 5.3.3.

The analysis of the machine learning classifications in section 5.3.3 is important because it provides an understanding of which classes were miss-classified. This is crucial because it provides insight into the whether or not the explainable method is credible.

6.2.2 Decision Tree and Random Forest

For a decision tree to be useful in a root cause analysis or for technicians when evaluating the results of a classification, the tree can not be too complex. A DT of depth 4 already has $2^0 + 2^1 + 2^2 + 2^3 + 2^4 = 31$ nodes. A decision by a tree with this depth produces at maximum 4 rules when classifying new samples. In relation to what Miller (2019) stated, this is on the limit of what a good explanation should be (2.5.3). The question one should ask is; will a DT with a maximal depth of 4 generalise well enough? That is, will the tree have understood the pattern in the training data well enough in order to classify new samples correctly, or will it *under fit* the data? From the results in section 5, one can see that the decision tree based on random forest has a rather low recall for some of the faults. This could potentially be critical since it means that some of the faults are uncaught.

However, a tree of maximum depth equals to three, like the one trained in this thesis, makes it possible for humans to traverse and understand it.

6.2.3 SVM and LIME

In this research paper, stratified CV was used for *evaluating* the different models, and not for tuning the models. Thus, the SVM most likely performed sub-optimally when classifying the faults. As a consequence, the explanations it produced with LIME may be inaccurate.

Potential Drawbacks of LIME

Ribeiro et al. (2016) acknowledge that the assumption that the underlying local model behaves linearly may be incorrect. The team suggests that estimating how well the simple model captures the underlying model locally will help to uncover

this kind of behaviour. However, the team has left this for further work and it has not been implemented yet.

According to Molnar (2019), LIME's main problem is to find the right definition of the neighbourhood, which influences the explanation. LIME uses a smoothing kernel to compute the proximity. It is possible to change the proximity by tweaking the kernel width, but it's no easy task to find the correct one.

How LIME samples the perturbed data set is problematic as well (Molnar, 2019). Currently, each feature value for a data point is sampled from a Gaussian distribution. It does not take the correlation between features into account, making it possible to sample unlikely data points, and creating bad explanations.

Molnar (2019) also raises concern about the instability of explanations. Experiments showed that two close data points' explanations varied greatly. In addition, he claims that the explanations may vary when repeating the sampling process. The results found in this thesis strengthen Molnar's concern. The explanation generated with LIME varied for the same instance, which is troubling. He concludes that LIME is a promising tool, but needs further development to be completely trustworthy.

6.2.4 Evaluation of Explainable Methods

The explainable models presented in the results will now be evaluated with the first category from the *five categories for evaluating explainable methods*, called *Explanation Goodness*. The five categories are shown in section 2.5.3. The second category, *explanation satisfaction*, would have been interesting to evaluate as well. However, for this we would need test participants such as WT operators to try the models, which was beyond the scope of this thesis.

Explanation Goodness

The method for evaluating explanation goodness by Hoffman et al. (2018b) was applied to the decision tree and LIME results (table 6.1). The reasoning is further discussed underneath.

	Explanation Goodness Question	DT	LIME
1	The explanation helps me understand how the model works	yes	no
2	The explanation of how the model works is satisfying	yes	no
3	The explanation of the model is sufficiently detailed	no	yes
4	The explanation of how the model works is sufficiently complete	yes	yes
5	The explanation is actionable, that is, it helps me know how to use the model	no	no
6	The explanation lets me know how accurate or reliable the model is	yes	no
7	The explanation lets me know how trustworthy the model is	no	no

Table 6.1: DT goodness evaluation. Source: (Hoffman et al., 2018b)

1: The splits attained from the DT enable us to understand how the model works. Even though LIME approximates how the model works on a local level, it does not explain how the model works in general.

2: The splits in the DT can be transformed into a set of rules, 5.1. These rules are easy to follow and provide a global interpretation of how the model works. LIME returns the most important features for a classified instance. The weight of each feature from the linear model is provided and visualised, which explains how the model is approximated locally. However, as it is only sufficient locally, it is considered unsatisfying in total.

3: The DT model is limited to a depth of three to ensure that the explanation is not too complex. This is not detailed enough. LIME lets the user select an arbitrary amount of features to be evaluated and is therefore considered sufficiently detailed.

4: Both methods are considered complete as they always end up with a classification label for an instance.

5: Neither DT nor LIME are actionable, since the explanations fail to inform how to use the explainable model.

6: Reliability is here interpreted as a models ability to behave consistently. The DT is reliable since it will output the same result every time when run with the same parameters. LIME is unreliable as it returns different explanations for the same data instance with the same parameters.

7: For a DT, it is possible to interpret uncertainty associated with a prediction by studying the leaf nodes. This is shown in Figure 6.1. The leaf node

is an example cut-out from the result in Figure 5.3. The array *value* shows how many data points of the training set are in the final leaf node classification on the form [N_{class0},N_{class1},N_{class2},N_{class3}]. A data sample "ending" up in this leaf node will be classified as class 1, since N_{class1} is the largest integer in the *value* list. However, it is not far off from being classified as class 3. Only $N_{class1} - N_{class0} = 4013 - 3924 = 89$ samples separated the two decisions. Results like this is not uncommon in shallow decision trees. This high level of uncertainty affects its trustworthiness and makes it untrustworthy. In LIME, the probability for each class is provided, but it does not measure how well the simple model fits the complex model at the local level, as discussed in 6.2.3. It is possible that it does not fit well which makes it untrustworthy.

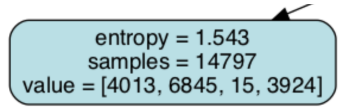


Figure 6.1: Section of figure 5.3 showing a leaf node

6.3 Potential of DTs and LIME for Diagnosis of Wind Turbines

As stated in the introduction, the main scope of this thesis was to research how explainable ML models could be built using SCADA data to (1) perform root cause analysis for turbine faults, as well as (2) provide explanations with decisions to human users, in order to increase their confidence and trust in models and their decision.

6.3.1 Potential of DT

From the comparison of the model performance, section 5.3.3, it is clear that the DT with depth 3 had a rather low recall– 62 % on average. This means that many actual faults are not classified as faults. So, even though the DT produces a visual tree, it is in many cases producing *incorrect* classifications for new samples. Therefore, a DT of depth 3 is not ideal for root cause analysis. A deeper DT would have better classification capabilities whilst sacrificing explainability.

In regards to the second research questions, the DT provides clear explanations to human users. One prerequisite for these explanations is that the user must have excellent knowledge of the domain, in order to separate correct explanations from incorrect ones. For example, if an internal node in a DT has the *rule if temp* $<$ *285 deg*, the operator must be able to know if this is a possible measurement.

6.3.2 Potential of LIME and SVM

The average recall for SVM with SMOTE for all classes was 66 % (5.3.2). This is slightly better than the DT, but still low. However, further optimisation of the SVMs parameters and more targeted feature selection would likely increase the performance, enabling more reliable classifications.

To answer the research questions; LIME with SVMs has a greater potential to produce valuable explanations than DT, but as we have seen, LIME does not produce reliable explanations, making correct root cause analysis difficult to obtain. Based on the evaluation previous in this chapter, the LIME explanations are not sufficient to increase confidence and trust in the model or its decision. There might be other model agnostic XAI methods, such as SHAP, which could be more valuable (further discussed in chapter 8).

6.4 Relevance of Status Log and Operational Data

After performing a literary survey on the topic of machine learning with SCADA data, it is clear that there are a number of issues associated with this data. One of them being that alarm codes are not standardised (Leahy et al., 2019). Thus, building on other research is often troublesome. Leahy et al. (2019) therefore suggested there be a unified standard for e.g. turbine taxonomies, alarm codes and SCADA operational data.

Many of the parameters in the SCADA data in this project were measured from temperature sensors. Figure 6.2 displays how an arbitrary component's condition varies over time. Temperature sensors are able to detect faults at a much later time, than e.g. vibration sensors (Madsen, 2011). This is one major drawback of using this type of measurements to classify faults in components. The use of vibration sensors is discussed in further work.

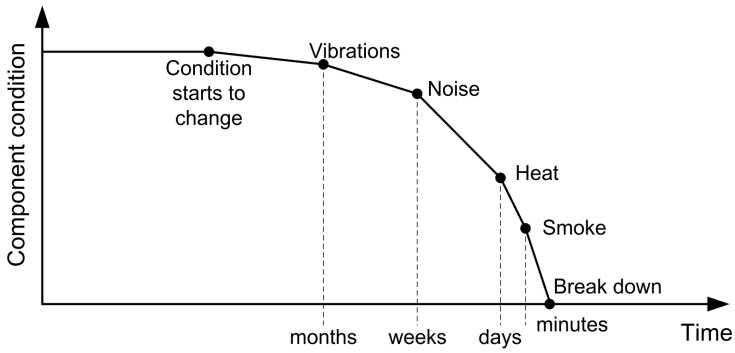


Figure 6.2: Development of mechanical faults. Source: (Madsen, 2011).

It was discovered during the research trip to Bessaker (4.1.2) that Bessaker has an unusually high availability given its age. This suggests that developing faults are discovered before they become real faults, and likely before they are measured by temperature sensors. Whilst this is advantageous for TrønderEnergi, it means that the data set contains few *real* faults, making it impossible to train ML models on these relevant faults.

Chapter 7

Conclusion

Continuous monitoring of wind turbine systems using SCADA data can provide useful insight into the state of the system. Using a machine learning approach and XAI frameworks, it is possible to build models that can diagnose and explain wind turbine states, based on input parameters such as temperatures in different components, wind speed and power outputs.

Our research was based on data from a SCADA system from 25 wind turbines at Bessaker, Norway. Initially, 34 parameters were downloaded for the time frame October 2017 to October 2019. This data was matched with a status log to build a labelled data set. This data set was used to train machine learning models to diagnose four WT states: *Other*, *Generator Heating fault*, *Feeding fault (Diff. P-set/P-actual)* and *Feeding fault (Diff. P-set/P-actual)*.

The main objective of this thesis was to explore how explainable machine learning methods could be applied to the data set. Specifically, we (1) investigated whether XAI methods could be used to perform root cause analysis for faults in wind turbines, and (2) if they could provide explanations for the models' decisions, so human operators could rely on the models and use them correctly. Explanations were created using a decision tree model and the LIME XAI framework.

A limitation in our research was the data. For one, the labelled data set was highly imbalanced. Ignoring the *Other* class, the *generator heating* fault consisted of more than 200,000 samples whilst one of the *feeding faults* only had 383 samples. This complicated the training process, and under- and over sampling was applied. Over sampling was done using SMOTE, and the mentioned minority class was grown from 383 to 10,000 samples. Over sampling of this magnitude could pose problems for the performance of ML models. We recommend tuning the "increase" parameter in SMOTE with e.g. cross validation, to find a suitable sample size to maximise model performance.

The other problem with the data set was the relevance of the operational data and status log. The WTs at Bessaker have a high availability. Thus, we experienced that the status log consisted mostly of faults that were not considered *faults* by the WT operators. Performing root cause analysis and creating explanations for WT operators based on these *faults* was therefore not really valuable.

Before the explainable models were trained, an alarm sequence analysis was carried out using the status log data (5.1). This analysis provided a different approach to understanding correlations between the alarms than the ML approach, and could be relevant in examining the root cause for some faults for other data sets.

From the evaluation method (proposed by DARPA) of the explainable methods, it is clear that the DT provides an overall better interpretable explanation than the LIME model. This is mainly because DT is more intuitive and reliable than LIME.

The results show that the DT provides an interpretable model just by studying the tree after it is trained. However, it falls short on several other points. Firstly, the random forest *feature importance* used to build the DT is not a reliable approach for reducing the feature space from 34 to 5. Secondly, the shallow depth of 3 limits the models' generalisation capabilities, i.e. it is not capable of learning which parameter values usually point to a certain fault class, as seen from the average recall of 62 %. Whilst a deeper tree generalises better, interpretability is sacrificed. The two aforementioned reasons make the model unsuitable for finding interesting root cause analysis for the three selected faults, using the SCADA data.

The DT model is consistent in the way that it always produces the same diagnosis for the same sample. Based on this property, a decision tree with better performance would be relevant for providing reliable explanations to e.g. wind turbine operators. However, a shallow DT trained on real world data will most likely not perform too well.

The LIME model based on the SVMs showed greater potential than the DT, because the SVMs could be optimised, as opposed to the DT. Optimising the SVM could in turn produced more correct explanations through LIME. However, the LIME method was not considered reliable, since different explanations were produced during the testing of one sample. Therefore, the LIME explanations were not considered suitable for performing a root cause analysis or creating reliable explanations for e.g. WT operators.

To summarise, the XAI methods applied in this thesis provide valuable insight and understanding of how explainable methods can be applied to predictive maintenance tasks with real world data, and what challenges they may be able to solve. However, the chosen models did not perform adequately. In conclusion; to perform root cause analysis on wind turbines and create reliable and useful explanations for

users like WT operators and other WT experts, a new angle of attack is needed. New and different methods must be explored and further research is necessary, some of which are presented in the following chapter.

Chapter 8

Further Work

8.1 Improving model performance

8.1.1 DT and Random Forest

As mentioned in section 5.3.1, the random forest feature importance method does not produce reliable results. Since the 5 "most important" features or variables were selected from the data set to use when training the decision tree from the RF, a bad choice could greatly affect the performance of the shallow DT. Further work includes using different ways of calculating features importances for training a decision tree.

Strobl et al. (2007) suggested permutation importance as an alternative to measuring feature importance. Permutation importance is performed by first measuring the baseline classification accuracy for a model (Strobl et al., 2007). Then, in a test set, the column of a single predictor variable is changed (permuted) and fed into the trained classifier. The classifier is then tested on the permuted set, and the accuracy is compared to the baseline accuracy. The feature importance is equal to the difference between the baseline accuracy and the drop in overall accuracy from permuting the column (Strobl et al., 2007). This method could be valuable to apply when creating decision trees.

8.1.2 SVM and LIME

To get more precise explanations, it is desirable to apply models that perform optimally. Chih-Wei Hsu and Lin (2016) have from their own experience created a step-by-step document for optimising the performance of SVMs. This is relevant to look at when building Support Vector Machine classifiers at another time.

8.2 Explainable AI

8.2.1 SHAP

An interesting model agnostic, explainable AI framework worth exploring is SHAP (SHapley Additive exPlanations). It is based upon Shapley values; a well known method from cooperative game theory, which is able to determine each feature's contribution to a model's prediction (Molnar, 2019). According to Molnar (2019), it has several advantages. Firstly, it is built on a solid theory unlike LIME which assumes linear behaviour when there is no theory to support it. Secondly, SHAP is able to produce *contrastive* explanations, which is known to be suitable to for humans 2.5.3. Thirdly, it is able to produce *global* explanations for tree based ML methods like decision trees, random forest, etc. One of its disadvantages is that it is computationally heavy for non-tree based models. Consequently it is difficult to create global explanations for these kinds of models.

8.3 Predictions with SCADA Data

Since the topic for this paper was to explore how to analyse machine learning models built on SCADA data, a natural next step could be to analyse *future* predictions with the same data. Leahy et al. (2018) explored fault predictions using SCADA data and various ML techniques, consisting of ensemble learners and variations of SVMs. They were able to predict faults up to 24 hours in advance with recall scores of around 0.8.

8.4 Interpretable Predictions

It would be interesting to determine if it's possible to provide explanations for future fault predictions using XAI methods. This is a relatively new subfield and has experienced an increased interest among researchers recently. Giurgiu and Schumann (2019) explored failure explainable predictions on time series data, using RNN classifiers. They applied LIME on top of a RNN classifier, and incorporated explainability directly into the classifier and compared the results.

Schlegel et al. (2019) explored XAI methods on various time series data. The team used CNN (convolutional neural network) and RNN (recurrent neural network) as baseline models. They identified and applied the five most most prominent XAI methods; LIME, LRP, DeepLIFT, Saliency Maps and SHAP, and compared the resulte. Schlegel et al. (2019) concluded that SHAP was the most robust framework.

8.5 Other Sensor Data

Other sensor data could be worth studying for fault detection, diagnostics, and prediction. Machine learning methods built with vibration data could be be able to detect faults at an early stage (Madsen, 2011). Figure 6.2 illustrates the typical development of a mechanical fault, and when atypical vibration frequencies could be detected (Madsen, 2011). Vibration analysis of different components on a WT could be an interesting topic for a Master's thesis.

Bibliography

Abdallah, I., Dertimanis, V., Mylonas, H., Tatsis, K., Chatzi, E., Dervilis, N., Worden, K., Maguire, E., 2018. Fault diagnosis of wind turbine structures using decision tree learning algorithms with big data. Safety and Reliability - Safe Societies in a Changing World - Proceedings of the 28th International European Safety and Reliability Conference, ESREL 2018 (iii), 3053–3062.

Barros, A., 2019. TPK4450 Data Driven Prognostic and Predictive Maintenance.

Ben-Hur, A., Ong, C. S., Sonnenburg, S., Schölkopf, B., Rätsch, G., 2008. Support vector machines and kernels for computational biology. PLoS Computational Biology 4 (10).

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

Breiman, L., 2001. Random Forest, 1–33.

Breiman, L., Friedman, J. H., Olshen, R. A., Stone, C. J., 1984. Classification And Regression Trees.

Chawla, N. V., Bowyer, K. W., Hall, L. O., Kegelmeyer, W. P., 2002. SMOTE: Synthetic minority over-sampling technique. Journal of Artificial Intelligence Research 16 (Sept. 28), 321–357.
URL `https://arxiv.org/pdf/1106.1813.pdf%0Ahttp://www.snopes.com/horrors/insects/telamonia.asp`

Chih-Wei Hsu, C.-C. C., Lin, C.-J., 2016. A Practical Guide to Support Vector Classification. Tech. Rep. 1.
URL `https://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf`

Dumais, S., Platt, J., Heckerman, D., Sahami, M., 1998. Inductive learning algorithms and representations for text categorization, 148–155.

García Márquez, F. P., Tobias, A. M., Pinar Pérez, J. M., Papaelias, M., 2012. Condition monitoring of wind turbines: Techniques and methods. Renewable Energy.

Giurgiu, I., Schumann, A., 2019. Explainable Failure Predictions with RNN Classifiers based on Time Series Data.
URL http://arxiv.org/abs/1901.08554

Godwin, J. L., Matthews, P., 2013. Classification and detection of wind turbine pitch faults through SCADA data analysis. International Journal of Prognostics and Health Management 4 (SPECIAL ISSUE 2).

Hahn, B., Durstewitz, M., 2005. Reliability of wind turbines - Experiences of 15 years with 1500 WTs. EUROMECH Colloquium 464b: Wind Energy.
URL http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Reliability+of+wind+turbines+-+Experiences+of+15+years+with+1500+WTs#1

Hameed, Z., Hong, Y. S., Cho, Y. M., Ahn, S. H., Song, C. K., 2009. Condition monitoring and fault detection of wind turbines and related algorithms: A review. Renewable and Sustainable Energy Reviews 13 (1), 1–39.

Hoffman, R. R., Klein, G., Mueller, S. T., 2018a. Explaining explanation for "explainable AI. Proceedings of the Human Factors and Ergonomics Society 1 (September), 197–201.

Hoffman, R. R., Mueller, S. T., Klein, G., Litman, J., 2018b. Metrics for Explainable AI: Challenges and Prospects (December).
URL http://arxiv.org/abs/1812.04608

James, G., Witten, D., Hastie, T., Tibshirani, R., 2013. An Introduction to Statistical Learning, 1st Edition. Vol. 64.

Kohavi, R., 1995. A study of Cross-Validation (0), 0–6.

Kraus, M., Feuerriegel, S., 2019. Forecasting remaining useful life: Interpretable deep learning approach via variational Bayesian inferences. Decision Support Systems 125 (April), 113100.
URL https://doi.org/10.1016/j.dss.2019.113100

Krüger, F., 2018. Activity, Context, and Plan Recognition with Computational Causal Behaviour Models. ResearchGate (August).
URL https://www.researchgate.net/figure/Confusion-matrix-for-multi-class-classification-The-confusion-matrix-of-a_fig7_314116591

Kusiak, A., Li, W., 2011. The prediction and diagnosis of wind turbine faults. Renewable Energy.

Leahy, K., Gallagher, C., O'Donovan, P., O'Sullivan, D. T., 2019. Issues with data quality for wind turbine condition monitoring and reliability analyses. Energies 12 (2), 1–22.

Leahy, K., Hu, R. L., Konstantakopoulos, I. C., Spanos, C. J., Agogino, A. M., O'Sullivan, D. T., 2018. Diagnosing and predicting wind turbine faults from scada data using support vector machines. International Journal of Prognostics and Health Management 9 (1), 1–11.

Lemaître, G., Nogueira, F., Aridas, C. K., 2017. Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning. Journal of Machine Learning Research 18 (17), 1–5.
URL http://jmlr.org/papers/v18/16-365

Lipton, P., 3 1990. Contrastive Explanation. Royal Institute of Philosophy Supplement 27, 247–266.
URL https://www.cambridge.org/core/product/identifier/S1358246100005130/type/journal_article

Madsen, B. N., 2011. Condition monitoring of wind turbines by electric signature analysis. A cost effective alternative or a redundant option for geared wind turbines (October), 118.

McKinney, W., et al., 2010. Data structures for statistical computing in python.

Miller, T., 2019. Explanation in artificial intelligence: Insights from the social sciences. Artificial Intelligence 267, 1–38.

Mitchell, T. M., 1997. Machine Learning. McGraw-Hill Science/Engineering/Math; (March 1, 1997).

Molnar, C., 2019. Interpretable Machine Learning.
URL https://christophm.github.io/interpretable-ml-book

Murphy, K. P., 2012. Machine Learning: A Probabilistic Perspective. The MIT press.

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

Poole, M. A., O'Farrell, P. N., 1971. The Assumptions of the Linear Regression Model. Transactions of the Institute of British Geographers 52 (52), 145.

Ribeiro, M. T., Singh, S., Guestrin, C., 2016. "Why should i trust you?" Explaining the predictions of any classifier. Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining 13-17-Augu, 1135–1144.

Rokach, L., Maimon, O., 2005. Top-down induction of decision trees classifiers - A survey. IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews 35 (4), 476–487.

Safavian, S. R., Landgrebe, D., 1991. A Survey of Decision Tree Classifier Methodology. IEEE Transactions on Systems, Man and Cybernetics 21 (3), 660–674.

Schlegel, U., Arnout, H., El-Assady, M., Oelke, D., Keim, D. A., 2019. Towards a Rigorous Evaluation of XAI Methods on Time Series (Ml).
URL http://arxiv.org/abs/1909.07082

Sciences, C., 2017. Machine Learning for Data-driven Prognostics: Methods and Applications.

Shin, S. H., Kim, S. R., Seo, Y. H., 2018. Development of a fault monitoring technique for wind turbines using a hidden Markov model. Sensors (Switzerland) 18 (6), 1–15.

Siekmann, J., Wahlster, W., 2019. Arti fi cial Intelligence, 253.

Stetco, A., Dinmohammadi, F., Zhao, X., Robu, V., Flynn, D., Barnes, M., Keane, J., Nenadic, G., 2019. Machine learning methods for wind turbine condition monitoring: A review.

Strobl, C., Boulesteix, A. L., Zeileis, A., Hothorn, T., 2007. Bias in random forest variable importance measures: Illustrations, sources and a solution. BMC Bioinformatics 8.

Tahir, M. A., Kittler, J., Yan, F., 2012. Inverse random under sampling for class imbalance problem and its application to multi-label classification. Pattern Recognition.

Tchakoua, P., Wamkeue, R., Ouhrouche, M., Slaoui-Hasnaoui, F., Tameghe, T. A., Ekemb, G., 2014. Wind turbine condition monitoring: State-of-the-art review, new trends, and future challenges. Energies 7 (4), 2595–2630.

The International Renewable Energy Agency (IRENA), 2012. Wind Power.

Vurderingsform, E., Sluttid, S., Pdf, S., 2019. TKT4915 1 Beregningsmekanikk, masteroppgave.

Wirtz, T., 1999. Table of Contents TABLE OF CONTENTS Publisher. International Nonwovens Journal 8 (1), 2–4.

Appendix A

Data Plots

A.1 Correlation plot for data set

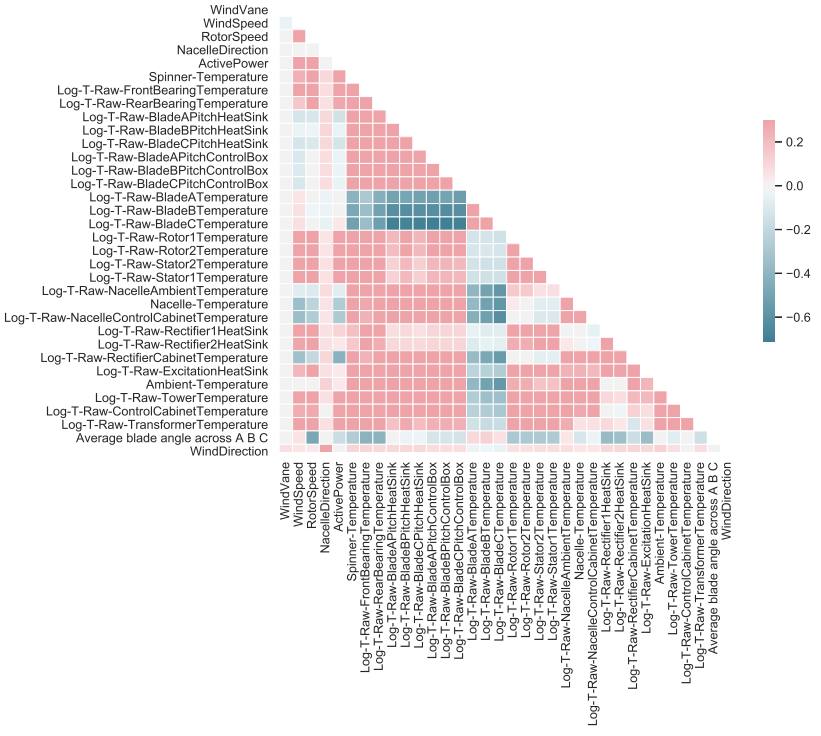
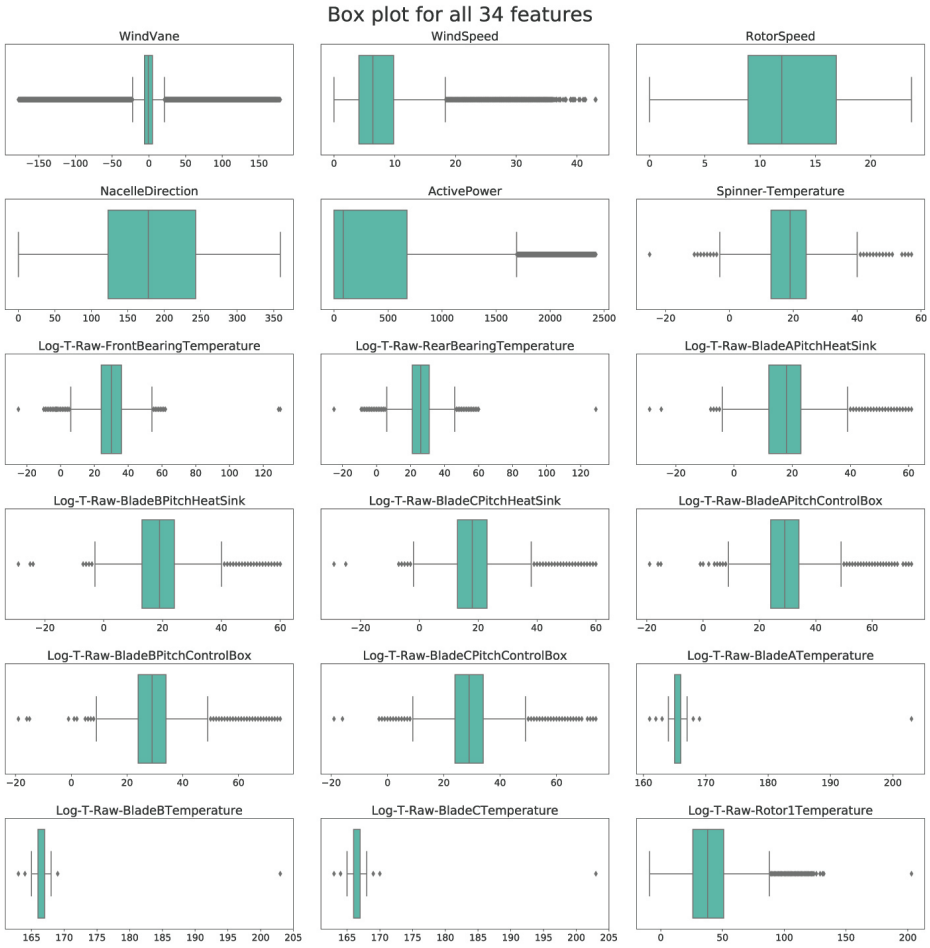
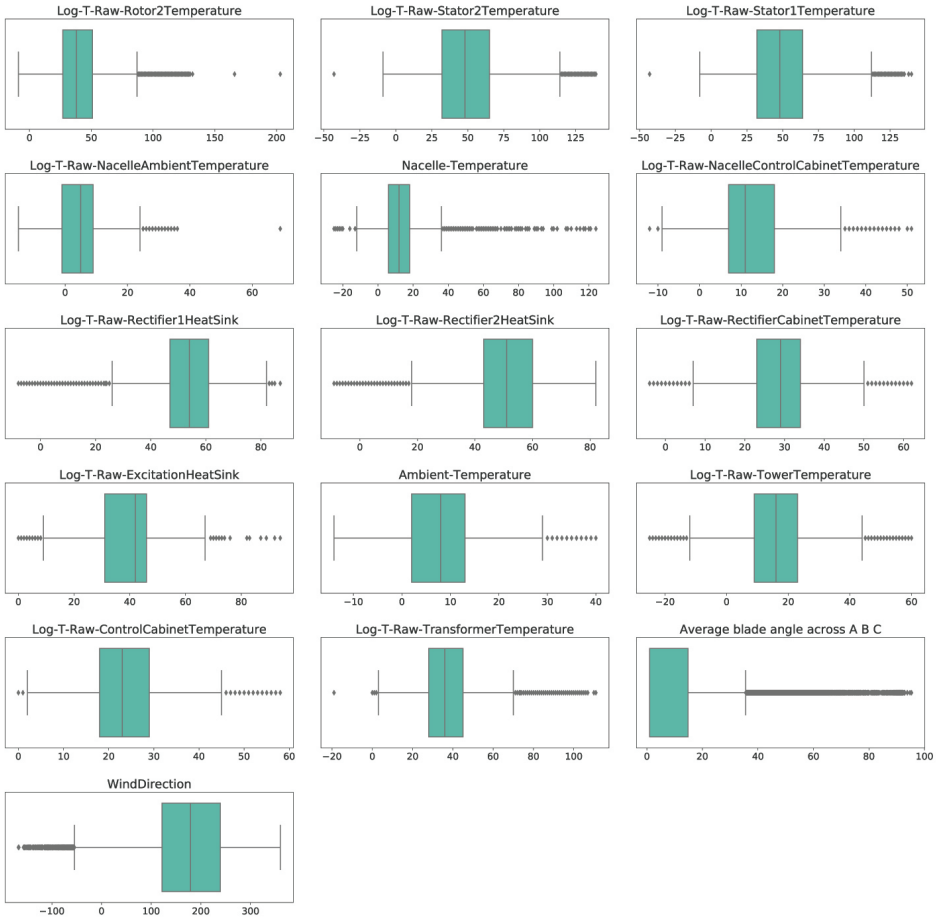


Figure A.1: Correlation plot of all the 34 features/parameters.

A.2 Box plot for data set




Log-T-Raw-Rotor2Temperature
Log-T-Raw-Stator2Temperature
Log-T-Raw-Stator1Temperature
Log-T-Raw-NacelleAmbientTemperature
Nacelle-Temperature
Log-T-Raw-NacelleControlCabinetTemperature
Log-T-Raw-Rectifier1HeatSink
Log-T-Raw-Rectifier2HeatSink
Log-T-Raw-RectifierCabinetTemperature
Log-T-Raw-ExcitationHeatSink
Ambient-Temperature
Log-T-Raw-TowerTemperature
Log-T-Raw-ControlCabinetTemperature
Log-T-Raw-TransformerTemperature
Average blade angle across A B C
WindDirection

A.3 LIME plots

Figure A.2: LIME explanation for a correct *Other* SVM prediction

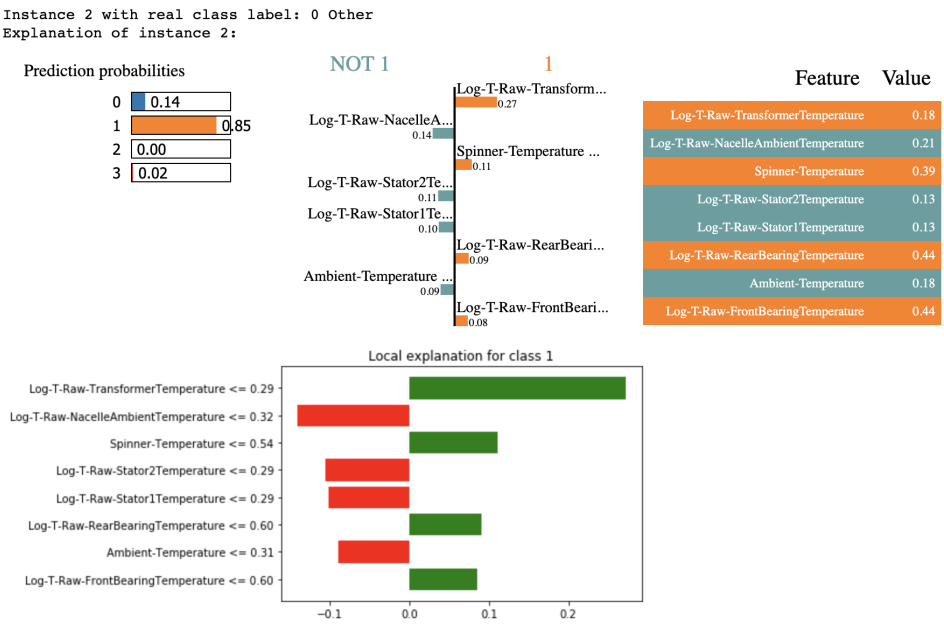


Figure A.3: LIME explanation for a correct *Generator heating - (Manual)* SVM prediction

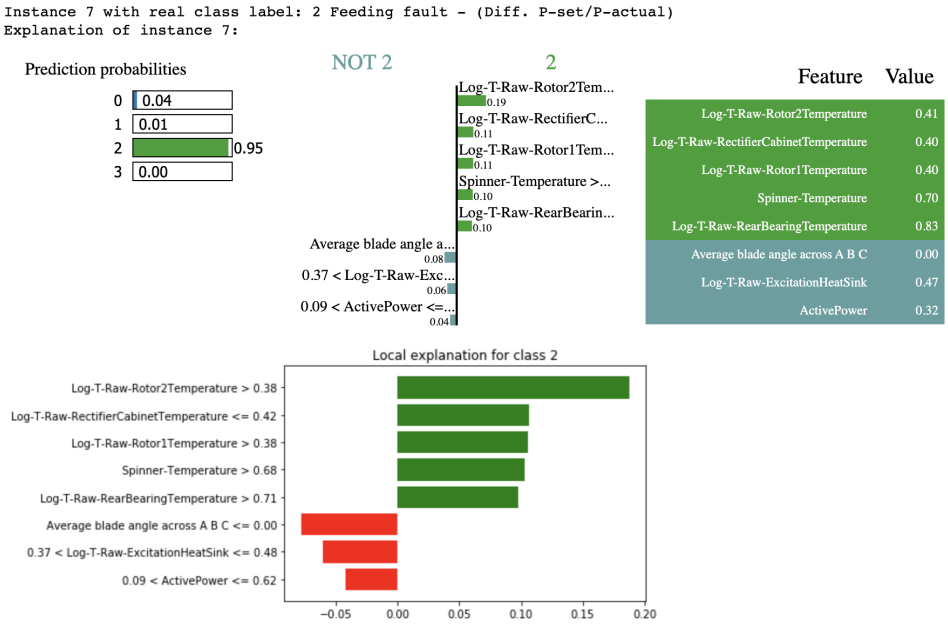


Figure A.4: LIME explanation for a correct *Feeding fault - (Diff. P-set/P-actual)* SVM prediction

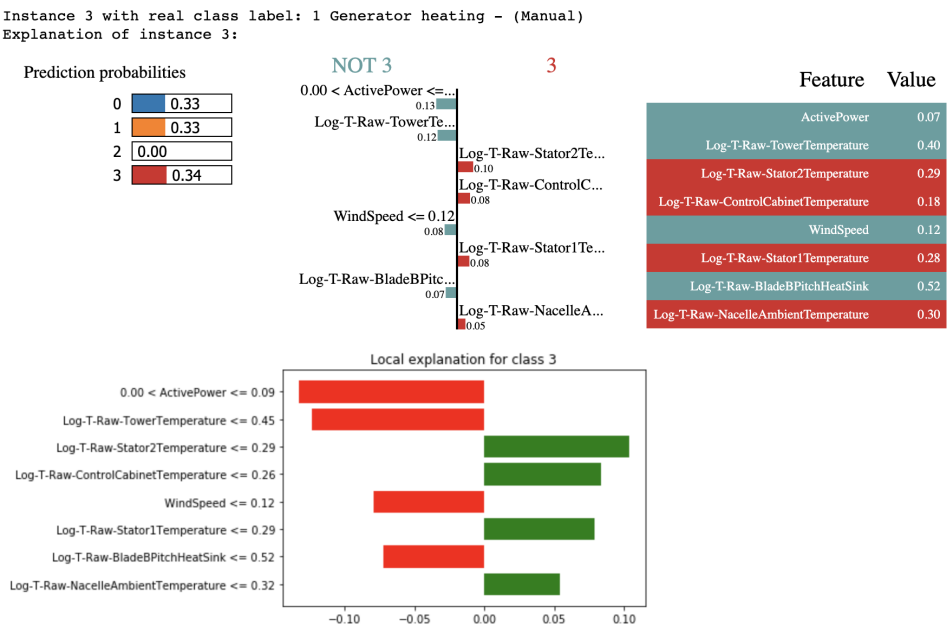


Figure A.5: LIME explanation for a an incorrect SVM prediction. Predicted *Feeding fault - (Feeding safety circuit faulty)* but was *Generator heating - (Manual)*

Appendix B

Tables

	Description	Count
0	N 2:1 Lack of wind - (Wind speed too low)	10700
1	N 8:0 Maintenance - (Maintenance)	1772
2	N 21:2 Cable twisted - (Right (2-3 turns))	710
3	N 9:8 Generator heating - (Manual)	459
4	R 60:15 Mains failure - (Overvoltage L2)	440
5	R 240:0 STATUS: Remote control PC (switched on)	430
6	R 62:7 Feeding fault - (Diff. P-set/P-actual)	424
7	N 1:1 Turbine Stopped - (Control cabinet)	363
8	N 17:0 Test safety system - (Test safety system)	338
9	R 240:246 STATUS: Remote control PC (Timeout receivebuffer)	333
10	N 3:12 Storm - (Average windspeed - (10min))	317
11	R 240:1 STATUS: Remote control PC (switched off)	306
12	R 222:1 Turbine reset - (Power failure)	300
13	R 220:33 Processor reset - (Power Control)	248
14	N 9:3 Generator heating - (Hygrostat inverter)	225
15	R 222:3 Turbine reset - (Scada system)	215
16	N 2:2 Lack of wind - (Rotor speed too low)	201
17	N 21:1 Cable twisted - (Left (2-3 turns))	118
18	N 15:1 Turbine moist - (Turbine moist Inverter 1)	115
19	R 67:508 Overtemperature - (Thermo switch heatsink inv. 8)	111
20	R 62:30 Feeding fault - (Feeding safety circuit faulty)	105
21	R 29:41 Anemonmeter Interface - (Fault 1 ultra sonic sensor)	105
22	N 9:1 Generator heating - (Isometer)	99
23	R 62:107 STATUS: Feeding fault (To low power inverter 7)	80
24	R 25:134 Fault yaw inverter - (Overspeed system 1)	75

Table B.1: The 50 most common faults at Bessaker (Part I)

	Description	Count
25	R 43:0 Main security stop capacitor - (Main security circuit fault)	73
26	R 62:104 STATUS: Feeding fault (To low power inverter 4)	70
27	R 42:334 Pitch control error - (Overtemperature motor blade C)	62
28	N 1:3 Turbine Stopped - (Scada system (ENERCON))	61
29	R 62:202 Feeding fault - (Too much power inverter 2)	55
30	R 228:90 Timeout warnmessage - (Prot. circuit-breaker tripped)	47
31	R 304:98 Data bus error (Timeout) - (Rectifier 2)	46
32	N * 3h 14:11 Ice detection - (Rotor (power measurement))	45
33	R 29:31 Anemonmeter Interface - (Error data transmission us-sensor)	45
34	R 60:12 Mains failure - (Undervoltage L2)	43
35	R 80:42 Fault excitation - (Excit. current too low during operation)	43
36	R 300:81 Turbine control bus error - (I/O-board control cabinet)	41
37	N 9:4 Generator heating - (Hygrostat rectifier)	40
38	R 25:136 Fault yaw inverter - (Yawing moment too low system 1)	40
39	R 70:12 Generator overtemperature - (Stator (slot 2))	38
40	R 31:11 Tower oscillation - (Transversal oscillation (max.))	37
41	R 42:234 Pitch control error - (Overtemperature motor blade B)	36
42	R 60:28 Mains failure - (Overfrequency (restart))	35
43	R 222:2 Turbine reset - (Quit button)	34
44	*11:0 Rotor brake activated manual - (Rotor brake activated manual)	33
45	* 25:201 Fault yaw inverter - (Cross short circuit system 2)	33
46	R 438:206 Error supply IGBT-driver - (Inverter Control 6)	32
47	R 42:105 Pitch control error - (Angle monitoring 3 blade A)	32
48	R 62:719 Feeding fault - (Overload chopper several inv.)	31
49	* 80:10 Fault excitation - (Overtemperature heatsink)	31

Table B.2: The 50 most common faults at Bessaker (Part II)

	Alarm Sequence	Count
0	Lack of wind - (Wind speed too low) → Lack of wind - (Wind speed too low)	905
1	Cable twisted - (Right (2-3 turns)) → Lack of wind - (Wind speed too low)	658
2	STATUS: Remote control PC (switched off) → STATUS: Remote control PC (switched on)	306
3	Mains failure - (Overvoltage L2) → Mains failure - (Overvoltage L2)	278
4	Maintenance - (Maintenance) → Generator heating - (Manual)	273
5	Lack of wind - (Wind speed too low) → Cable twisted - (Right (2-3 turns))	236
6	Turbine reset - (Power failure) → Processor reset - (Power Control)	233
7	STATUS: Remote control PC (Timeout receivebuffer) → Turbine reset - (Power failure)	223
8	STATUS: Remote control PC (Timeout receivebuffer) → Turbine reset - (Power failure) → Processor reset - (Power Control)	184
9	Maintenance - (Maintenance) → Maintenance - (Maintenance)	178
10	Maintenance - (Maintenance) → Turbine Stopped - (Control cabinet)	171
11	Maintenance - (Maintenance) → Lack of wind - (Wind speed too low)	163
12	Mains failure - (Overvoltage L2) → Mains failure - (Overvoltage L2) → Mains failure - (Overvoltage L2)	160
13	Turbine Stopped - (Control cabinet) → Maintenance - (Maintenance)	150
14	Lack of wind - (Wind speed too low) → Generator heating - (Manual)	130
15	Cable twisted - (Left (2-3 turns)) → Lack of wind - (Wind speed too low)	106
16	Test safety system - (Test safety system) → Lack of wind - (Wind speed too low)	93
17	Generator heating - (Manual) → Lack of wind - (Wind speed too low)	93
18	Storm - (Average windspeed - (10min)) → Storm - (Average windspeed - (10min))	86
19	Lack of wind - (Wind speed too low) → Generator heating - (Hygrostat inverter)	70
20	STATUS: Remote control PC (switched on) → STATUS: Remote control PC (switched off)	68
21	STATUS: Feeding fault (To low power inverter 7) → STATUS: Feeding fault (To low power inverter 7)	61
22	Generator heating - (Hygrostat inverter) → Maintenance - (Maintenance)	59
23	Lack of wind - (Wind speed too low) → Maintenance - (Maintenance)	52
24	Generator heating - (Hygrostat inverter) → Lack of wind - (Wind speed too low)	47
25	Maintenance - (Maintenance) → Generator heating - (Hygrostat inverter)	47
26	STATUS: Remote control PC (switched off) → STATUS: Remote control PC (switched on) → STATUS: Remote control PC (switched off)	46
27	Lack of wind - (Wind speed too low) → STATUS: Remote control PC (switched off)	46
28	Feeding fault - (Feeding safety circuit faulty) → Data bus error (Timeout) - (Rectifier 2)	44
29	Lack of wind - (Wind speed too low) → STATUS: Remote control PC (switched off) → STATUS: Remote control PC (switched on)	44
30	Turbine moist - (Turbine moist Inverter 1) → Lack of wind - (Wind speed too low)	44
31	STATUS: Remote control PC (switched on) → Lack of wind - (Wind speed too low)	43
32	Anemonmeter Interface - (Fault 1 ultra sonic sensor) → Anemonmeter Interface - (Fault 1 ultra sonic sensor)	40
33	STATUS: Feeding fault (To low power inverter 7) → STATUS: Feeding fault (To low power inverter 7) → STATUS: Feeding fault (To low power inverter 7)	40
34	Lack of wind - (Wind speed too low) → STATUS: Remote control PC (switched off) → STATUS: Remote control PC (switched on) → Lack of wind - (Wind speed too low)	39
35	STATUS: Remote control PC (switched off) → STATUS: Remote control PC (switched on) → Lack of wind - (Wind speed too low)	39
36	Turbine Stopped - (Control cabinet) → Lack of wind - (Wind speed too low)	38
37	14:11 Ice detection - (Rotor (power measurement)) → Maintenance - (Maintenance)	38
38	Overtemperature - (Thermo switch heatsink inv. 8) → Overtemperature - (Thermo switch heatsink inv. 8)	38
39	Lack of wind - (Wind speed too low) → Turbine moist - (Turbine moist Inverter 1)	36
40	Lack of wind - (Wind speed too low) → Turbine moist - (Turbine moist Inverter 1) → Lack of wind - (Wind speed too low)	36

Table B.3: Most common alarm sequences at Bessaker (Part I)

	Alarm Sequence	Count
41	Turbine reset - (Power failure) → Mains failure - (Overfrequency (restart))	35
42	Maintenance - (Maintenance) → Turbine Stopped - (Control cabinet) → Lack of wind - (Wind speed too low)	34
43	Lack of wind - (Wind speed too low) → Cable twisted - (Left (2-3 turns))	34
44	STATUS: Feeding fault (To low power inverter 4) → STATUS: Feeding fault (To low power inverter 4)	34
45	Feeding fault - (Feeding safety circuit faulty) → Lack of wind - (Wind speed too low)	34
46	Generator heating - (Hygrostat inverter) → Lack of wind - (Wind speed too low) → Turbine moist - (Turbine moist Inverter 1)	33
47	Generator heating - (Hygrostat inverter) → Lack of wind - (Wind speed too low) → Turbine moist - (Turbine moist Inverter 1) → Lack of wind - (Wind speed too low)	33
48	Maintenance - (Maintenance) → Turbine Stopped - (Control cabinet) → Maintenance - (Maintenance)	32
49	Mains failure - (Undervoltage L2) → STATUS: Remote control PC (Timeout receivebuffer)	31
50	STATUS: Remote control PC (switched on) → Maintenance - (Maintenance)	30
51	Turbine moist - (Turbine moist Inverter 1) → Generator heating - (Hygrostat inverter)	29
52	Anemonmeter Interface - (Fault 1 ultra sonic sensor) → Anemonmeter Interface - (Fault 1 ultra sonic sensor) → Anemonmeter Interface - (Fault 1 ultra sonic sensor)	27
53	STATUS: Remote control PC (Timeout receivebuffer) → Turbine reset - (Power failure) → Mains failure - (Overfrequency (restart))	25
54	Fault yaw inverter - (Cross short circuit system 2) → Fault yaw inverter - (Cross short circuit system 2)	25
55	Mains failure - (Undervoltage L2) → STATUS: Remote control PC (Timeout receivebuffer) → Turbine reset - (Power failure)	24
56	Timeout warnmessage - (Prot. circuit-breaker tripped) → Maintenance - (Maintenance)	24
57	STATUS: Remote control PC (switched off) → STATUS: Remote control PC (switched on) → STATUS: Remote control PC (switched off) → STATUS: Remote control PC (switched on)	24
58	STATUS: Remote control PC (switched on) → STATUS: Remote control PC (switched off) → STATUS: Remote control PC (switched on)	24
59	Maintenance - (Maintenance) → Generator heating - (Isometer)	24
60	STATUS: Feeding fault (To low power inverter 7) → STATUS: Feeding fault (To low power inverter 7) → STATUS: Feeding fault (To low power inverter 7) → STATUS: Feeding fault (To low power inverter 7)	24
61	STATUS: Remote control PC (switched off) → STATUS: Remote control PC (switched on) → STATUS: Remote control PC (switched on)	23
62	STATUS: Remote control PC (switched on) → STATUS: Remote control PC (switched on)	23
63	Pitch control error - (Angle monitoring 3 blade A) → Turbine reset - (Scada system)	23
64	Fault speed sensor - (Measurement pitch control) → Fault speed sensor - (Measurement pitch control)	23
65	Turbine Stopped - (Scada system (ENERCON)) → STATUS: Remote control PC (Timeout receivebuffer)	22
66	STATUS: Remote control PC (switched off) → STATUS: Remote control PC (switched on) → Maintenance - (Maintenance)	22
67	Anemonmeter Interface - (Fault 1 ultra sonic sensor) → Fault wind measurement - (Constant wind direction)	22
68	Maintenance - (Maintenance) → STATUS: Remote control PC (switched off)	21
69	Maintenance - (Maintenance) → STATUS: Remote control PC (switched off) → STATUS: Remote control PC (switched on)	21
70	Maintenance - (Maintenance) → STATUS: Remote control PC (switched off) → STATUS: Remote control PC (switched on) → Maintenance - (Maintenance)	21

Table B.4: Most common alarm sequences at Bessaker (Part II)

Appendix C

Evaluating Explainable AI

Explanation Satisfaction Scale

1. From the explanation, I **understand** how the [software, algorithm, tool] works.

5	4	3	2	1
I agree strongly	I agree somewhat	I'm neutral about it	I disagree somewhat	I disagree strongly

2. This explanation of how the [software, algorithm, tool] works is **satisfying**.

5	4	3	2	1
I agree strongly	I agree somewhat	I'm neutral about it	I disagree somewhat	I disagree strongly

3. This explanation of how the [software, algorithm, tool] works has **sufficient detail.**

5	4	3	2	1
I agree strongly	I agree somewhat	I'm neutral about it	I disagree somewhat	I disagree strongly

4. This explanation of how the [software, algorithm, tool] works seems **complete**.

5	4	3	2	1
I agree strongly	I agree somewhat	I'm neutral about it	I disagree somewhat	I disagree strongly

5. This explanation of how the [software, algorithm, tool] works **tells me how to use** it.

5	4	3	2	1
I agree strongly	I agree somewhat	I'm neutral about it	I disagree somewhat	I disagree strongly

6. This explanation of how the [software, algorithm, tool] works is **useful to my goals**.

5	4	3	2	1
I agree strongly	I agree somewhat	I'm neutral about it	I disagree somewhat	I disagree strongly

Figure C.1: Evaluating Explanation Satisfaction Page 1 Hoffman et al. (2018b)

7. This explanation of the [software, algorithm, tool] shows me how **accurate** the [software, algorithm, tool] is.

5	4	3	2	1
I agree strongly	I agree somewhat	I'm neutral about it	I disagree somewhat	I disagree strongly

8. This explanation lets me judge when I should **trust and not trust** the [software, algorithm, tool]

5	4	3	2	1
I agree strongly	I agree somewhat	I'm neutral about it	I disagree somewhat	I disagree strongly

Figure C.2: Evaluating Explanation Satisfaction Page 2 Hoffman et al. (2018b)

Appendix D

Source Code

D.1 Pseudo Code

Algorithm *SMOTE*(T, N, k)
Input: Number of minority class samples T; Amount of SMOTE N%; Number of nearest neighbors k
Output: (N/100) * T synthetic minority class samples
1. (∗ *If N is less than 100%, randomize the minority class samples as only a random percent of them will be SMOTEd.* ∗)
2. **if** $N < 100$
3. **then** Randomize the T minority class samples
4. $T = (N/100) * T$
5. $N = 100$
6. **endif**
7. $N = (int)(N/100)$ (∗ *The amount of SMOTE is assumed to be in integral multiples of 100.* ∗)
8. k = Number of nearest neighbors
9. *numattrs* = Number of attributes
10. *Sample*[ ][ ]: array for original minority class samples
11. *newindex*: keeps a count of number of synthetic samples generated, initialized to 0
12. *Synthetic*[ ][ ]: array for synthetic samples
(∗ *Compute k nearest neighbors for each minority class sample only.* ∗)
13. **for** $i \leftarrow 1$ **to** T
14. Compute k nearest neighbors for i, and save the indices in the *nnarray*
15. Populate(N, i, *nnarray*)
16. **endfor**

Populate(N, i, *nnarray*) (∗ *Function to generate the synthetic samples.* ∗)
17. **while** $N \neq 0$
18. Choose a random number between 1 and k, call it nn. This step chooses one of the k nearest neighbors of i.
19. **for** $attr \leftarrow 1$ **to** *numattrs*
20. Compute: $dif = Sample[nnarray[nn]][attr] - Sample[i][attr]$
21. Compute: gap = random number between 0 and 1
22. $Synthetic[newindex][attr] = Sample[i][attr] + gap * dif$
23. **endfor**
24. $newindex{+}{+}$
25. $N = N - 1$
26. **endwhile**
27. **return** (∗ *End of Populate.* ∗)
End of Pseudo-Code.

Figure D.1: SMOTE pseudocode by Chawla et al. (2002)