SUPERVISED MACHINE LEARNING BASED FAULT DIAGNOSIS AND PREDICTION FOR WIND TURBINE OPERATING IN EXTREME WEATHER

A report submitted in partial fulfillment of the requirements

for the degree of

Master of Science (by Research)

in

Disaster Management and Risk Reduction

by

Gaurav Jyoti Gogoi

(214357006)

Under the Supervision of

Prof. Arunasis Chakraborty

CENTRE FOR DISASTER MANAGEMENT & RESEARCH

INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI

June, 2023

Declaration

I declare that this written submission represents my ideas in my own words, and where others' ideas or words have been included, I have adequately cited and referenced the sources. I also declare that I have adhered to all academic honesty and integrity principles and have not misrepresented, fabricated, or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will cause disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been appropriately cited or from whom proper permission has not been taken when needed.

Date: 26-06-2023

Gaurav Jyoti Gogoi
(Roll no. 214357006)
Centre for Disaster Management & Research
Indian Institute of Technology, Guwahati
Guwahati - 781039, India

Certificate

It is certified that the work contained in this report entitled "**Wind Turbine Fault Diagnosis and Prediction using Supervised Machine Learning,**" submitted by **Gaurav Jyoti Gogoi (Roll No. 214357006)** for the partial fulfillment of the requirements for the award of **Master of Science (by Research)** in **Disaster Management and Risk Reduction** at the Indian Institute of Technology, Guwahati is an authentic work carried out by him under my supervision, and this work has not been submitted elsewhere for a degree.

Date: 26-06-2023

Prof. Arunasis Chakraborty
Department of Civil Engineering
and Centre for Disaster Management & Research
Indian Institute of Technology, Guwahati
Guwahati - 781039, India

Acknowledgment

I take this opportunity to express my sincere thanks and gratitude to my project supervisor Prof. Arunais Chakraborty for his invaluable support throughout my thesis with his patience and knowledge. His effusive cooperation and coordination immensely helped me to complete this research work. His suggestions and encouraging interactions gave a sound basis for the present thesis.

I thank Mr. Debashish Panda, Mr. Arka Mitra, and Mr. Sanjoy Pal for their unconditional support. I can never forget their timely help in carrying out my work and their positive attitude in solving problems.

Finally, I take this opportunity to extend my deep appreciation to my family and friends for their support and encouragement.

Abstract

Wind energy is a widely used and environmentally friendly resource that benefits the economy. However, wind turbines are prone to faults that can spread to nearby components and result in complex and expensive problems. Unscheduled maintenance due to component failure can lead to significant downtime and revenue loss. To address this issue, efforts have been made to develop Condition Monitoring Systems (CMSs) based on retrofitting costly sensors to turbines.

This project aims to gain valuable insight into turbine performance at a lower cost by performing complex analyses of existing data from the turbine's Supervisory Control and Data Acquisition (SCADA) system. The study analyzes fault and status data for a turbine located on the southern coast of Ireland to identify periods of normal and abnormal operation. Classification techniques are applied to SCADA data to detect and diagnose faults, and the approach is extended to allow prediction and diagnosis in advance of specific faults.

The study considered several types of faults and used multiple machine learning algorithms, including Support Vector Machine (SVM), K-Nearest Neighbor (KNN), Random Forest, Extreme Gradient Boosting (XGBoost), and Bayesian Neural Network (BNN). The performance of the models was assessed using evaluation metrics such as recall, precision, F1-score, and accuracy. The results of the study show that the proposed methodology can predict failures with an accuracy of up to 75%. This demonstrates the potential of machine learning techniques for optimizing predictive maintenance strategies.

Despite some limitations, such as a lack of testing on a different dataset, the results suggest that supervised machine learning has the potential to be a valuable tool for improving the reliability and efficiency of wind turbines. Future research could focus on developing more accurate models and testing them on real-time datasets.

Content

List of Tables

List of Figures

Chapter 1
Introduction

Wind turbines are a renewable and clean source of energy, but they are also prone to various faults that can affect their performance and reliability. Fault diagnosis and prediction are essential tasks for wind turbine condition monitoring and maintenance, which aim to detect and identify the occurrence and location of faults, as well as to estimate their severity and future evolution.

Supervised machine learning is a data-driven approach that can learn from historical operational data and fault labels to build models that can perform fault diagnosis and prediction tasks. This thesis report focuses on applying supervised machine learning methods to wind turbine fault diagnosis and prediction using SCADA data, which are commonly used as indicators of the health condition of wind turbines.

The motivation for this thesis report stems from the increasing importance of wind energy as a renewable and clean source of energy, as well as the challenges and costs associated with maintaining wind turbines. The purpose of this thesis report is to compare different supervised machine learning methods for wind turbine fault diagnosis and prediction and to evaluate their performance on real-world vibration data collected from an offshore wind farm. The methods include traditional machine learning algorithms, such as Support Vector Machines, Decision Trees, and Random Forests, as well as deep learning algorithms, such as Bayesian Neural Networks.

1.1 Condition Monitoring of Wind Turbines

In the case of wind turbines, CM is an integral part of operation and maintenance (O&M), which includes the management, monitoring, and high-level onshore control of the wind farm, while maintenance involves interventions required to maintain the installation. Reactive (or corrective, run-to-failure) maintenance is the costliest and does not utilize CM with components being replaced whenever defects occur or accumulate; preventive (scheduled) maintenance involves replacing components at the next intervention, hopefully before a related fault occurs; CM-based predictive maintenance strategies enable maintenance to determine when components are likely to fail in order to replace them in advance.

In terms of CM, we can view it from several perspectives. CM can be applied at various levels of granularity; at the most granular level, we can monitor the condition of wind turbine sub-components (for instance, drivetrain); at the most coarse-grained uppermost level, we can consider the entire wind farm. By combining the signals provided by different models, a higher-level warning can be provided for the entire turbine.

Secondly, the manner in which monitoring is performed can have a physical effect on the component being monitored. The literature identifies two primary types of monitoring.

- **Intrusive Monitoring**: involves vibration analysis, oil debris monitoring, shock pulse methods, etc. Such methods impose a penalty (wear) on the component being monitored.
- **Non-intrusive Monitoring**: involves ultrasonic testing techniques, visual inspection, acoustic emission, thermography, performance monitoring using power signal analysis, etc.

Finally, CM can be used to detect faults either in real-time or in the future. Consequently, we distinguish between

- **CM for diagnosis (fault detection)**, where we identify a fault when it happens. Ensuring that the CM can identify the presence of failure should be a prerequisite for building an ML model for prognosis.
- **CM for prognosis (fault prediction)**, where the underlying model finds patterns in the signal data that are predictive of failures in the future.

When deciding which components to monitor, it is critical to consider the failure rate and downtime per failure of different sub-components. Since they may incur the greatest potential impact, components that are more likely to fail or can cause long downtimes are prioritized. An analysis of seven surveys by Pfaffel et al. determined the annual failure rates of various subsystems of a wind turbine and the mean daily downtime. The study concluded that certain components (such as the rotor (particularly the pitch system), transmission, and power system) tend to fail at a higher rate than others. Carroll et al. In their study of over 300 offshore wind turbines, Carroll et al. found that the failure rate per offshore turbine per year was approximately 10. Approximately 80% required minor repairs (1K Euro), 17.5% required major repairs (1-10K Euro), and 2.5% required major replacements (>10K Euro). They

identified the pitch/hydraulic, generator, and other subsystems (door/hatch issues, covers, bolts, lightning) as contributing the most to failure rates. Generators and converters tend to have higher levels of failure rates in offshore wind turbines than in onshore ones. Typical failures in gearboxes include slip ring, grease pipe, rotor issues failures in planetary gears and bearings, intermediate and highspeed shaft bearings, and lubrication system malfunction.

According to Crabtree et al., despite the wide variety of commercial CM systems in use, there is no consensus regarding future research directions. According to their report, current CM for wind turbines relies heavily upon established methodologies derived from conventional rotating machine industries. The most common methods of conducting CM are acoustic measurements, electrical effects monitoring, power quality and temperature monitoring, oil debris monitoring, vibration analysis, physics-based data analytics, etc.

1.2 Machine Learning for Condition Monitoring

ML is the process of building an inductive model that learns from a limited amount of data without specialist intervention. In the taxonomy of ML models (Fig. 1), supervised learning predicts an output variable using labeled input data, while unsupervised learning draws inferences from data without labeled inputs (such as done by clustering algorithms, recommender systems, etc.). For supervised learning, we distinguish between models that predict a numeric variable (regression) or a categorical variable (classifiers). Learning in models translates into fitting a model's parameters to a specific dataset, iteratively updating them with several passes through the data until a specific predefined function is minimized.

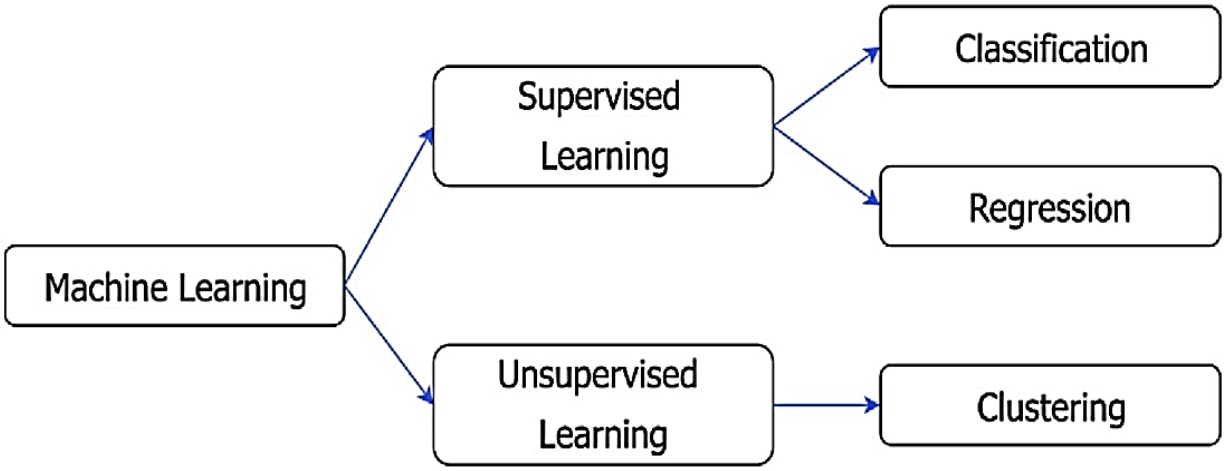


Figure 1: Taxonomy of ML models (Stetco et al., 2018)

The ML process can be represented as a series of steps

- **Data acquisition and pre-processing,** where possibly different data sets and modalities are integrated, cleaned of outliers, etc.
- **Feature selection and extraction**, important signals and characteristics are identified and extracted from the data.
- **Model selection,** an appropriate model is chosen, taking into consideration the task to be solved.
- **Validation,** a performance measure is used that is specific to the task, including accuracy (classification) and mean absolute error (regression), evaluated on a validation set of data.

Fig. 2 illustrates a typical detailed workflow for two common ML tasks.

- **Classification/prediction:** Important steps include data pre-processing (dealing with missing data, outliers, etc.), classes equalization (ensuring the classes to be predicted have equal distribution so that the model is not biased), filter/wrapper feature selection and extraction (for keeping only relevant features), classification model fitting (where the model's parameters are estimated), cross-validation (where the model's generalizability is tested).

  The solution can be cast as a prediction problem if the features are fed into the model with labels at future times (e.g., $t + 1$), which extends the solution from diagnostic to prognostic.

- **Regression-based anomaly detection:** Here, the task is to identify how signals and features are related to outputs in different components. This relationship is captured by fitting regression models when the system is in a healthy state. When new data comes in, it is compared to what the model predicts for a healthy state, and if a deviation is found for several consecutive time intervals, an alarm is raised. The behavior of a component (from low to high granularities) can be captured through regressions of different complexities (from a simple linear model to a complex non-linear one).

The ML model selection step is particularly significant as it is the core functionality that learns from past data and generalizes into the future. Such models have been used for different tasks, including classification, regression, anomaly detection, synthesis and sampling,

imputation of missing values, denoising, density estimation, and many others. Several different models have been suggested for learning from data. Support vector machines (SVMs) and neural networks (NNs) are two common models that have been used in ML for diagnostics and prognostics.

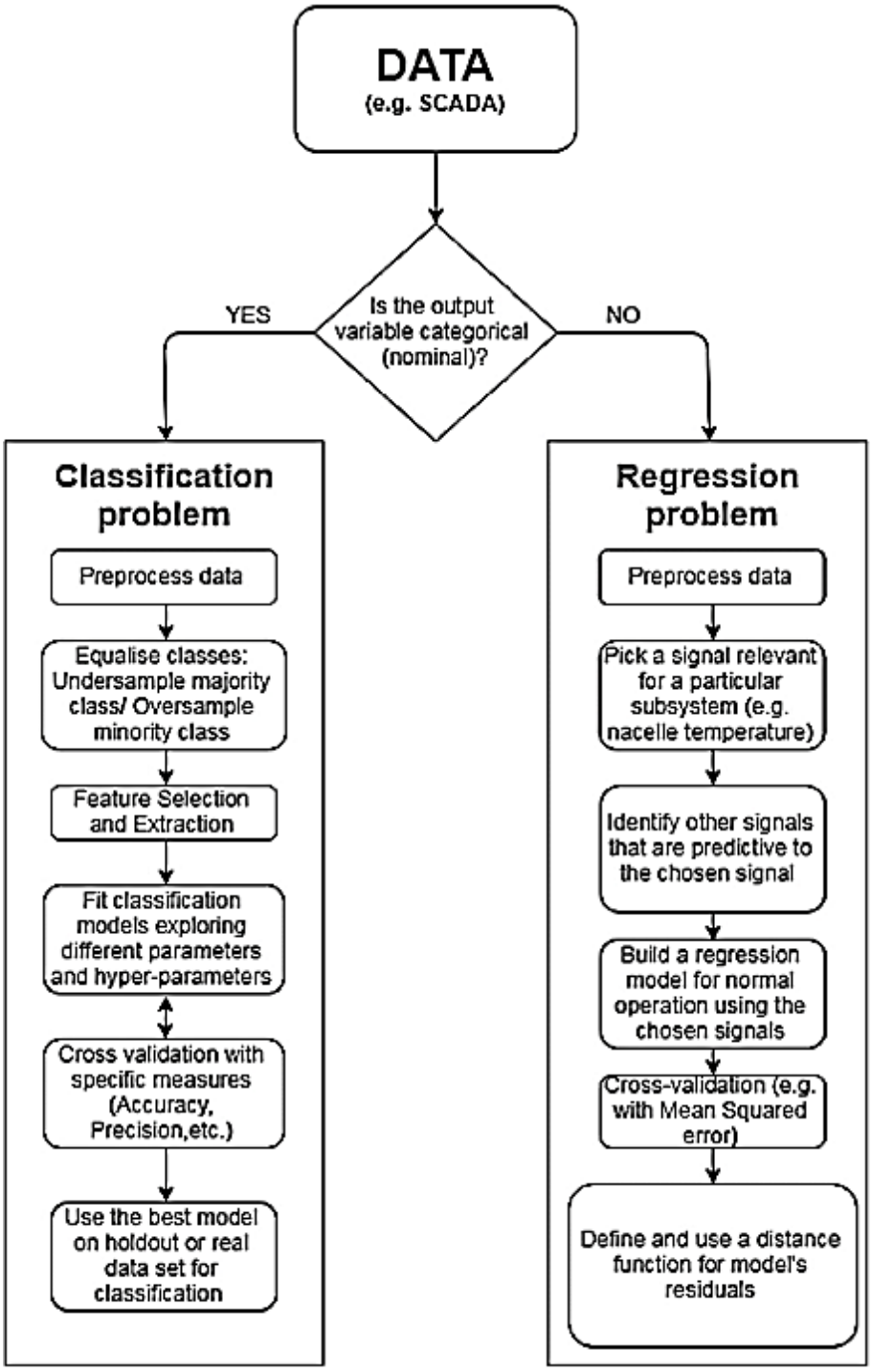


Figure 2: Two main approaches to learning from data (Stetco et al., 2018)

The ML model selection step is particularly significant as it is the core functionality that learns from past data and generalizes into the future. Such models have been used for different tasks, including classification, regression, anomaly detection, synthesis and sampling, imputation of missing values, denoising, density estimation, and many others. Several different models have been suggested for learning from data. Support vector machines (SVMs) and neural networks (NNs) are two common models that have been used in ML for diagnostics and prognostics.

Connectionist models, such as NNs, consist of simple replicated computing units called neurons, which can communicate with and pass information to each other through links between the synapses. While the main principles behind NNs have been around for some time, the availability of larger data sets, better initialization algorithms, larger sets of neuron activation functions, and more powerful machines have made it possible to train NNs composed of hundreds of stacked hidden layers. Although the training time of an NN can be potentially long, when it comes to actual classification or regression, the application of models is comparatively very fast. However, results obtained using NNs are highly dependent on the choice of architecture used, weight initialization, activation function, optimization procedure etc., and the process can require much effort and expertise. Moreover, if transparency, explanation or audit of a model is important, NNs are not well suited.

SVMs are often used in fault detection and CM, and for complex data sets in general. They perform linear/non-linear classification or regression by finding decision boundary hyperplanes that best separate classes of instances, i.e. by leaving the widest possible margin to the instances closest to the margin (see Fig. 4). It is not always possible to clearly separate instances (e.g. in the presence of outliers) and a parameter C (slack variable) allows control of margin violations. For non-linear problems, adding polynomial features created from existing ones can make the problem linearly separable in a higher dimensional space. Implementations of SVMs (e.g. SCIKIT-Learn) have several ways to transform a problem into a linearly separable one with the use of kernels (polynomial, RBF, etc.).

As with other classes of learning algorithms, NNs and SVMs can be used as classifiers (where a nominal variable is predicted) or regressors (where a numeric variable is predicted). Compared to NNs, not only do SVMs always find the global minimum when performing

optimization on the training set, but they also have an intuitive graphical interpretation. SVMs can be slow and training on large dataset remains a challenge; time complexity is usually between O (m2n) and O (m3n), where m is the number of instances and n is the number of features. Having a multitude of kernels to choose from is another complication and without any assumptions about the underlying data distribution, the search for optimum kernel (and kernel parameter) can introduce a significant time burden.

Validity of ML models can be estimated through several specific measures in combination with an out-of-sample technique such as n-fold cross validation which assess how well the results of the model will generalize beyond the training data. Classification models are typically evaluated using accuracy, sensitivity, specificity and F1-measure.

1.3 Aim and Organisation of the Project

In this thesis, we compare various machine learning algorithms for fault detection and diagnosis in wind turbines using SCADA data. Wind turbines are complex and interconnected systems that require effective fault diagnosis to ensure their safety and performance. Fault diagnosis is the process of identifying and locating the source of a malfunction in a system based on observed symptoms. Machine learning is a branch of artificial intelligence that uses data to train algorithms that can learn from patterns and make predictions. We use five machine learning algorithms, namely Support Vector Machine (SVM), K-Nearest Neighbors (KNN), Random Forest (RF), XGBoost, and Bayesian Network, to classify different fault types based on data collected from a coastal site in the South of Ireland where a 3 MW turbine has been installed at a large biomedical device manufacturing facility to offset energy costs. We balance the class weights among the fault classes to account for the imbalance in datasets, which means that some fault types are more frequent or rare than others. We also perform hyperparameter tuning to optimize the f1-score of the classes, which is a measure of the trade-off between precision and recall, in order to minimize the probability of false positives, which are cases where the algorithm predicts a fault when there is none. We use Python version 3.9.12 for the computation of the entire machine learning algorithm.

The rest of this thesis is organized as follows:

In Section 2, we present a literature review on wind turbine fault detection and diagnosis using machine learning. We provide an overview of the existing state-of-the-art methods and techniques, as well as their advantages and limitations. We also describe the mathematical formulation of each machine learning algorithm and its variants that we employ in this thesis.

In Section 3, we describe the mathematical formulation of each machine learning algorithm and its variants that we employ in this thesis. We describe the SCADA data used in our analysis and provide details on its collection and preprocessing. SCADA stands for Supervisory Control And Data Acquisition, which is a system that monitors and controls industrial processes such as wind turbines. We explain how we extract relevant features from the raw data and how we label them according to different fault types.

In Section 4, we outline our methodology for analyzing the data and developing our models. We describe how we split the data into training, validation, and testing sets, how we perform cross-validation and grid search to tune the hyperparameters of each machine learning model, and how we evaluate their performance using various metrics such as accuracy, precision, recall, f1-score and confusion matrix.

In Section 5, we present our results and discuss their implications for wind turbine fault detection and diagnosis. We compare the performance of different machine learning models with different parameters and features on different fault types and scenarios. We also analyze the sources of errors and limitations of our models and suggest possible ways to improve them.

In Section 6, we conclude by summarizing the research presented in this thesis and drawing conclusions from our results. We highlight the main contributions and findings of our work and provide some directions for future research.

Chapter 2
Literature Review

Wind turbines are equipped with SCADA systems that collect and transmit various measurements of the turbine's operation and performance, such as power output, wind speed, generator speed, temperatures, currents, and voltages. These measurements can be used for condition monitoring purposes, as they can reflect the health status of the turbine and its components. By applying data analysis techniques to the SCADA data, it is possible to detect and diagnose faults in wind turbines, as well as to predict their future occurrence and evolution. This can help to improve the efficiency and reliability of wind turbines, as well as to reduce the maintenance costs and downtime. However, SCADA data has some limitations, such as low sampling rate, high noise level, and large volume, which pose challenges for data analysis methods (Tautz-Weinert & Watson, 2017).

2.1 Power curve analysis for fault detection

One of the common methods for wind turbine fault detection using SCADA data is based on the analysis of the power curve, which is the relationship between the power output and the wind speed at the hub height of the turbine. The power curve can be modelled under normal operating conditions using various techniques, such as polynomial regression, artificial neural networks, or Gaussian processes. Then, the deviation of the actual power output from the modelled curve can be used as an indicator of abnormal operation or fault occurrence. For example, a reduced power output for a given wind speed can be caused by faulty controller settings, blade pitch errors, or generator faults. A cumulative residual can be calculated by comparing the actual and modelled power outputs over time, and a threshold can be set to trigger an alarm when the residual exceeds a certain value. This method can detect faults in wind turbines without requiring additional sensors or prior knowledge of the fault types (Yang et al., 2014).

2.2 Anomaly detection and machine learning for fault diagnosis and prediction

Another method for wind turbine condition monitoring using SCADA data is based on anomaly detection and machine learning techniques. Anomaly detection aims to identify patterns or events in the data that deviate from normal behavior or expectations. Machine learning aims to learn from data and build models that can perform tasks such as classification, regression, or clustering. By combining these techniques, it is possible to diagnose different types of faults in wind turbines based on their signatures or features in the SCADA data. For example, blade pitch faults can be diagnosed by using decision trees that are trained on SCADA parameters such as generator speed, power output, and pitch angle (Kusiak & Verma, 2011). Moreover, some machine learning methods can also predict faults in advance by using historical data and fault labels to learn the temporal patterns or trends that precede fault occurrence. For example, neural networks can be used to predict diverter malfunctions by using SCADA parameters such as active and reactive power, voltage, and current (Du et al., 2016; Kusiak & Li, 2011).

However, these methods also have some limitations and challenges. For example, they require a large amount of labelled data for training and testing purposes, which may not be available or reliable in practice. They also need to deal with the imbalance between normal and faulty instances in the data, which may affect their performance and generalization ability. Furthermore, they need to consider the trade-off between prediction accuracy and lead time, as well as the complexity and interpretability of the models.

2.3 Machine learning algorithms for fault diagnosis and prediction

Machine learning algorithms are powerful tools for wind turbine condition monitoring, as they can learn from data and build models that can perform tasks such as fault diagnosis and prediction. Fault diagnosis aims to identify the type and location of a fault in a wind turbine based on its symptoms or features in the SCADA data. Fault prediction aims to estimate the remaining useful life or the probability of failure of a wind turbine component based on its historical data and current condition. Machine learning algorithms can be classified into

supervised, unsupervised, and semi-supervised learning, depending on the availability and use of labelled data for training and testing purposes.

Supervised learning algorithms require labelled data, which means that each data instance has a corresponding class label that indicates the fault type or the health status of the wind turbine. Supervised learning algorithms can be further divided into classification and regression algorithms, depending on the nature of the output variable. Classification algorithms aim to assign a discrete class label to a data instance, such as normal or faulty, or blade pitch fault or generator fault. Regression algorithms aim to predict a continuous value for a data instance, such as the remaining useful life or the probability of failure of a wind turbine component. Some examples of supervised learning algorithms that have been applied for wind turbine fault diagnosis and prediction are artificial neural networks (Stetco et al., 2019; Du et al., 2016; Kusiak & Li, 2011), decision trees (Stetco et al., 2019; Kusiak & Verma, 2011), support vector machines (Stetco et al., 2019), random forests (Chen et al., 2022), and k-nearest neighbors (Chen et al., 2022).

Unsupervised learning algorithms do not require labelled data, which means that they can discover patterns or structures in the data without any prior knowledge or guidance. Unsupervised learning algorithms can be further divided into clustering and dimensionality reduction algorithms, depending on the goal of the analysis. Clustering algorithms aim to group similar data instances together based on some similarity or distance measure, such as k-means, hierarchical clustering, or density-based clustering (Stetco et al., 2019). Dimensionality reduction algorithms aim to reduce the number of features or dimensions in the data while preserving the most relevant information, such as principal component analysis, independent component analysis, or autoencoders (Stetco et al., 2019). Unsupervised learning algorithms can be useful for wind turbine fault diagnosis and prediction when labelled data is scarce or unreliable, or when new or unknown faults need to be detected.

Semi-supervised learning algorithms combine labelled and unlabelled data, which means that they can leverage both types of information to improve their performance and generalization ability. Semi-supervised learning algorithms can be further divided into self-training, co-training, and transfer learning algorithms, depending on the strategy used to exploit the unlabelled data. Self-training algorithms use their own predictions on unlabelled

data to augment their training set, such as self-organizing maps (Stetco et al., 2019). Co-training algorithms use two different views or feature sets of the data to train two classifiers separately and then exchange their predictions on unlabelled data to enhance their agreement, such as co-training support vector machines (Stetco et al., 2019). Transfer learning algorithms use knowledge learned from a source domain or task to help a target domain or task that has less or different data available, such as Inception V3 and TrAdaBoost (Chen et al., 2021). Semi-supervised learning algorithms can be beneficial for wind turbine fault diagnosis and prediction when labelled data is limited or imbalanced, or when data distributions change over time or across different wind turbines.

2.4 Summary

In this literature review, we have discussed two main methods for wind turbine condition monitoring using SCADA data: power curve analysis and anomaly detection/machine learning. Both methods have their advantages and disadvantages for fault detection, diagnosis, and prediction tasks. The choice of the most suitable method depends on various factors such as data availability and quality, fault types and characteristics, prediction goals and requirements, and computational resources and constraints.

This section also reviews the machine learning algorithms for wind turbine fault diagnosis and prediction using SCADA data. It categorizes the algorithms into supervised, unsupervised, and semi-supervised learning, and discusses their advantages and disadvantages. It also provides some examples of algorithms that have been applied for different types of faults and tasks.

Chapter 3

Supervised Machine Learning Algorithm for Fault Diagnosis and Prediction

Supervised machine learning algorithm is a type of algorithm that learns from labelled data, which means that each data instance has a corresponding class label that indicates the fault type or the health status of the wind turbine. Supervised machine learning algorithm can perform tasks such as fault diagnosis and prediction by building models that can assign a class label or predict a value for a new data instance based on its features. Supervised machine learning algorithm is useful for wind turbine fault detection and prediction because it can identify and classify different types of faults based on their signatures or features in the SCADA data. It can also estimate the remaining useful life or the probability of failure of a wind turbine component based on its historical data and current condition. Supervised machine learning algorithm can help to improve the efficiency and reliability of wind turbines, as well as to reduce the maintenance costs and downtime.

Following are the different types of supervised machine learning algorithms that we have applied for our study, along with their mathematical formulations, advantages and disadvantages.

3.1 Support Vector Machines (SVM)

A support vector machine constructs a hyper-plane or set of hyper-planes in a high or infinite dimensional space, which can be used for classification, regression or other tasks. Intuitively, a good separation is achieved by the hyper-plane that has the largest distance to the nearest training data points of any class (so-called functional margin), since in general the larger the margin the lower the generalization error of the classifier.

The following figure shows the decision function for a linearly separable problem, with two samples on the margin boundaries, called “support vectors”.

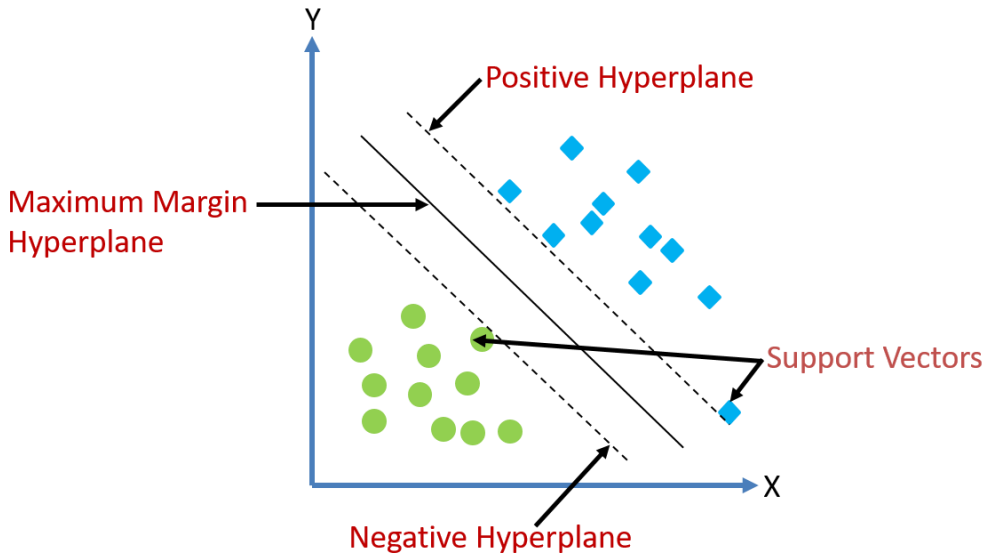


Figure 3: SVM algorithm

3.1.1 Mathematical Formulation

In general, when the problem isn't linearly separable, the support vectors are the samples within the margin boundaries.

Given training vectors $x_i \in \mathbb{R}^p$, i = 1, … , n, in two classes, and a vector $y \in \{1, -1\}^n$, our goal is to find $w \in \mathbb{R}^p$ and $b \in \mathbb{R}^{\text{such}}$ that the prediction given by sign $(w^T \phi(x) + b)$ is correct for most samples.

SVC solves the following primal problem:

$$\min_{w,b,\zeta} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \zeta_i \qquad (1)$$

$$subject\ to\ y_i(w^T \phi(x_i) + b) \geq 1 - \zeta_i$$

$$\zeta_i \geq 0, i = 1, \dots, n$$

Intuitively, we're trying to maximize the margin (by minimizing $\| w \|^2 = w^T w$), while incurring a penalty when a sample is misclassified or within the margin boundary. Ideally, the value $y_i(w^T \phi(x_i) + b$) would be ≥ 1 for all samples, which indicates a perfect prediction. But problems are usually not always perfectly separable with a hyperplane, so we allow some samples to be at a distance ζ_i from their correct margin boundary. The penalty term c controls

the strength of this penalty, and as a result, acts as an inverse regularization parameter (see note below).

The dual problem to the primal is

$$\min_{\alpha} \frac{1}{2}\alpha^T Q\alpha - e^T\alpha \qquad (2)$$

$$subject\ to\ \ y^T\alpha = 0$$

$$\zeta_i \geq 0, i = 1, \dots, n$$

where e is the vector of all ones, and Q is an n by n positive semidefinite matrix, $Q_{ij} \equiv y_i y_j K(x_i, x_j)$ where $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$ is the kernel. The terms α_i are called the dual coefficients, and they are upper-bounded by C. This dual representation highlights the fact that training vectors are implicitly mapped into a higher (maybe infinite) dimensional space by the function ϕ. Thus, in addition to performing linear classification, SVCs can efficiently perform non-linear classification by implicitly mapping their inputs into high-dimensional feature spaces (which is called the kernel trick).

Once the optimization problem is solved, the output of decision function for a given sample $\boldsymbol{x}$ becomes

$$\sum_{i \in SV} y_i \alpha_i K(x_i, x) + b \qquad (3)$$

and the predicted class correspond to its sign. We only need to sum over the support vectors (i.e. the samples that lie within the margin) because the dual coefficients α_i are zero for the other samples.

3.1.2 Advantages and Disadvantages

Some of the advantages & disadvantages of Support Vector Machines are (Zhang et al., 2020):

Advantages:

- **Strong theoretical foundation:** It has a strong theoretical foundation based on statistical learning theory and structural risk minimization

- **Can use different kernel functions:** It can use different kernel functions to adapt to different types of data and problems.
- **Good generalization ability:** It has good generalization ability and can avoid overfitting by using regularization and margin maximization.

Disadvantages:

- **Computationally expensive for large datasets:** It is computationally expensive and time-consuming for large datasets, as the algorithm requires solving a quadratic optimization problem
- **Does not provide probabilistic outputs or confidence estimates:** It does not provide probabilistic outputs or confidence estimates for the predictions, only binary labels
- **Does not guarantee finding the global optimum:** It does not guarantee finding the global optimum, as it may depend on the initial conditions or the choice of kernel function

3.2 K-Nearest Neighbours (KNN)

K-Nearest Neighbours is one of the most basic yet essential classification algorithms in Machine Learning. It belongs to the supervised learning domain and finds intense application in pattern recognition, data mining, and intrusion detection.

It is widely disposable in real-life scenarios since it is non-parametric, meaning, it does not make any underlying assumptions about the distribution of data (as opposed to other algorithms such as GMM, which assume a Gaussian distribution of the given data). We are given some prior data (also called training data), which classifies coordinates into groups identified by an attribute.

The working principle of KNN involves assigning a new data point to a class based on its proximity to the existing data points in the training set. The algorithm uses the labeled training data to identify groups or classes defined by a specific attribute. When a new data point is encountered, KNN determines its class by finding the K nearest neighbors from the training set and assigning the majority class among those neighbors to the new data point.

As an example, consider the following data points containing two features.

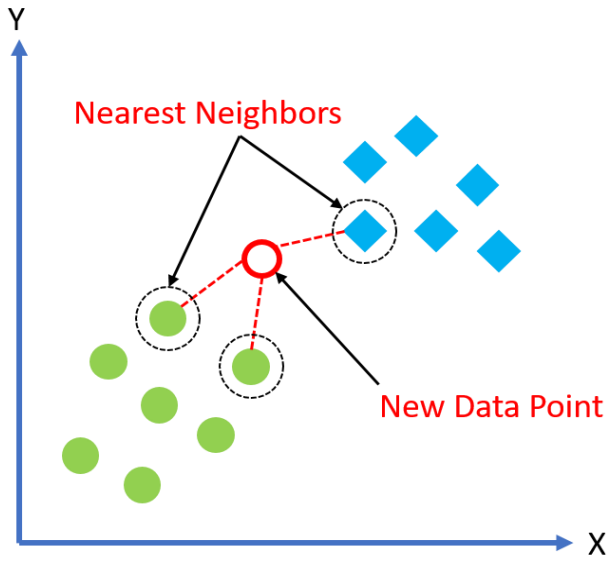


Figure 4: KNN algorithm

Now, given another set of data point (also called testing data), allocate these points to a group by analyzing the training set. Note that the unclassified points are marked as 'White'.

If we plot these points on a graph, we may be able to locate some clusters or groups. Now, given an unclassified point, we can assign it to a group by observing what group its nearest neighbors belong to. This means a point close to a cluster of points classified as 'Green' has a higher probability of getting classified as 'Green'.

3.2.1 Mathematical Formulation

In the classification problem, the K-nearest neighbor algorithm essentially said that for a given value of K algorithm will find the K nearest neighbor of unseen data point and then it will assign the class to unseen data point by having the class which has the highest number of data points out of all classes of K neighbors.

The nearest points or groups for a given query point can be evaluated based on some distance metric. The distance metric measures the similarity or dissimilarity between two points in a feature space. Different distance metrics can be used depending on the problem and the data. Some of the commonly used distance metrics are (Santos et al., 2020):

- **Euclidean Distance**: This is the square root of the sum of the squared differences between the coordinates of the points in n-dimensions. It corresponds to the length of the shortest line segment connecting the two points. This metric is suitable for measuring the displacement between two states of an object.

$$d(x,y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \quad (4)$$

- **Manhattan Distance** (**Gokte, 2020)**: This is the sum of the absolute differences between the coordinates of the points in n-dimensions. It corresponds to the total distance traveled along the axes when moving from one point to another. This metric is suitable for measuring the total distance traveled by an object.

$$d(x,y) = \sum_{i=1}^{n} |x_i - y_i| \quad (5)$$

- **Minkowski Distance:** This is a generalization of the Euclidean and Manhattan distances that allows different weights for different dimensions. It is defined by a power parameter p that determines the shape of the distance function. When p = 2, it reduces to the Euclidean distance, and when p = 1, it reduces to the Manhattan distance.

$$d(x,y) = \left(\sum_{i=1}^{n} (x_i - y_i)^p\right)^{\frac{1}{p}} \quad (6)$$

There are other distance metrics that can be used for specific problems and data types, such as Hamming Distance, which is useful for comparing binary or categorical vectors. The choice of the distance metric can affect the performance and accuracy of the KNN algorithm.

3.2.2 Advantages and disadvantages

The KNN algorithm has some advantages and disadvantages that need to be considered before applying it to a problem. Some of them are:

Advantages

- **Easy to implement**: The algorithm is relatively simple and does not require complex computations or assumptions.
- **Adapts easily:** The algorithm can learn from new data points as they are added to the training set and update its predictions accordingly.
- **Few hyperparameters:** The algorithm only needs two parameters to be tuned: the value of k and the distance metric.

Disadvantages

- **Does not scale:** The algorithm is computationally expensive and memory intensive, as it requires storing all the training data and calculating distances for each query point. This makes the algorithm slow and inefficient for large datasets.
- **Curse of dimensionality:** The algorithm suffers from the peaking phenomenon, which means that the performance of the algorithm decreases as the number of dimensions increases. This is because the distance between points becomes less meaningful and more noisy in high-dimensional spaces.
- **Prone to overfitting:** The algorithm can overfit the data if the value of k is too small or the distance metric is not appropriate for the problem. This can lead to poor generalization and high variance. Therefore, feature selection and dimensionality reduction techniques are often applied to reduce the risk of overfitting.

3.3 Random Forest

A Random Forest Algorithm is a supervised machine learning algorithm that is extremely popular and is used for Classification and Regression problems in Machine Learning. A forest comprises numerous trees, and the more trees more it will be robust. Similarly, the greater the number of trees in a Random Forest Algorithm, the higher its accuracy and problem-solving ability. Random Forest is a classifier that contains several decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset. It is based on the concept of ensemble learning which is a process of combining multiple classifiers to solve a complex problem and improve the performance of the model.

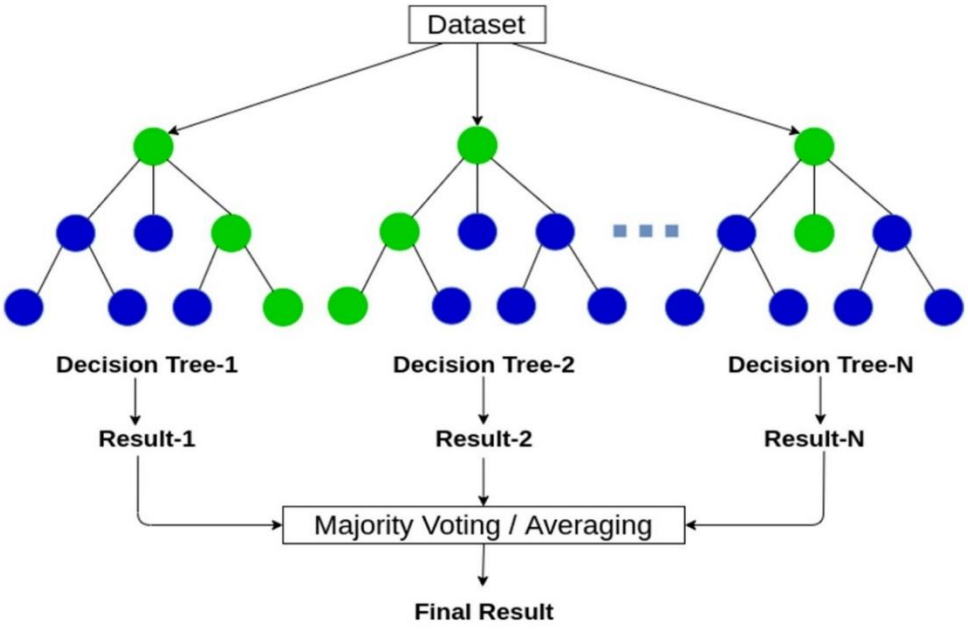


Figure 5: Random Forest Algorithm

The following steps explain the working Random Forest Algorithm:

Step 1: Select random samples from a given data or training set.

Step 2: This algorithm will construct a decision tree for every training data.

Step 3: Voting will take place by averaging the decision tree.

Step 4: Finally, select the most voted prediction result as the final prediction result.

3.3.1 Mathematical Formulation

For each decision tree, Scikit-learn calculates a node's importance using Gini Importance, assuming only two child nodes (binary tree):

$$n_j = w_j C_j - w_{left(j)} C_{left(j)} - w_{right(j)} C_{right(j)} \quad (7)$$

n_j = importance of node j

w_j = weighted number of samples reaching node j

C_j = impurity value of node j

$left(j)$ = child node from left split on node j

right(j) = child node from right split on node j

The importance for each feature on a decision tree is then calculated as:

$$f_i = \frac{\sum_{j:\ node\ j\ splits\ on\ feature\ i} n_j}{\sum_{k \in\ all\ nodes} n_k} \quad (8)$$

f_i = importance of feature i

n_j = importance of node j

These can then be normalized to a value between 0 and 1 by dividing by the sum of all feature importance values:

$$f_i(normalised) = \frac{f_i}{\sum_{j \in\ all\ features} f_j} \quad (9)$$

The final feature importance, at the Random Forest level, is its average over all the trees. The sum of the feature's importance value on each tree is calculated and divided by the total number of trees:

$$f_i(random\ forest) = \frac{\sum_{j \in\ all\ trees} f_{ij}(normalised)}{T} \quad (10)$$

f_i(random forest) = importance of feature i calculated from all trees in the Random Forest model

f_{ij}(normalised) = normalized feature importance for i in tree j

T = total number of trees

3.3.2 Advantages and disadvantages

Some of the advantages and disadvantages of Random Forest are (Schonlau and Zou, 2020):

Advantages

- **Handle high dimensional data**: It can handle high dimensional data and does not require feature selection or dimensionality reduction
- **Measure feature importance:** It can measure the feature importance and the interaction between different features

- **Robust to noisy data:** It is robust to noisy data and outliers and less likely to overfit the data

Disadvantages

- **Not easily interpretable:** It is not easily interpretable as it consists of many decision trees and does not provide complete visibility into the coefficients
- **Computationally and memory intensive:** It is computationally intensive and memory intensive as it requires storing all the training data and building multiple decision trees
- **Black box algorithm:** It is a black box algorithm that has very little control over what the model does

3.4 Extreme Gradient Boosting (XGBoost)

XGBoost is an optimized distributed gradient boosting library designed for efficient and scalable training of machine learning models. It is an ensemble learning method that combines the predictions of multiple weak models to produce a stronger prediction. XGBoost stands for "Extreme Gradient Boosting" and it has become one of the most popular and widely used machine learning algorithms due to its ability to handle large datasets and its ability to achieve state-of-the-art performance in many machine learning tasks such as classification and regression.

Before understanding the XGBoost, we first need to understand the trees, especially the decision tree. Decision trees, in their basic form, are easy to visualize and understand. However, comprehending the more advanced tree-based algorithms can be a bit difficult. Here is a simple analogy to help better understand how these algorithms have evolved.

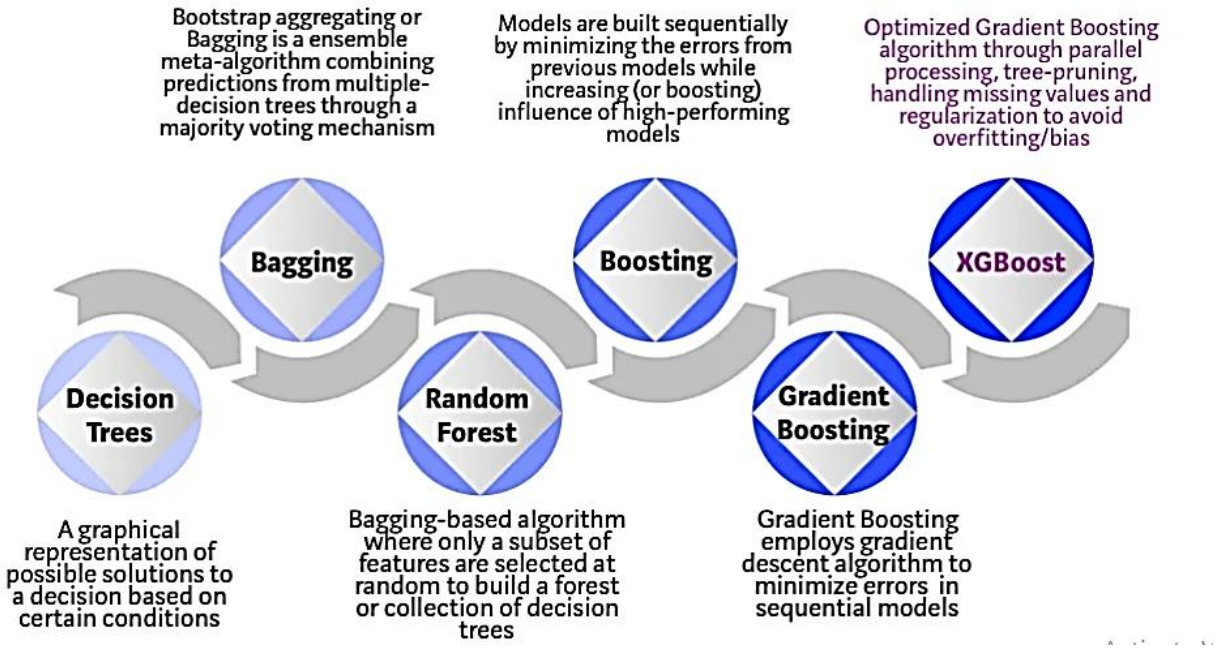


Figure 6: Evolution of XGBoost Algorithm from Decision Trees (Krishnan et al., 2021)

- **Decision Trees:** In the context of machine learning, a decision tree is a supervised learning algorithm that can be used for both classification and regression problems. Decision trees are made up of a series of nodes, each of which represents a decision or split in the data. The leaves of the tree represent the different classes or outcomes that the model can predict.

  A hiring manager can be thought of as a decision tree. The hiring manager has a set of criteria that they use to decide about which candidate to hire. These criteria could include education level, number of years of experience, interview performance, etc. The hiring manager would start by asking the candidate questions about their education level. If the candidate meets the hiring manager's criteria for education level, the hiring manager would then ask the candidate questions about their number of years of experience. This process would continue until the hiring manager had enough information to decide about which candidate to hire.

- **Bagging:** Bagging, or bootstrap aggregating, is an ensemble learning technique that can be used to improve the performance of decision trees. Bagging works by creating multiple decision trees from bootstrap samples of the training data. A bootstrap sample is a random sample of the training data, with replacement. This means that some data points may be

included in multiple bootstrap samples. Bagging helps to reduce the variance of the decision trees, which can improve their accuracy.

In the context of the hiring analogy, bagging can be thought of as having multiple hiring managers interview the same candidates. Each hiring manager would have their own set of criteria, and each hiring manager would make their own decision about which candidate to hire. The final decision about which candidate to hire would be made by a majority vote of the hiring managers.

- **Random Forest:** Random forest is a type of bagging ensemble algorithm that uses decision trees as its base learners. Random forest works by creating multiple decision trees from bootstrap samples of the training data, where each tree is trained on a random subset of the features. This helps to reduce the variance of the decision trees and improve their accuracy.

  In the context of the hiring analogy, random forest can be thought of as having multiple hiring managers interview the same candidates, but each hiring manager is only allowed to ask questions about a subset of the candidates' qualifications. This helps to ensure that all of the candidates are evaluated fairly, and it helps to reduce the chances that any one hiring manager will make a biased decision.

- **Boosting:** Boosting is another ensemble learning technique that can be used to improve the performance of decision trees. Boosting works by creating multiple decision trees sequentially, where each tree is trained to correct the errors of the previous trees.

  In the context of the hiring analogy, boosting can be thought of as having a series of hiring managers interview the same candidates. Each hiring manager would be given feedback from the previous hiring managers, and they would use this feedback to make their own decision about which candidate to hire. This helps to ensure that the final decision about which candidate to hire is as accurate as possible.

- **Gradient Boosting:** Gradient boosting is a type of boosting algorithm that uses gradient descent to minimize the error of the model. Gradient descent is an iterative optimization algorithm that works by repeatedly adjusting the model's parameters in order to minimize a loss function.

In the context of the hiring analogy, gradient boosting can be thought of as a hiring manager who uses a feedback loop to improve their decision-making process. The hiring manager would start by deciding about which candidate to hire. They would then get feedback from other hiring managers about their decision. They would then use this feedback to make a better decision about which candidate to hire. This process would continue until the hiring manager was confident that they were making the best possible decision.

- **XGBoost:** XGBoost is a type of gradient boosting algorithm that has been shown to be very effective for a variety of machine learning problems. XGBoost uses a number of techniques to improve the performance of gradient boosting algorithms, such as regularization, tree pruning, and parallelization.

  In the context of the hiring analogy, XGBoost can be thought of as a hiring manager who is using all of the latest technology to improve their decision-making process. The hiring manager would be using a computer to help them to collect data about the candidates, to analyze the data, and to decide about which candidate to hire. The hiring manager would also be using a computer to help them to get feedback from other hiring managers, and to use this feedback to improve their decision-making process.

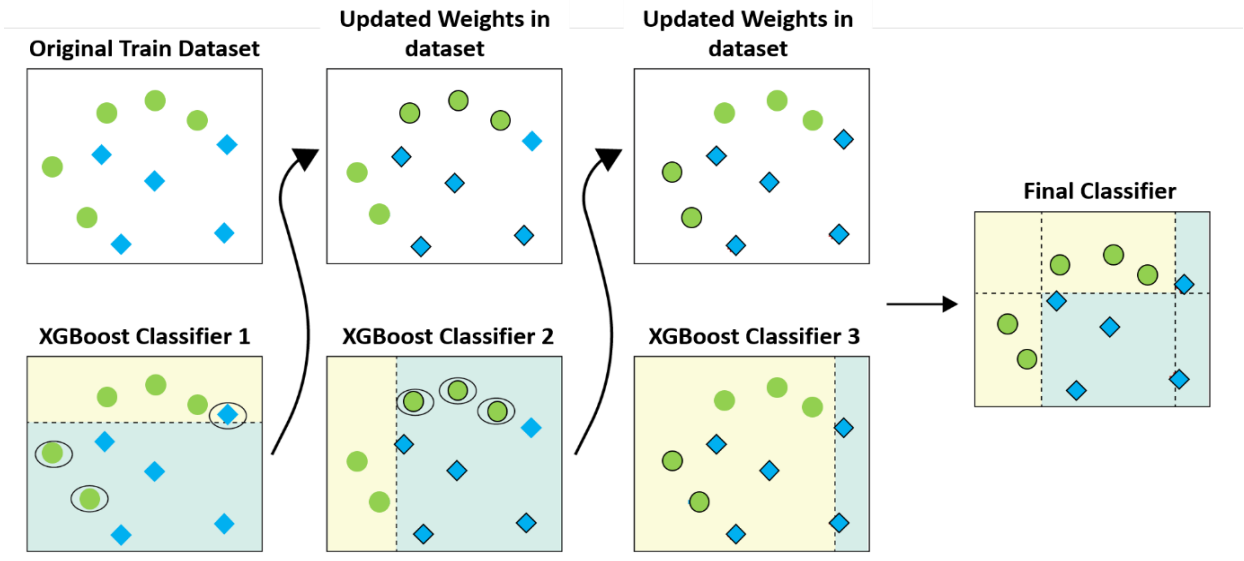


Figure 7: XGBoost algorithm

3.4.1 Mathematical Formulation

XGBoost employs several predictors that combine to provide the solution. Instead of using the bagging technique, it uses boosting. The predictors are created sequentially, rather than independently, in boosting. It makes it simple to obtain the precise ideal number of boosting iterations in a single run because it permits cross-validation at every iteration of the boosting process. We can define the predicted target $\hat{y}$ if we have m regression trees, where each tree improves the previous one, as shown in Eqn. (11)

$$\hat{y}_i = \sum_{m=1}^{M} f_m(x_i), f_m \epsilon \boldsymbol{F} \tag{11}$$

The objective function described in Eq. (3) must be minimized. Loss and regularization are components of this function where l denotes the explanation of the loss function, and Ω is the regularization function.

$$obj(\theta) = \sum_{i}^{n} l(y_i, \hat{y}_i) + \sum_{m=1}^{M} \Omega(f_m) \tag{12}$$

3.4.2 Advantages and disadvantages

Some of the advantages and disadvantages of XGBoost are (Zhang et al., 2022):

Advantages:

- **Can be tuned to improve performance and avoid overfitting:** XGBoost has a number of hyper-parameters that can be tuned to improve the performance of the model and avoid overfitting. This can be done by trial and error or by using a grid search algorithm.
- **Can handle missing values:** XGBoost has an in-built capability to handle missing values. This means that it does not require imputation, which can be a time-consuming and error-prone process.
- **Faster and more robust than other gradient boosting methods:** XGBoost provides various features such as parallelization, distributed computing, cache optimization, and regularization that make it faster and more robust than other gradient boosting methods.

Disadvantages:

- **Requires manual encoding of categorical features:** XGBoost does not support categorical features directly. This means that categorical features must be manually encoded before they can be used in the model. There are a number of different ways to encode categorical features, and the best approach will depend on the specific data set.
- **Not easily interpretable:** XGBoost is not easily interpretable. This means that it can be difficult to understand how the model makes predictions. This can make it difficult to use XGBoost for problems where it is important to understand the model.
- **May consume more memory and computational resources:** XGBoost may consume more memory and computational resources than other algorithms. This is because it builds a large number of trees, and each tree requires some memory and computational resources. If the data set is large or the number of trees is large, this can be a problem.

3.5 Bayesian Neural Network (BNN)

Bayesian neural networks (BNNs) are a type of neural network that uses probabilistic modeling to account for uncertainty in the data. This makes BNNs more robust to overfitting than traditional neural networks, and they can also be used to generate probabilistic predictions.

BNNs are made up of two parts: a neural network and a probabilistic model. The neural network is responsible for learning the underlying relationship between the input and output data. The probabilistic model is responsible for representing the uncertainty in the data. The neural network is trained using supervised learning. The input data is used to train the network to predict the output data. The probabilistic model is trained using Bayesian inference. The prior distribution of the parameters is used to generate a posterior distribution of the parameters, given the input data. Once the BNN is trained, it can be used to make probabilistic predictions. The probabilistic prediction is a distribution over the possible outputs, given a new input. This distribution can be used to quantify the uncertainty in the prediction.

BNNs are a powerful tool for machine learning. They are more robust to overfitting than traditional neural networks, and they can also be used to generate probabilistic predictions. This makes them a valuable tool for a variety of applications, such as classification and regression.

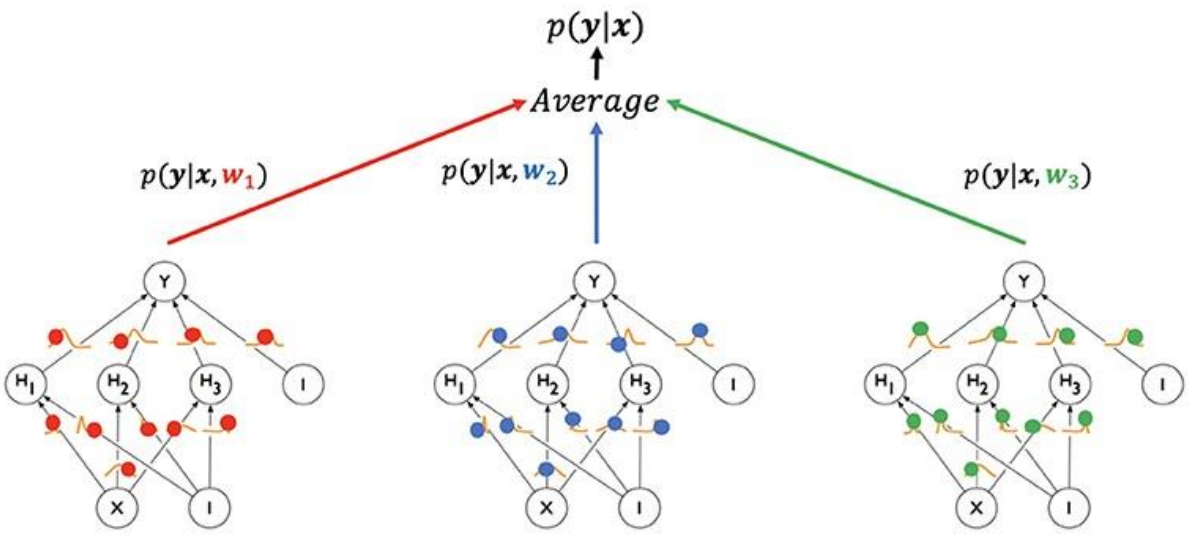


Figure 8: BNN algorithm

3.5.1 Mathematical Formulation

Learning algorithms of neural networks use a set of training data to iteratively update the parameters of a neural network such that some error measure is decreased or some performance measure is increased. The data $\mathcal{D}$ consists of vectors $\mathcal{D}_i = \{\boldsymbol{y}_i, \boldsymbol{x}_i\}$, with $\boldsymbol{x}_i$ representing an input and $\boldsymbol{y}_i$ the corresponding target for $i = 1, \dots, N$. Let $\hat{\boldsymbol{y}}_i$ denote the output of the network corresponding to the sample $\boldsymbol{x}_i$, that is

$$\hat{\boldsymbol{y}}_i = NN_{\boldsymbol{\theta}}(\boldsymbol{x}_i) \qquad (13)$$

with $\mathrm{NN}_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$ denoting a Neural Network parametrized over $\boldsymbol{\theta}$ and evaluated at $\boldsymbol{x}_i$. An error function $E(\mathcal{D}, \boldsymbol{\theta})$ is defined at a particular parameter $\boldsymbol{\theta}$, which is used to guide the learning process.

The goal of approximating a function relating the input to the output in classical ANNs is treated under an entirely deterministic perspective. Switching towards a Bayesian perspective in mathematical terms is rather straightforward. In place of estimating the parameter vector $\boldsymbol{\theta}$, BNNs target the estimation of the posterior distribution $p(\boldsymbol{\theta} \mid \mathcal{D}_x, \mathcal{D}_y)$, that is (Jospin et al. 2022):

$$p(\boldsymbol{\theta} \mid \mathcal{D}_x, \mathcal{D}_y) = \frac{p(\mathcal{D}_y \mid \mathcal{D}_x, \boldsymbol{\theta}) p(\boldsymbol{\theta})}{\int_{\Theta} p(\mathcal{D}_y \mid \mathcal{D}_x, \boldsymbol{\theta}') p(\boldsymbol{\theta}') d\boldsymbol{\theta}'} \quad (14)$$

which stands as a simple application of the Bayes theorem. Here we assume, as it is usually the case, that the data $\mathcal{D}$ is composed of an input set $\mathcal{D}_x$ and the corresponding set of outputs $\mathcal{D}_y$. In general, $\mathcal{D}_x$ is a matrix of regressors, and $\mathcal{D}_y$ is either the vector or matrix (depending on whether the nature of the output is univariate or not) of the variables that the networks aim at modeling based on $\mathcal{D}_x$. Alternatively, but analogously, $\mathcal{D}$ can be thought as the collection of all input-output pairs $\mathcal{D} = \{\boldsymbol{y}_i, \boldsymbol{x}_i\}_{i=1}^{N}$, where N denotes the sample size, and $\boldsymbol{x}_i$ and $\boldsymbol{y}_i$ are the input and output vectors of observations for the ith sample, respectively. Using this notation, $\mathcal{D}_x = \{\boldsymbol{x}_i\}_{i=1}^{N}$ and $\mathcal{D}_y = \{\boldsymbol{y}_i\}_{i=1}^{N}$.

While Eq. (14) provides a theoretical prescription for obtaining the posterior distribution, in practice solving for the form of the posterior distribution and retrieving its parameters is a very challenging task.

3.5.2 Advantages and disadvantages

Some of the advantages and disadvantages of Bayesian Neural Network are (Kevin P. Murphy, 2012):

Advantages:

- **Probabilistic predictions:** Bayesian neural networks (BNNs) can make probabilistic predictions, which means that they can provide a measure of uncertainty in their predictions. This can be useful for tasks such as risk assessment and decision-making.
- **Robust to noisy data:** BNNs are robust to noisy data, which means that they can still perform well even when the data contains errors or outliers. This is because BNNs use Bayesian inference to learn the probability distribution of the output, rather than simply fitting a line or curve to the data.
- **Interpretability:** BNNs can be more interpretable than other machine learning algorithms, such as deep neural networks. This is because BNNs can provide information about the importance of different features in the model, as well as the uncertainty in the model's predictions.

Disadvantages:

- **Computationally expensive:** BNNs can be computationally expensive to train, especially for large datasets. This is because BNNs require the computation of the posterior distribution of the model parameters, which can be a time-consuming process.
- **Require prior knowledge:** BNNs require prior knowledge about the distribution of the data. This prior knowledge can be obtained from expert knowledge or from previous data. If the prior knowledge is incorrect, it can lead to inaccurate predictions.
- **Not always better than other models:** BNNs are not always better than other machine learning models, such as deep neural networks, especially for tasks that require a large number of parameters to be learned.

Chapter 4
Methodology

We attempt to classify faults at three levels; namely fault detection, fault diagnosis and fault prediction. A general methodology for all three types of classification is shown in Figure 9. As can be seen, there are four main steps following a general machine learning process, described in detail in this section.

4.1 Data Labelling

Following are the processes for labelling the data for each classification level.

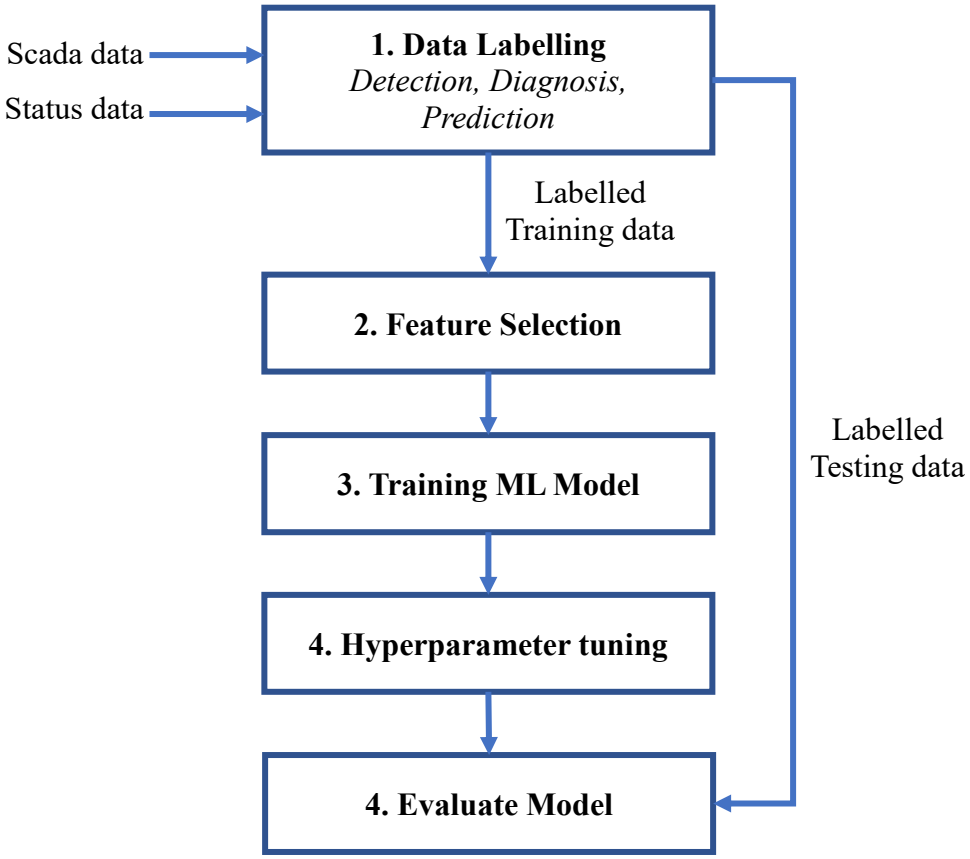


Figure 9: Methodology, following a typical machine learning approach

4.1.1 Fault Detection

The first level of classification is distinguishing between two classes i.e. “fault” and “no-fault”. The fault data corresponds to times of operation under a set of specific faults

mentioned in Section 3.1. For these faults, status messages with codes corresponding to the faults were selected. Next, a time band of 600s before the start, and after the end, of these turbine states was used to match up the associated 10-minute operational data. The 10 minutes time-band was selected so as to definitely capture any 10-minute period where a fault occurred, e.g., if a power feeding fault occurred from 11:49-13:52, this would ensure the 11:40-11:50 and 13:50-14:00 operational data points were labelled as faults. No matter the type of fault, all faults were simply labelled as the "fault" class. All the remaining data in the operational dataset was then given the label "other", representing all other data. This is because the remaining data didn't necessarily represent fault-free data; it just meant it didn't contain the faults mentioned in Section 2.3, but could have included other, less frequent faults or times when the turbine power output was being curtailed for any one of a number of reasons mentioned previously.

4.1.2 Fault Diagnosis

Fault diagnosis represents a more advanced level of classification than simply fault detection. The aim of fault diagnosis is to identify specific faults from the rest of the data. Faults were labelled in the same way as in the previous section, but this time, each fault was given its own specific label. Again, a time band of 600s before the start and after the end of each fault status was used to match up corresponding 10-minute operational data. Any data that remained, i.e., was not labelled as one of the faults mentioned in Section 2.3, was again given the "no fault" label, for a total of five different classes (the four fault classes as well as the "no fault" class). If two faults occurred concurrently, then the data point was duplicated, with each point having a different label.

4.1.3 Fault Prediction

Fault prediction represents an even more advanced level of classification than fault diagnosis. The aim of this level of classification was to see if it was possible to identify that a specific fault was imminent from the full set of operational data. After initial tests, it was decided to focus prediction only on generator heating and excitation faults as these showed the greatest promise for early detection. Details of this can be found in Section 3.1. The other faults were included in the "other" label along with the rest of the data, for a total of three classes.

For fault prediction, the times during which the turbine was in faulty operation were not labelled as such. Instead, operational data points leading up to each fault were labelled as "pre-fault", for each specific fault. A number of separate cases representing different values of T and W were tried to see how far in advance an accurate prediction could be made. These can be seen in Table 4. For example, for case F, all data points with timestamps between 24 and 12 hours before a generator heating fault occurred were labelled as "generator heating fault". The same was applied to excitation faults. All remaining unlabelled data points were labelled "other". Once again, if different faults occurred concurrently, the data points were duplicated and given different labels.

This would mean that, for case F, for example, if the trained classifier detected a new, live, data point as "generator fault", it would mean that a generator heating fault is likely to occur in the next 12 hours. The technician or maintenance operator would then have between 12 and 24 hours between this point being detected and the fault actually occurring to remotely or manually inspect the generator and organise any necessary maintenance actions ahead of time. If maintenance is needed, this can reduce the logistics lead time or allow it to be scheduled in conjunction with other maintenance activities for maximum economic benefit.

4.2 Feature Selection

The full operational dataset had more than sixty features, many of which were redundant, incorrect or irrelevant. Because of this, only a subset of specific features was chosen to be included for training purposes. It was found that a number of the original features corresponded to sensors on the turbine which were broken, e.g., they had frozen or blatantly incorrect values, while others contained duplicate or redundant values. These were removed. A number of the remaining features which were deemed as obviously irrelevant based on some basic domain knowledge were also excluded, e.g. features relating to the cumulative uptime of the anemometer or the open/closed state of the tower base door. This resulted in 39 remaining features. A subset of *these*, corresponding to 12 temperature sensors on the inverter cabinets in the turbine, all had very similar readings. Because of this, it was decided to instead consolidate these and use the average and standard deviation of the 12 inverter temperatures. This resulted in 29 features being used to train the models. It was decided to scale all features

individually to unit norm because some, e.g., power output, had massive ranges from zero to thousands, whereas others, e.g., temperature, ranged from zero to only a few tens.

4.3 Model Selection

The model parameters were tuned via Random Search, and split in 5 groups using time-series cross validation. The class weight approach is utilized to address the heavily unbalanced dataset.

4.3.1 Support Vector Machines (SVM)

This model uses a hyperplane to separate data into different classes. A hyperplane is a line or plane that separates the data points into different classes. The goal of the SVM algorithm is to find the hyperplane that maximizes the margin between the two classes. The margin is defined as the distance between the hyperplane and the closest data points from each class, which are called support vectors.

The main parameters of the SVM model are:

- **Kernel type:** The kernel function is used to transform the data into a higher dimensional space where it is easier to separate the data using a hyperplane. There are several types of kernel functions, including linear, polynomial, and radial basis function (RBF). The choice of kernel function depends on the nature of the data and can significantly affect the performance of the model.
- **Regularization parameter (C):** This parameter controls the trade-off between maximizing the margin and minimizing classification errors. A small value of C creates a large margin but may allow some misclassifications, while a large value of C creates a small margin but tries to minimize misclassifications.
- **Kernel coefficient gamma (γ):** This parameter is specific to the RBF kernel and controls the shape of the decision boundary. A small value of gamma creates a flexible decision boundary, while a large value of gamma creates a rigid decision boundary.During cross-validation, the train set was divided into train set and validation set. Therefore, to ensure that the train set is balanced, the class weights are put inside via pipeline.

4.3.2 K-Nearest Neighbours (KNN)

This model classifies data based on the majority vote of its k nearest neighbors. For a given data point, the algorithm finds the k data points in the training set that are closest to it, and then assigns the class label based on the majority vote of these k neighbors.

The main parameter of the KNN model is:

- **Number of neighbors k:** This parameter determines how many neighbors are used to classify a new data point. A small value of k results in a flexible decision boundary, while a large value of k results in a more rigid decision boundary. The choice of k depends on the nature of the data and can significantly affect the performance of the model.
- **Distance metric** used to determine the nearest neighbors can also affect the performance of the model. Common distance metrics include Euclidean distance and Manhattan distance

4.3.3 Random Forest

This model is an ensemble of decision trees that uses bagging and feature randomness to improve accuracy. Bagging is a technique where multiple models are trained on different subsets of the training data, and their predictions are combined to make the final prediction. Feature randomness refers to the process of randomly selecting a subset of features to consider when splitting a node in a decision tree.

The main parameters of the Random Forest model are:

- **Number of trees in the forest:** This parameter determines how many decision trees are used in the ensemble. A larger number of trees can improve the accuracy of the model, but also increases the computational cost.
- **Maximum depth of each tree:** This parameter controls the maximum depth of each decision tree in the forest. A deeper tree can capture more complex relationships between the features and the target variable, but may also lead to overfitting.
- **Minimum number of samples required to split an internal node:** This parameter determines the minimum number of samples required to split an internal node in a decision tree. A larger value can help prevent overfitting by reducing the complexity of the tree.

4.3.4 Extreme Gradient Boosting (XGBoost)

This model is an ensemble of decision trees that uses gradient boosting to improve accuracy. Gradient boosting is a technique where multiple models are trained sequentially, with each model trying to correct the errors made by the previous model. The final prediction is made by combining the predictions of all the models.

The main parameters of the XGBoost model are:

- **Learning rate:** This parameter controls the contribution of each tree to the final prediction. A smaller learning rate means that each tree has a smaller impact on the final prediction, which can help prevent overfitting.
- **Maximum depth of each tree:** This parameter controls the maximum depth of each decision tree in the ensemble. A deeper tree can capture more complex relationships between the features and the target variable, but may also lead to overfitting.
- **Minimum child weight:** This parameter determines the minimum sum of instance weight needed in a child node. A larger value can help prevent overfitting by reducing the complexity of the tree.

4.3.5 Bayesian Neural Network (BNN)

This model is a neural network that uses Bayesian inference to estimate the posterior distribution of its weights. Bayesian inference is a statistical technique that allows us to update our beliefs about unknown parameters (in this case, the weights of the neural network) based on observed data. The posterior distribution of the weights represents our updated beliefs about their values after observing the data.

The main parameters of the Bayesian Neural Network model are:

- **Number of hidden layers and nodes:** These parameters determine the architecture of the neural network. The number of hidden layers and nodes in each layer can affect the ability of the model to capture complex relationships between the features and the target variable.
- **Prior distribution of the weights:** This parameter represents our initial beliefs about the values of the weights before observing any data. The choice of prior distribution can affect the performance of the model.

- **Activation function:** This parameter determines the function used to compute the output of each node in the neural network. Common activation functions include sigmoid, tanh, and ReLU.
- **Kullback-Leibler (KL) divergence loss**: This parameter controls the trade-off between the reconstruction loss and the KL divergence loss in a variational autoencoder (VAE). The KL divergence loss measures how close the posterior distribution of the latent variables is to the prior distribution. A higher KL weight means more emphasis on matching the posterior and prior distributions, while a lower KL weight means more emphasis on reconstructing the input data.
- **Learning rate**: This parameter controls how much the weights are updated in each iteration of gradient descent. Gradient descent is an optimization algorithm that finds the optimal values of the weights by iteratively moving in the direction of steepest descent. A higher learning rate means faster convergence but may cause overshooting or instability, while a lower learning rate means slower convergence but may cause getting stuck in local minima or underfitting.
- **Number of iterations:** For a gradient descent algorithm, it determines how many times the model parameters are updated based on the input data and the cost function. A higher number of steps means more training but may also cause overfitting or slow convergence. A lower number of steps means less training but may also cause underfitting or poor performance.

4.4 Model Evaluation

A number of scoring metrics were used to evaluate final performance on the test sets for fault detection and fault diagnosis, as well as the six test sets representing the time windows A-E for fault prediction. A high number of false positives can lead to unnecessary checks or corrections carried out on the turbine, and this was captured with the precision score (where a higher score represents a lower false positive rate). On the other hand, a high number of false negatives can lead to failure of the component with no detection having taken place (Saxena et al., 2008). This is captured by the recall score, where a higher number indicates a low ratio of false negatives. The F1-Score was also used, which is the harmonic mean of precision and

recall. Confusion matrices were used where appropriate to give a visual overview of performance and show absolute numbers.

The formulae for calculating precision, recall and the F1-score can be seen below

$$Recall = \frac{tp}{tp + fn} \tag{15}$$

$$Precision = \frac{tp}{tp + fp} \tag{16}$$

$$F1 = \frac{2tp}{2tp + fp + fn} \tag{17}$$

$$Specificity = \frac{tn}{fp} + tn \tag{18}$$

where *tp* is the number of true positives, i.e., correctly predicted fault samples, *fp* is false positives, *fn* is false negatives, i.e., fault samples incorrectly labelled as no-fault, and *tn* is true negatives.

The overall accuracy of the classifier on each test set was not used as a metric due to the massive imbalance seen in the data. For example, if 4990 samples were correctly labelled as fault-free, and the only 20 fault samples were also incorrectly labelled as such, the overall accuracy of the classifier would still stand at 99.6%. Specificity was not used as a metric for a similar reason, though was used in one specific case for benchmarking against specificity scores in a previous study.

Chapter 5

Results and Discussion

The results and discussion of applying various supervised machine learning algorithms to wind turbine fault diagnosis and prediction using SCADA data. We compare the performance of different techniques in terms of accuracy.

5.1 Description of Data and Faults

The data in this study are derived from a 3 MW direct-drive turbine that supplies power to a large biomedical device manufacturing facility near the coast of Ireland. Two separate data sets were obtained from the turbine SCADA system "operational" and "status" data having different time spans. The status data has the longest record timespan from January 2014 to December 2015 while the shortest is SCADA data from April 2014 to April 2015. Therefore, when observing the SCADA records, we can refer to status and fault data to see what happens on the turbine at certain timestamps.

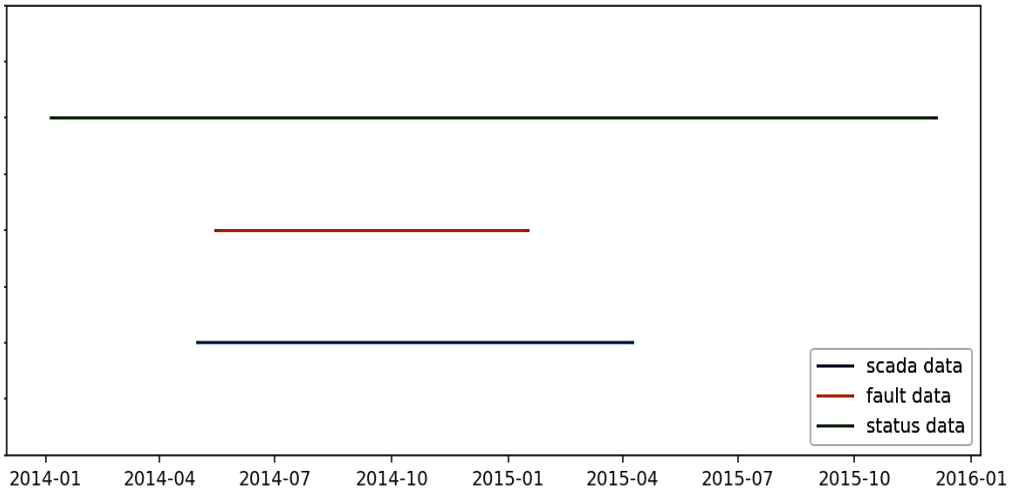


Figure 10: Time series analysis of the given data

5.1.1 Operational Data

The turbine control system is responsible for controlling and monitoring the operation of the wind turbine. It collects various instantaneous parameters that reflect the working conditions and performance of the wind turbine, such as wind speed and ambient temperature, real and reactive power, currents and voltages within the electrical equipment, and the

temperature of generator bearings and rotors. These parameters are important for detecting and diagnosing faults in the wind turbine components.

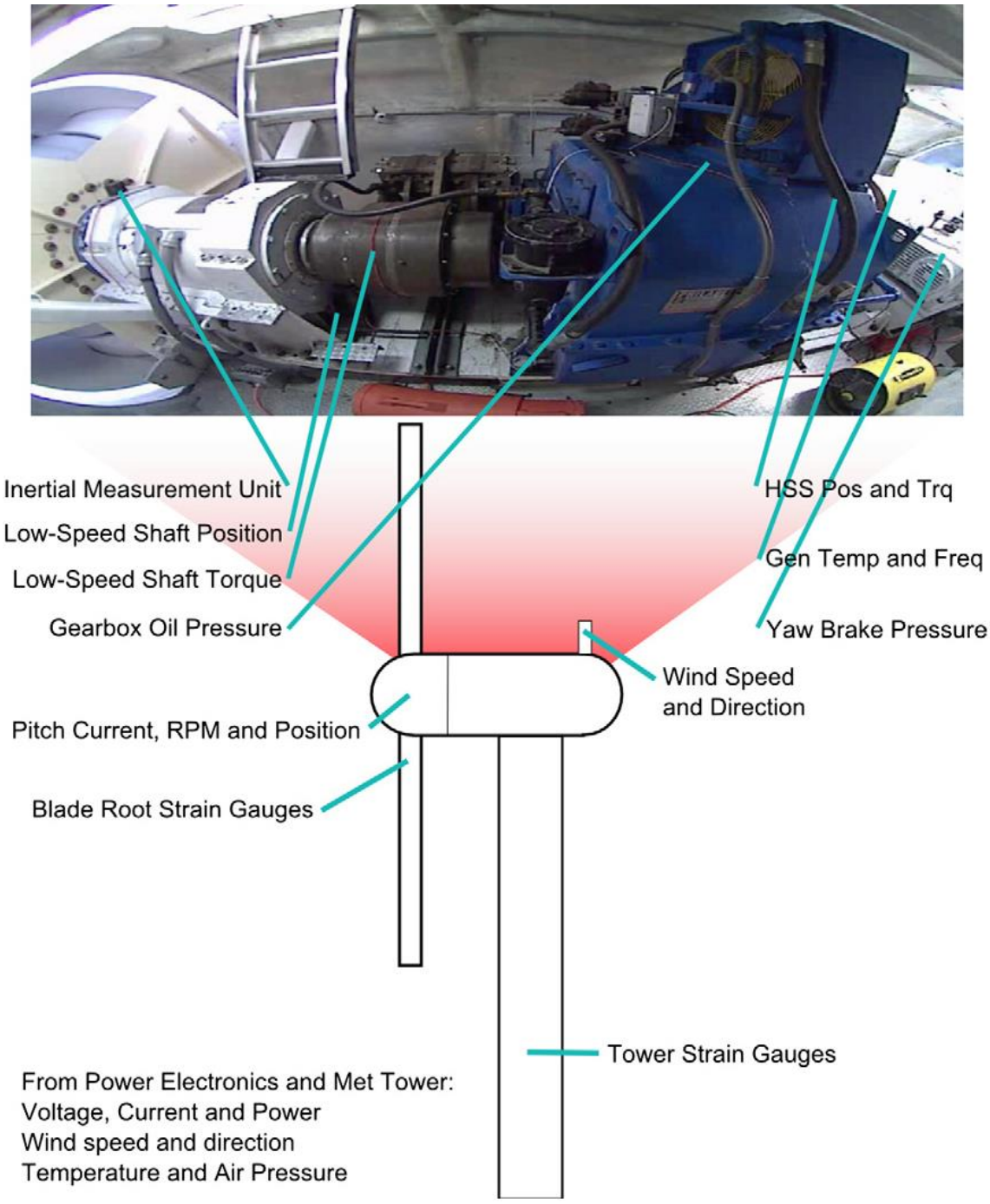


Figure 11: Sensor configuration

SCADA systems are used to store and process the data collected by the turbine control system. SCADA systems aggregate the data over a period of ten minutes and record the average,

the maximum, and the minimum of each parameter. This data is referred to as operational data. Table 1 shows a sample of this data, which includes 12 parameters and a label indicating the fault type. The operational data used in this research consists of approximately 45,000 data points, representing the 11-months analysed in this research. The operational data was used to train different machine learning classifiers that can predict the fault type based on the input parameters. The operational data was labelled based on three different processes explained in Section 3, which involve using expert knowledge, fault codes, and anomaly detection methods.

Table 1: Operational SCADA data

Date-Time	WEC: ava. Windspeed (m/s)	WEC: max. windspeed (m/s)	WEC: min. windspeed (m/s)	WEC: Operating Hours	WEC: Production (kWh)	Ambient temp. (°C)	…
05-01-14 0:00	6.9	9.4	2.9	881	939053	12	…
05-01-14 0:09	5.3	8.9	1.6	881	939053	12	…
05-01-14 0:20	5	9.5	1.4	881	939053	12	…
05-01-14 0:30	4.4	8.3	1.3	881	939053	12	…
05-01-14 0:39	5.7	9.7	1.2	881	939053	12	…

5.1.2 Status Data

The turbine can operate in different modes depending on the wind conditions and the control system. For example, when the wind speeds are within the normal range, the turbine is producing power normally. When the wind speeds are below the cut-in speed, which is the minimum speed required for power generation, the turbine is idle. When the wind speeds exceed a certain threshold, which is the maximum speed allowed for safe operation, the turbine is operating in storm mode, which means it shuts down and locks its blades to prevent damage. The status messages data records the changes in the turbine mode and other events that affect the operation of the turbine. The status messages data contains a timestamp and two codes: a main status code and a sub-status code.

The main status code indicates the general category of the event, such as normal operation, lack of wind, fault, maintenance, etc. The sub-status code indicates the specific type of event within that category, such as power production, low wind hysteresis, generator overtemperature, manual stop, etc. The status messages data can be used to track abnormal or faulty operations of the turbine by checking the main status code. Any main status code above zero indicates abnormal behaviour, but not necessarily a fault. For example, status code 2 means “lack of wind”, which is an abnormal but not faulty condition.

Table 2: WEC status data

Date-Time	Main Status	Sub Status	Status Text
05-06-14 17:35	0	5	Calibration of load control
05-06-14 17:39	0	1	Turbine starting
05-06-14 17:39	0	0	Turbine in operation
09-06-14 0:00	228	100	Timeout warn message: Malfunction air-cooling
09-06-14 0:04	222	3	Turbine reset: Scada system

5.2 Fault Classification

In Section 5.1.2, it is stated that any “main status” above zero indicates abnormal behaviour, but not necessarily a fault. Table 3 summarizes these faults. It is important to note that fault frequency refers to specific incidents of each fault, rather than the number of data points of operational data associated with it. The six operational data points of a generator heating fault lasting one hour, for example, would constitute one fault instance, even though it lasted one hour.

Table 3: Frequently occurring faults, listed by status code and number of corresponding 10-minute SCADA data points

Fault modes		Description	Main Status	No. of pts.
Feeding Fault	**(FF)**	Faults in the power feeder cables of the turbine (van Bussel and Zaaijer, 2001) **Causes:** Grid failures, lightning strikes, cablefaults, or circuit breaker trips **Effects:** Reduced power output, increased wear and tear, or damage to the components	62	254
Excitation Error	**(EF)**	Problems with the generator excitation system (Qiao and Lu, 2015) **Causes:** Faults in the generator windings, brushes, slip rings, or converters **Effects:** Low efficiency, poor power quality, or instability of the generator	80	174
Malfunction Air Cooling	**(AF)**	Problems with air circulation and internal temperature circulation within the turbine (Santelo et al., 2022) **Causes:** Faults in the fans, filters, ducts, or sensors **Effects:** Overheating, thermal stress, or degradation of the components	228	62
Generator Heating Fault	**(GF)**	Overheating in the generator (Santelo et al., 2022) **Causes**: Overloading, friction, misalignment, imbalance, or malfunction air cooling **Effects:** Reduced efficiency, increased losses, or failure of the generator	9	43
Mains failure fault	**(MF)**	Problems in mains power supply due to undervoltage or delay in start (van Bussel and Zaaijer, 2001) **Causes:** Grid faults, power outages, voltage fluctuations, or frequency deviations **Effects:** Loss of generation revenue, increased downtime, or damage to the components	60	20

NB: Fault modes are abbreviated within parenthesis

5.3 Exploratory data analysis

According to the SCADA data, it can be observed in Fig. 7 that wind power production dropped in October 2014, December 2014, and January 2015, with the most significant drop being in January 2015

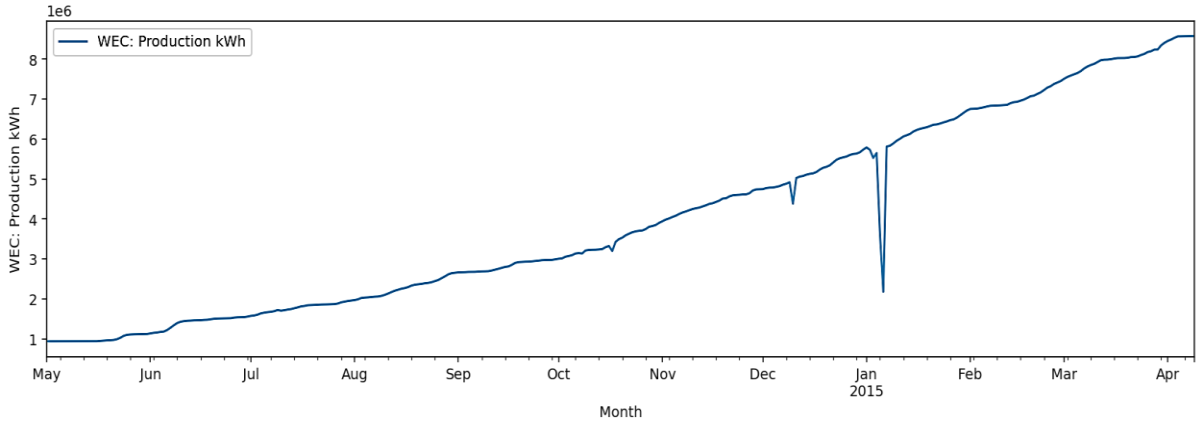


Figure 12: Production of wind energy over the period of time

On the other hand, there has been a significant increase in wind turbine faults in October 2014 as can be observed in Fig.8. In October and November 2014, a significant number of EF events occurred, and FF events also increased.

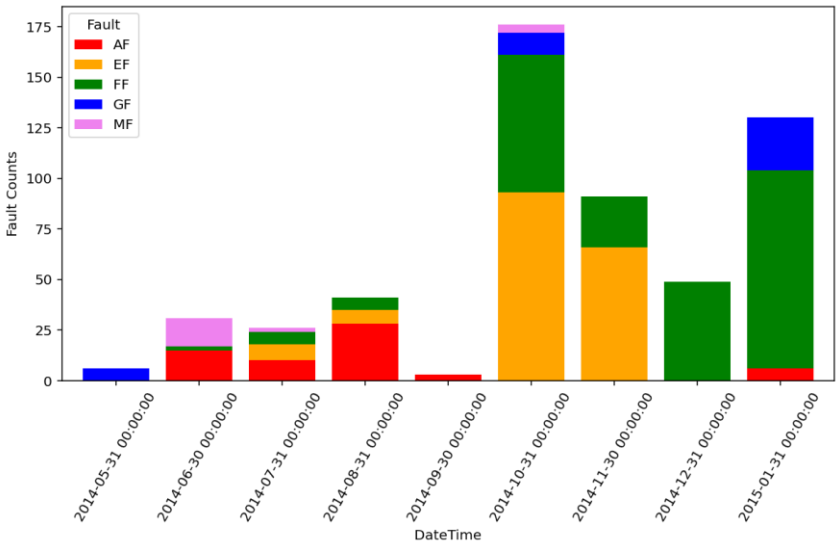


Figure 13: Wind turbine faults

On averaging the various faults with respect to the selected features, we could identify several anomalous behaviors of Fault Modes in Table 4. When compared to No Fault (NF), FF

has a lower active reactive power ava, while EF has a higher active reactive power ava, min, and max. In FF and MF, the nacelle cable twisting is greater than in NF. In AF and GF, it is negative. AF and MF have lower production. Generally, all faults have a significant blade angle, with the highest blade angle being FF.

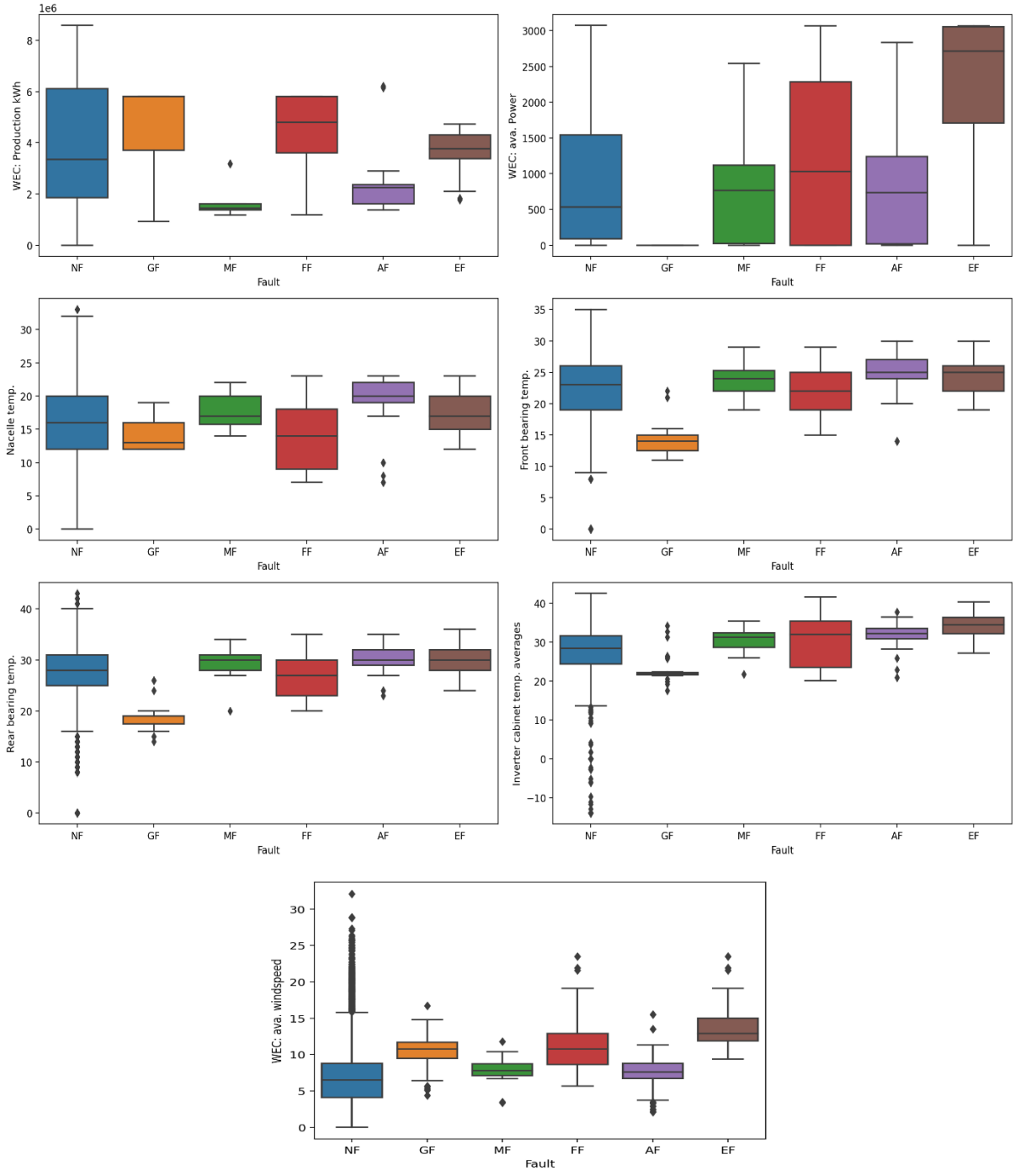


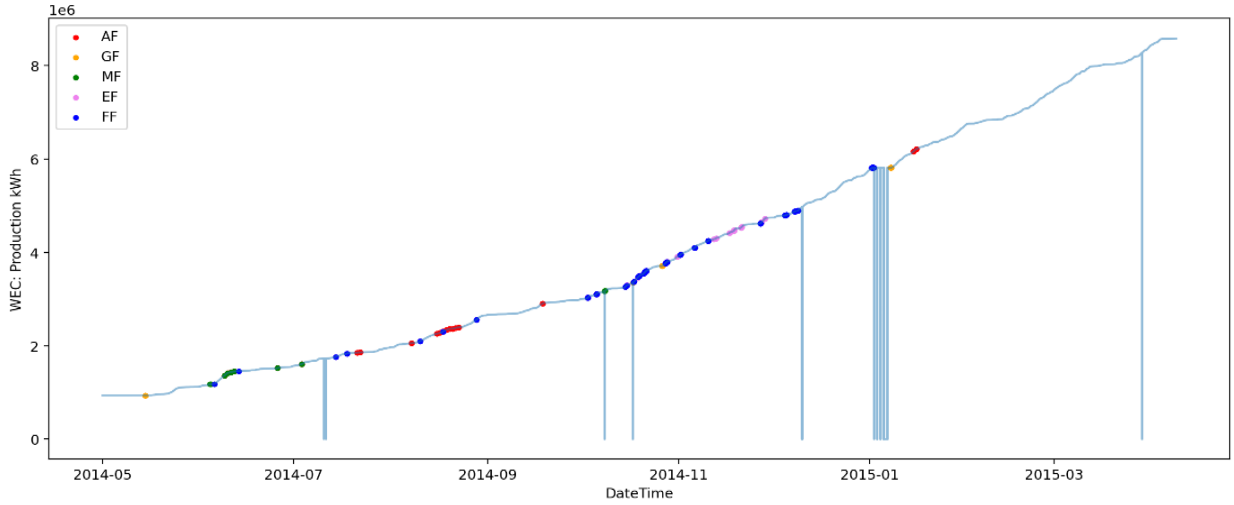
Figure 14: Distribution of SCADA data features for different fault types

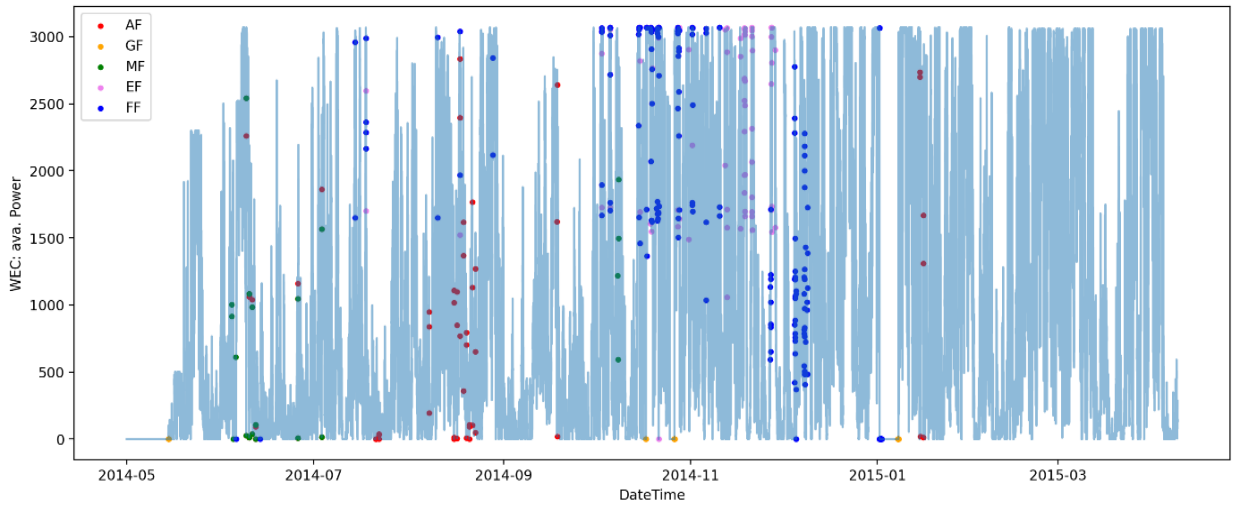
Table 4: Averaging various fault modes across selected features

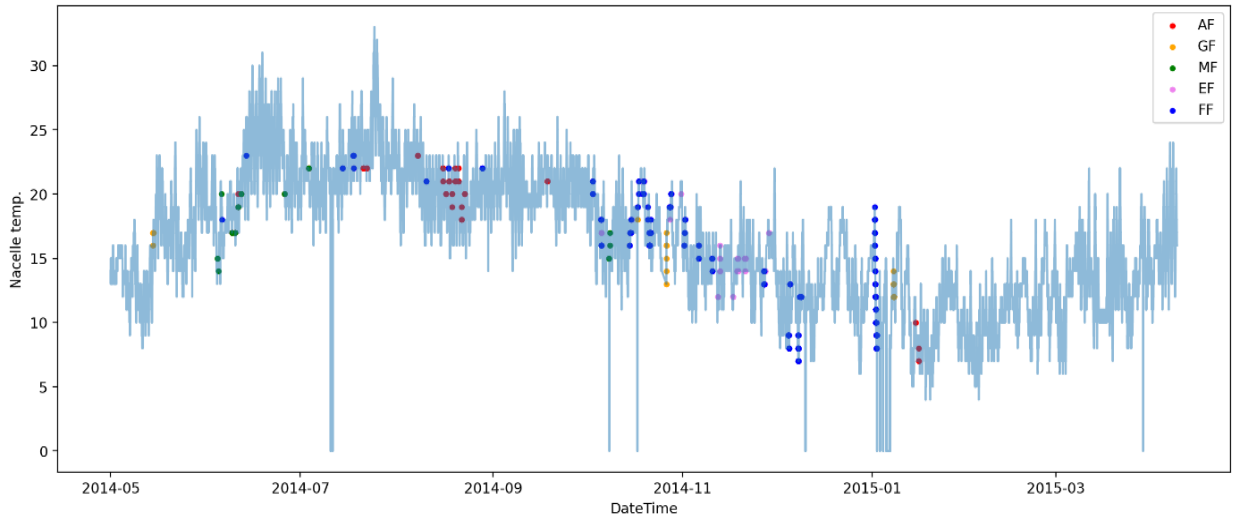
Features	AF	EF	FF	GF	MF	NF
WEC: ava. windspeed	7.376	13.594	11.184	10.3	7.79	6.912
WEC: ava. Rotation	7.382	12.074	7.486	9.068	7.263	9.09
WEC: ava. Power	803.71	2428.069	1207.335	0	760.7	990.487
WEC: ava. reactive Power	45.5	201.069	107.906	0	21.15	87.29
WEC: ava. blade angle A	23.525	16.517	33.658	24.323	27.217	8.278
Spinner temp.	20.694	19.264	16.965	12.628	19.05	17.667
Front bearing temp.	25.226	24.54	21.906	14.093	24.2	21.88
Pitch cabinet blade A temp.	35.903	36.954	34.504	30.442	33.4	33.62
Pitch cabinet blade B temp.	36.113	36.586	34.051	30.674	33.9	33.61
Pitch cabinet blade C temp.	35.113	35.741	33.043	29.814	33.1	32.847
Rotor temp. 1	55.565	100.368	66.071	34.14	52.3	53.543
Rotor temp. 2	55.677	99.92	65.925	34.86	52.5	53.613
Stator temp. 1	68.903	101.592	70.425	42.837	66.35	62.767
Stator temp. 2	68.323	100.454	69.72	42.465	65.85	62.237
Nacelle ambient temp. 1	15.629	14.5	11.016	12.116	14.8	12.287
Nacelle ambient temp. 2	15.629	14.299	10.917	11.93	14.65	12.183
Nacelle temp.	19.29	17.368	14.051	14.093	17.85	16.127
Nacelle cabinet temp.	22.919	20.736	17.831	19.395	21	19.503
Main carrier temp.	20.548	19.632	15.866	13.279	19.1	16.387
Rectifier cabinet temp.	32.629	30.385	26.697	30.767	30	30.253
Yaw inverter cabinet temp.	27.694	27.322	22.614	21.14	25.9	24.237
Fan inverter cabinet temp.	32.032	29.948	26.323	24.86	30.05	28.727
Ambient temp.	16.758	14.937	11.378	12.581	16.75	13.243
Tower temp.	28.129	28.891	21.638	16.349	27.15	23.65
Control cabinet temp.	36.516	38.833	31.551	24.884	35.2	32.213
Transformer temp.	46.903	64.121	51.063	30.233	45.25	44.55
RTU: ava. Setpoint 1	3042.194	3049.425	3047.48	2973.395	2940.2	2992.87
Inverter cabinet temp. (avg)	31.812	34.222	30.144	22.973	30.345	27.718
Inverter cabinet temp. (std dev)	1.439	1.617	1.639	1.141	1.456	1.902

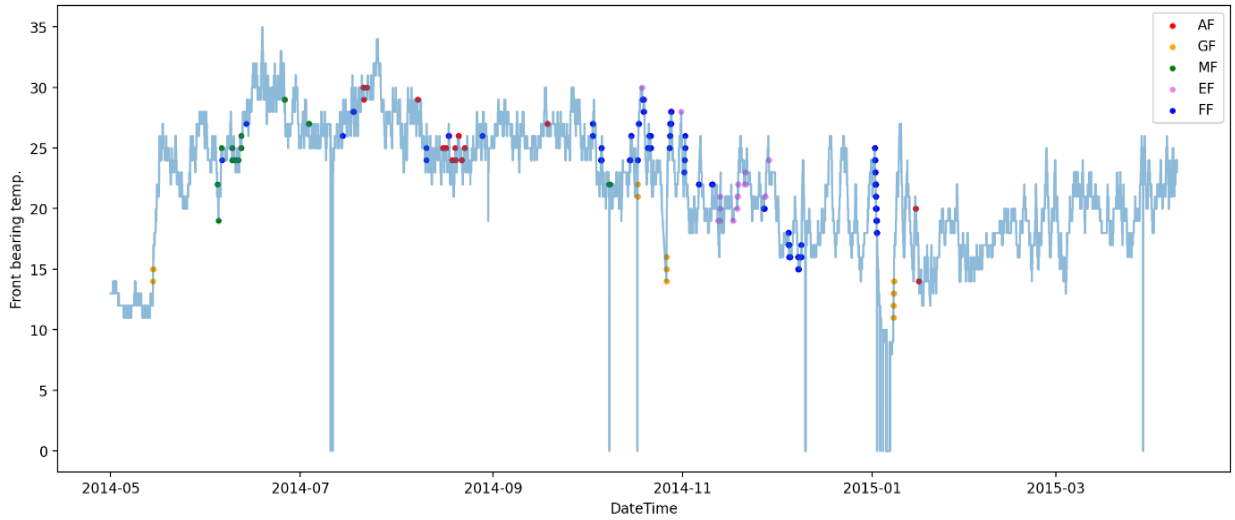
In general, GF has the lowest temperature across all components (cabinet temperature, spinner, front bearing, etc). While other faults (FF, AF, MF, EF) have higher temperatures. Cabinet, pitch, rotor, stator, ambient, control, tower, and transformer have the highest EF temperatures. Spinners, front bearings, rare bearings, nacelles, main carriers, rectifiers, yaw, and fan inverters have the highest AF temperatures.

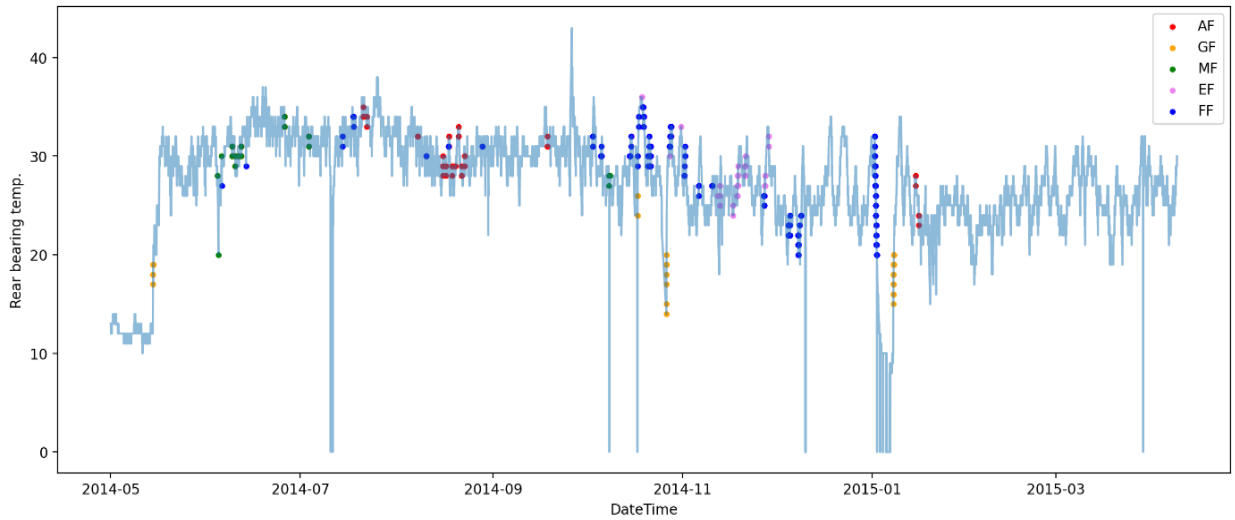
The following plots depict various SCADA data features with fault points showing the relationship between different data points collected by the SCADA system and how they relate to the occurrence of faults. For example, in the context of wind turbines, it shows the relationship between the temperature of the inverter cabinet and the occurrence of faults. By analyzing the data and identifying patterns or trends, operators can use the information to detect faults early and take appropriate action to prevent or mitigate their impact.






AF
GF
MF
EF
FF
Nacelle temp.
DateTime
2014-05
2014-07
2014-09
2014-11
2015-01
2015-03


AF
GF
MF
EF
FF
Front bearing temp.
DateTime
2014-05
2014-07
2014-09
2014-11
2015-01
2015-03


AF
GF
MF
EF
FF
Rear bearing temp.
DateTime
2014-05
2014-07
2014-09
2014-11
2015-01
2015-03

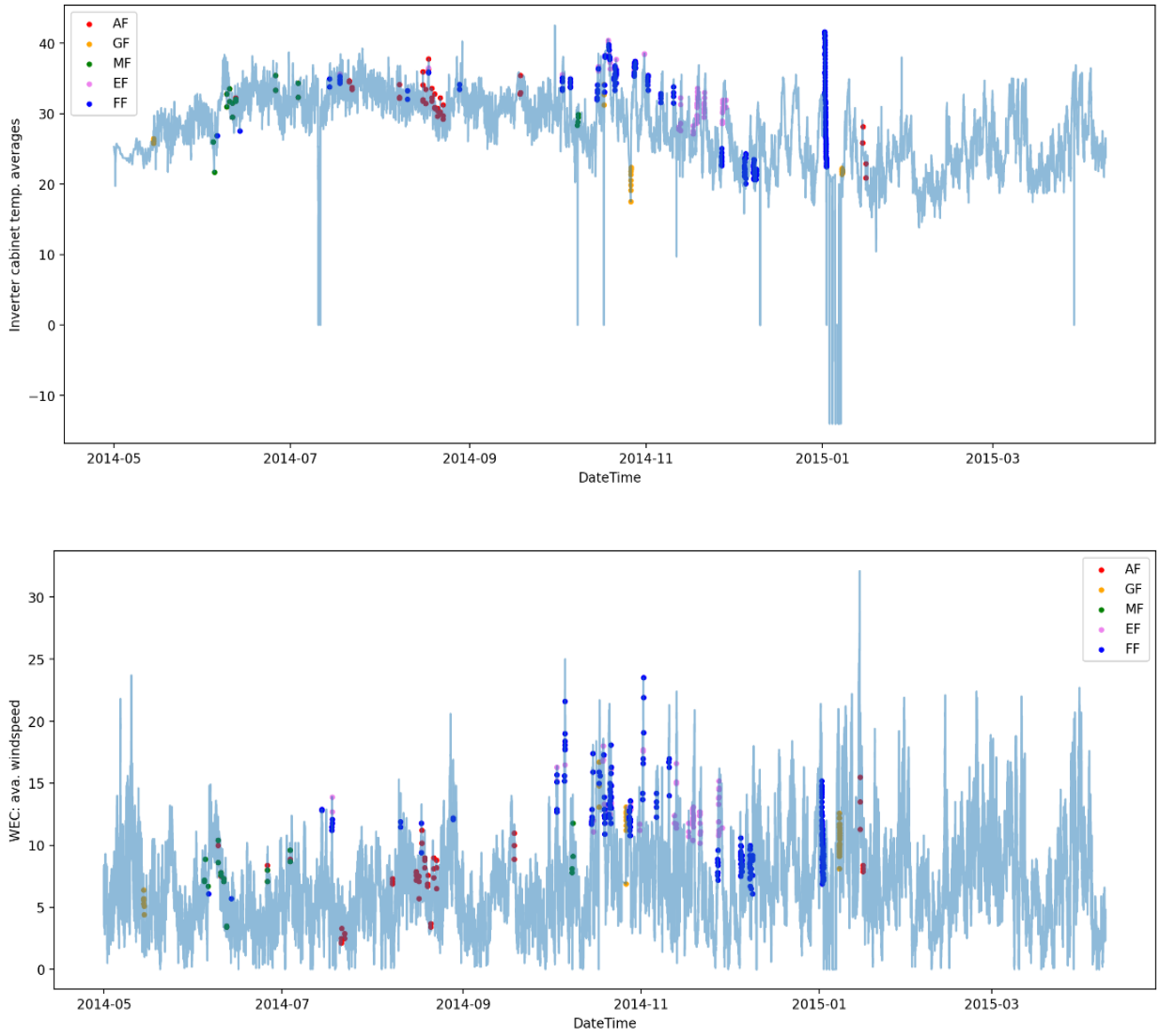


Figure 15: SCADA data features with fault points

In addition to temperature data, windspeed and energy production data are also included. By combining multiple data points and analyzing their relationships, it may be possible to identify more complex fault conditions or even predict potential failures before they occur.

5.4 Supervised Machine Learning-based Fault Diagnosis and Prediction

During the preparation of the datasets for machine learning, we observed that there were a greater number of NF (normal conditions) records than faulty records, which inevitably leads

to an imbalanced dataset. As a consequence, we sampled only 300 records from the No Fault dataset in order to avoid bias

We are going to build a predictive model to classify fault modes of wind turbine based on the information or status of wind turbine components (gear box, tower, nacelle, bearing, etc.) from SCADA system. This is a multiclass classification task.

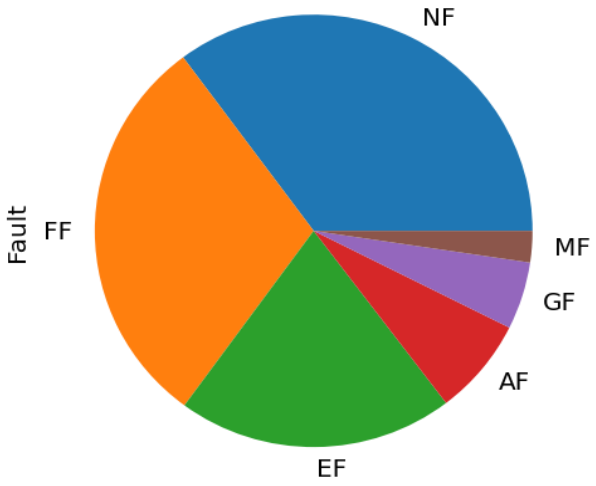


Figure 16: Fault modes

Because our training data is largely imbalanced for each fault modes, we use class weights in algorithms in order to modify the loss function directly by giving a penalty to the classes with different weights. It means purposely increasing the power of the minority class and reducing the power of the majority class which gives better results than SMOTE (Synthetic Minority Over-sampling Technique).

5.4.1 Hyperparameter selection

Machine learning models have various parameters that can be adjusted to improve their performance. These parameters, also known as hyperparameters, control the behavior of the model and can have a significant impact on its accuracy and effectiveness. Tuning these parameters involves finding the optimal combination of values that results in the best performance of the model.

In this case, random search was used to tune the parameters of a variety of machine learning models, including Support Vector Machines (SVMs), K-Nearest Neighbors (KNNs), Random Forests (RFs), XGBoost, and Bayesian Neural Networks (BNNs). The parameter space

for each model was defined based on the literature, meaning that the range of possible values for each parameter was determined based on previous research and best practices.

Table 5: Model hyperparameters

Model	Variables	Values
SVM	Kernel type	rbf
	γ	'scale'
	C	1.0
KNN	n_neighbors	5
	metric	Minkowski
	leaf_size	30
Random Forest	n_estimators	100
	max_depth	None
	min_samples_split	2
XGBoost	learning_rate	None
	max_depth	None
	min_child_weight	None
BNN	Hidden layer neurons	5
	prior_mu	0
	prior_sigma	0.1
	Hidden layer activation function	ReLU
	Output layer activation function	Sigmoid
	Iterations	3000
	Batch size	64
	Learning rate	1E-05

5.4.2 Model Evaluation

The dataset was split into three parts: 70% for training, 15% for validation, and 15% for testing. The optimal parameters were selected and then fine-tuned on a Windows 10 system with an Intel Core i7 processor and 16 GB of RAM.

The performance of the models was evaluated using a confusion matrix and a classification report. The confusion matrix shows the number of instances that were correctly

classified and the number of instances that were incorrectly classified. The classification report shows the accuracy, precision, recall, and F1 score for each model.

By using random search to tune the parameters of these models, it is possible to improve their performance and achieve better results. This can lead to more accurate predictions and more effective decision-making.

- **Support Vector Machine (SVM)**

SVM has the lowest F1 score for both failure and no-failure classes, which means that it is not very good at predicting either class correctly. It has a low recall for the no-failure class and a low precision for the failure class.

Table 6: Classification report of SVM model

	Precision	Recall	F1-score	Support
AF	0.39	0.60	0.47	20
EF	0.59	1	0.74	57
FF	0.97	0.49	0.65	76
GF	1	1	1	10
MF	0.15	0.29	0.2	7
NF	0.79	0.62	0.69	86
accuracy			0.67	0.67
macro avg	0.65	0.66	0.63	256
weighted avg	0.76	0.67	0.67	256

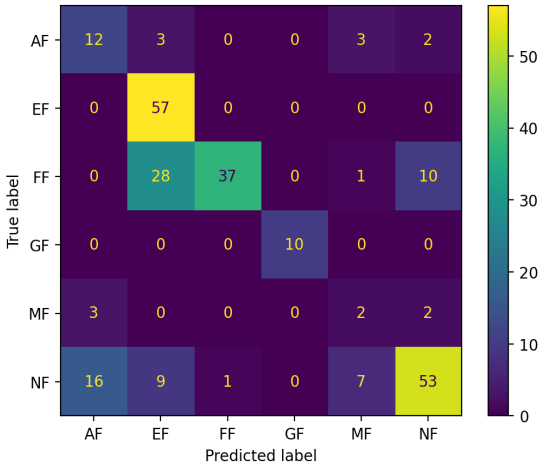


Figure 17: Confusion matrix of SVM model

Figure 18 illustrates the evaluation of a Support Vector Machine (SVM) model in predicting fault points within a wind turbine system. The figure comprises three subplots, each displaying a distinct parameter plotted against time. The x-axis represents time in chronological order, while the y-axis denotes the value of the respective parameter. The parameters WEC: ava windspeed, WEC: ava Power, and nacelle temperature are represented as continuous lines. Fault points predicted by the model are indicated on the graph, with correctly predicted fault points denoted by '.' markers and falsely predicted fault points denoted by 'x' markers.

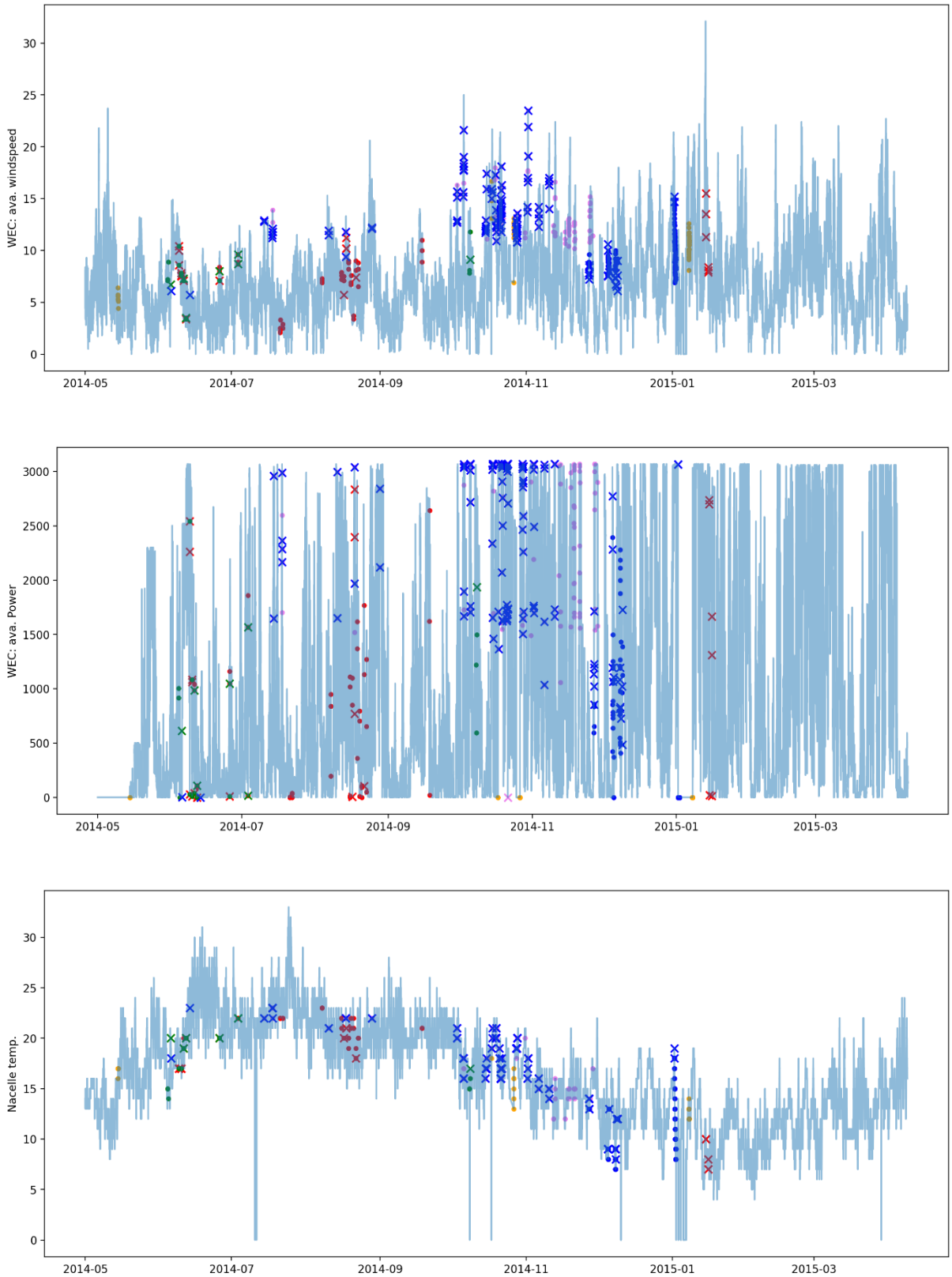


Figure 18: Prediction of Fault Points Using SVM Model

- **K-Nearest Neighbors (KNN)**

KNN has the highest precision and F1 score for the no-failure class, but the lowest recall for the failure class. This means that it is very good at predicting no-failures correctly, but also misses many failures.

Table 7: Classification report of KNN model

	Precision	Recall	F1-score	Support
AF	0.57	0.65	0.60	20
EF	0.46	0.53	0.49	57
FF	0.60	0.61	0.60	76
GF	1	1	1	10
MF	0.25	0.14	0.18	7
NF	0.92	0.83	0.87	86
accuracy			0.67	0.67
macro avg	0.63	0.63	0.63	256
weighted avg	0.68	0.67	0.67	256

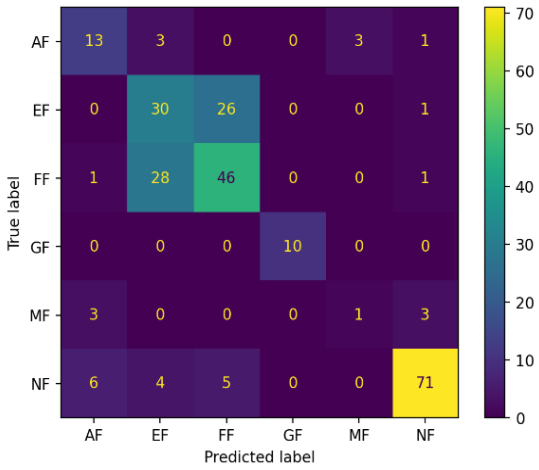
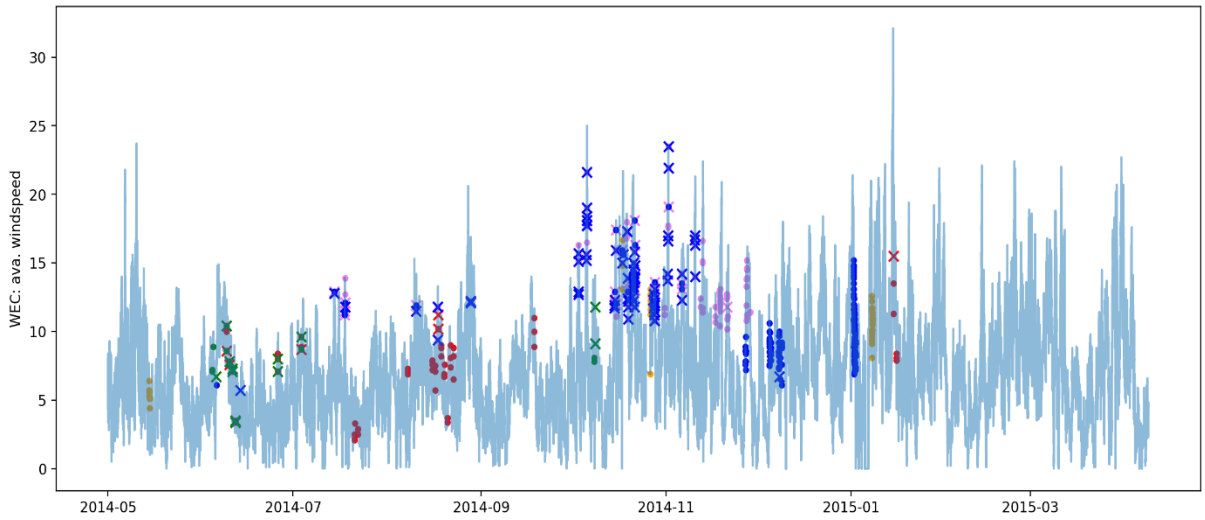


Figure 19: Confusion matrix of KNN model

This following figure presents a visual representation of the relationship of WEC: Ava Windspeed, WEC: Ava Power, and Nacelle Temperature with various types of fault within a wind turbine system. It also evaluates the performance of a K-Nearest Neighbors (KNN) model in predicting fault points. By examining the trends and comparing them with the predicted fault points, one can assess the accuracy model in detecting faults within the wind turbine system**.**



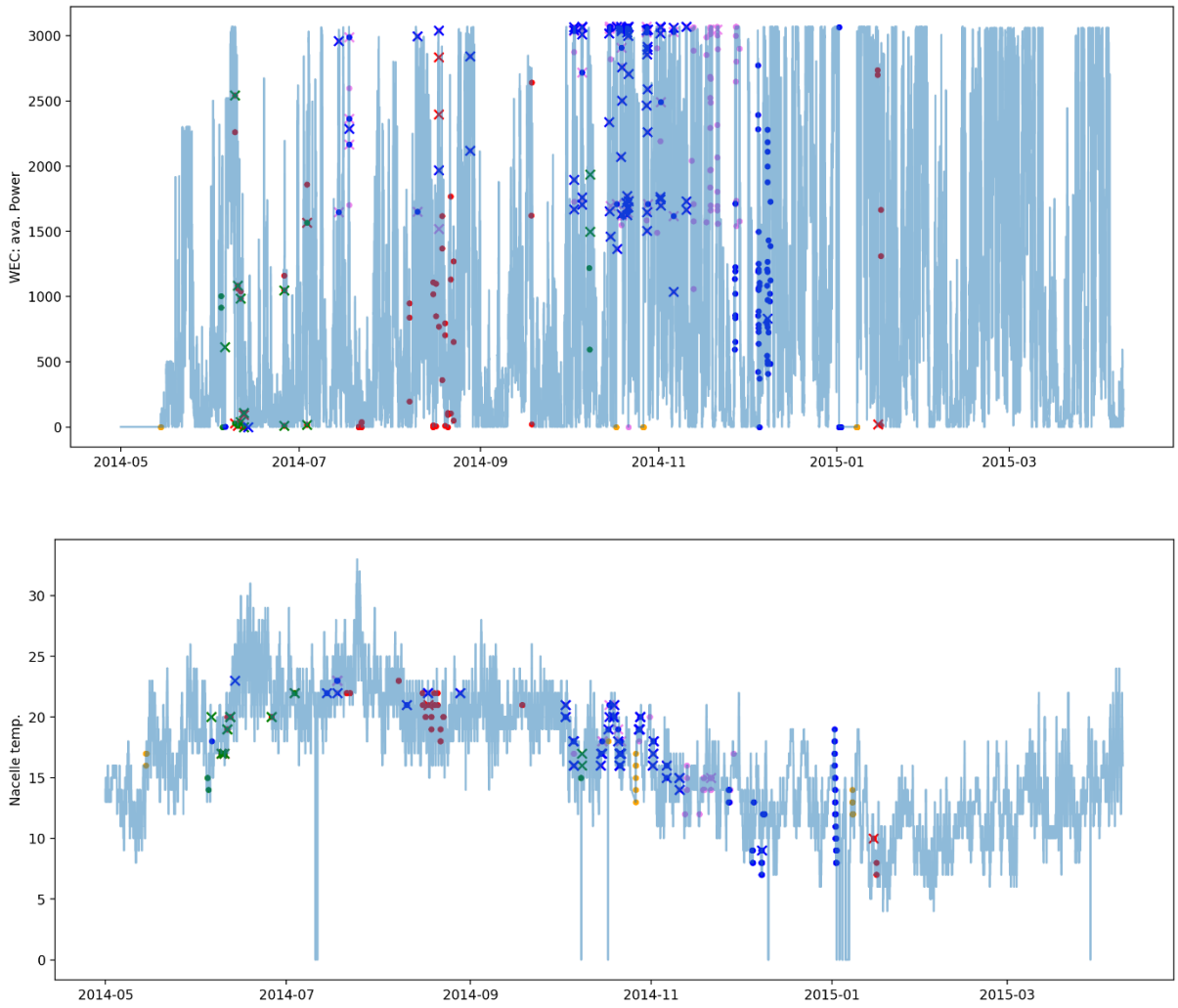


Figure 20: Prediction of Fault Points Using KNN Model

- **Random Forest**

	Precision	Recall	F1-score	Support
AF	0.72	0.65	0.68	20
EF	0.51	0.53	0.52	57
FF	0.62	0.66	0.64	76
GF	1	1	1	10
MF	0.25	0.14	0.18	7
NF	0.94	0.92	0.93	86
accuracy			0.71	0.71
macro avg	0.67	0.65	0.66	256
weighted avg	0.71	0.71	0.71	256

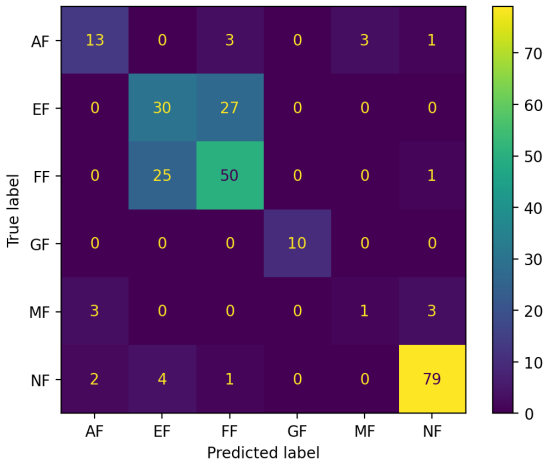


Figure 21: Confusion matrix of Random Forest model

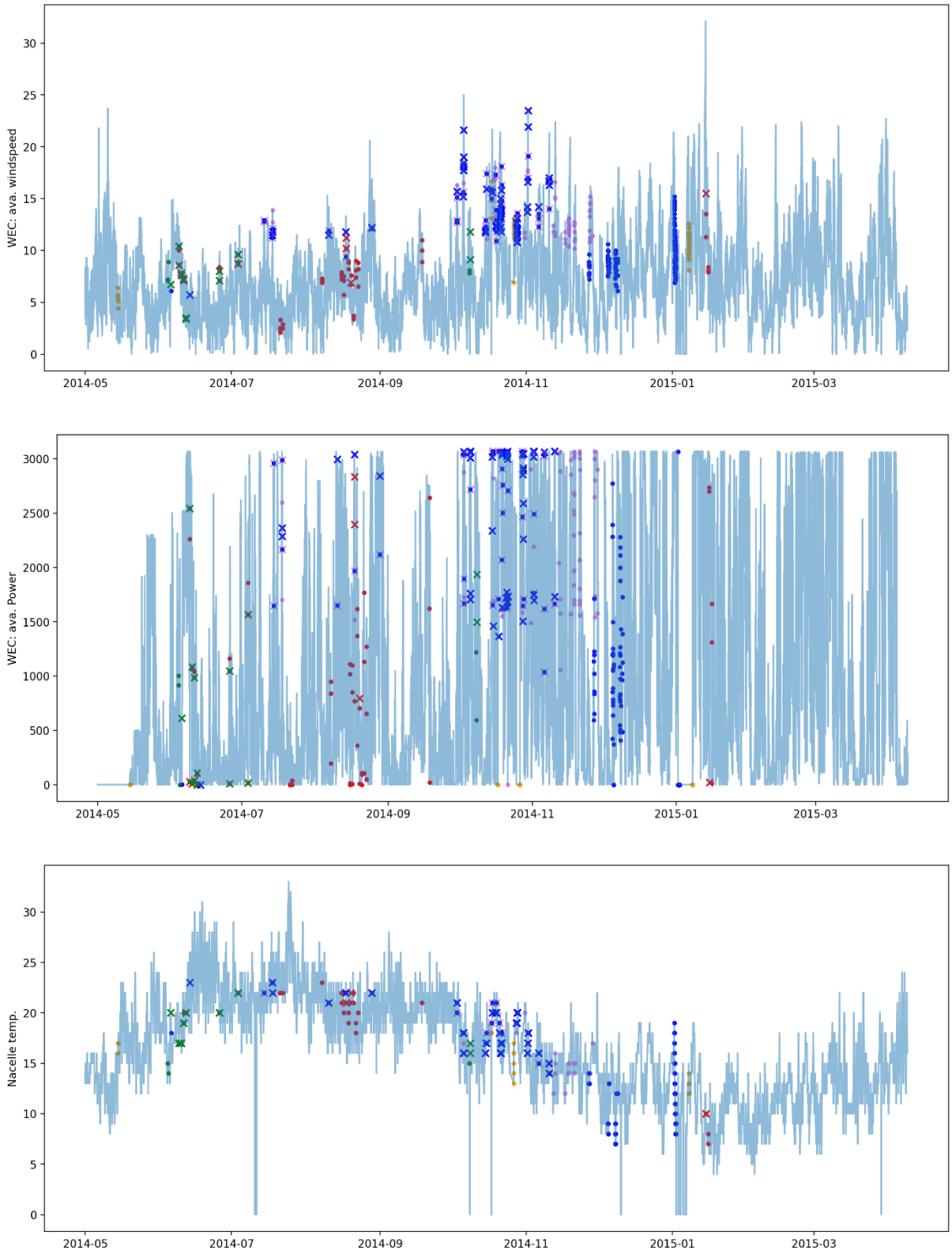


Figure 22: Prediction of Fault Points Using Random Forest Model

- **XGBoost**

  According to the classification reports, Random Forest and XGBoost have similar and balanced performance across both classes, with high precision, recall and F1 score. These metrics indicate that Random Forest and XGBoost are able to predict both classes correctly and consistently. Therefore, they might be the best models to use if both classes are equally important for the problem domain and the evaluation criteria.

Table 8: Classification report of XGBoost model

	Precision	Recall	F1-score	Support
AF	0.6	0.6	0.6	20
EF	0.46	0.46	0.46	57
FF	0.58	0.66	0.62	76
GF	1	0.9	0.95	10
MF	0.2	0.14	0.17	7
NF	0.95	0.87	0.91	86
accuracy			0.68	0.68
macro avg	0.63	0.60	0.62	256
weighted avg	0.68	0.68	0.68	256

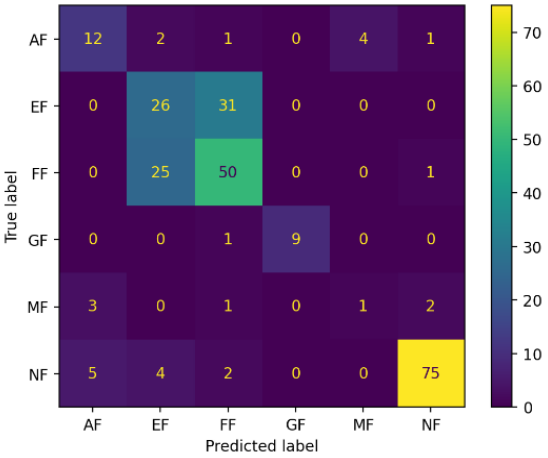
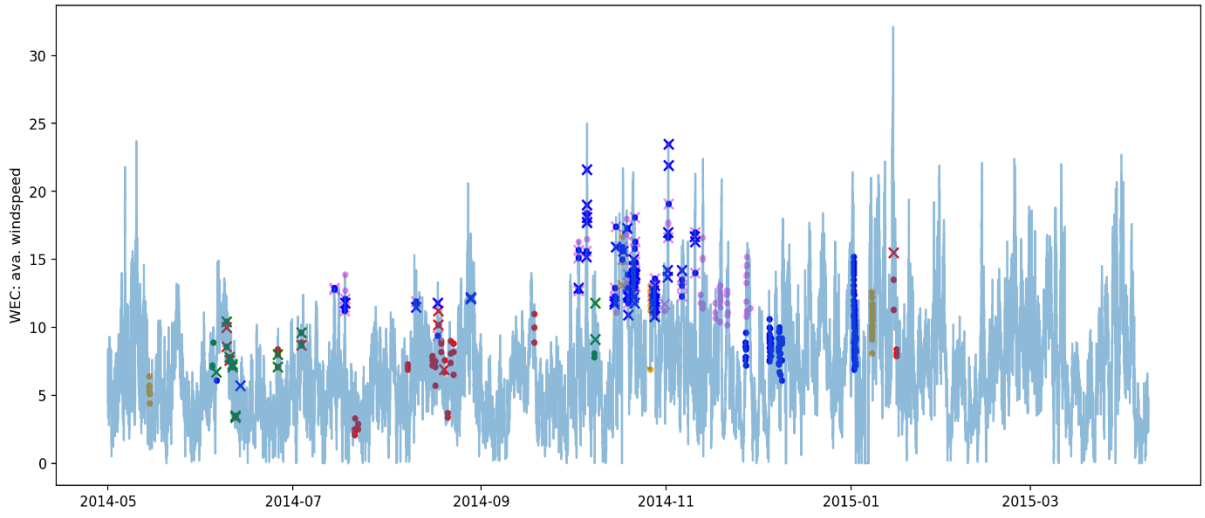


Figure 23: Confusion matrix of XGBoost model

The following figure shows that the fault point predictions are similar to those of random forest, indicating that XGBoost is a more robust and accurate algorithm for identifying faults in wind turbines. XGBoost can deal with unbalanced data, prevent overfitting, and enhance the performance of decision trees by applying gradient boosting and regularization methods .



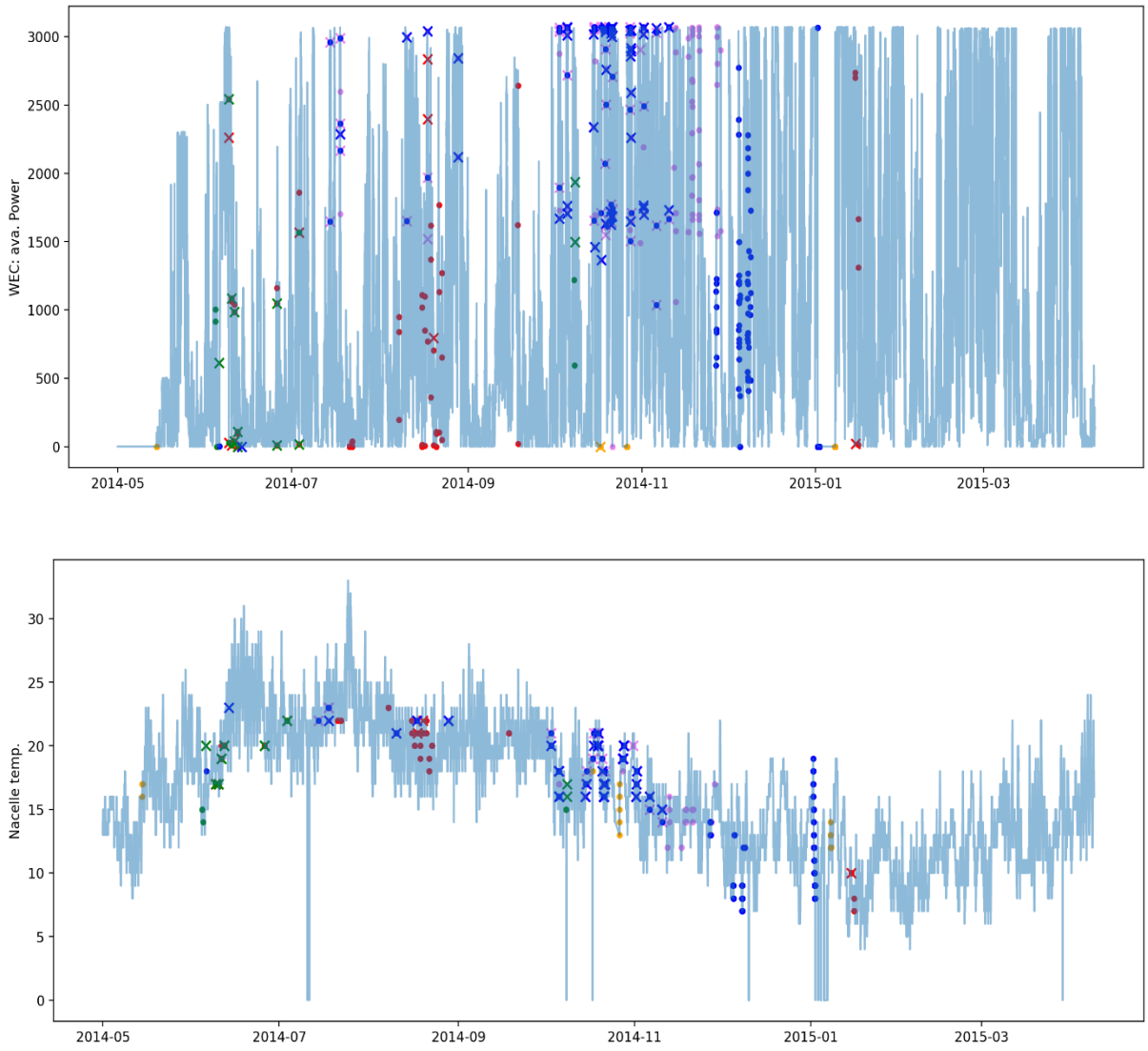


Figure 24: Prediction of Fault Points Using XGBoost Model

- **Bayesian Neural Network (BNN)**

BNN has the highest recall and F1 score for the failure class, but the lowest precision and accuracy for both classes. A high recall and F1 score for the failure class indicates that the model can capture most of the failures, but a low precision and accuracy means that it also misclassifies many non-failures as failures and vice versa. This could be problematic if the cost of false positives and false negatives are very different or high.

Table 9: Classification report of BNN model

	Precision	Recall	F1-score	Support
AF	0.33	0.24	0.28	21
EF	0.40	0.89	0.56	47
FF	0.73	0.45	0.56	84
GF	1	1	1	8
MF	0	0	0	5
NF	0.94	0.60	0.67	91
accuracy			0.75	0.58
macro avg	0.64	0.53	0.51	256
weighted avg	0.64	0.58	0.75	256

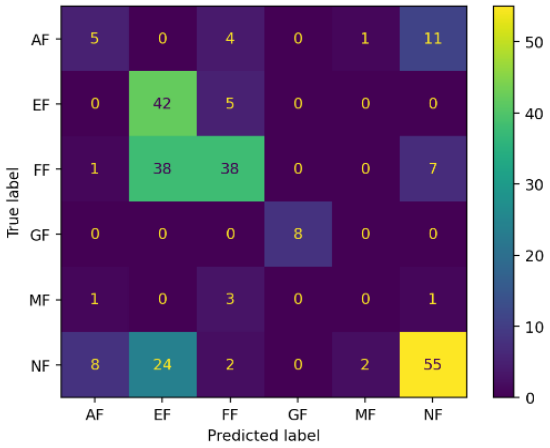


Figure 25: Confusion matrix of BNN model

The learning curve of Bayesian Neural Network is depicted in Figure 26. It shows the performance of a machine learning model as a function of time or iterations. It can be used to evaluate the convergence and generalization of the model, as well as to compare different models or hyperparameters1. In this case, the learning curve shows how the Bayesian Neural Network learns from the data and improves its predictions over time.

The model tries to minimize the loss by updating its parameters during the training process. The lower the loss, the better the model fits the data

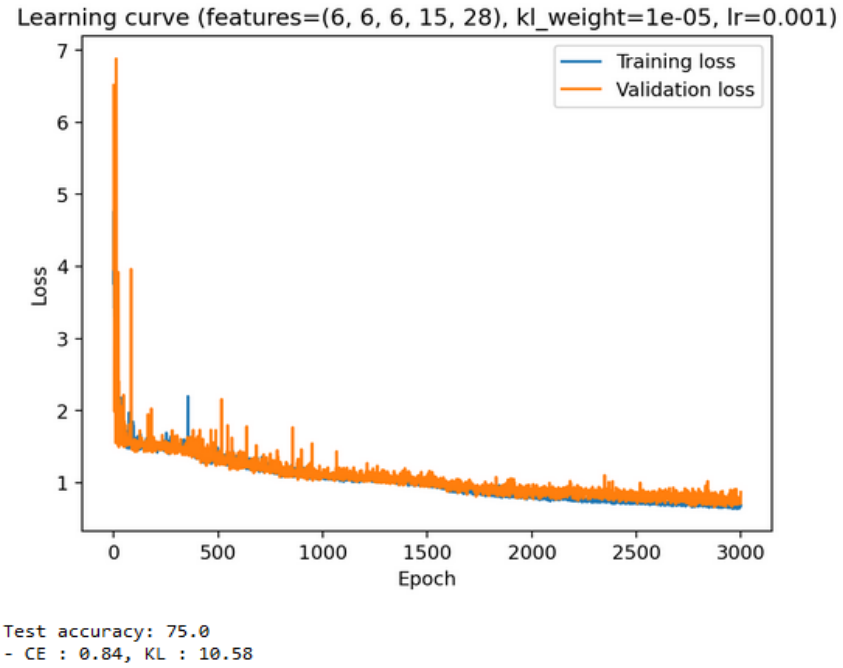


Figure 26: Learning of BNN model

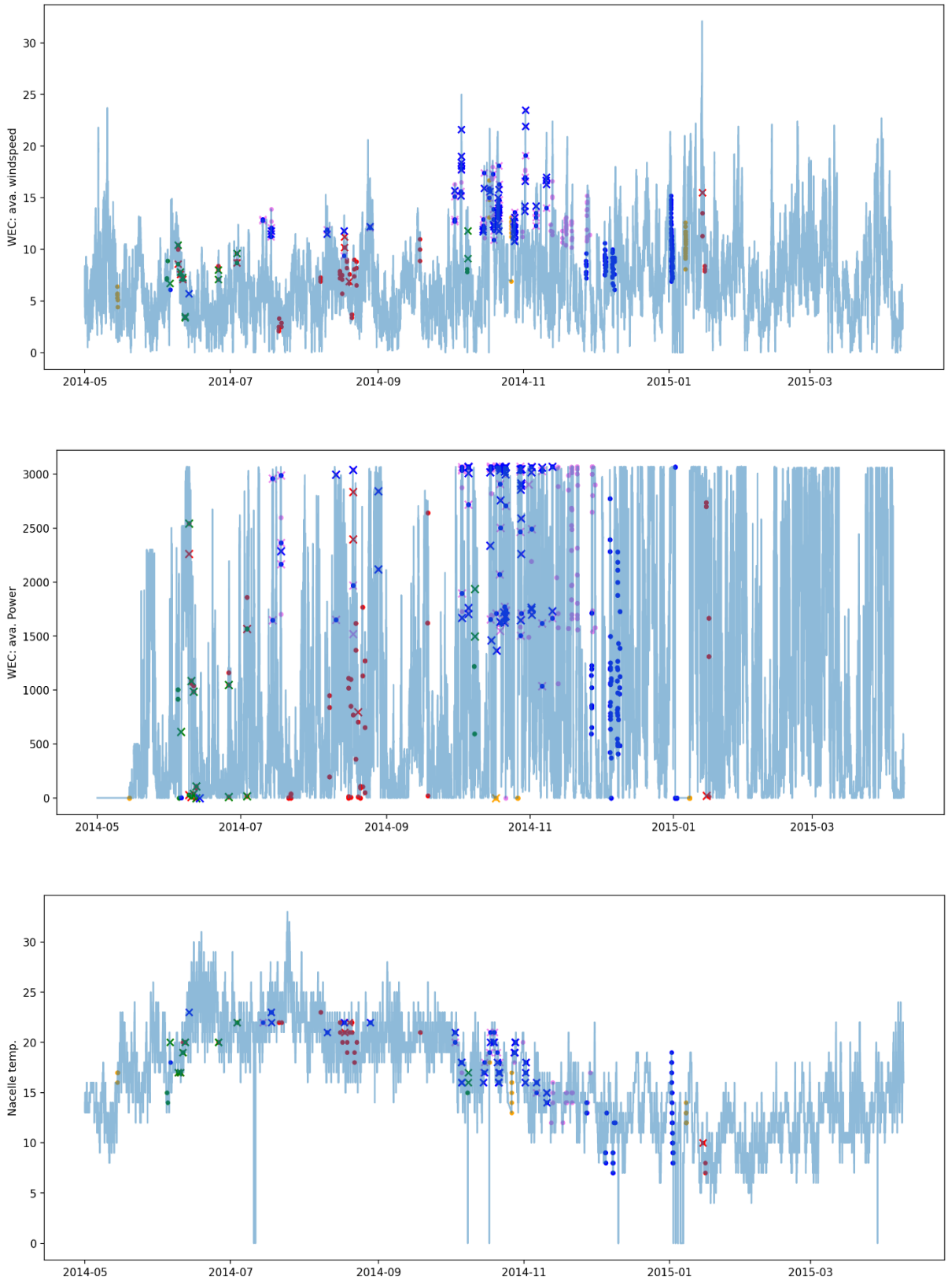


Figure 27: Prediction of Fault Points Using BNN Model

Table 10 shows the results obtained from the models. The models have similar performance in terms of precision, recall, and F1-score. The high precision and recall values for predicting no failure can be attributed to the large number of samples. The recall for predicting faulty states is reasonable and consistent with the typical forecast scores expected from the machine learning tools studied here.

Table 10: Model test results

Prediction	Model	Precision	Recall	F1 score	Data
No-failure	SVM	0.79	0.62	0.69	
	KNN	0.99	0.83	0.87	
	Random Forest	0.94	0.92	0.93	7432
	XGBoost	0.95	0.87	0.91	
	BNN	0.74	0.60	0.67	
Failure	SVM	0.76	0.67	0.67	
	KNN	0.68	0.67	0.67	
	Random Forest	0.71	0.71	0.71	441
	XGBoost	0.68	0.68	0.68	
	BNN	0.64	0.58	0.75	

Overall, the results showed that all of the models were able to achieve high accuracy, precision, recall, and F1 scores. However, the BNN model had the highest performance, followed by the Random Forest model, the XGBoost model, the SVM model and the KNN model.

Chapter 6
Conclusion and Future Work

6.1 Conclusion

This study investigated the feasibility of predicting faulty behavior in wind turbines one hour in advance using machine learning models. Five types of faults were considered in this study: generator overheating, main power supply failure, feeding mechanism fault, air cooling system malfunction due to timeout warning, and DC-link overvoltage due to excitation error.

To detect and classify these faults, different machine learning models were trained and tested using real data from a wind farm in Southern Ireland. The data were obtained from the supervisory control and data acquisition (SCADA) system, which monitors the operational parameters of the wind turbines. The data were preprocessed and feature selection was performed using a combination of domain knowledge and data analysis. The domain knowledge helped to identify and remove the features that were broken, incorrect, redundant, or irrelevant for the fault prediction task. The data analysis helped to identify and consolidate the features that had very similar readings, such as the inverter temperatures. The feature selection reduced the number of features from 60 to 29. The features were then scaled to unit norm to avoid the influence of different ranges and scales on the machine learning models.

Five machine learning models were used: Support Vector Machines (SVMs), K-Nearest Neighbors (KNN), Random Forest , XGBoost, and Bayesian Neural Network (BNN). The models were evaluated using metrics such as precision, recall, and F1-score to compare their performance in predicting the likelihood of a fault occurring in the following hour in real time. The results showed that the BNN model achieved the highest accuracy of 75%, followed by Random Forest with 71%, XGBoost with 68%, and SVMs & KNN with 67%.

The main contribution of this study is to demonstrate that machine learning models can be used to predict faulty behavior in wind turbines with high accuracy using SCADA data. This can significantly reduce wind farm downtime, which can save money and improve the reliability of wind energy generation. Moreover, this study provides a comprehensive comparison of different machine learning models for fault detection and classification in wind turbines, which can help researchers and practitioners to choose the best model for their

applications. Furthermore, this study suggests that SCADA data can be a valuable source of information for condition monitoring and predictive maintenance of wind turbines, which can enhance their performance and lifespan.

6.2 Future Work

This study has demonstrated the feasibility and effectiveness of using machine learning models to predict faulty behavior in wind turbines one hour in advance using SCADA data. However, there are still some limitations and challenges that need to be addressed in future work. Some possible directions for future work are:

- **Extending the types and sources of data:** This study only considered five types of faults in wind turbines and used SCADA data as the sole source of information. However, there may be other types of faults that affect the wind turbine performance and reliability, such as blade icing, gearbox failure, and pitch system malfunction. Moreover, there may be other sources of data that can provide useful information for fault detection and classification, such as vibration signals, acoustic emissions, thermography, and image analysis. Therefore, future work can explore the use of more diverse and comprehensive data to improve the accuracy and robustness of the machine learning models.
- **Improving the feature selection and extraction methods:** This study used correlation analysis to select the features from the SCADA data. However, correlation analysis may not capture the nonlinear and complex relationships between the features and the faults. Moreover, correlation analysis may not be able to handle high-dimensional and noisy data effectively. Therefore, future work can investigate the use of more advanced feature selection and extraction methods, such as Principal Component Analysis (PCA), Independent Component Analysis (ICA), autoencoders, and deep neural networks. These methods can reduce the dimensionality and noise of the data, as well as extract more meaningful and representative features for fault prediction.
- **Exploring different machine learning models and techniques:** This study used five machine learning models: SVMs, KNN, Random Forest, XG Boost, and BNN. However, there may be other machine learning models that can achieve better performance in fault prediction, such as Convolutional Neural Networks (CNNs), Recurrent Neural Networks

(RNNs), Long Short-Term Memory (LSTM), Gated Recurrent Units (GRU), and attention mechanisms. These models can capture the temporal and spatial patterns of the data, as well as learn from long-term dependencies and context information. Moreover, there may be other machine learning techniques that can enhance the performance of the models, such as ensemble learning, transfer learning, active learning, and reinforcement learning. These techniques can improve the generalization ability, adaptability, efficiency, and interactivity of the models.

- **Evaluating the practical impact and value of the machine learning models:** This study evaluated the performance of the machine learning models using metrics such as precision, recall, and F1-score. However, these metrics may not reflect the practical impact and value of the models in terms of reducing wind farm downtime, operation and maintenance costs, levelized cost of energy, and greenhouse gas emissions. Therefore, future work can conduct a more comprehensive and realistic evaluation of the machine learning models using metrics such as availability, reliability, maintainability, profitability, return on investment, net present value, internal rate of return, payback period, carbon footprint, and life cycle assessment. These metrics can provide a more holistic and meaningful assessment of the benefits and drawbacks of using machine learning models for wind turbine fault prediction.

Appendix A

Data Analytics and Classification Model for Failure Detection of Wind Turbine from IIoT Data

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

import matplotlib.pyplot as plt
import missingno as msno
import seaborn as sns
import xgboost as xgb

from imblearn.over_sampling import SMOTE
from imblearn.pipeline import make_pipeline
from sklearn.utils import class_weight
from sklearn.model_selection import train_test_split, cross_validate, StratifiedK
Fold, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import ConfusionMatrixDisplay, classification_report
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier

%config InlineBackend.figure_format = 'retina'
```

A1. Read data

We have 3 data:

- *scada_data.csv*: Contains >60 information (or status) of wind turbine components recorded by SCADA system
- *fault_data.csv*: Contains wind turbine fault types (or modes)
- *status_data.csv*: Contains description of status of wind turbine operational

```
scada_df = pd.read_csv('scada_data.csv')
scada_df['DateTime'] = pd.to_datetime(scada_df['DateTime'])
# scada_df.set_index('DateTime', inplace=True)

scada_df
```

```
                DateTime        Time  Error  WEC: ava. windspeed  \
0     2014-05-01 00:00:00  1398920448      0                  6.9
1     2014-05-01 00:09:00  1398920960      0                  5.3
2     2014-05-01 00:20:00  1398921600      0                  5.0
3     2014-05-01 00:30:00  1398922240      0                  4.4
4     2014-05-01 00:39:00  1398922752      0                  5.7
...                   ...         ...    ...                  ...
49022 2015-04-08 23:20:00  1428553216      0                  3.9
49023 2015-04-08 23:30:00  1428553856      0                  3.9
```

```
49024 2015-04-08 23:39:00  1428554368      0                 4.2
49025 2015-04-08 23:50:00  1428555008      0                 4.1
49026 2015-04-09 00:00:00  1428555648      0                 4.8

       WEC: max. windspeed  WEC: min. windspeed  WEC: ava. Rotation  \
0                      9.4                  2.9                0.00
1                      8.9                  1.6                0.00
2                      9.5                  1.4                0.00
3                      8.3                  1.3                0.00
4                      9.7                  1.2                0.00
...                    ...                  ...                 ...
49022                  5.5                  2.2                6.75
49023                  5.6                  2.9                6.64
49024                  6.7                  2.6                7.18
49025                  6.6                  2.7                7.02
49026                  6.0                  3.3                8.39

       WEC: max. Rotation  WEC: min. Rotation  WEC: ava. Power  ...  \
0                    0.02                0.00                0  ...
1                    0.01                0.00                0  ...
2                    0.04                0.00                0  ...
3                    0.08                0.00                0  ...
4                    0.05                0.00                0  ...
...                   ...                 ...              ...  ...
49022                7.40                6.01              147  ...
49023                7.06                6.33              128  ...
49024                8.83                6.22              163  ...
49025                7.94                6.20              160  ...
49026                9.48                7.14              284  ...

       Rectifier cabinet temp.  Yaw inverter cabinet temp.  \
0                           24                           20
1                           24                           20
2                           24                           20
3                           23                           21
4                           23                           21
...                        ...                          ...
49022                       33                           23
49023                       34                           23
49024                       34                           23
49025                       33                           23
49026                       33                           22

       Fan inverter cabinet temp.  Ambient temp.  Tower temp.  \
0                              25             12           14
1                              25             12           14
2                              25             12           14
3                              25             12           14
4                              25             12           14
...                           ...            ...          ...
49022                          28              9           17
49023                          28              9           17
49024                          28              9           18
49025                          28              9           17
```

```
49026                          28              9           17

       Control cabinet temp.  Transformer temp.  RTU: ava. Setpoint 1  \
0                         24                 34                  2501
1                         24                 34                  2501
2                         24                 34                  2501
3                         24                 34                  2501
4                         23                 34                  2501
...                      ...                ...                   ...
49022                     27                 35                  3050
49023                     27                 35                  3050
49024                     27                 34                  3050
49025                     27                 34                  3050
49026                     27                 34                  3050

       Inverter averages  Inverter std dev
0              25.272728         1.103713
1              25.272728         1.103713
2              25.272728         1.103713
3              25.272728         1.103713
4              25.272728         1.103713
...                  ...              ...
49022          24.454546         3.474583
49023          24.454546         3.445683
49024          24.363636         3.413876
49025          24.000000         3.376389
49026          23.818182         3.250175

[49027 rows x 66 columns]
```

```
status_df = pd.read_csv('status_data.csv')
status_df['DateTime'] = pd.to_datetime(status_df['DateTime'])
# status_df.set_index('DateTime', inplace=True)

status_df
```

```
                DateTime  Main Status  Sub Status Full Status  \
0    2014-04-24 12:37:00            0           0        0:00
1    2014-04-25 19:27:00           71         104    71 : 104
2    2014-04-26 09:30:00            8           0        8:00
3    2014-04-26 10:05:00            8           0        8:00
4    2014-04-26 10:05:00            8           0        8:00
...                  ...          ...         ...         ...
1844 2015-04-27 07:26:00            0           0        0:00
1845 2015-04-28 22:14:00           26         373    26 : 373
1846 2015-04-28 22:14:00            0           2        0:02
1847 2015-04-28 22:17:00            0           1        0:01
1848 2015-04-28 22:18:00            0           0        0:00

                                          Status Text  T  Service  FaultMsg  \
0                                Turbine in operation  1    False     False
1     Insulation monitoring : Insulation fault Phase U2  6    False      True
2                                         Maintenance  6     True     False
3                                         Maintenance  6    False     False
4                                         Maintenance  6     True     False
```

```
...                                                  ... ..      ...       ...
1844                               Turbine in operation  1    False     False
1845  Malfunction fan-inverter : Other control board...  6    False     False
1846                                Turbine operational  1    False     False
1847                                   Turbine starting  1    False     False
1848                               Turbine in operation  1    False     False

      Value0
0        7.4
1       20.5
2       17.1
3        8.7
4       10.6
...      ...
1844     7.0
1845     8.1
1846     9.5
1847    11.1
1848    11.5

[1849 rows x 9 columns]
```

```
fault_df = pd.read_csv('fault_data.csv')
fault_df['DateTime'] = pd.to_datetime(fault_df['DateTime'])
# fault_df.set_index('DateTime', inplace=True)

fault_df
```

```
               DateTime          Time Fault
0   2014-05-14 14:39:44  1.400096e+09    GF
1   2014-05-14 14:50:24  1.400097e+09    GF
2   2014-05-14 14:58:56  1.400098e+09    GF
3   2014-05-14 15:09:36  1.400098e+09    GF
4   2014-05-14 15:20:16  1.400099e+09    GF
..                  ...           ...   ...
548 2015-01-14 23:00:48  1.421298e+09    AF
549 2015-01-14 23:09:20  1.421299e+09    AF
550 2015-01-15 22:50:08  1.421384e+09    AF
551 2015-01-15 23:00:48  1.421384e+09    AF
552 2015-01-15 23:09:20  1.421385e+09    AF

[553 rows x 3 columns]
```

In the fault data, there are 5 types of faults, or fault modes:

- gf: generator heating fault
- mf: mains failure fault
- ff: feeding fault
- af: timeout warnmessage : malfunction aircooling
- ef: excitation error : overvoltage DC-link

```
fault_df.Fault.unique()
```

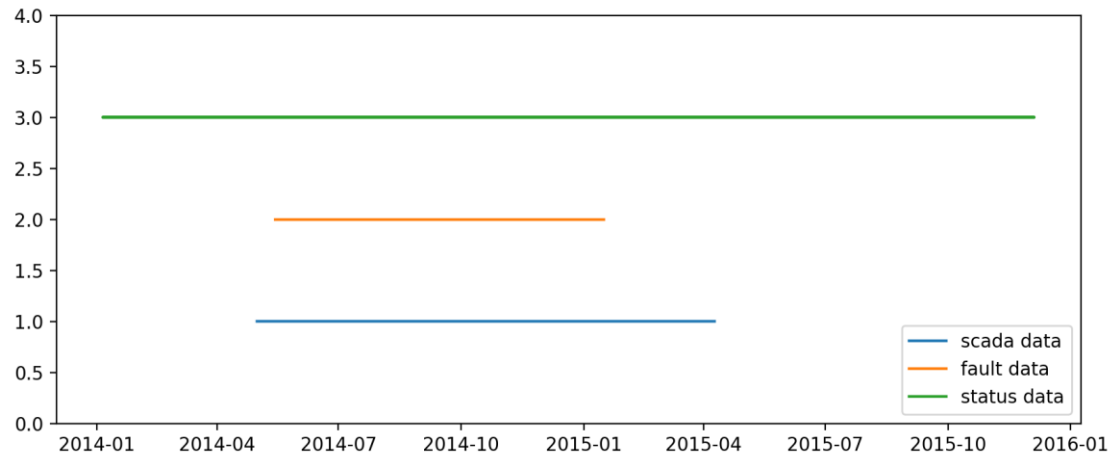
```
array(['GF', 'MF', 'FF', 'AF', 'EF'], dtype=object)
```

A2. Time series analysis

The 3 data have different time spans. The status data has the longest record timespan from January 2014 to December 2015. The shortest is SCADA data from April 2014 to April 2015. Therefore, when seeing the SCADA records, we can refer to status and fault data to see what happens on the turbine at certain timestamps.

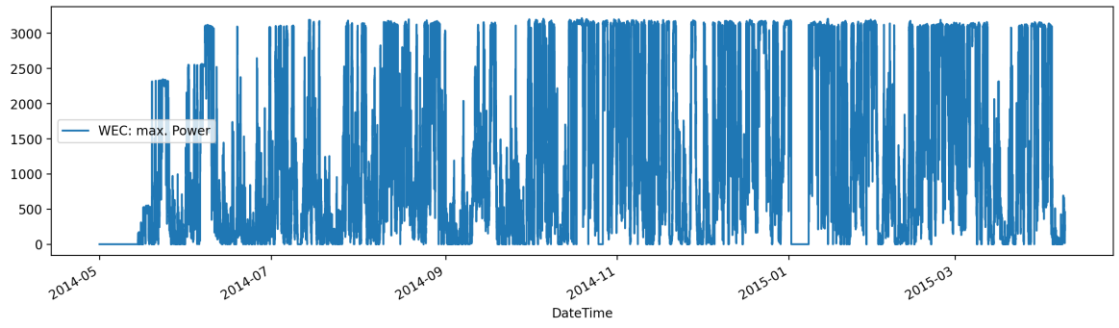
```
# Plot time span of all data
t_scada = scada_df.DateTime
t_fault = fault_df.DateTime
t_status = status_df.DateTime

plt.figure(figsize=(10,4))
plt.plot(t_scada, np.full(len(scada_df), 1), label='scada data')
plt.plot(t_fault, np.full(len(fault_df), 2), label='fault data')
plt.plot(t_status, np.full(len(status_df), 3), label='status data')
plt.legend(loc='lower right')
plt.ylim(0,4)
```

```
(0.0, 4.0)
```



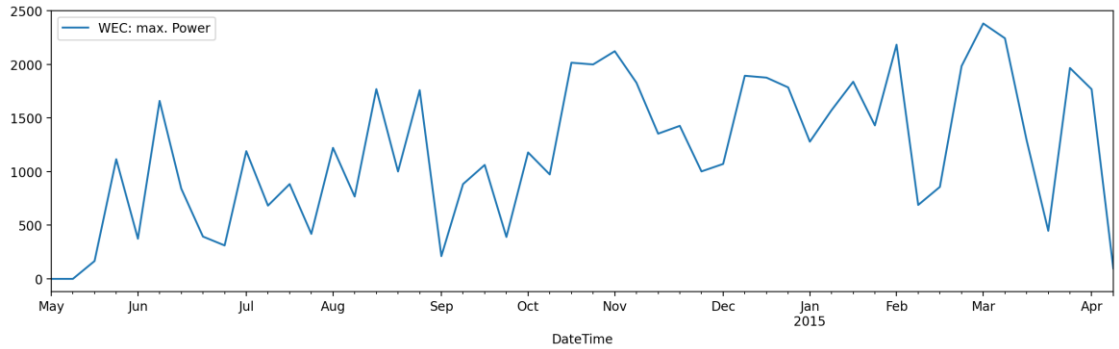

```
# Plot of max power from SCADA data
scada_df.plot(x='DateTime', y='WEC: max. Power', figsize=(15,4))
```

```
<AxesSubplot:xlabel='DateTime'>
```



```
# Plot of max power on weekly resampled data
y = 'WEC: max. Power'
scada_df.resample('W', on='DateTime').mean().plot(y=y, figsize=(15,4))
```

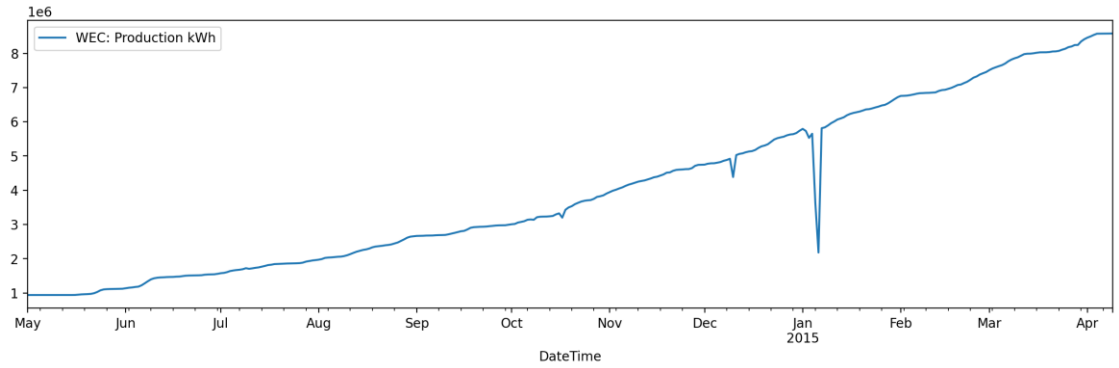
```
<AxesSubplot:xlabel='DateTime'>
```



There were times when power dropped, for example in October 2014, December 2014, and the most significant in January 2015.

```
# Plot of power production on monthly resampled data
y = 'WEC: Production kWh'
scada_df.resample('D', on='DateTime').mean().plot(y=y, figsize=(15,4))
```

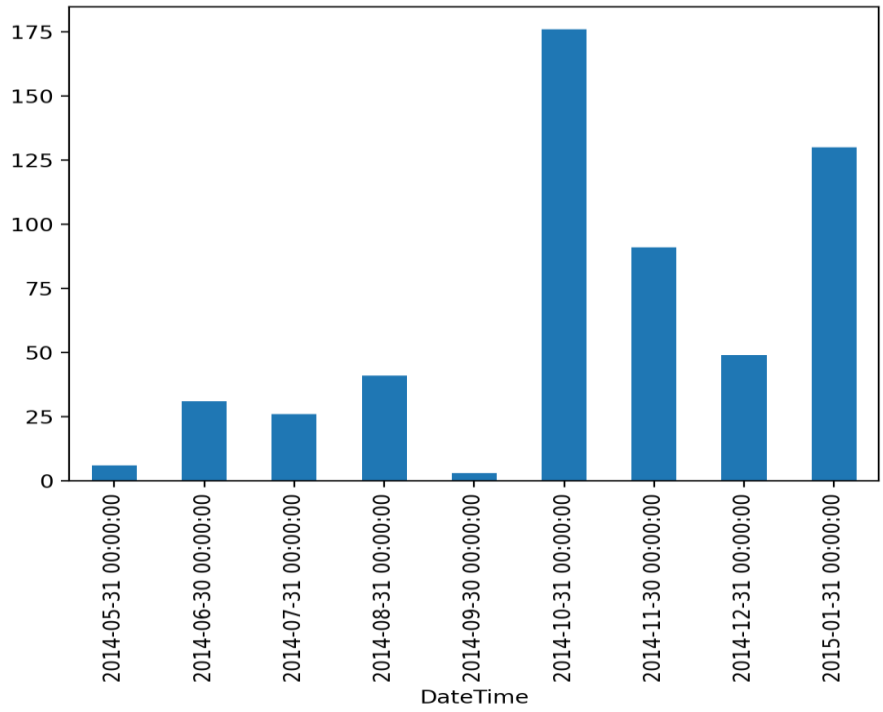
```
<AxesSubplot:xlabel='DateTime'>
```



The number of wind turbine faults significantly increases on October 2014.

```
# Plot of number of faults on monthly resampled data
fault_df.resample('M', on='DateTime').Fault.count().plot.bar()
```

```
<AxesSubplot:xlabel='DateTime'>
```



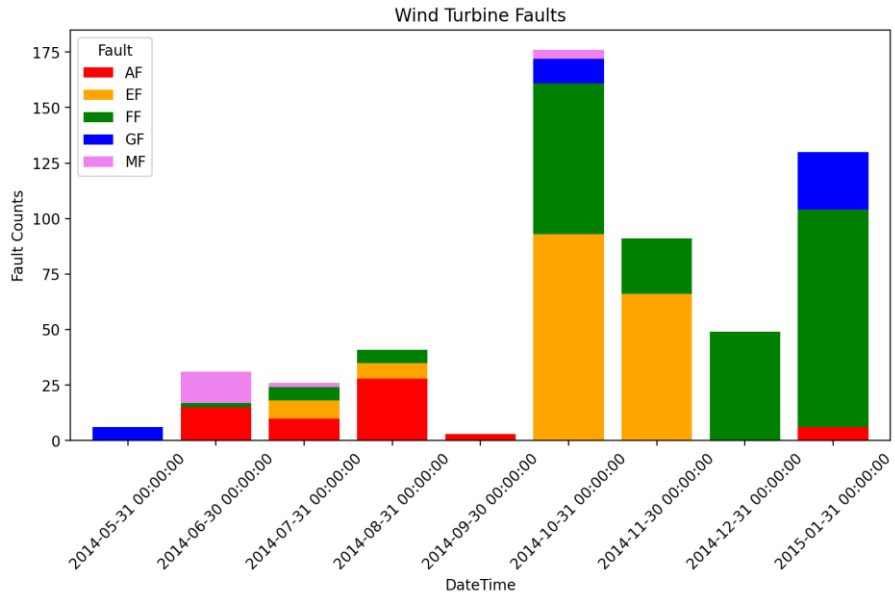

```
fault_df.resample('M', on='DateTime').Fault.value_counts()
```

```
DateTime    Fault
2014-05-31  GF        6
2014-06-30  AF       15
            MF       14
            FF        2
2014-07-31  AF       10
            EF        8
            FF        6
            MF        2
2014-08-31  AF       28
            EF        7
            FF        6
2014-09-30  AF        3
2014-10-31  EF       93
            FF       68
            GF       11
            MF        4
2014-11-30  EF       66
            FF       25
2014-12-31  FF       49
2015-01-31  FF       98
            GF       26
            AF        6
Name: Fault, dtype: int64
```

Let's plot the faults grouped by its fault modes. There are lots of EF events in October and November 2014, and lots of FF events from October 2014 - January 2015.

```
def line_format(label):
    """
    Convert time label to the format of pandas line plot
    """
    month = label.month_name()[:3]
    if month == 'Jan':
        month += f'\n{label.year}'
    return month

c = ['red', 'orange', 'green', 'blue', 'violet']
fault_df.resample('M', on='DateTime').Fault.value_counts().unstack().plot.bar(sta
cked=True, width=0.8, figsize=(10,5), color=c, rot=45,
                                                                        tit
le='Wind Turbine Faults', ylabel='Fault Counts')
```

```
<AxesSubplot:title={'center':'Wind Turbine Faults'}, xlabel='DateTime', ylabel='F
ault Counts'>
```

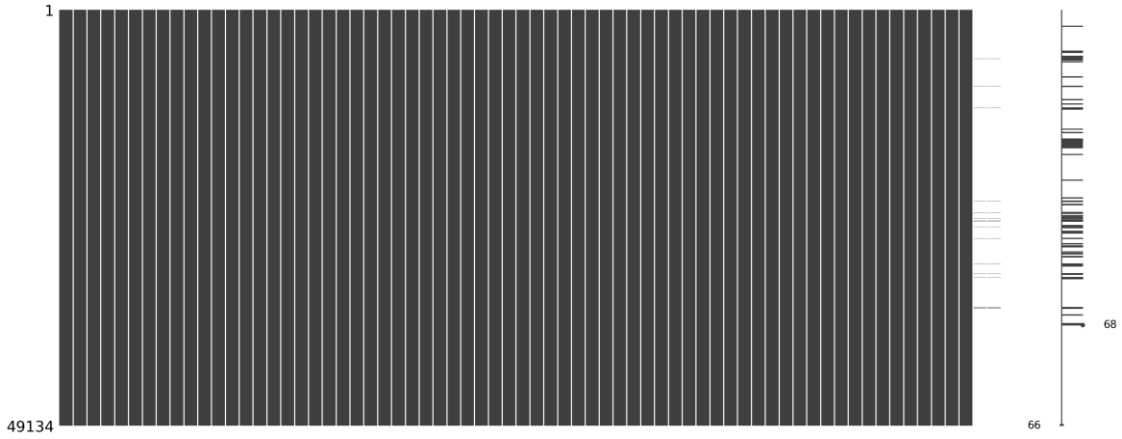


A3. Combine SCADA and faults data

We combine SCADA and fault data to pair each measurements with associated faults.

```
# Combine scada and fault data
df_combine = scada_df.merge(fault_df, on='Time', how='outer')
msno.matrix(df_combine)
```

```
<AxesSubplot:>
```



There are lots of NaNs, or unmatched SCADA timestamps with fault timestamps, simply because there are no faults happen at certain time. For these NaNs, we will replace with "NF".

NF is No Fault (normal condition)

```
# Replace records that has no fault label (NaN) as 'NF' (no fault)
df_combine['Fault'] = df_combine['Fault'].replace(np.nan, 'NF')

df_combine
```

```
               DateTime_x        Time  Error  WEC: ava. windspeed  \
0     2014-05-01 00:00:00  1398920448      0                  6.9
1     2014-05-01 00:09:00  1398920960      0                  5.3
2     2014-05-01 00:20:00  1398921600      0                  5.0
3     2014-05-01 00:30:00  1398922240      0                  4.4
4     2014-05-01 00:39:00  1398922752      0                  5.7
...                   ...         ...    ...                  ...
49129 2015-04-08 23:20:00  1428553216      0                  3.9
49130 2015-04-08 23:30:00  1428553856      0                  3.9
49131 2015-04-08 23:39:00  1428554368      0                  4.2
49132 2015-04-08 23:50:00  1428555008      0                  4.1
49133 2015-04-09 00:00:00  1428555648      0                  4.8

       WEC: max. windspeed  WEC: min. windspeed  WEC: ava. Rotation  \
0                      9.4                  2.9                0.00
1                      8.9                  1.6                0.00
2                      9.5                  1.4                0.00
3                      8.3                  1.3                0.00
4                      9.7                  1.2                0.00
...                    ...                  ...                 ...
49129                  5.5                  2.2                6.75
49130                  5.6                  2.9                6.64
49131                  6.7                  2.6                7.18
49132                  6.6                  2.7                7.02
49133                  6.0                  3.3                8.39

       WEC: max. Rotation  WEC: min. Rotation  WEC: ava. Power  ...  \
0                    0.02                0.00                0  ...
1                    0.01                0.00                0  ...
```

```
2                   0.04               0.00                0  ...
3                   0.08               0.00                0  ...
4                   0.05               0.00                0  ...
...                  ...                ...              ...  ...
49129               7.40               6.01              147  ...
49130               7.06               6.33              128  ...
49131               8.83               6.22              163  ...
49132               7.94               6.20              160  ...
49133               9.48               7.14              284  ...

       Fan inverter cabinet temp.  Ambient temp.  Tower temp.  \
0                              25             12           14
1                              25             12           14
2                              25             12           14
3                              25             12           14
4                              25             12           14
...                           ...            ...          ...
49129                          28              9           17
49130                          28              9           17
49131                          28              9           18
49132                          28              9           17
49133                          28              9           17

       Control cabinet temp.  Transformer temp.  RTU: ava. Setpoint 1  \
0                         24                 34                  2501
1                         24                 34                  2501
2                         24                 34                  2501
3                         24                 34                  2501
4                         23                 34                  2501
...                      ...                ...                   ...
49129                     27                 35                  3050
49130                     27                 35                  3050
49131                     27                 34                  3050
49132                     27                 34                  3050
49133                     27                 34                  3050

       Inverter averages  Inverter std dev  DateTime_y  Fault
0              25.272728          1.103713         NaT     NF
1              25.272728          1.103713         NaT     NF
2              25.272728          1.103713         NaT     NF
3              25.272728          1.103713         NaT     NF
4              25.272728          1.103713         NaT     NF
...                  ...               ...         ...    ...
49129          24.454546          3.474583         NaT     NF
49130          24.454546          3.445683         NaT     NF
49131          24.363636          3.413876         NaT     NF
49132          24.000000          3.376389         NaT     NF
49133          23.818182          3.250175         NaT     NF

[49134 rows x 68 columns]
```

A4. Exploratory Data Analysis

Print the averages of SCADA values grouped by fault modes.

```
# Suppress scientific notations
pd.set_option('display.float_format', lambda x: '%.3f' % x)

# Group by fault and take average
df_summary = df_combine.groupby('Fault').mean().T
df_summary.tail(20)
```

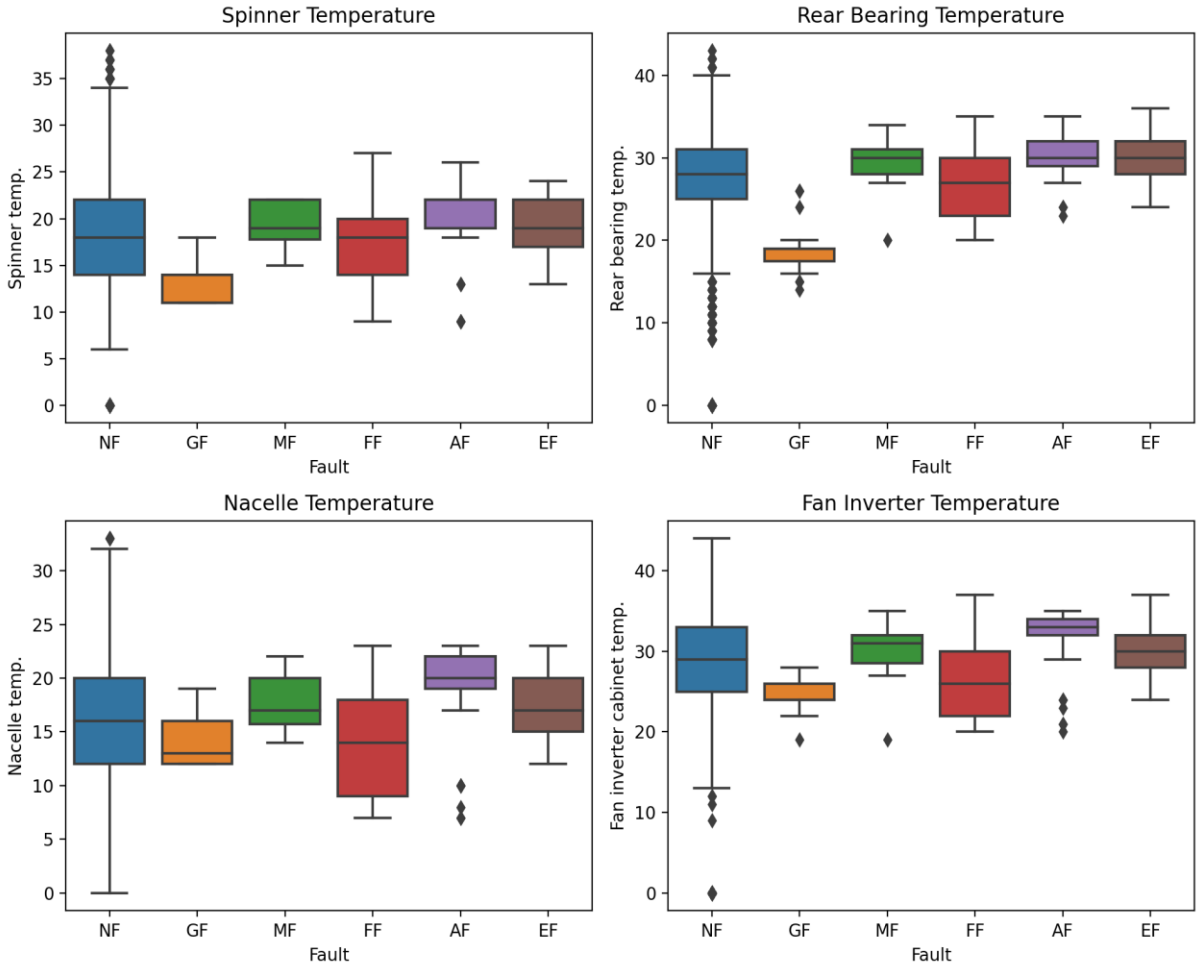
```
Fault                           AF       EF       FF       GF       MF  \
Blade C temp.              166.032  165.925  166.236  166.512  166.050
Rotor temp. 1               55.565  100.368   66.071   34.140   52.300
Rotor temp. 2               55.677   99.920   65.925   34.860   52.500
Stator temp. 1              68.903  101.592   70.425   42.837   66.350
Stator temp. 2              68.323  100.454   69.720   42.465   65.850
Nacelle ambient temp. 1     15.629   14.500   11.016   12.116   14.800
Nacelle ambient temp. 2     15.629   14.299   10.917   11.930   14.650
Nacelle temp.               19.290   17.368   14.051   14.093   17.850
Nacelle cabinet temp.       22.919   20.736   17.831   19.395   21.000
Main carrier temp.          20.548   19.632   15.866   13.279   19.100
Rectifier cabinet temp.     32.629   30.385   26.697   30.767   30.000
Yaw inverter cabinet temp.  27.694   27.322   22.614   21.140   25.900
Fan inverter cabinet temp.  32.032   29.948   26.323   24.860   30.050
Ambient temp.               16.758   14.937   11.378   12.581   16.750
Tower temp.                 28.129   28.891   21.638   16.349   27.150
Control cabinet temp.       36.516   38.833   31.551   24.884   35.200
Transformer temp.           46.903   64.121   51.063   30.233   45.250
RTU: ava. Setpoint 1      3042.194 3049.425 3047.480 2973.395 2940.200
Inverter averages           31.812   34.222   30.144   22.973   30.345
Inverter std dev             1.439    1.617    1.639    1.141    1.456

Fault                           NF
Blade C temp.              165.640
Rotor temp. 1               52.517
Rotor temp. 2               52.614
Stator temp. 1              60.702
Stator temp. 2              60.204
Nacelle ambient temp. 1     12.515
Nacelle ambient temp. 2     12.435
Nacelle temp.               16.339
Nacelle cabinet temp.       19.852
Main carrier temp.          16.551
Rectifier cabinet temp.     30.354
Yaw inverter cabinet temp.  24.325
Fan inverter cabinet temp.  28.815
Ambient temp.               13.386
Tower temp.                 23.115
Control cabinet temp.       31.758
Transformer temp.           43.933
RTU: ava. Setpoint 1      2988.201
Inverter averages           27.807
Inverter std dev             1.858
```

Seeing the averages above, we could identify the anomalous behavior of Fault Modes:

- WF has lower ava, min, max active reactive power than No Fault (NF)
- EF has higher ava, min, max active reactive power than No Fault (NF)
- GF has ZERO ava, min, max active reactive power
- FF and MF have higher nacelle cable twisting than NF
- AF and GF have negative nacelle cable twisting
- AF and MF have lower production
- All faults have higher blade angle, the highest is FF
- GF in general has the lowest temperature in ALL components (cabinet temp, T spinner, T front bearing, ..., T transformer)
- While other faults (FF, AF, MF, EF) have higher temperature
- EF temperature is highest in cabinet, pitch, rotor, stator, ambient, control, tower, and transformer
- AF temperature is highest in spinner, front bearing, rare bearing, nacelle, main carrier, rectifier, yaw, and fan inverter

The boxplots of temperatures (at spinner, bearing, nacelle, and fan inverter) shows that during GF, the temperatures are anomalously lower than normal condition. However, temperatures are higher than normal during AF and EF.

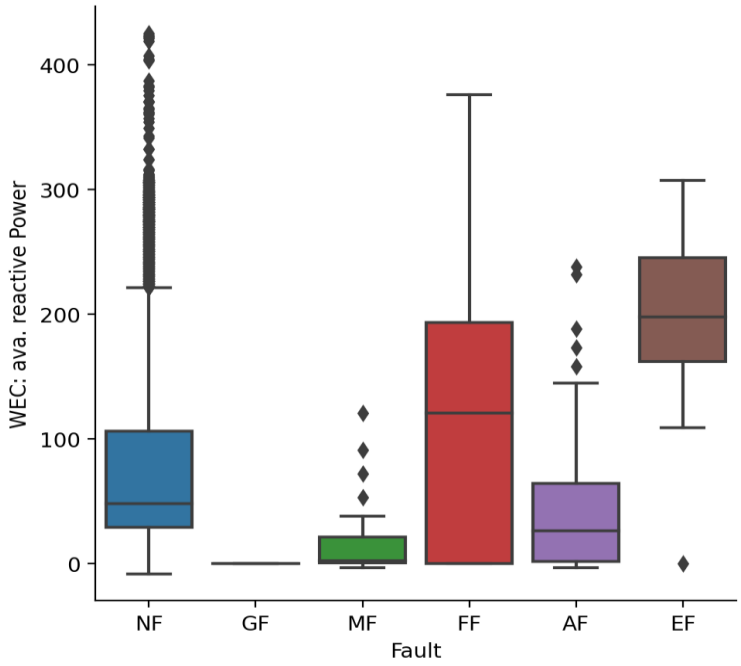
```
# Boxplots of temperature
f, axes = plt.subplots(nrows=2, ncols=2,figsize=(10,8))

sns.boxplot(x='Fault', y='Spinner temp.', data=df_combine, ax=axes[0][0])
axes[0][0].set_title('Spinner Temperature')
sns.boxplot(x='Fault', y='Rear bearing temp.', data=df_combine, ax=axes[0][1])
axes[0][1].set_title('Rear Bearing Temperature')
sns.boxplot(x='Fault', y='Nacelle temp.', data=df_combine, ax=axes[1][0])
axes[1][0].set_title('Nacelle Temperature')
sns.boxplot(x='Fault', y='Fan inverter cabinet temp.', data=df_combine, ax=axes[1
][1])
axes[1][1].set_title('Fan Inverter Temperature')

plt.tight_layout()
```



Boxplot of reactive power (power from the generator?) shows the power during EF is anomalously high, while the power during MF is lower than normal condition.

```
sns.catplot(data=df_combine, x='Fault', y='WEC: ava. reactive Power', kind='box')
```

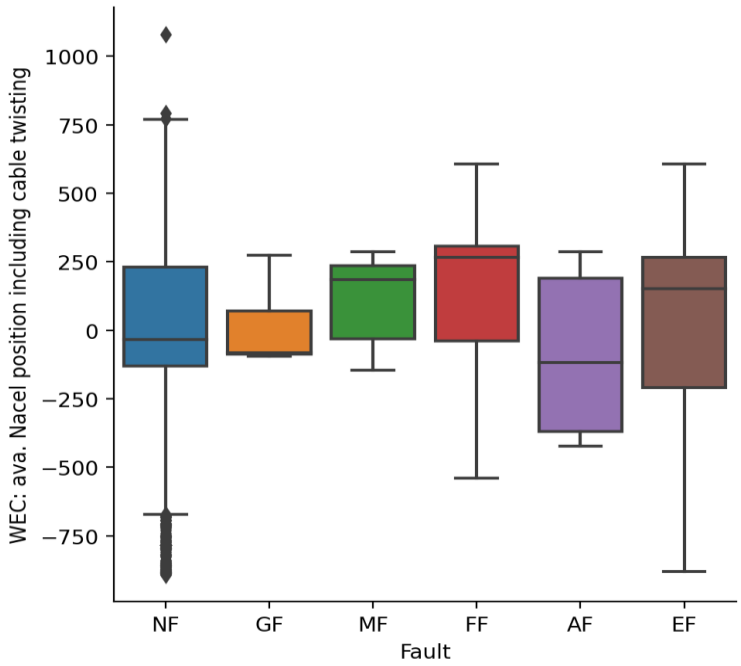
```
<seaborn.axisgrid.FacetGrid at 0x20b02b18400>
```



The boxplot of nacelle position and cable twisting shows that during AF, nacelle position is negative (up to -500) while during MF, FF, and EF, nacelle position is positive (up to +500).

```
sns.catplot(data=df_combine, x='Fault', y='WEC: ava. Nacel position including cab
le twisting', kind='box')
```

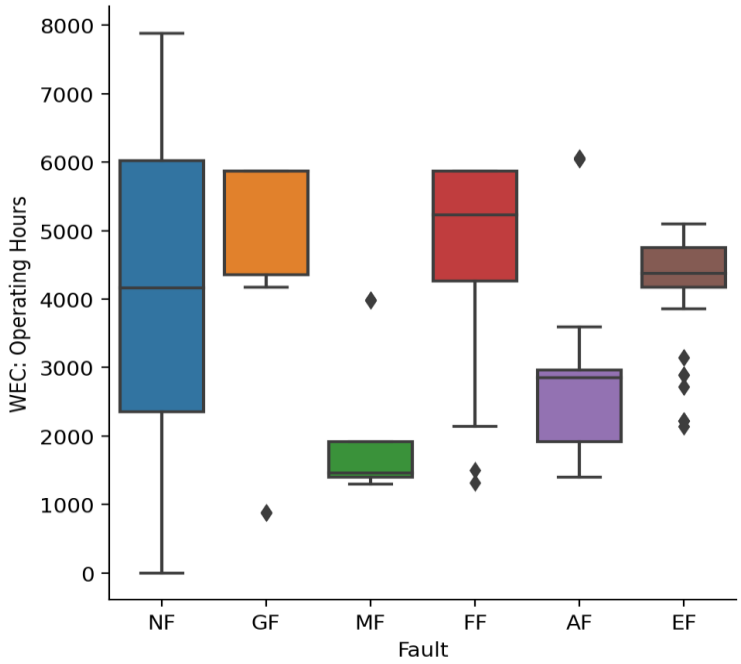
```
<seaborn.axisgrid.FacetGrid at 0x20b035df910>
```



The boxplot of operating hours shows that during MF and AF, the operating hours are shorter than normal condition. However, during FF, the operating hours are longer than normally are.

```
sns.catplot(data=df_combine, x='Fault', y='WEC: Operating Hours', kind='box')
```

```
<seaborn.axisgrid.FacetGrid at 0x20b035df700>
```



A5. Data preparation for ML

There are far more records of NF (normal condition) than faulty records - imbalanced dataset. We will sample the No Fault dataframe and pick only 300 records.

```
df_combine.Fault.value_counts()
```

```
NF    48581
FF      254
EF      174
AF       62
GF       43
MF       20
Name: Fault, dtype: int64
```

```
# No Fault mode data
df_nf = df_combine[df_combine.Fault=='NF'].sample(300, random_state=42)

df_nf
```

```
               DateTime_x        Time  Error  WEC: ava. windspeed  \
7340  2014-06-21 19:29:00  1403396992      0                3.500
949   2014-05-07 16:39:00  1399498752      0                5.300
37369 2015-01-17 00:00:00  1421474432      0                9.200
```

```
15200 2014-08-15 07:30:00  1408105856      0               7.200
42798 2015-02-23 22:30:00  1424752256      0              14.900
...                   ...         ...    ...                 ...
6298  2014-06-14 13:50:00  1402771840      0               2.200
22374 2014-10-04 01:38:00  1412404736      0               7.100
39742 2015-02-02 16:41:00  1422916864      0               2.100
34504 2014-12-28 02:40:00  1419756032      0               7.800
10632 2014-07-14 15:09:00  1405368576      0              11.300

       WEC: max. windspeed  WEC: min. windspeed  WEC: ava. Rotation  \
7340                 4.600                2.400               6.370
949                 15.000                3.000               0.000
37369               10.800                8.100              13.480
15200                9.600                5.000              10.450
42798               20.900               11.200              14.690
...                    ...                  ...                 ...
6298                 4.300                0.400               4.600
22374                9.000                5.500              10.120
39742                2.600                1.700               5.050
34504                9.500                5.200              10.670
10632               15.200                6.100              13.560

       WEC: max. Rotation  WEC: min. Rotation  WEC: ava. Power  ...  \
7340                6.640               6.160               87  ...
949                 0.000               0.000                0  ...
37369              13.950              12.920             2078  ...
15200              11.620               9.700              998  ...
42798              15.540              14.130             3060  ...
...                   ...                 ...              ...  ...
6298                5.120               4.150                1  ...
22374              10.770               9.500              875  ...
39742               5.320               4.940                6  ...
34504              11.720               9.360             1138  ...
10632              14.870              10.740             2638  ...

       Fan inverter cabinet temp.  Ambient temp.  Tower temp.  \
7340                           40             20           26
949                            26             16           15
37369                          20              3           18
15200                          30             21           30
42798                          22              6           28
...                           ...            ...          ...
6298                           40             24           25
22374                          27             13           26
39742                          21              4            8
34504                          22              4           19
10632                          38             23           36

       Control cabinet temp.  Transformer temp.  RTU: ava. Setpoint 1  \
7340                      36                 45                  3050
949                       24                 33                  2501
37369                     25                 40                  3050
15200                     37                 48                  3050
42798                     39                 67                  3050
```

```
...                     ...                ...                   ...
6298                     35                 45                  3050
22374                    34                 43                  3050
39742                    17                 27                  3050
34504                    27                 38                  3050
10632                    44                 64                  3050

       Inverter averages  Inverter std dev  DateTime_y  Fault
7340              34.273             3.228         NaT     NF
949               24.455             0.934         NaT     NF
37369             19.091             1.446         NaT     NF
15200             30.545             1.036         NaT     NF
42798             28.273             1.902         NaT     NF
...                  ...               ...         ...    ...
6298              35.000             1.414         NaT     NF
22374             28.091             1.136         NaT     NF
39742             19.909             1.044         NaT     NF
34504             19.818             0.603         NaT     NF
10632             35.909             1.973         NaT     NF

[300 rows x 68 columns]
```

```
# With fault mode data
df_f = df_combine[df_combine.Fault!='NF']

df_f
```

```
              DateTime_x        Time  Error  WEC: ava. windspeed  \
1945  2014-05-14 14:39:00  1400096384      0                5.700
1946  2014-05-14 14:50:00  1400097024      0                6.400
1947  2014-05-14 14:58:00  1400097536      0                5.600
1948  2014-05-14 15:09:00  1400098176      0                5.300
1949  2014-05-14 15:20:00  1400098816      0                5.100
...                   ...         ...    ...                  ...
37075 2015-01-14 23:00:00  1421298048      0               13.500
37076 2015-01-14 23:09:00  1421298560      0               15.500
37218 2015-01-15 22:50:00  1421383808      0                8.400
37219 2015-01-15 23:00:00  1421384448      0                8.100
37220 2015-01-15 23:09:00  1421384960      0                7.900

       WEC: max. windspeed  WEC: min. windspeed  WEC: ava. Rotation  \
1945                 7.400                4.100               9.980
1946                 8.600                2.700               4.120
1947                 7.500                4.000               9.990
1948                 6.600                4.100               9.980
1949                 7.200                3.800               9.990
...                    ...                  ...                 ...
37075               22.000                7.700              14.690
37076               25.700                1.900               2.530
37218                9.700                6.800              12.790
37219                9.800                6.000              11.710
37220               11.600                1.900               1.400

       WEC: max. Rotation  WEC: min. Rotation  WEC: ava. Power  ...  \
1945               10.230               9.720                0  ...
```

```
1946              10.710              0.700               0  ...
1947              10.200              9.800               0  ...
1948              10.170              9.790               0  ...
1949              10.170              9.800               0  ...
...                  ...                ...             ...  ...
37075             15.190             13.830            2736  ...
37076             14.670              0.000              20  ...
37218             13.300             11.640            1667  ...
37219             13.270              9.990            1311  ...
37220             11.180              0.000              14  ...

       Fan inverter cabinet temp.  Ambient temp.  Tower temp.  \
1945                           27             17           26
1946                           27             17           26
1947                           27             17           26
1948                           27             18           26
1949                           28             18           26
...                           ...            ...          ...
37075                          24              8           29
37076                          23              7           22
37218                          21              5           21
37219                          20              5           21
37220                          21              5           17

       Control cabinet temp.  Transformer temp.  RTU: ava. Setpoint 1  \
1945                      35                 34                  2501
1946                      35                 34                  2501
1947                      35                 34                  2501
1948                      35                 34                  2501
1949                      35                 34                  2501
...                      ...                ...                   ...
37075                     35                 57                  3050
37076                     35                 57                  3050
37218                     28                 42                  3050
37219                     28                 42                  3050
37220                     28                 42                  3050

       Inverter averages  Inverter std dev          DateTime_y  Fault
1945              25.818            0.603 2014-05-14 14:39:44     GF
1946              26.091            0.944 2014-05-14 14:50:24     GF
1947              26.455            0.820 2014-05-14 14:58:56     GF
1948              26.182            0.874 2014-05-14 15:09:36     GF
1949              26.182            0.982 2014-05-14 15:20:16     GF
...                  ...              ...                 ...    ...
37075             25.909            1.814 2015-01-14 23:00:48     AF
37076             28.182            1.834 2015-01-14 23:09:20     AF
37218             20.909            1.300 2015-01-15 22:50:08     AF
37219             20.909            1.300 2015-01-15 23:00:48     AF
37220             22.909            1.300 2015-01-15 23:09:20     AF

[553 rows x 68 columns]
```

```
# Combine no fault and faulty dataframes
```

```
df_combine = pd.concat((df_nf, df_f), axis=0).reset_index(drop=True)

df_combine
```

```
             DateTime_x        Time  Error  WEC: ava. windspeed  \
0   2014-06-21 19:29:00  1403396992      0                3.500
1   2014-05-07 16:39:00  1399498752      0                5.300
2   2015-01-17 00:00:00  1421474432      0                9.200
3   2014-08-15 07:30:00  1408105856      0                7.200
4   2015-02-23 22:30:00  1424752256      0               14.900
..                  ...         ...    ...                  ...
848 2015-01-14 23:00:00  1421298048      0               13.500
849 2015-01-14 23:09:00  1421298560      0               15.500
850 2015-01-15 22:50:00  1421383808      0                8.400
851 2015-01-15 23:00:00  1421384448      0                8.100
852 2015-01-15 23:09:00  1421384960      0                7.900

     WEC: max. windspeed  WEC: min. windspeed  WEC: ava. Rotation  \
0                  4.600                2.400               6.370
1                 15.000                3.000               0.000
2                 10.800                8.100              13.480
3                  9.600                5.000              10.450
4                 20.900               11.200              14.690
..                   ...                  ...                 ...
848               22.000                7.700              14.690
849               25.700                1.900               2.530
850                9.700                6.800              12.790
851                9.800                6.000              11.710
852               11.600                1.900               1.400

     WEC: max. Rotation  WEC: min. Rotation  WEC: ava. Power  ...  \
0                 6.640               6.160               87  ...
1                 0.000               0.000                0  ...
2                13.950              12.920             2078  ...
3                11.620               9.700              998  ...
4                15.540              14.130             3060  ...
..                  ...                 ...              ...  ...
848              15.190              13.830             2736  ...
849              14.670               0.000               20  ...
850              13.300              11.640             1667  ...
851              13.270               9.990             1311  ...
852              11.180               0.000               14  ...

     Fan inverter cabinet temp.  Ambient temp.  Tower temp.  \
0                            40             20           26
1                            26             16           15
2                            20              3           18
3                            30             21           30
4                            22              6           28
..                          ...            ...          ...
848                          24              8           29
849                          23              7           22
850                          21              5           21
851                          20              5           21
852                          21              5           17
```

```
     Control cabinet temp.  Transformer temp.  RTU: ava. Setpoint 1  \
0                       36                 45                  3050
1                       24                 33                  2501
2                       25                 40                  3050
3                       37                 48                  3050
4                       39                 67                  3050
..                     ...                ...                   ...
848                     35                 57                  3050
849                     35                 57                  3050
850                     28                 42                  3050
851                     28                 42                  3050
852                     28                 42                  3050

     Inverter averages  Inverter std dev          DateTime_y  Fault
0               34.273             3.228                 NaT     NF
1               24.455             0.934                 NaT     NF
2               19.091             1.446                 NaT     NF
3               30.545             1.036                 NaT     NF
4               28.273             1.902                 NaT     NF
..                 ...               ...                 ...    ...
848             25.909             1.814 2015-01-14 23:00:48     AF
849             28.182             1.834 2015-01-14 23:09:20     AF
850             20.909             1.300 2015-01-15 22:50:08     AF
851             20.909             1.300 2015-01-15 23:00:48     AF
852             22.909             1.300 2015-01-15 23:09:20     AF

[853 rows x 68 columns]
```

Preparing for the training dataset, we **drop irrelevant features**. First we drop datetime, time, and error columns. Next, features that "de facto" are output of wind turbine, such as power from wind, operating hours, and kWh production, are dropped. Also, climatic variable such as wind speed are not useful.

```
# Feature selection
train_df = df_combine.loc[:, ['WEC: ava. windspeed',
                              'WEC: ava. Rotation',
                              'WEC: ava. Power',
                              'WEC: ava. reactive Power',
                              'WEC: ava. blade angle A',
                              'Spinner temp.',
                              'Front bearing temp.',
                              'Rear bearing temp.',
                              'Pitch cabinet blade A temp.',
                              'Pitch cabinet blade B temp.',
                              'Pitch cabinet blade C temp.',
                              'Rotor temp. 1',
                              'Rotor temp. 2',
                              'Stator temp. 1',
                              'Stator temp. 2',
                              'Nacelle ambient temp. 1',
                              'Nacelle ambient temp. 2',
                              'Nacelle temp.',
                              'Nacelle cabinet temp.',
```

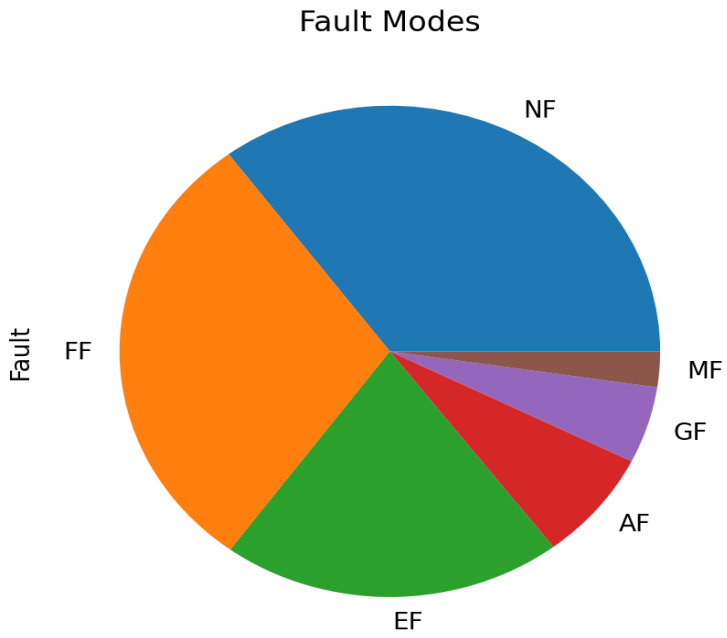
```
                            'Main carrier temp.',
                            'Rectifier cabinet temp.',
                            'Yaw inverter cabinet temp.',
                            'Fan inverter cabinet temp.',
                            'Ambient temp.',
                            'Tower temp.',
                            'Control cabinet temp.',
                            'Transformer temp.',
                            'Inverter averages',
                            'Inverter std dev',
                            'Fault']]
train_df
```

```
     WEC: ava. windspeed  WEC: ava. Rotation  WEC: ava. Power  \
0                  3.500               6.370               87
1                  5.300               0.000                0
2                  9.200              13.480             2078
3                  7.200              10.450              998
4                 14.900              14.690             3060
..                   ...                 ...              ...
848               13.500              14.690             2736
849               15.500               2.530               20
850                8.400              12.790             1667
851                8.100              11.710             1311
852                7.900               1.400               14

     WEC: ava. reactive Power  WEC: ava. blade angle A  Spinner temp.  \
0                          32                   1.000             28
1                           0                  91.930             15
2                         173                   1.000              8
3                          64                   1.000             17
4                         296                  12.780             12
..                        ...                     ...            ...
848                       238                   8.260             13
849                         4                  66.850             13
850                       145                   1.000              9
851                       111                   1.000              9
852                         3                  66.750              9

     Front bearing temp.  Rear bearing temp.  Pitch cabinet blade A temp.  \
0                     30                  34                           45
1                     12                  12                           28
2                     12                  23                           22
3                     22                  26                           31
4                     18                  26                           31
..                   ...                 ...                          ...
848                   20                  28                           32
849                   20                  27                           32
850                   14                  24                           23
851                   14                  24                           23
852                   14                  23                           24

     Pitch cabinet blade B temp.  ...  Rectifier cabinet temp.  \
0                             45  ...                       39
1                             27  ...                       25
```

```
2                            22  ...                      21
3                            31  ...                      27
4                            31  ...                      23
..                          ...  ...                     ...
848                          32  ...                      27
849                          32  ...                      27
850                          24  ...                      23
851                          24  ...                      23
852                          24  ...                      24

     Yaw inverter cabinet temp.  Fan inverter cabinet temp.  Ambient temp.  \
0                            36                          40             20
1                            22                          26             16
2                            16                          20              3
3                            24                          30             21
4                            19                          22              6
..                          ...                         ...            ...
848                          21                          24              8
849                          21                          23              7
850                          17                          21              5
851                          16                          20              5
852                          17                          21              5

     Tower temp.  Control cabinet temp.  Transformer temp.  Inverter averages  \
0             26                      36                 45             34.273
1             15                      24                 33             24.455
2             18                      25                 40             19.091
3             30                      37                 48             30.545
4             28                      39                 67             28.273
..           ...                     ...                ...                ...
848           29                      35                 57             25.909
849           22                      35                 57             28.182
850           21                      28                 42             20.909
851           21                      28                 42             20.909
852           17                      28                 42             22.909

     Inverter std dev  Fault
0               3.228     NF
1               0.934     NF
2               1.446     NF
3               1.036     NF
4               1.902     NF
..                ...    ...
848             1.814     AF
849             1.834     AF
850             1.300     AF
851             1.300     AF
852             1.300     AF

[853 rows x 30 columns]
```

```
# Imbalanced fault modes
train_df.Fault.value_counts().plot.pie(title='Fault Modes')
```

```
<AxesSubplot:title={'center':'Fault Modes'}, ylabel='Fault'>
```



A6. Machine learning - fault modes classification

```
# Feature and target
# X = df_combine.iloc[:,3:-2]
# y = df_combine.iloc[:,-1]
X = train_df.iloc[:,:-1]
y = train_df.iloc[:,-1]

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_s
tate=42)
```

A6.1 Support Vector Classification

```
# Make pipeline of scaling, and classifier
svc_pipe = make_pipeline(StandardScaler(), SVC(random_state=42, class_weight='bal
anced'))

# Define multiple scoring metrics
scoring = {'acc': 'accuracy',
           'prec_macro': 'precision_macro',
           'rec_macro': 'recall_macro',
           'f1_macro': 'f1_macro'}

# Stratified K-Fold
stratkfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Return a dictionary of all scorings
cv_svc_scores = cross_validate(svc_pipe, X_train, y_train, cv=stratkfold, scoring
=scoring)
```

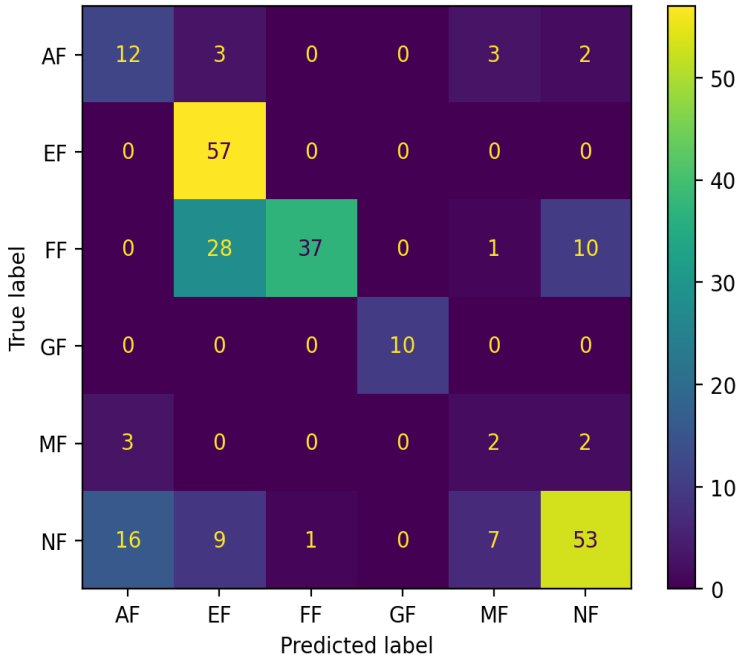
```
# Print scoring results from dictionary
for metric_name, metric_value in cv_svc_scores.items():
    mean = np.mean(metric_value)
    print(f'{metric_name}: {np.round(metric_value, 4)}, Mean: {np.round(mean, 4)}
')
```

```
fit_time: [0.008  0.008  0.0101 0.007  0.007 ], Mean: 0.008
score_time: [0.005 0.007 0.005 0.005 0.004], Mean: 0.0052
test_acc: [0.6917 0.675  0.6807 0.6891 0.6639], Mean: 0.6801
test_prec_macro: [0.6568 0.6193 0.6728 0.6675 0.6449], Mean: 0.6523
test_rec_macro: [0.7065 0.6554 0.6908 0.7304 0.7087], Mean: 0.6984
test_f1_macro: [0.6471 0.6029 0.6288 0.6525 0.6343], Mean: 0.6331
```

```
# Fit pipeline to train set
svc_pipe.fit(X_train, y_train)

# Predict on test set
y_pred = svc_pipe.predict(X_test)

# Get the parameters of the model
params = svc_pipe.get_params()

# Print the parameters
print(params)
```

```
{'memory': None, 'steps': [('standardscaler', StandardScaler()), ('svc', SVC(clas
s_weight='balanced', random_state=42))], 'verbose': False, 'standardscaler': Stan
dardScaler(), 'svc': SVC(class_weight='balanced', random_state=42), 'standardscal
er__copy': True, 'standardscaler__with_mean': True, 'standardscaler__with_std': T
rue, 'svc__C': 1.0, 'svc__break_ties': False, 'svc__cache_size': 200, 'svc__class
_weight': 'balanced', 'svc__coef0': 0.0, 'svc__decision_function_shape': 'ovr', '
svc__degree': 3, 'svc__gamma': 'scale', 'svc__kernel': 'rbf', 'svc__max_iter': -1
, 'svc__probability': False, 'svc__random_state': 42, 'svc__shrinking': True, 'sv
c__tol': 0.001, 'svc__verbose': False}
```

```
# Confusion matrix of test set
ConfusionMatrixDisplay.from_predictions(y_test, y_pred)
plt.show()
```




```
# Get the classification report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

          AF       0.39      0.60      0.47        20
          EF       0.59      1.00      0.74        57
          FF       0.97      0.49      0.65        76
          GF       1.00      1.00      1.00        10
          MF       0.15      0.29      0.20         7
          NF       0.79      0.62      0.69        86

    accuracy                           0.67       256
   macro avg       0.65      0.66      0.63       256
weighted avg       0.76      0.67      0.67       256
```

```
#Get the classification report
svm_report = classification_report(y_test, y_pred, output_dict=True)

# Create a Pandas DataFrame from the report
df = pd.DataFrame(svm_report).transpose()

# Save the DataFrame to an Excel file
df.to_excel('svm_report.xlsx')
```

A6.2 K-Neighbours Classification

```
# Make pipeline of scaling, and classifier
knn_pipe = make_pipeline(StandardScaler(), KNeighborsClassifier(weights='distance
'))
```

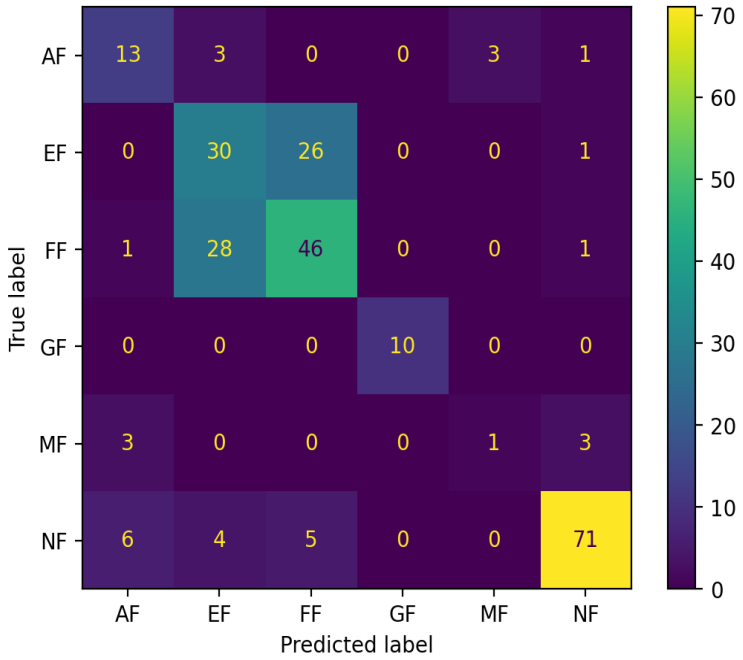
```
# Define multiple scoring metrics
scoring = {'acc': 'accuracy',
           'prec_macro': 'precision_macro',
           'rec_macro': 'recall_macro',
           'f1_macro': 'f1_macro'}

# Stratified K-Fold
stratkfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Return a dictionary of all scorings
cv_knn_scores = cross_validate(knn_pipe, X_train, y_train, cv=stratkfold, scoring
=scoring)

# Print scoring results from dictionary
for metric_name, metric_value in cv_knn_scores.items():
    mean = np.mean(metric_value)
    print(f'{metric_name}: {np.round(metric_value, 4)}, Mean: {np.round(mean, 4)}
')
```

```
fit_time: [0.002 0.002 0.002 0.002 0.001], Mean: 0.0018
score_time: [0.004 0.003 0.003 0.002 0.004], Mean: 0.0032
test_acc: [0.625  0.6917 0.6723 0.6975 0.7143], Mean: 0.6801
test_prec_macro: [0.5342 0.6132 0.5846 0.606  0.5947], Mean: 0.5865
test_rec_macro: [0.5383 0.6399 0.5524 0.6023 0.6124], Mean: 0.5891
test_f1_macro: [0.5331 0.6236 0.5652 0.6022 0.602 ], Mean: 0.5852
```

```
# Fit pipeline to train set
knn_pipe.fit(X_train, y_train)

# Predict on test set
y_pred = knn_pipe.predict(X_test)

# Get the parameters of the model
params = knn_pipe.get_params()

# Print the parameters
print(params)
```

```
{'memory': None, 'steps': [('standardscaler', StandardScaler()), ('kneighborsclas
sifier', KNeighborsClassifier(weights='distance'))], 'verbose': False, 'standards
caler': StandardScaler(), 'kneighborsclassifier': KNeighborsClassifier(weights='d
istance'), 'standardscaler__copy': True, 'standardscaler__with_mean': True, 'stan
dardscaler__with_std': True, 'kneighborsclassifier__algorithm': 'auto', 'kneighbo
rsclassifier__leaf_size': 30, 'kneighborsclassifier__metric': 'minkowski', 'kneig
hborsclassifier__metric_params': None, 'kneighborsclassifier__n_jobs': None, 'kne
ighborsclassifier__n_neighbors': 5, 'kneighborsclassifier__p': 2, 'kneighborsclas
sifier__weights': 'distance'}
```

```
# Confusion matrix of test set
ConfusionMatrixDisplay.from_predictions(y_test, y_pred)
plt.show()
```




```
# Classification report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

          AF       0.57      0.65      0.60        20
          EF       0.46      0.53      0.49        57
          FF       0.60      0.61      0.60        76
          GF       1.00      1.00      1.00        10
          MF       0.25      0.14      0.18         7
          NF       0.92      0.83      0.87        86

    accuracy                           0.67       256
   macro avg       0.63      0.63      0.63       256
weighted avg       0.68      0.67      0.67       256
```

```
#Get the classification report
knn_report = classification_report(y_test, y_pred, output_dict=True)

# Create a Pandas DataFrame from the report
df = pd.DataFrame(knn_report).transpose()

# Save the DataFrame to an Excel file
df.to_excel('knn_report.xlsx')
```

A6.3 Random Forest

```
from sklearn.metrics import make_scorer, precision_score
# Make pipeline of scaling and classifier
rf_pipe = make_pipeline(StandardScaler(), RandomForestClassifier(n_estimators=100
, random_state=42))
```

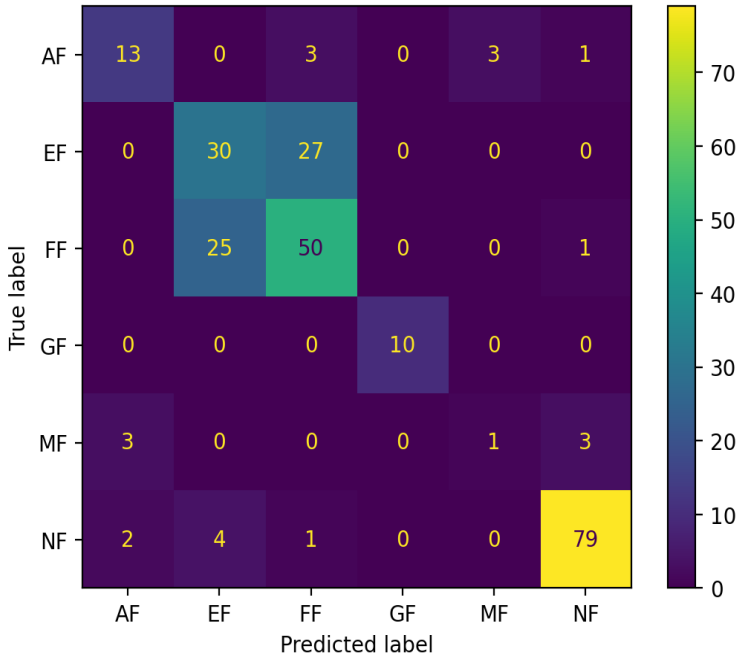
```
# Define multiple scoring metrics
scoring = {'acc': 'accuracy',
           'prec_macro': make_scorer(precision_score, average='macro', zero_divis
ion=0),
           'rec_macro': 'recall_macro',
           'f1_macro': 'f1_macro'}

# Stratified K-Fold
stratkfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Return a dictionary of all scorings
cv_rf_scores = cross_validate(rf_pipe, X_train, y_train, cv=stratkfold, scoring=s
coring)

# Print scoring results from dictionary
for metric_name, metric_value in cv_rf_scores.items():
    mean = np.mean(metric_value)
    print(f'{metric_name}: {np.round(metric_value, 4)}, Mean: {np.round(mean, 4)}
')
```

```
fit_time: [0.0913 0.0905 0.0938 0.0781 0.0781], Mean: 0.0864
score_time: [0.006  0.     0.     0.0156 0.    ], Mean: 0.0043
test_acc: [0.7167 0.7167 0.7059 0.7395 0.7227], Mean: 0.7203
test_prec_macro: [0.6167 0.6384 0.7198 0.6475 0.6101], Mean: 0.6465
test_rec_macro: [0.6226 0.6694 0.6421 0.6088 0.5946], Mean: 0.6275
test_f1_macro: [0.619  0.6503 0.6712 0.6211 0.5973], Mean: 0.6318
```

```
# Fit pipeline to train set
rf_pipe.fit(X_train, y_train)

# Predict on test set
y_pred = rf_pipe.predict(X_test)

# Get the parameters of the model
params = rf_pipe.get_params()

# Print the parameters
print(params)
```

```
{'memory': None, 'steps': [('standardscaler', StandardScaler()), ('randomforestcl
assifier', RandomForestClassifier(random_state=42))], 'verbose': False, 'standard
scaler': StandardScaler(), 'randomforestclassifier': RandomForestClassifier(rando
m_state=42), 'standardscaler__copy': True, 'standardscaler__with_mean': True, 'st
andardscaler__with_std': True, 'randomforestclassifier__bootstrap': True, 'random
forestclassifier__ccp_alpha': 0.0, 'randomforestclassifier__class_weight': None,
'randomforestclassifier__criterion': 'gini', 'randomforestclassifier__max_depth':
None, 'randomforestclassifier__max_features': 'auto', 'randomforestclassifier__ma
x_leaf_nodes': None, 'randomforestclassifier__max_samples': None, 'randomforestcl
assifier__min_impurity_decrease': 0.0, 'randomforestclassifier__min_samples_leaf'
: 1, 'randomforestclassifier__min_samples_split': 2, 'randomforestclassifier__min
_weight_fraction_leaf': 0.0, 'randomforestclassifier__n_estimators': 100, 'random
forestclassifier__n_jobs': None, 'randomforestclassifier__oob_score': False, 'ran
domforestclassifier__random_state': 42, 'randomforestclassifier__verbose': 0, 'ra
ndomforestclassifier__warm_start': False}
```

```
# Confusion matrix of test set
ConfusionMatrixDisplay.from_predictions(y_test, y_pred)
plt.show()
```




```
# Classification report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

          AF       0.72      0.65      0.68        20
          EF       0.51      0.53      0.52        57
          FF       0.62      0.66      0.64        76
          GF       1.00      1.00      1.00        10
          MF       0.25      0.14      0.18         7
          NF       0.94      0.92      0.93        86

    accuracy                           0.71       256
   macro avg       0.67      0.65      0.66       256
weighted avg       0.71      0.71      0.71       256
```

```
#Get the classification report
rf_report = classification_report(y_test, y_pred, output_dict=True)

# Create a Pandas DataFrame from the report
df = pd.DataFrame(rf_report).transpose()

# Save the DataFrame to an Excel file
df.to_excel('rf_report.xlsx')
```

A6.4 XGBoost

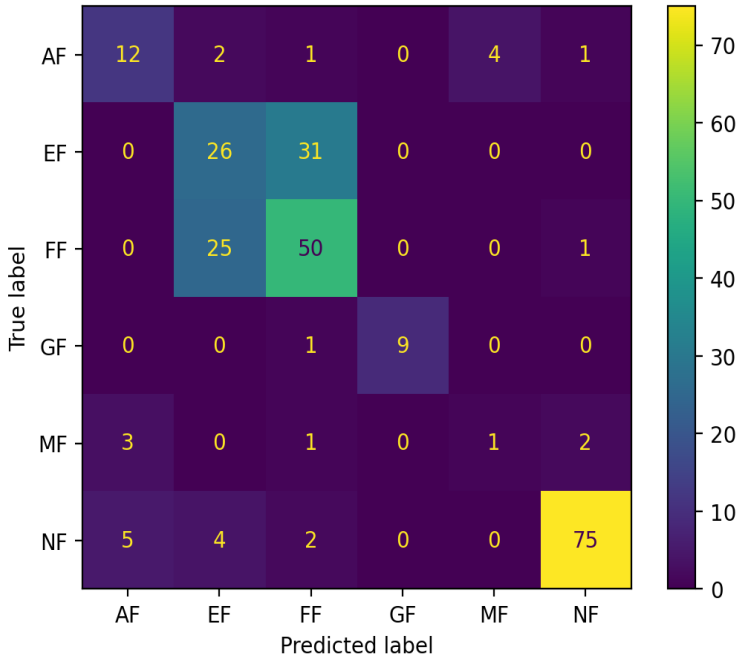
```
# Encode target variable
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(y)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_s
tate=42)

from xgboost import XGBClassifier

# Make pipeline of scaling and classifier
xgb_pipe = make_pipeline(StandardScaler(), XGBClassifier(random_state=42))

# Define multiple scoring metrics
scoring = {'acc': 'accuracy',
           'prec_macro': 'precision_macro',
           'rec_macro': 'recall_macro',
           'f1_macro': 'f1_macro'}

# Stratified K-Fold
stratkfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Return a dictionary of all scorings
cv_xgb_scores = cross_validate(xgb_pipe, X_train, y_train, cv=stratkfold, scoring
=scoring)

# Print scoring results from dictionary
for metric_name, metric_value in cv_xgb_scores.items():
    mean = np.mean(metric_value)
    print(f'{metric_name}: {np.round(metric_value, 4)}, Mean: {np.round(mean, 4)}
')
```

```
fit_time: [0.1532 0.1274 0.1223 0.1433 0.1186], Mean: 0.1329
score_time: [0.003 0.005 0.    0.004 0.   ], Mean: 0.0024
test_acc: [0.7    0.7    0.7143 0.7479 0.7059], Mean: 0.7136
test_prec_macro: [0.604  0.6204 0.709  0.647  0.7665], Mean: 0.6694
test_rec_macro: [0.6094 0.6542 0.6772 0.5999 0.6306], Mean: 0.6343
test_f1_macro: [0.6056 0.6352 0.6912 0.6116 0.6679], Mean: 0.6423
```

```
# Fit pipeline to train set
xgb_pipe.fit(X_train, y_train)

# Predict on test set
y_pred = xgb_pipe.predict(X_test)

# Get the parameters of the model
params = xgb_pipe.get_params()

# Print the parameters
print(params)
```

```
{'memory': None, 'steps': [('standardscaler', StandardScaler()), ('xgbclassifier'
, XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              objective='multi:softprob', predictor=None, ...))], 'verbose': Fals
e, 'standardscaler': StandardScaler(), 'xgbclassifier': XGBClassifier(base_score=
None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              objective='multi:softprob', predictor=None, ...), 'standardscaler__
copy': True, 'standardscaler__with_mean': True, 'standardscaler__with_std': True,
'xgbclassifier__objective': 'multi:softprob', 'xgbclassifier__use_label_encoder':
None, 'xgbclassifier__base_score': None, 'xgbclassifier__booster': None, 'xgbclas
sifier__callbacks': None, 'xgbclassifier__colsample_bylevel': None, 'xgbclassifie
r__colsample_bynode': None, 'xgbclassifier__colsample_bytree': None, 'xgbclassifi
er__early_stopping_rounds': None, 'xgbclassifier__enable_categorical': False, 'xg
bclassifier__eval_metric': None, 'xgbclassifier__feature_types': None, 'xgbclassi
fier__gamma': None, 'xgbclassifier__gpu_id': None, 'xgbclassifier__grow_policy':
None, 'xgbclassifier__importance_type': None, 'xgbclassifier__interaction_constra
ints': None, 'xgbclassifier__learning_rate': None, 'xgbclassifier__max_bin': None
, 'xgbclassifier__max_cat_threshold': None, 'xgbclassifier__max_cat_to_onehot': N
one, 'xgbclassifier__max_delta_step': None, 'xgbclassifier__max_depth': None, 'xg
bclassifier__max_leaves': None, 'xgbclassifier__min_child_weight': None, 'xgbclas
sifier__missing': nan, 'xgbclassifier__monotone_constraints': None, 'xgbclassifie
r__n_estimators': 100, 'xgbclassifier__n_jobs': None, 'xgbclassifier__num_paralle
l_tree': None, 'xgbclassifier__predictor': None, 'xgbclassifier__random_state': 4
2, 'xgbclassifier__reg_alpha': None, 'xgbclassifier__reg_lambda': None, 'xgbclass
ifier__sampling_method': None, 'xgbclassifier__scale_pos_weight': None, 'xgbclass
ifier__subsample': None, 'xgbclassifier__tree_method': None, 'xgbclassifier__vali
date_parameters': None, 'xgbclassifier__verbosity': None}
```

```
# Confusion matrix of test set
y_test = le.inverse_transform(y_test)
y_pred = le.inverse_transform(y_pred)
ConfusionMatrixDisplay.from_predictions(y_test, y_pred)
plt.show()
```




```
# Classification report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

          AF       0.60      0.60      0.60        20
          EF       0.46      0.46      0.46        57
          FF       0.58      0.66      0.62        76
          GF       1.00      0.90      0.95        10
          MF       0.20      0.14      0.17         7
          NF       0.95      0.87      0.91        86

    accuracy                           0.68       256
   macro avg       0.63      0.60      0.62       256
weighted avg       0.68      0.68      0.68       256
```

Appendix B

Bayesian Classification Model for Failure Detection of Wind Turbine from IIoT Data

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

import matplotlib.pyplot as plt
import missingno as msno
import seaborn as sns

from imblearn.over_sampling import SMOTE
from imblearn.pipeline import make_pipeline
from sklearn.utils import class_weight
from sklearn.model_selection import train_test_split, cross_validate, Stra
tifiedKFold, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import ConfusionMatrixDisplay, classification_report

%config InlineBackend.figure_format = 'retina'
```

B1. Read data

We have 3 data:

- *scada_data.csv*: Contains >60 information (or status) of wind turbine components recorded by SCADA system
- *fault_data.csv*: Contains wind turbine fault types (or modes)
- *status_data.csv*: Contains description of status of wind turbine operational

```
scada_df = pd.read_csv('scada_data.csv')
scada_df['DateTime'] = pd.to_datetime(scada_df['DateTime'])
# scada_df.set_index('DateTime', inplace=True)

scada_df
```

```
                DateTime        Time  Error  WEC: ava. windspeed  \
0     2014-05-01 00:00:00  1398920448      0                  6.9
1     2014-05-01 00:09:00  1398920960      0                  5.3
2     2014-05-01 00:20:00  1398921600      0                  5.0
3     2014-05-01 00:30:00  1398922240      0                  4.4
4     2014-05-01 00:39:00  1398922752      0                  5.7
...                   ...         ...    ...                  ...
49022 2015-04-08 23:20:00  1428553216      0                  3.9
49023 2015-04-08 23:30:00  1428553856      0                  3.9
49024 2015-04-08 23:39:00  1428554368      0                  4.2
49025 2015-04-08 23:50:00  1428555008      0                  4.1
```

```
49026 2015-04-09 00:00:00  1428555648      0                 4.8

       WEC: max. windspeed  WEC: min. windspeed  WEC: ava. Rotation  \
0                      9.4                  2.9                0.00
1                      8.9                  1.6                0.00
2                      9.5                  1.4                0.00
3                      8.3                  1.3                0.00
4                      9.7                  1.2                0.00
...                    ...                  ...                 ...
49022                  5.5                  2.2                6.75
49023                  5.6                  2.9                6.64
49024                  6.7                  2.6                7.18
49025                  6.6                  2.7                7.02
49026                  6.0                  3.3                8.39

       WEC: max. Rotation  WEC: min. Rotation  WEC: ava. Power  ...  \
0                    0.02                0.00                0  ...
1                    0.01                0.00                0  ...
2                    0.04                0.00                0  ...
3                    0.08                0.00                0  ...
4                    0.05                0.00                0  ...
...                   ...                 ...              ...  ...
49022                7.40                6.01              147  ...
49023                7.06                6.33              128  ...
49024                8.83                6.22              163  ...
49025                7.94                6.20              160  ...
49026                9.48                7.14              284  ...

       Rectifier cabinet temp.  Yaw inverter cabinet temp.  \
0                           24                          20
1                           24                          20
2                           24                          20
3                           23                          21
4                           23                          21
...                        ...                         ...
49022                       33                          23
49023                       34                          23
49024                       34                          23
49025                       33                          23
49026                       33                          22

       Fan inverter cabinet temp.  Ambient temp.  Tower temp.  \
0                              25             12           14
1                              25             12           14
2                              25             12           14
3                              25             12           14
4                              25             12           14
...                           ...            ...          ...
49022                          28              9           17
49023                          28              9           17
49024                          28              9           18
49025                          28              9           17
49026                          28              9           17

       Control cabinet temp.  Transformer temp.  RTU: ava. Setpoint 1  \
```

```
0                        24                34                2501
1                        24                34                2501
2                        24                34                2501
3                        24                34                2501
4                        23                34                2501
...                     ...               ...                 ...
49022                    27                35                3050
49023                    27                35                3050
49024                    27                34                3050
49025                    27                34                3050
49026                    27                34                3050

       Inverter averages  Inverter std dev
0              25.272728         1.103713
1              25.272728         1.103713
2              25.272728         1.103713
3              25.272728         1.103713
4              25.272728         1.103713
...                  ...              ...
49022          24.454546         3.474583
49023          24.454546         3.445683
49024          24.363636         3.413876
49025          24.000000         3.376389
49026          23.818182         3.250175

[49027 rows x 66 columns]
```

```
status_df = pd.read_csv('status_data.csv')
status_df['DateTime'] = pd.to_datetime(status_df['DateTime'])
# status_df.set_index('DateTime', inplace=True)

status_df
```

```
                DateTime  Main Status  Sub Status Full Status  \
0    2014-04-24 12:37:00            0           0        0:00
1    2014-04-25 19:27:00           71         104    71 : 104
2    2014-04-26 09:30:00            8           0        8:00
3    2014-04-26 10:05:00            8           0        8:00
4    2014-04-26 10:05:00            8           0        8:00
...                  ...          ...         ...         ...
1844 2015-04-27 07:26:00            0           0        0:00
1845 2015-04-28 22:14:00           26         373    26 : 373
1846 2015-04-28 22:14:00            0           2        0:02
1847 2015-04-28 22:17:00            0           1        0:01
1848 2015-04-28 22:18:00            0           0        0:00

                                              Status Text  T  Service  FaultMsg  \
0                                  Turbine in operation  1    False     False
1     Insulation monitoring : Insulation fault Phase U2  6    False      True
2                                          Maintenance  6     True     False
3                                          Maintenance  6    False     False
4                                          Maintenance  6     True     False
...                                                  ... ..      ...       ...
1844                               Turbine in operation  1    False     False
1845  Malfunction fan-inverter : Other control board...  6    False     False
```

```
1846                                Turbine operational  1    False     False
1847                                   Turbine starting  1    False     False
1848                               Turbine in operation  1    False     False

      Value0
0        7.4
1       20.5
2       17.1
3        8.7
4       10.6
...      ...
1844     7.0
1845     8.1
1846     9.5
1847    11.1
1848    11.5

[1849 rows x 9 columns]
```

```
fault_df = pd.read_csv('fault_data.csv')
fault_df['DateTime'] = pd.to_datetime(fault_df['DateTime'])
# fault_df.set_index('DateTime', inplace=True)

fault_df
```

```
                DateTime          Time Fault
0   2014-05-14 14:39:44  1.400096e+09    GF
1   2014-05-14 14:50:24  1.400097e+09    GF
2   2014-05-14 14:58:56  1.400098e+09    GF
3   2014-05-14 15:09:36  1.400098e+09    GF
4   2014-05-14 15:20:16  1.400099e+09    GF
..                  ...           ...   ...
548 2015-01-14 23:00:48  1.421298e+09    AF
549 2015-01-14 23:09:20  1.421299e+09    AF
550 2015-01-15 22:50:08  1.421384e+09    AF
551 2015-01-15 23:00:48  1.421384e+09    AF
552 2015-01-15 23:09:20  1.421385e+09    AF

[553 rows x 3 columns]
```

In the fault data, there are 5 types of faults, or fault modes:

- gf: generator heating fault
- mf: mains failure fault
- ff: feeding fault
- af: timeout warnmessage : malfunction aircooling
- ef: excitation error : overvoltage DC-link

```
fault_df.Fault.unique()
```

```
array(['GF', 'MF', 'FF', 'AF', 'EF'], dtype=object)
```

B2. Data preparation for ML

There are far more records of NF (normal condition) than faulty records - imbalanced dataset. We will sample the No Fault dataframe and pick only 300 records.

```
df_combine.Fault.value_counts()
```

```
NF    48581
FF      254
EF      174
AF       62
GF       43
MF       20
Name: Fault, dtype: int64
```

```
# No Fault mode data
df_nf = df_combine[df_combine.Fault=='NF'].sample(300, random_state=42)

df_nf
```

```
               DateTime_x        Time  Error  WEC: ava. windspeed  \
7340  2014-06-21 19:29:00  1403396992      0                3.500
949   2014-05-07 16:39:00  1399498752      0                5.300
37369 2015-01-17 00:00:00  1421474432      0                9.200
15200 2014-08-15 07:30:00  1408105856      0                7.200
42798 2015-02-23 22:30:00  1424752256      0               14.900
...                   ...         ...    ...                  ...
6298  2014-06-14 13:50:00  1402771840      0                2.200
22374 2014-10-04 01:38:00  1412404736      0                7.100
39742 2015-02-02 16:41:00  1422916864      0                2.100
34504 2014-12-28 02:40:00  1419756032      0                7.800
10632 2014-07-14 15:09:00  1405368576      0               11.300

       WEC: max. windspeed  WEC: min. windspeed  WEC: ava. Rotation  \
7340                 4.600                2.400               6.370
949                 15.000                3.000               0.000
37369               10.800                8.100              13.480
15200                9.600                5.000              10.450
42798               20.900               11.200              14.690
...                    ...                  ...                 ...
6298                 4.300                0.400               4.600
22374                9.000                5.500              10.120
39742                2.600                1.700               5.050
34504                9.500                5.200              10.670
10632               15.200                6.100              13.560

       WEC: max. Rotation  WEC: min. Rotation  WEC: ava. Power  ...  \
7340                6.640               6.160               87  ...
949                 0.000               0.000                0  ...
37369              13.950              12.920             2078  ...
15200              11.620               9.700              998  ...
42798              15.540              14.130             3060  ...
...                   ...                 ...              ...  ...
6298                5.120               4.150                1  ...
```

```
22374              10.770              9.500              875  ...
39742               5.320              4.940                6  ...
34504              11.720              9.360             1138  ...
10632              14.870             10.740             2638  ...

       Fan inverter cabinet temp.  Ambient temp.  Tower temp.  \
7340                           40             20           26
949                            26             16           15
37369                          20              3           18
15200                          30             21           30
42798                          22              6           28
...                           ...            ...          ...
6298                           40             24           25
22374                          27             13           26
39742                          21              4            8
34504                          22              4           19
10632                          38             23           36

       Control cabinet temp.  Transformer temp.  RTU: ava. Setpoint 1  \
7340                      36                 45                  3050
949                       24                 33                  2501
37369                     25                 40                  3050
15200                     37                 48                  3050
42798                     39                 67                  3050
...                      ...                ...                   ...
6298                      35                 45                  3050
22374                     34                 43                  3050
39742                     17                 27                  3050
34504                     27                 38                  3050
10632                     44                 64                  3050

       Inverter averages  Inverter std dev  DateTime_y  Fault
7340              34.273             3.228         NaT     NF
949               24.455             0.934         NaT     NF
37369             19.091             1.446         NaT     NF
15200             30.545             1.036         NaT     NF
42798             28.273             1.902         NaT     NF
...                  ...               ...         ...    ...
6298              35.000             1.414         NaT     NF
22374             28.091             1.136         NaT     NF
39742             19.909             1.044         NaT     NF
34504             19.818             0.603         NaT     NF
10632             35.909             1.973         NaT     NF

[300 rows x 68 columns]
```

```
# With fault mode data
df_f = df_combine[df_combine.Fault!='NF']

df_f
```

```
              DateTime_x        Time  Error  WEC: ava. windspeed  \
1945  2014-05-14 14:39:00  1400096384      0                5.700
1946  2014-05-14 14:50:00  1400097024      0                6.400
1947  2014-05-14 14:58:00  1400097536      0                5.600
```

```
1948  2014-05-14 15:09:00  1400098176      0               5.300
1949  2014-05-14 15:20:00  1400098816      0               5.100
...                   ...         ...    ...                 ...
37075 2015-01-14 23:00:00  1421298048      0              13.500
37076 2015-01-14 23:09:00  1421298560      0              15.500
37218 2015-01-15 22:50:00  1421383808      0               8.400
37219 2015-01-15 23:00:00  1421384448      0               8.100
37220 2015-01-15 23:09:00  1421384960      0               7.900

       WEC: max. windspeed  WEC: min. windspeed  WEC: ava. Rotation  \
1945                 7.400                4.100               9.980
1946                 8.600                2.700               4.120
1947                 7.500                4.000               9.990
1948                 6.600                4.100               9.980
1949                 7.200                3.800               9.990
...                    ...                  ...                 ...
37075               22.000                7.700              14.690
37076               25.700                1.900               2.530
37218                9.700                6.800              12.790
37219                9.800                6.000              11.710
37220               11.600                1.900               1.400

       WEC: max. Rotation  WEC: min. Rotation  WEC: ava. Power  ...  \
1945               10.230               9.720                0  ...
1946               10.710               0.700                0  ...
1947               10.200               9.800                0  ...
1948               10.170               9.790                0  ...
1949               10.170               9.800                0  ...
...                   ...                 ...              ...  ...
37075              15.190              13.830             2736  ...
37076              14.670               0.000               20  ...
37218              13.300              11.640             1667  ...
37219              13.270               9.990             1311  ...
37220              11.180               0.000               14  ...

       Fan inverter cabinet temp.  Ambient temp.  Tower temp.  \
1945                           27             17           26
1946                           27             17           26
1947                           27             17           26
1948                           27             18           26
1949                           28             18           26
...                           ...            ...          ...
37075                          24              8           29
37076                          23              7           22
37218                          21              5           21
37219                          20              5           21
37220                          21              5           17

       Control cabinet temp.  Transformer temp.  RTU: ava. Setpoint 1  \
1945                      35                 34                  2501
1946                      35                 34                  2501
1947                      35                 34                  2501
1948                      35                 34                  2501
1949                      35                 34                  2501
...                      ...                ...                   ...
```

```
37075                    35                57                3050
37076                    35                57                3050
37218                    28                42                3050
37219                    28                42                3050
37220                    28                42                3050

       Inverter averages  Inverter std dev          DateTime_y  Fault
1945              25.818             0.603 2014-05-14 14:39:44     GF
1946              26.091             0.944 2014-05-14 14:50:24     GF
1947              26.455             0.820 2014-05-14 14:58:56     GF
1948              26.182             0.874 2014-05-14 15:09:36     GF
1949              26.182             0.982 2014-05-14 15:20:16     GF
...                  ...               ...                 ...    ...
37075             25.909             1.814 2015-01-14 23:00:48     AF
37076             28.182             1.834 2015-01-14 23:09:20     AF
37218             20.909             1.300 2015-01-15 22:50:08     AF
37219             20.909             1.300 2015-01-15 23:00:48     AF
37220             22.909             1.300 2015-01-15 23:09:20     AF

[553 rows x 68 columns]
```

```
# Combine no fault and faulty dataframes

df_combine = pd.concat((df_nf, df_f), axis=0).reset_index(drop=True)

df_combine
```

```
             DateTime_x        Time  Error  WEC: ava. windspeed  \
0   2014-06-21 19:29:00  1403396992      0                3.500
1   2014-05-07 16:39:00  1399498752      0                5.300
2   2015-01-17 00:00:00  1421474432      0                9.200
3   2014-08-15 07:30:00  1408105856      0                7.200
4   2015-02-23 22:30:00  1424752256      0               14.900
..                  ...         ...    ...                  ...
848 2015-01-14 23:00:00  1421298048      0               13.500
849 2015-01-14 23:09:00  1421298560      0               15.500
850 2015-01-15 22:50:00  1421383808      0                8.400
851 2015-01-15 23:00:00  1421384448      0                8.100
852 2015-01-15 23:09:00  1421384960      0                7.900

     WEC: max. windspeed  WEC: min. windspeed  WEC: ava. Rotation  \
0                  4.600                2.400               6.370
1                 15.000                3.000               0.000
2                 10.800                8.100              13.480
3                  9.600                5.000              10.450
4                 20.900               11.200              14.690
..                   ...                  ...                 ...
848               22.000                7.700              14.690
849               25.700                1.900               2.530
850                9.700                6.800              12.790
851                9.800                6.000              11.710
852               11.600                1.900               1.400

     WEC: max. Rotation  WEC: min. Rotation  WEC: ava. Power  ...  \
0                 6.640               6.160               87  ...
```

```
1                0.000              0.000               0  ...
2               13.950             12.920            2078  ...
3               11.620              9.700             998  ...
4               15.540             14.130            3060  ...
..                 ...                ...             ...  ...
848             15.190             13.830            2736  ...
849             14.670              0.000              20  ...
850             13.300             11.640            1667  ...
851             13.270              9.990            1311  ...
852             11.180              0.000              14  ...

     Fan inverter cabinet temp.  Ambient temp.  Tower temp.  \
0                            40             20           26
1                            26             16           15
2                            20              3           18
3                            30             21           30
4                            22              6           28
..                          ...            ...          ...
848                          24              8           29
849                          23              7           22
850                          21              5           21
851                          20              5           21
852                          21              5           17

     Control cabinet temp.  Transformer temp.  RTU: ava. Setpoint 1  \
0                       36                 45                  3050
1                       24                 33                  2501
2                       25                 40                  3050
3                       37                 48                  3050
4                       39                 67                  3050
..                     ...                ...                   ...
848                     35                 57                  3050
849                     35                 57                  3050
850                     28                 42                  3050
851                     28                 42                  3050
852                     28                 42                  3050

     Inverter averages  Inverter std dev          DateTime_y  Fault
0               34.273             3.228                 NaT     NF
1               24.455             0.934                 NaT     NF
2               19.091             1.446                 NaT     NF
3               30.545             1.036                 NaT     NF
4               28.273             1.902                 NaT     NF
..                 ...               ...                 ...    ...
848             25.909             1.814 2015-01-14 23:00:48     AF
849             28.182             1.834 2015-01-14 23:09:20     AF
850             20.909             1.300 2015-01-15 22:50:08     AF
851             20.909             1.300 2015-01-15 23:00:48     AF
852             22.909             1.300 2015-01-15 23:09:20     AF

[853 rows x 68 columns]
```

Preparing for the training dataset, we **drop irrelevant features**. First we drop datetime, time, and error columns. Next, features that "de facto" are output of wind turbine, such as power

from wind, operating hours, and kWh production, are dropped. Also, climatic variable such as wind speed are not useful.

```
# Feature selection
train_df = df_combine.loc[:, ['WEC: ava. windspeed',
                              'WEC: ava. Rotation',
                              'WEC: ava. Power',
                              'WEC: ava. reactive Power',
                              'WEC: ava. blade angle A',
                              'Spinner temp.',
                              'Front bearing temp.',
                              'Rear bearing temp.',
                              'Pitch cabinet blade A temp.',
                              'Pitch cabinet blade B temp.',
                              'Pitch cabinet blade C temp.',
                              'Rotor temp. 1',
                              'Rotor temp. 2',
                              'Stator temp. 1',
                              'Stator temp. 2',
                              'Nacelle ambient temp. 1',
                              'Nacelle ambient temp. 2',
                              'Nacelle temp.',
                              'Nacelle cabinet temp.',
                              'Main carrier temp.',
                              'Rectifier cabinet temp.',
                              'Yaw inverter cabinet temp.',
                              'Fan inverter cabinet temp.',
                              'Ambient temp.',
                              'Tower temp.',
                              'Control cabinet temp.',
                              'Transformer temp.',
                              'Inverter averages',
                              'Inverter std dev',
                              'Fault']]
train_df
```

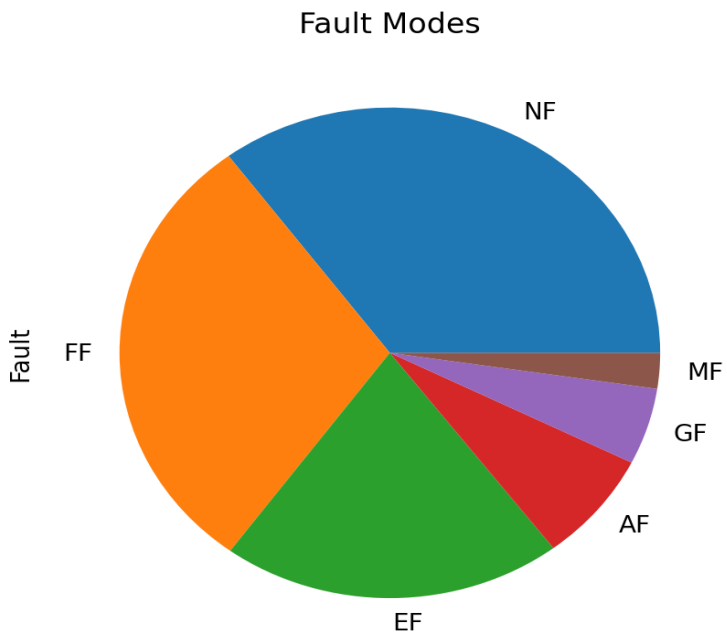
```
     WEC: ava. windspeed  WEC: ava. Rotation  WEC: ava. Power  \
0                  3.500              6.370               87
1                  5.300              0.000                0
2                  9.200             13.480             2078
3                  7.200             10.450              998
4                 14.900             14.690             3060
..                   ...                ...              ...
848               13.500             14.690             2736
849               15.500              2.530               20
850                8.400             12.790             1667
851                8.100             11.710             1311
852                7.900              1.400               14

     WEC: ava. reactive Power  WEC: ava. blade angle A  Spinner temp.  \
0                          32                   1.000             28
1                           0                  91.930             15
```

```
2                        173                   1.000             8
3                         64                   1.000            17
4                        296                  12.780            12
..                       ...                     ...           ...
848                      238                   8.260            13
849                        4                  66.850            13
850                      145                   1.000             9
851                      111                   1.000             9
852                        3                  66.750             9

     Front bearing temp.  Rear bearing temp.  Pitch cabinet blade A temp.  \
0                     30                  34                           45
1                     12                  12                           28
2                     12                  23                           22
3                     22                  26                           31
4                     18                  26                           31
..                   ...                 ...                          ...
848                   20                  28                           32
849                   20                  27                           32
850                   14                  24                           23
851                   14                  24                           23
852                   14                  23                           24

     Pitch cabinet blade B temp.  ...  Rectifier cabinet temp.  \
0                             45  ...                       39
1                             27  ...                       25
2                             22  ...                       21
3                             31  ...                       27
4                             31  ...                       23
..                           ...  ...                      ...
848                           32  ...                       27
849                           32  ...                       27
850                           24  ...                       23
851                           24  ...                       23
852                           24  ...                       24

     Yaw inverter cabinet temp.  Fan inverter cabinet temp.  Ambient temp.  \
0                            36                          40             20
1                            22                          26             16
2                            16                          20              3
3                            24                          30             21
4                            19                          22              6
..                          ...                         ...            ...
848                          21                          24              8
849                          21                          23              7
850                          17                          21              5
851                          16                          20              5
852                          17                          21              5

     Tower temp.  Control cabinet temp.  Transformer temp.  Inverter averages  \
0             26                     36                 45             34.273
1             15                     24                 33             24.455
2             18                     25                 40             19.091
3             30                     37                 48             30.545
4             28                     39                 67             28.273
```

```
..          ...                  ...                ...              ...
848          29                   35                 57           25.909
849          22                   35                 57           28.182
850          21                   28                 42           20.909
851          21                   28                 42           20.909
852          17                   28                 42           22.909

     Inverter std dev  Fault
0               3.228     NF
1               0.934     NF
2               1.446     NF
3               1.036     NF
4               1.902     NF
..                ...    ...
848             1.814     AF
849             1.834     AF
850             1.300     AF
851             1.300     AF
852             1.300     AF

[853 rows x 30 columns]
```

```
# Imbalanced fault modes
train_df.Fault.value_counts().plot.pie(title='Fault Modes')
```

```
<AxesSubplot:title={'center':'Fault Modes'}, ylabel='Fault'>
```



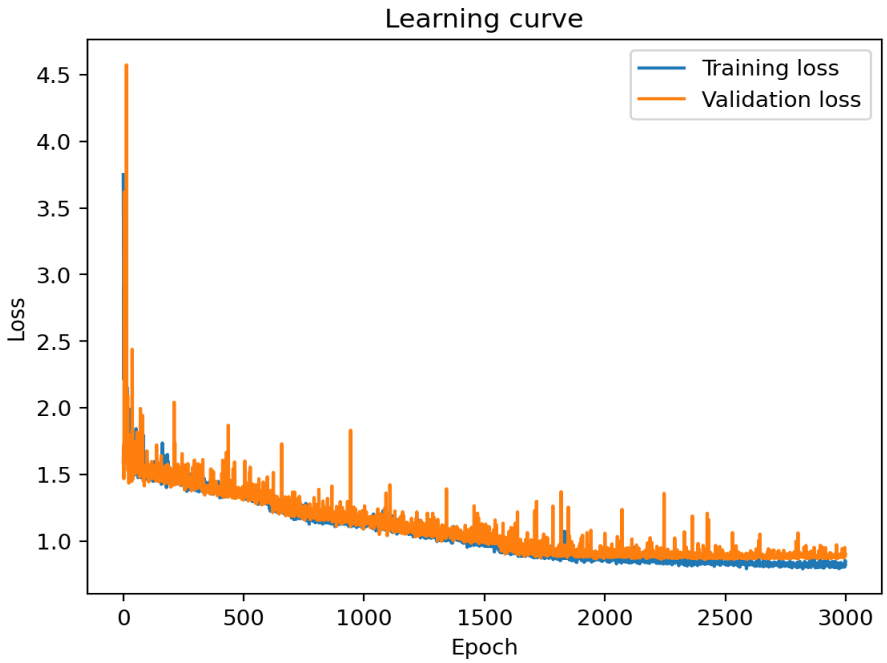
B3. Machine learning - fault modes classification

```
import numpy as np
import torch
import torch.nn as nn
```

```
import torch.optim as optim
import torchbnn as bnn
from sklearn.preprocessing import LabelEncoder

#Fix the random seed
torch.manual_seed(0)
np.random.seed(0)

# Feature and target
X = train_df.iloc[:,:-1]
y = train_df.iloc[:,-1]

# Encode target variable
le = LabelEncoder()
y = le.fit_transform(y)

%%time

from torch.utils.data import TensorDataset, DataLoader
import matplotlib.pyplot as plt

# Split data into training, validation, and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_s
ize=0.2)

# Convert to torch tensors
x_train, y_train = torch.from_numpy(X_train.values).float(), torch.from_nu
mpy(y_train).long()
x_val, y_val = torch.from_numpy(X_val.values).float(), torch.from_numpy(y_
val).long()
x_test, y_test = torch.from_numpy(X_test.values).float(), torch.from_numpy
(y_test).long()

batch_size = 64

# Create data loaders
train_data = TensorDataset(x_train, y_train)
train_loader = DataLoader(train_data,
                          shuffle=True,
                          batch_size=batch_size)

# Make pipeline of scaling and classifier
model = nn.Sequential(
            bnn.BayesLinear(prior_mu=0, prior_sigma=0.1, in_features=x_tra
in.shape[1], out_features=6),
            nn.ReLU(),
            bnn.BayesLinear(prior_mu=0, prior_sigma=0.1, in_features=6, ou
t_features=6),
            nn.ReLU(),
```

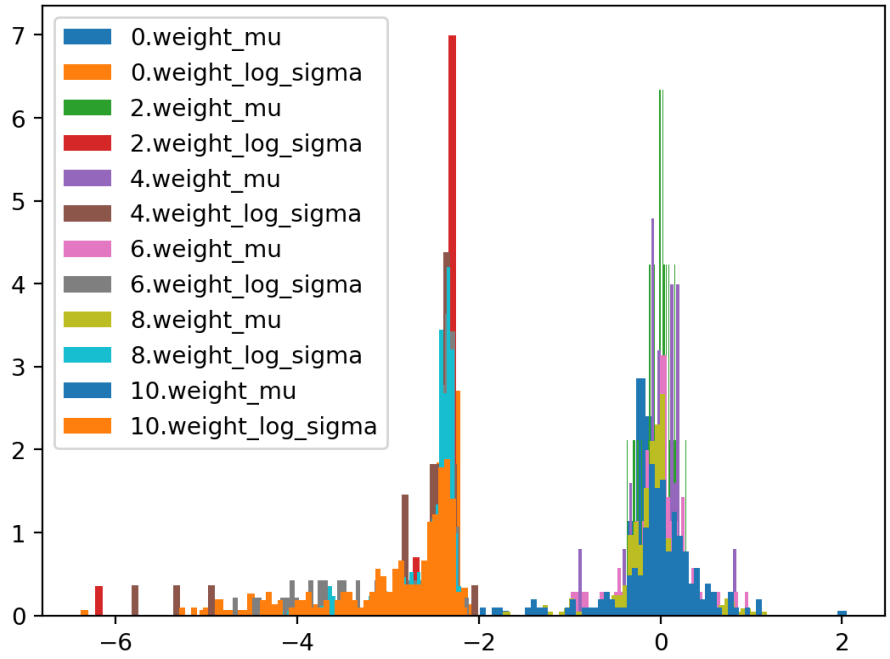
```
            bnn.BayesLinear(prior_mu=0, prior_sigma=0.1, in_features=6, ou
t_features=6),
            nn.ReLU(),
            bnn.BayesLinear(prior_mu=0, prior_sigma=0.1, in_features=6, ou
t_features=15),
            nn.ReLU(),
            bnn.BayesLinear(prior_mu=0, prior_sigma=0.1, in_features=15, o
ut_features=28),
            nn.ReLU(),
            bnn.BayesLinear(prior_mu=0, prior_sigma=0.1, in_features=28, o
ut_features=len(le.classes_)),
        )

# Define loss functions
ce_loss = nn.CrossEntropyLoss()
kl_loss = bnn.BKLLoss(reduction='mean', last_layer_only=False)
kl_weight = 0.00001

# Define optimizer
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Initialize lists to store loss values
train_losses = []
val_losses = []

# Train model
for step in range(3000):

    # Compute training loss
    train_loss = 0
    for x_batch, y_batch in train_loader:
        pre = model(x_batch)
        ce = ce_loss(pre, y_batch)
        kl = kl_loss(model)
        cost = ce + kl_weight * kl
        train_loss += cost.item()

        # Update model parameters
        optimizer.zero_grad()
        cost.backward()
        optimizer.step()

    # Record training loss
    train_losses.append(train_loss / len(train_loader))

    # Compute validation loss
    pre_val = model(x_val)
    ce_val = ce_loss(pre_val, y_val)
    kl_val = kl_loss(model)
    cost_val = ce_val + kl_weight * kl_val
```

```
    # Record validation loss
    val_losses.append(cost_val.item())

# Evaluate model on validation data
pre_val = model(x_val)
_, predicted_val = torch.max(pre_val.data, 1)
total_val = y_val.size(0)
correct_val = (predicted_val == y_val).sum()
accuracy = 100 * float(correct_val) / total_val

# Update best hyperparameters if necessary
#if accuracy > best_accuracy:
#    best_accuracy = accuracy
#    best_params = params

# Plot learning curve
plt.plot(train_losses, label='Training loss')
plt.plot(val_losses, label='Validation loss')
plt.title(f'Learning curve')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()

print('Test accuracy:', accuracy)
print('- CE : %2.2f, KL : %2.2f' % (ce.item(), kl.item()))
```

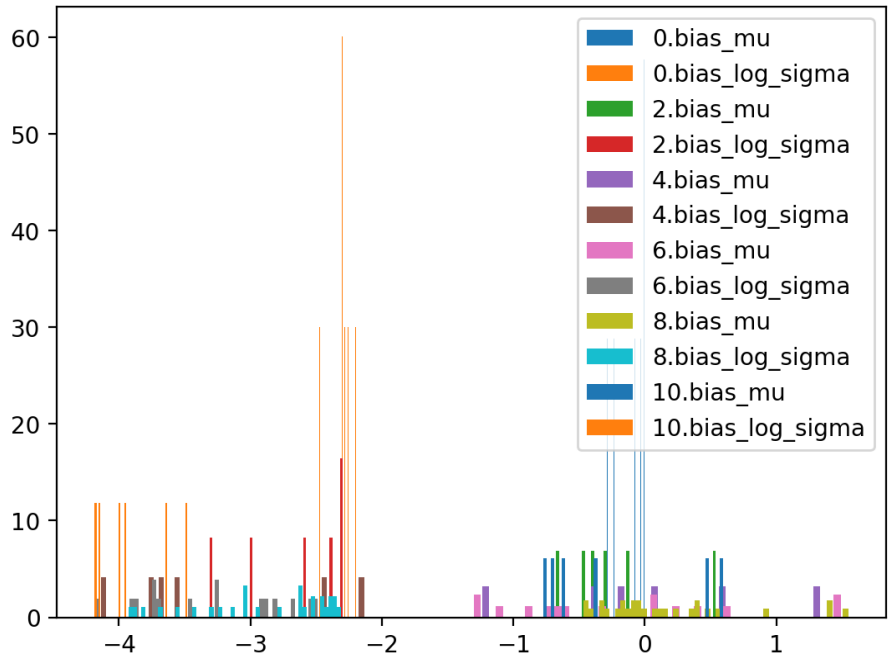


```
Test accuracy: 67.5
- CE : 0.68, KL : 8.39
Wall time: 2min 56s
```

```
#check posterior distribution of weights
for name, param in model.named_parameters():
    if 'weight' in name:
        plt.hist(param.data.cpu().numpy().flatten(), bins=50, density=True
, label=name)
plt.legend()
plt.show()
```

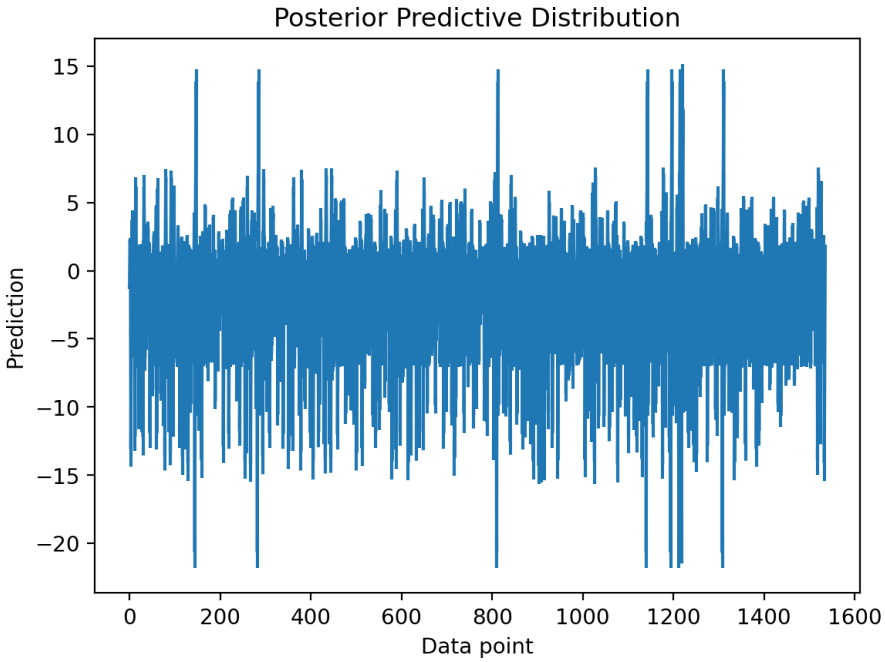



```
#check posterior distribution of biases
for name, param in model.named_parameters():
    if 'bias' in name:
        plt.hist(param.data.cpu().numpy().flatten(), bins=50, density=True
, label=name)
plt.legend()
plt.show()
```




```
# Number of samples
n_samples = 100

# Get weight samples
weight_samples = [model.state_dict(keep_vars=True) for _ in range(n_sample
s)]

# Make predictions using weight samples
predictions = []
for weight_sample in weight_samples:
    model.load_state_dict(weight_sample)
    predictions.append(model(x_test).detach().numpy())

# Convert predictions to numpy array
predictions = np.array(predictions)

# Compute mean and standard deviation of predictions
mean_prediction = predictions.mean(axis=0)
std_prediction = predictions.std(axis=0)

mean_prediction = mean_prediction.reshape(-1)
std_prediction = std_prediction.reshape(-1)

# Plot mean and standard deviation of predictions
plt.errorbar(range(len(mean_prediction)), mean_prediction, yerr=std_predic
tion)
plt.xlabel('Data point')
plt.ylabel('Prediction')
```

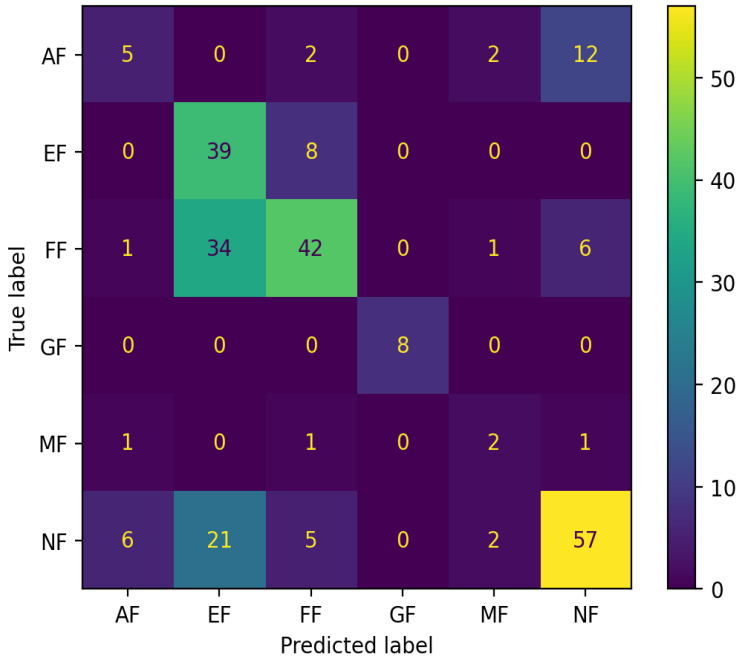
```
plt.title('Posterior Predictive Distribution')
plt.show()
```



The precision, recall, and F1 score are 66%.

```
#Confusion matrix of test data
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
y_pred = model(x_test)
_, predicted = torch.max(y_pred.data, 1)
ConfusionMatrixDisplay(confusion_matrix(y_test, predicted), display_labels
=le.classes_).plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x29909ec13d0>
```



We can see 2 problematic (false predicted) classes here are FF and EF. There are 28 FF predicted as EF.

```
#classification report
print(classification_report(y_test, predicted, target_names=le.classes_))
```

```
              precision    recall  f1-score   support

          AF       0.38      0.24      0.29        21
          EF       0.41      0.83      0.55        47
          FF       0.72      0.50      0.59        84
          GF       1.00      1.00      1.00         8
          MF       0.29      0.40      0.33         5
          NF       0.75      0.63      0.68        91

    accuracy                           0.60       256
   macro avg       0.59      0.60      0.58       256
weighted avg       0.65      0.60      0.60       256
```

References

Boser, B. E., Guyon, I. M., & Vapnik, V. N. (1992). A Training Algorithm for Optimal Margin Classifiers. Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory, 144–152. doi::: 10.1.1.21.3818

Breiman, L. (1996). Bagging Predictors. Machine Learning, 24(421), 123–140. doi: 10.1007/BF00058655

Chang, C., & Lin, C. (2011). LIBSVM A Library for Support Vector Machines. ACM Transactions on Intelligent Systems and Technology (TIST), 2, 1–39. doi: 10.1145/1961189.1961199

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE Synthetic minority oversampling technique. Journal of Artificial Intelligence Research, 16, 321–357. doi: 10.1613/jair.953

Du, M., Tjernberg, L. B., Ma, S., He, Q., Cheng, L., & Guo, J. (2016). A SOM based Anomaly Detection Method for Wind Turbines Health Management through SCADA Data. International Journal of Prognostics and Health Management, 7, 1–13.

Freund, Y., & Schapire, R. (1995). A decision-theoretic generalization of on-line learning and an application to boosting. Computational learning theory, 55(1), 119– 139. doi: 10.1006/jcss.1997.1504

Gill, S., Stephen, B., & Galloway, S. (2012). Wind turbine condition assessment through power curve copula modelling. IEEE Transactions on Sustainable Energy, 3(1), 94–101. doi: 10.1109/TSTE.2011.2167164

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, 11

Hu, R. L., Leahy, K., Konstantakopoulos, I. C., Auslander, D. M., Spanos, C. J., & Agogino, A. M. (2016). Using Domain Knowledge Features for Wind Turbine Diagnostics. In Icmla.

Knerr, S., Personnaz, L., & Dreyfus, G. (1990). Single-layer learning revisited a stepwise procedure for building and training a neural network. Neurocomputing (68), 41–50

Kusiak, A., & Li, W. (2011). The prediction and diagnosis of wind turbine faults. Renewable Energy, 36(1), 16–23. doi: 10.1016/j.renene.2010.05.014

Kusiak, A., & Verma, A. (2011). A data-driven approach for monitoring blade pitch faults in wind turbines. IEEE Transactions on Sustainable Energy, 2(1), 87–96. doi: 10.1109/TSTE.2010.2066585

Laouti, N., Sheibat-othman, N., & Othman, S. (2011). Support Vector Machines for Fault Detection in Wind Turbines. 18th IFAC World Congress, 7067–7072.

Lapira, E., Brisset, D., Davari Ardakani, H., Siegel, D., & Lee, J. (2012). Wind turbine performance assessment using multi-regime modelling approach. Renewable Energy, 45, 86–95. doi: 10.1016/j.renene.2012.02.018

Leahy, K., Hu, R. L., Konstantakopoulos, I. C., Spanos, C. J., & Agogino, A. M. (2016). Diagnosing Wind Turbine Faults Using Machine Learning Techniques Applied to Operational Data. In IEEE international conference on prognostics and health management.

Liu, X. Y., Wu, J., & Zhou, Z. H. (2006). Exploratory under-sampling for class-imbalance learning. Proceedings - IEEE International Conference on Data Mining, ICDM, 39(2), 965–969. doi: 10.1109/ICDM.2006.68

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Duchesnay, E. (2012, Jan). ´ Scikit-learn Machine Learning in Python. The Journal of Machine Learning Research, 12, 2825–2830. doi: 10.1007/s13398-014-0173-7.2

Saxena, A., Celaya, J., Balaban, E., Goebel, K., Saha, B., Saha, S., & Schwabacher, M. (2008). Metrics for evaluating performance of prognostic techniques. 2008 International Conference on Prognostics and Health Management, PHM 2008. doi: 10.1109/PHM.2008.4711436

Skrimpas, G. A., Sweeney, C. W., Marhadi, K. S., Jensen, B. B., Mijatovic, N., & Holboll, J. (2015). Employment of Kernel Methods on Wind Turbine Power Performance Assessment. IEEE Transactions on Sustainable Energy, 6(3), 698–706. doi: 10.1109/TSTE.2015.2405971

Tamilselvan, P., Wang, P., Sheng, S., & Twomey, J. M. (2013). A Two-Stage Diagnosis Framework for Wind Turbine Gearbox Condition Monitoring. International Journal of Prognostics and Health Management (Special Issue on Wind Turbines PHM)

Widodo, A., & Yang, B.-S. (2007). Support vector machine in machine condition monitoring and fault diagnosis. Mechanical Systems and Signal Processing, 21(6), 2560–2574. doi: 10.1016/j.ymssp.2006.12.007

Wilson, D. L. (1972). Asymptotic Properties of Nearest Neighbour Rules Using Edited Data. IEEE Transactions on Systems, Man and Cybernetics, 2(3), 408–421. doi:10.1109/TSMC.1972.4309137

Yang, W., Tavner, P. J., Crabtree, C. J., Feng, Y., & Qiu, Y. (2014). Wind turbine condition monitoring Technical and commercial challenges. Wind Energy, 17(5), 673– 693. doi: 10.1002/we.1508

Zaher, A., McArthur, S., Infield, D., & Patel, Y. (2009, sep). Online wind turbine fault detection through automated SCADA data analysis. Wind Energy, 12(6), 574–593. doi: 10.1002/we.319