NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

DIOGO JORGE GATO REBIMBA

Master in Computer Science

# ARTIFICIAL INTELLIGENCE FOR THE PREDICTION OF FAILURES IN WIND TURBINES

DISSERTATION TO OBTAIN THE DEGREE OF MASTER IN COMPUTER SCIENCE

MASTER IN COMPUTER SCIENCE

NOVA University Lisbon
July, 2022

NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

# ARTIFICIAL INTELLIGENCE FOR THE PREDICTION OF FAILURES IN WIND TURBINES

## DISSERTATION TO OBTAIN THE DEGREE OF MASTER IN COMPUTER SCIENCE

**DIOGO JORGE GATO REBIMBA**

Master in Computer Science

**Adviser:** Nuno Ferreira
*Coordenador de Consultores, CGI IT Portugal*

**Co-adviser:** Carlos Viegas Damásio
*Associate Professor, NOVA University Lisbon*

MASTER IN COMPUTER SCIENCE

NOVA University Lisbon
July, 2022

**Artificial Intelligence for the Prediction of Failures In Wind Turbines**

*Dedicada aos meus pais e amigos.*

# Acknowledgements

Por todas as orientações e esforços que foram feitos para poder entregar esta preparação de tese, gostaria de agradecer ao Professor Carlos Viegas Damásio do Departamento de Informática da Faculdade de Ciências e Tecnologias da Universidade Nova de Lisboa. Gostaria também de agradecer ao meu coordenador, Nuno Ferreira, colega Fernando Subtil e à empresa CGI IT Portugal, por terem aceite a realização da minha tese de mestrado na empresa, fornecendo-me todas as condições para tal. Por fim, agradecer aos meus pais e família, por me terem proporcionado todas as condições que necessitei para a realização deste curso, por todos os conselhos que me deram e por todos os esforços que fizeram para frequentar esta faculdade e curso. Deixo também uma palavra aos meus amigos mais próximos por me terem acompanhado em todo o meu percurso académico e me terem ajudado sempre que precisei.

# Abstract

With the increase of the global environmental awareness and demand nowadays, the request for renewable energy sources by the companies responsible for distributing the energy itself highly increased as well. The sources, such as wind turbines, are highly exposed to several external factors that can result in mechanical failures on their complex components. These faults lead to multiple consequences due to their failure time: loss of energy production, which means that less renewable energy will be transmitted to the electrical grid and more money will be spent on repairing these components. The content of this thesis consists of an analysis to the implementation of a prediction system allied with a monitoring system. This system will allow failure detection on wind assets, anticipating them and thus reducing maintenance costs and the increase of the longevity of the components. By reducing the number of failures, it will allow companies to increase the profit of the energy production on a long term. With the work that was made, it was implemented an experiment that could be applied to several faults of wind turbines that created machine learning models that reliably predict these faults.

**Keywords:** Renewable energy, machine learning, renewable monitoring system, wind parks, wind turbine, failure prediction

# Resumo

Com a crescente preocupação ambiental existente nos dias de hoje, fontes de energia renovável são cada vez mais procuradas pelas empresas de distribuição de energia. Estas fontes, como as turbinas eólicas, são expostas a diversos fatores externos e a falhas mecânicas dos seus complexos componentes. Estas falhas levam a diversas consequências devido ao tempo fora de serviço: perda de produção de energia, levando a que menos energia renovável seja transmitida para a rede elétrica e também maiores custos de reparação. O conteúdo desta tese apresenta uma análise à implementação de um sistema de predição de falhas aliado a um sistema de monitorização. Este sistema irá permitir a deteção de falhas nos ativos eólicos, antecipando as mesmas e assim reduzindo todos os custos de manutenção, aumentando a longevidade dos componentes. A redução das falhas irá permitir à empresa aumentar o lucro derivado da produção de energia a longo prazo. Com o trabalho realizado, foi possível implementar uma experiência que pode ser aplicada às diversas falhas de turbinas eólicas que crie modelos de aprendizagem automática que consigam prever estas mesmas falhas de um modo fiável.

**Palavras-chave:** energias renováveis, aprendizagem automática, sistema de monitorização de renováveis, parques eólicos, turbinas eólicas, previsão de falhas

# Contents

# List of Figures

# List of Tables

# Glossary

**Balance of Plant** "power engineering which refers to the various supporting auxiliary components of a power plant system required to produce energy. BoP systems provide the sup-port needed to keep the plant running stably and efficiently" [24] 10

# Acronyms

**KPI** Key Performance Indicator 22

**RMS** Renewable Management System 1, 3, 4, 6, 7, 14, 21, 22, 23

**SVM** Support Vector Machine 18

# 1

# Introduction

## 1.1 Context and Motivation

Over the past two decades, many governmental entities have implemented a wide array of policies aimed to reduce the $CO_2$ emissions of the electricity sector by increasing the market penetration of renewable energy technologies [41]. Due to this, there is an increase on the research of new, safe, and sustainable green power technologies, such as wind-based ones. The primary focus of renewable energy technologies research is to convert renewable resources into electrical energy to feed the utility grid or consumer loads [28].

Despite all the technologies that exist to assist the monitoring of renewable energy systems, failures in their assets are still one of the major problems that decrease the potential of those assets. "Ensuring that wind turbines perform at their optimal level over their lifetime (usually 20-25 years) costs around 25% of the offshore installation" [38]. Also, "technical faults that require unscheduled maintenance are increasingly important" and represent a large part of the total of failures on wind turbines [23].

This is where the CGI Renewable Management System (RMS) enters. Monitoring systems are essential to maintain optimal performance of renewable systems assets. With this master thesis work, we are trying to improve CGI RMS system by implementing a fault detection tool, to help our system clients in detecting possible failures soon and thus, improving the overall performance of their systems and reducing maintenance costs.

## 1.2 Problem Definition

Renewable energy technologies, like wind turbines, are based in complex mechanic components that are subject to multiple external conditions. These external factors allied with the complexity and degradation of the components of wind turbines, may lead to failures in the assets that compose a power plant. These failures lead to downtimes in the production of energy to the grid and repairs, that cost money and resources to the owners of the power plants.

In wind farms, many failures of wind turbines are unpredictable. According to [17] and [23], major faults in wind turbines that cause more than one day of downtime, only represent 25% of all the failures that occur in a wind turbine. Despite this low percentage, these major faults are responsible for 95% of the total downtime that occur in wind turbines.

A wind turbine is composed by complex components interconnected between them. The complexity of each component varies and consequently the mean time between failures (MTBF) and cost of repair of each one also changes. In [5], this is exactly what is analyzed, considering as a base a 2.3 MW onshore wind turbine. Considering the alarms history (by hours) from two years, it was recorded the failures duration for different assemblies. As a cost reference, we selected two of the variables that [5] used: (i) the C0, that represents the cost of the non-delivery energy and (ii) the Cf (Cost of Failures), that represents the cost of failure of the component. With these variables presented before, in Figure 1.1, we present a chart from the data collected in Table 1 of [5] and that correlates the number of hours of failure and the different costs (C0 and Cf) from each assembly that composes a wind turbine.

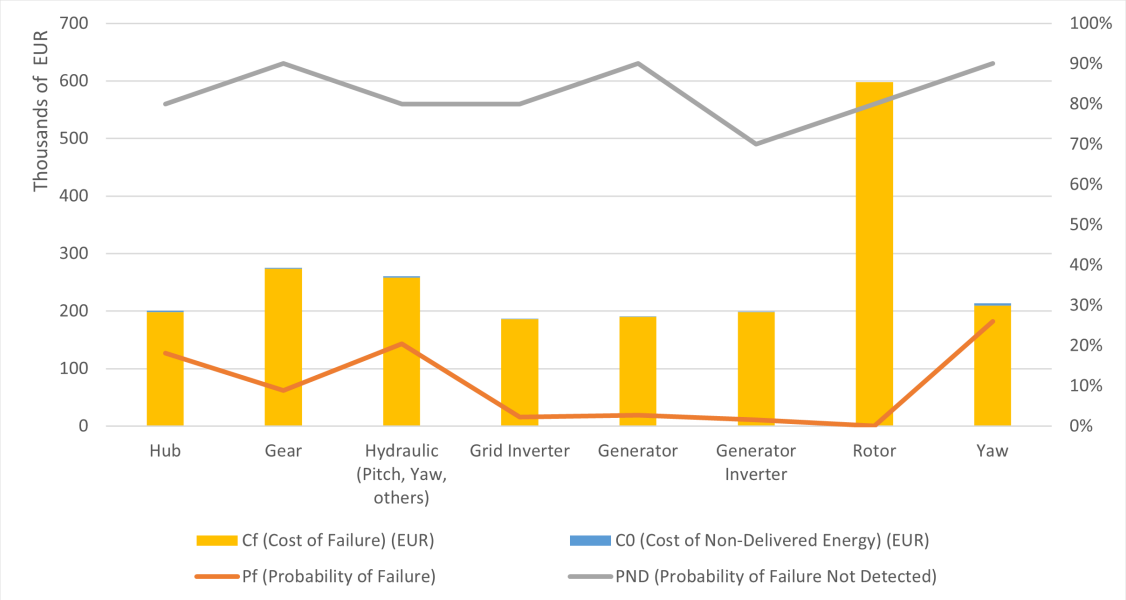



Figure 1.1: Costs of non-delivered energy and failure correlated with the Probability of Failure and Probability of Failure Not Detected for each component of a Wind Turbine. Data Source: [5]

As observed in Figure 1.1, the probability of failures that are not detected, in all key components that are represented, are in average above 80% . Despite the fact of actual failures in the Rotor (that is the component that the total cost is higher) being almost 0%, the next two components that have the higher cost of failure, Gear and Hydraulic failures, have a probability of failure of 9% and 20%. The costs of these, two combined, surpasses the 500,000 €, that is a much higher cost compared to the costs of maintenance

and monitoring of the components.

In [32], it is presented an analysis of the costs of failures in wind turbines, compared to the costs of a correct OM (Operation and Maintenance) (including predictive maintenance). According to this study, the cost of O&M during a 20-years lifetime in a 2 MW onshore wind turbine in a 200-MW project, is around 1.7M € and in a 4,14 MW offshore wind turbine in a 600-MW project, the cost is around 12.6M €. These costs were in addition to the cost for lost energy production during the failure and maintenance period.

Further in this article, it is presented that "employing a condition monitoring system would result to savings of 190,000 € for a 3 MW wind turbine". If no preventive maintenance is employed, the cost of the corrective maintenance is 436,299 €, whereas the "total O&M cost would be 66,129 € if there are two preventive maintenance events per year per turbine in its entire life", according to the average presented by OM databases [32]. This result is a total of 370,170 € reduced costs, by using a predictive maintenance.

## 1.3 Goals

To maximize energy production, increase availability, control energy losses and improve overall operational performance, CGI Renewable Management System (RMS) [13] features an integrated set of tools. This master thesis study has the objective to improve this system, more particularly to study the base of the tool Failure Prediction from the Predictive module by using Machine Learning. By extracting and studying the data available, our goal is to analyze all the steps needed to build a machine learning experiment in Azure Machine Learning Studio, that will work as a foundation to predict faults in a wind turbine and to be used in the Failure Prediction tool, thus providing to the client a tool that predict exceptional situations and failures in the operation of renewable energies assets.

With the use of CGI RMS, we will have available real-time and historical data from real assets that exists on the field. With this data, we will be able to build a data-driven based method for fault detection with the help of machine learning algorithm and a pipeline. We will study, implement, and adapt supervised learning algorithms that are adequate to failure prediction, study the historical data available in CGI RMS from multiple signals and feed them to the previous referred algorithms. We will analyze the obtained results, comparing them against the historical data that we have available, so that we can check which algorithms and set of features are the more adequate to predict faults of multiple wind turbines. To conclude our work, we will then prepare these build models to be feed to the Failure Prediction tool available In CGI RMS.

To achieve this, we are going to divide the study of our thesis in five different scopes:

1. First, we are going to select the type of exceptional situations and failures in renewable energies assets we shall try to predict.

2. Then, we are going to study data so that we can identify the variables that allow us to predict such situations. These variables can be signals from the field and historical data events (like downtimes and faults that were registered) and we will do some previous analysis on the signals that may be related to each fault.

3. We will perform some feature engineering, by using the Azure Machine Learning Studio built-in modules to analyze which features have more correlation with the fault events, building new features from the initial set of variables and to analyze our historical failure events that we have available to confirm that we have an unbalanced training set (since, naturally, the number of failures events is less that the normal operation events) and prepare a future step on balancing the training set for model training.

4. By using the features identified on the previous step, we are going to develop the machine learning models for each wind turbine fault, that can be integrated with CGI RMS to help to predict and classify the major failures according to the real-time data that the system monitors.

5. After our models are trained and we acquire an output from our experiments, we will analyze and present our conclusions of our built models and experiment steps, and present the future improvements that must be done in order to implement the Failure Prediction tool on CGI RMS.

## 1.4 Document Structure

This master thesis document will be organized as the following:

1. First we will provide a background of what this master thesis will focus. We start by the monitoring system that provide us the data to build our machine learning models, the CGI RMS, then we advance to provide some context of the area were the study focuses, the wind turbines and we will finish by providing a basic introduction on the machine learning area, that is the area that will be used to accomplish our master thesis goals.

2. Then, we will provide a basic introduction on the technologies that are going to be used to accomplish our work.

3. We advance through an analysis of the main literature that was used to guide our work and backup our decisions of the procedures that were used when building our datasets and experiment.

4. After this, we introduce our methodology by presenting our data, our dataset and faults, the machine learning algorithms that were used in this study and all our

experiment procedures that were used to build our dataset and to build and evaluate our machine learning models.

5. Then, we present our experimental results, with the results of all the steps of our experiment using the metrics presented earlier and the best hyperparameters and model from each fault scope.

6. We conclude this master thesis document by analyzing the previously presented results and taking some conclusions of these results.

# 2

# Background

## 2.1 Introduction

In this chapter we will provide to the reader the necessary background. We will start by introducing the system in which our thesis work will be implemented and that provide us the data that will be used. Then, we will advance to a theoretical introduction to our main analysis theme, the wind turbines, providing a first introduction to its context in nowadays, how their downtimes are classified and their major components. We will finish this chapter by introducing the area that we will use to accomplish our goal, machine learning.

## 2.2 CGI RMS - Renewable Monitoring System

CGI Renewable Monitoring System (RMS) is a system that "comprises an integrated set of tools to maximize energy production, increase availability, control energy losses, and improve overall operational performance with direct impact on business revenues".

Its scope includes all the 3 major renewable energies: wind, solar and hydro. In 2019, it had multiple clients with a total combined portfolio of over 8,000 generators, and approximately 15 GW installed capacity distributed over 3 continents: North America (with 5,495 MW) South America (with 1,261 MW) and Europe (with 8,089 MW) [13]. It has 4 major modules [13]:

1. Monitor: with a screen with possible dashboards and tools with real time data, allows to assist both managers and field teams in acquire all the necessary information to manage their assets.

2. Analyze: provides a global overview of portfolio performance by providing trend analysis, benchmark indicators and key operational dashboards and reports.

3. Predict: by using machine-learning algorithms and continuous monitoring of real-time data, allows to anticipate potential failures. It allows also to build digital twins and use them as a baseline for behavior comparison.

4. Operate: this module is designed for control rooms. It provides multi-screen with a clear real-time overview of all assets and a step-by-step workflow to guide operators, remote control, and intelligent alarm management.

CGI RMS stands-out by providing real-time portfolio supervision (represented in Figure 2.1 by the monitoring dashboard) and control, automatic generation and sharing of reports and data exports, key performance indicators calculated and updated according to the real-time data, long-term storage of historical data for performance analysis and tools to analyze the data (like the Data Analysis tool, represented in Figure 2.2) and performance of their assets according to what it should be expected with dashboards like the power curve analysis, among other features.


Figure 2.1: Dashboard of general monitoring. The user can have an overview of the state, performance and notifications and alarms that exist on their parks and assets.Source: [12]

## 2.3 Wind Turbines

Our study will focus into one of the major renewable energy technologies: wind turbines. This renewable energy source is increasingly year by year. As we can see in Figure 2.3, in 2019 in the whole World combined, there was a growth of 12,14% in terms of wind energy generation. In Portugal only, these numbers are similar, with a total growth of 8,48% in terms of wind energy produced, compared to the previous year. According to the latest date of IRENA (International Renewable Energy Agency), the global investment in renewable energy technologies is 46,56% in wind energy [34].

The data available in terms of the grow in these two renewable energy sources, are major factors that motivate the development and improvement of technologies like the CGI RMS and studies like the one that was developed in this Master Thesis.

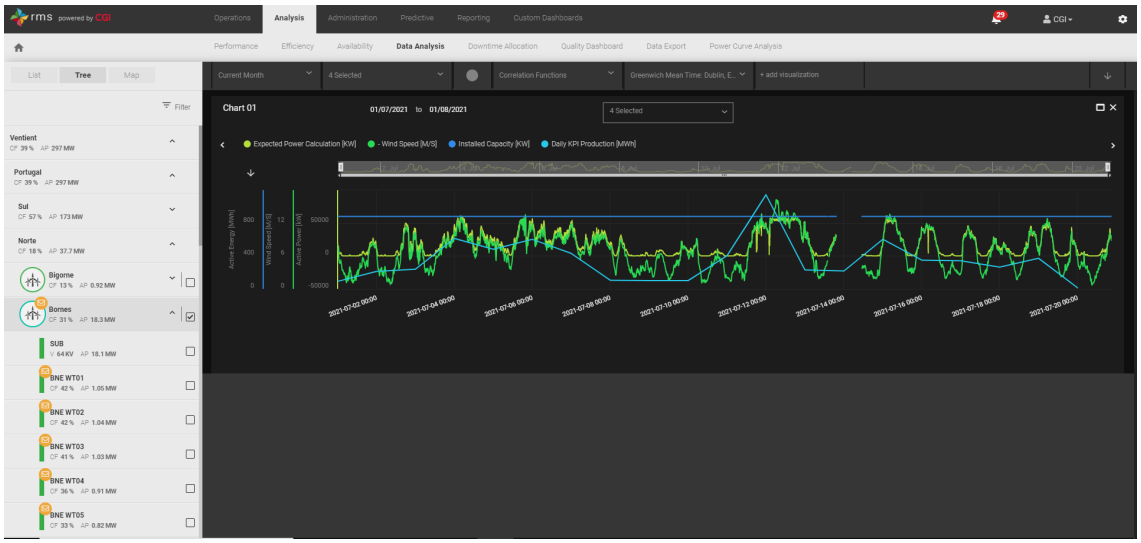



Figure 2.2: CGI RMS Data Analysis tool. It allows to select certain variables, key performance indicators and signals and compare their values in a certain time interval. It also allows to apply a correlation function on those variables. Source: [12]

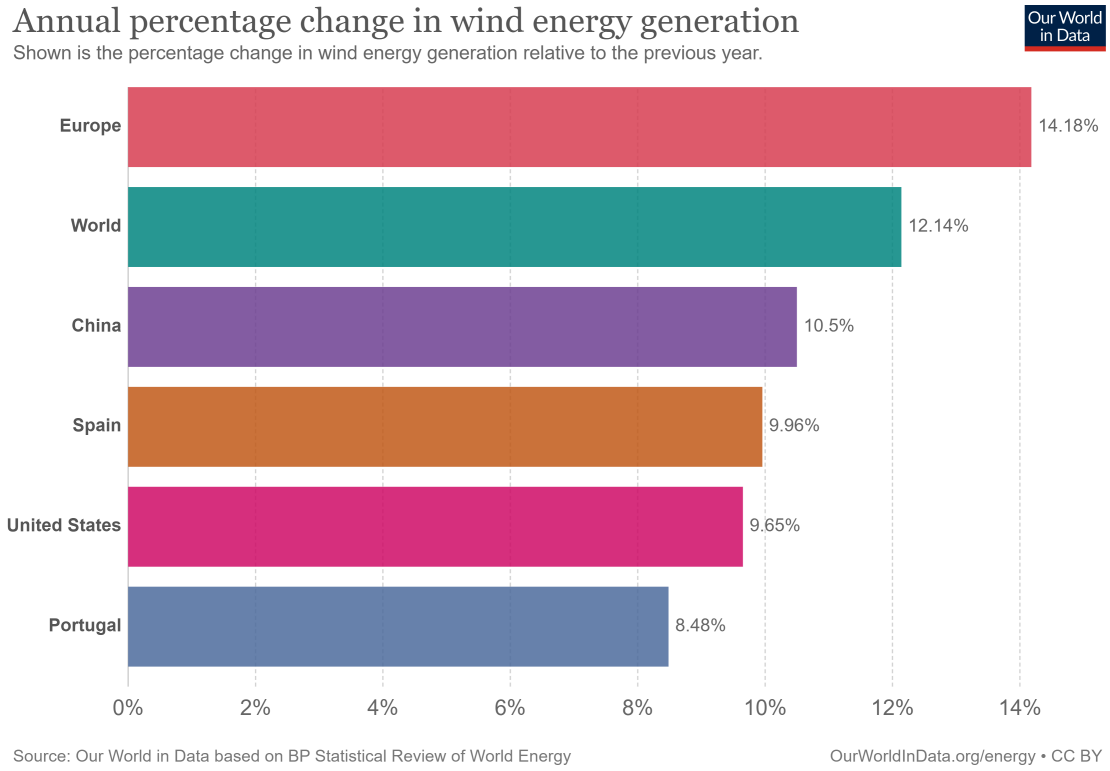



Figure 2.3: Shown is the percentage change in wind energy generation relative to the previous year, according to OurWordInData. [34]

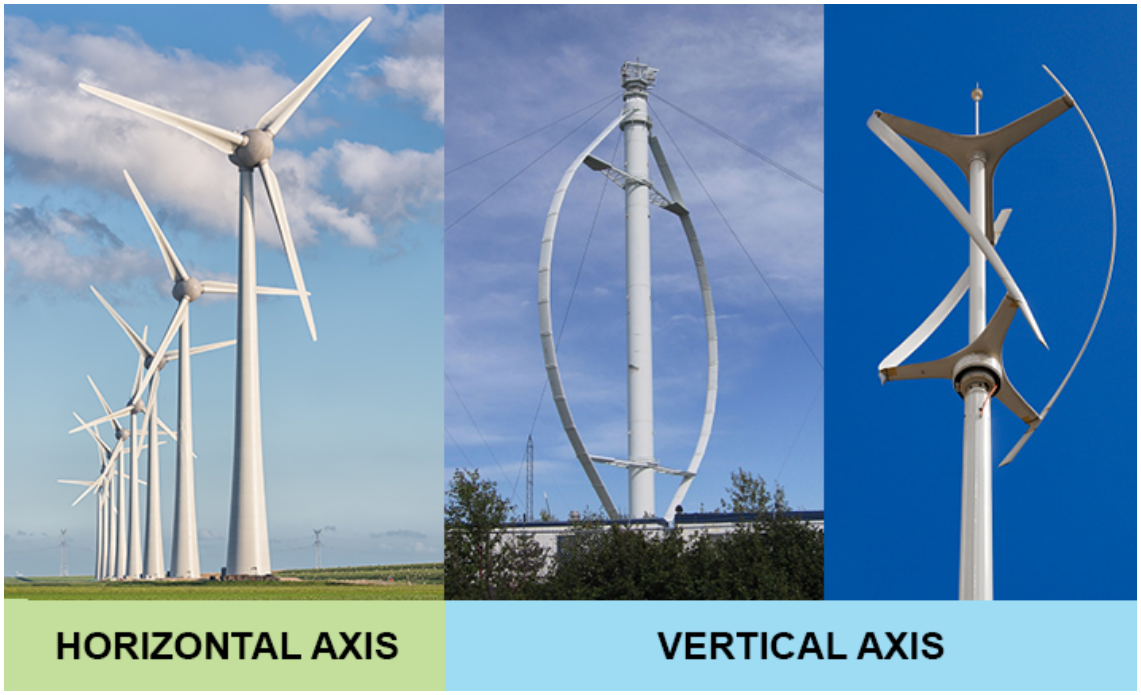



Figure 2.4: Different types of Wind Turbines: horizontal (left) and vertical (right) Source: Kid's Korner (apogee.net)

In this chapter, we are going to introduce the basics of each of the wind turbines that allow the reader to acquire a general knowledge of them and some key concepts that will provide the basics to understand this Master Thesis goals. We will start by introducing how a wind park is composed, presenting the major components of each of their asset and finalizing with the factors (internal or external) that may cause failures in these same assets.

The wind brings kinetic energy associated with it. This energy, when passes through the wind turbine, rotate the blades and consequently the rotor. With this, the kinetic energy is transformed into mechanical energy that can acquired and transformed into electric energy and supplied to the grid [16] [40]. Wind turbines may be classified as two major types: horizontal-axis turbines and vertical-axis turbines, represented in Figure 2.4 in the left and right images, respectively. The more common are the horizontal-axis turbines, that are also composed with gearboxes to accelerate the slow rotation of the blades, to make it strong enough for the generator. The length of the blades is the biggest factor that determines the amount of energy that a wind turbine can generate.

A wind power plant, also known as wind farm, is the combination of multiple turbines. A wind park may have hundreds of wind turbines, like the Capricon Ridge Wind Farm, in Texas, United States, that have more than 400 wind turbines making it a park with an installed capacity of 662,5 MW [33]. As we can see in the charts of Figure 2.5 in terms of wind energy sources, in 2019, the electricity generated per year in the World was 1.416,95 TWh and 13,69 TWh in Portugal. The total installed power worldwide was 622,25 GW

and in Portugal was 5,22 GW [34].

### 2.3.1 Downtime Classification

Faults in a wind turbine, causes unscheduled maintenance and consequently, loss in the production of energy. A wind turbine has many and complex components that, without the required maintenance and a predictive system of their conditions, may cause the wind turbine to stop working and, may cause the total failure on a component that may cause costs in terms of their replacement, as well as downtime on a wind turbine, which leads to a drop in production. Downtimes in a wind power plant are classified in two major categories, like in the wind power plant:

1. Internal Causes: These internal downtimes can be caused by wind turbine direct failures or internal Balance of Plant failures. These failures are registered also as planned, unplanned or caused by civil or electrical repairs, with each of these being classified in a lower level of the specific cause of the failure, like wind turbine maintenance, failures, among others.

2. External Causes: The external factors that may cause downtime to a wind power plant are grouped as external to the wind turbine, external Balance of Plant failures or curtailment downtimes. The specificities of these failures may be caused by environment conditions, exclusion causes, operations on the grid, among many others.

The failures that we are going to predict in this thesis are the ones concerning internal causes downtimes.

### 2.3.2 Major Components

The wind turbine and its blades form the base component that capture the wind, in order to, and with the help of other components, transform it into electric energy supplied to the grid. In Figure 8, we present an example of some of the main components that compose a wind turbine.

The wind cause lift and rotation of the blades. With this, the rotor [ Figure 2.6 - 1], that is composed together by the blades and hub, starts to spin. Since, most of the times, the wind Itself does not make the rotor spin enough to produce energy, the gearbox rotor [ Figure 2.6 - 3] that "connects the low-speed shaft to the high-speed shaft and increases the rotational speeds from around 30-60 rotations per minute (rpm), to about 1.000-1.800 rpm [ Figure 2.6 - 2 and 4, respectively]. This is the rotational speed required by most generators [ Figure 2.6 - 5] to produce electricity". To maintain the performance and guarantee that the high speed intensity may cause some failures and damages in the wind turbine, the anemometer and controller [ Figure 2.6 - 6] together, allows to control the machine according to the wind speed: "Starts up the machine at wind speeds of about 12

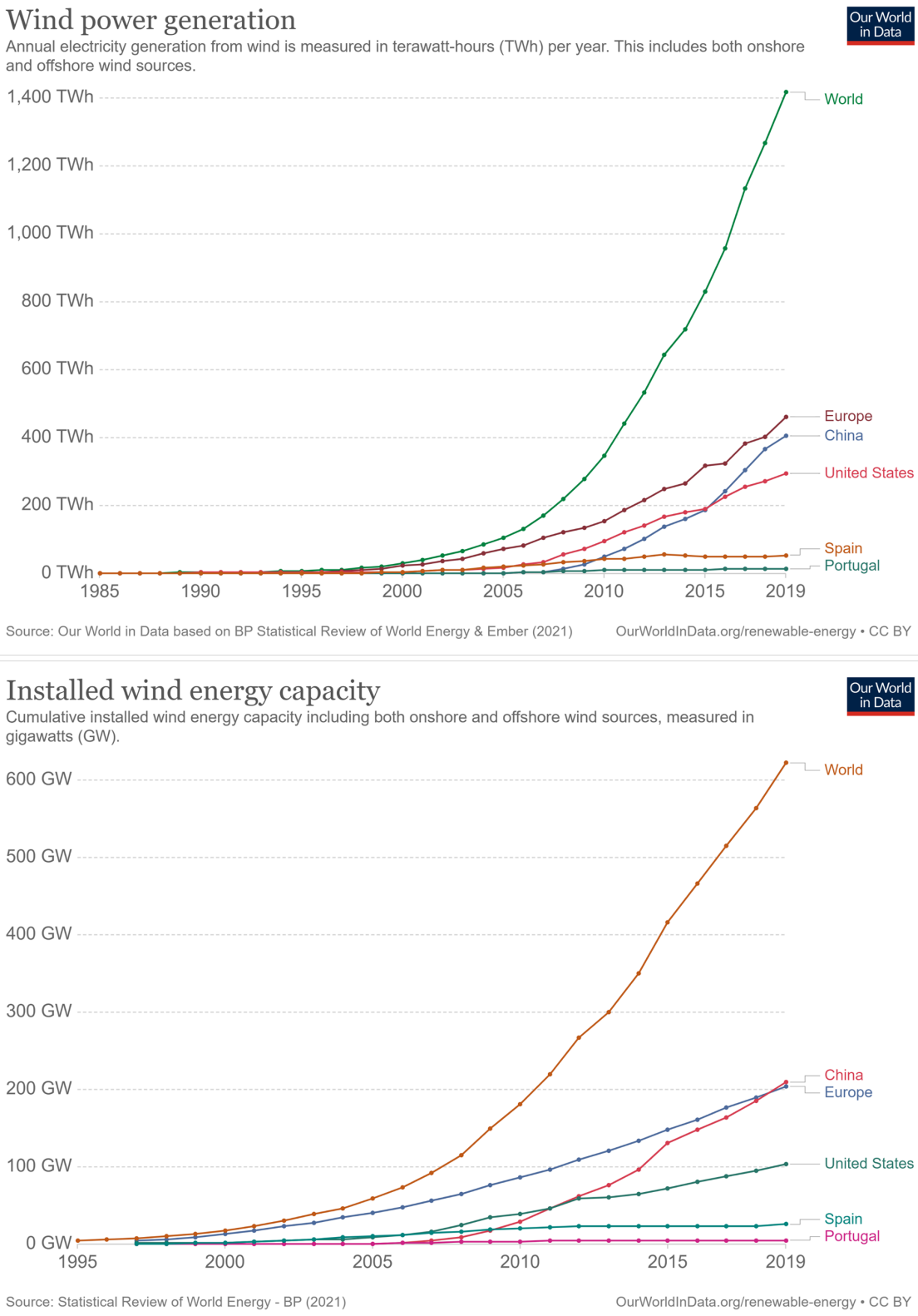



Figure 2.5: Electricity generated from wind energy sources (up image) and installed wind capacity (down image) in 2019 in two continents (World, Europe) and four countries (China, United States, Spain and Portugal). [34]

to 25 kilometers per hour (km/h) and shuts off the machine at about 88 km/h. Turbines do not operate at wind speeds above about 88 km/h because they may be damaged by the high winds." [16]. These components and others, that have a more secondary role, form the nacelle [ Figure 2.6 - 7].

The high complexity and cost of the gearbox, makes this system part one that is trying to be improved: if electricity can be produced by lower rotational speeds, then the necessity of gearbox disappears and, consequently, the price of a wind turbine drops and the faults that are caused by problems of this component also disappear. The major component that makes part of the wind turbine tower is the yaw [ Figure 2.6 - 8] (composed by the yaw drive and yaw motor): this component, in the overall, "orients upwind turbines to keep them facing the wind when the direction changes" [16].

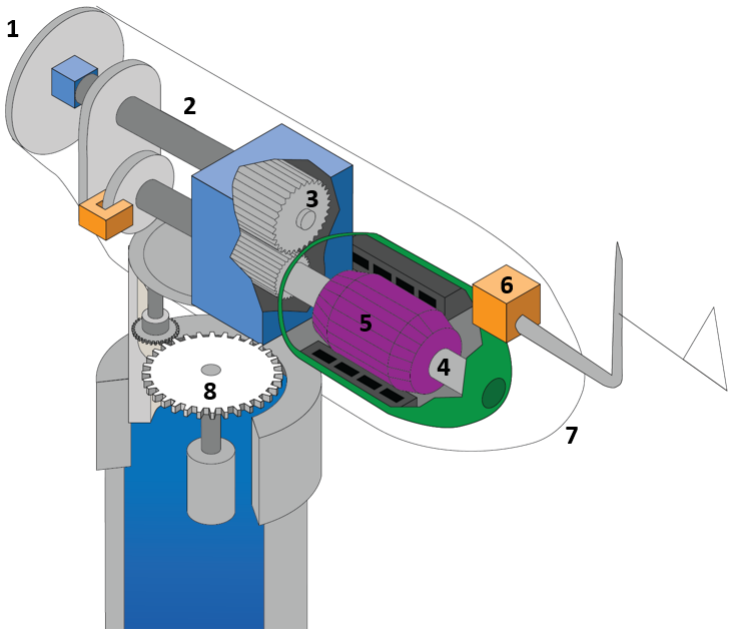



Figure 2.6: Wind turbine main components. [1] Rotor, [2] Low-Speed Shaft, [3] Gearbox, [4] High-Speed Shaft, [5] Generator, [6] Controller], [7] Nacelle, [8] Yaw (with Yaw Drive above and Yaw Motor below. Source: [16]

## 2.4 Machine Learning

"Machine learning is the science of building systems that improve with data"[26]. By providing data into a machine learning, the model can analyze it and discover patterns that are useful for understanding relationships in data. This model must pass through a learning process so that it can improve its accuracy and reliability in providing correct outputs [38] [15].

The taxonomy of ML models is divided into two major types, as represented in Figure 2.7. In each taxonomy we also present the most common algorithms that are used. Some

of these algorithms are going to be used in this thesis work.

1. Supervised Learning The main goal is to predict an output variable using labeled data as input. One important goal of these models is their adaptation to new input. To do that, we don't want our model to be overfit, meaning, too much adapted to our training data. We always want our model to be as general as possible, so that the "supervised model defines the real 'general' underlying relationship". A supervised learning algorithm may be a regression model, that predicts a numeric variable or a classification model, that it should classify the inputs into a set of classes.

2. Unsupervised Learning These models work only with input data, not having an output labelled data to compare. The main goal is to provide complex patterns that are hidden within data without any labels. The more used type of unsupervised learning is clustering. Clustering is the task of dividing the sample into a number of groups in which the data points from the same groups are similar to each other and dissimilar to the data points in other groups.

ML also have two less used types and that are also more complex to implement: semi-supervised learning, that as the name indicates, it is a mix between supervised and unsupervised learning. It is used for cases that we mix a small amount of labelled data with a much larger unlabeled dataset; reinforcement learning, that is implemented as Pavlov's dog study: occasional positive and negative feedback teaches the model to takes better decision in the next actions, meaning, in the next predictions [15].

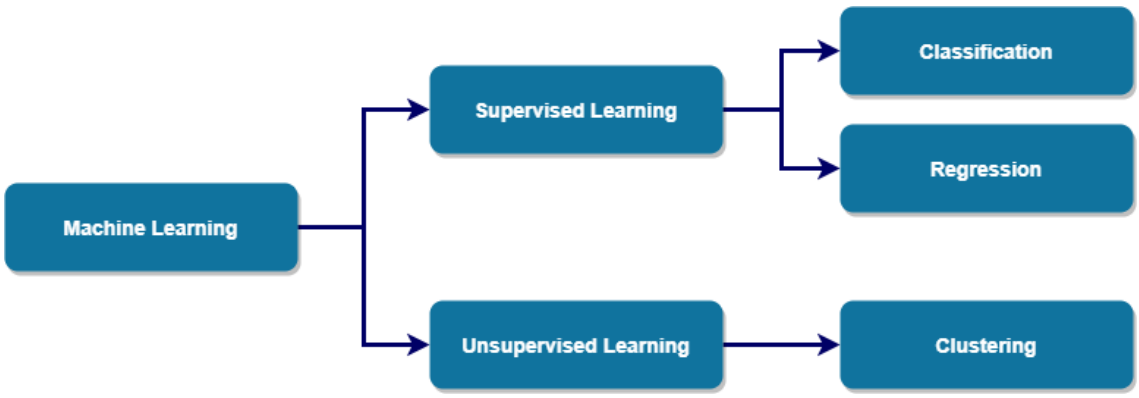



Figure 2.7: Machine Learning taxonomy of the main types. Source: [26]

There exist 3 basics terms that we should know to understand ML [15]:

1. Dataset: The set of data that it is provided to train the model.

2. Features: "Important pieces of data that help us understand a problem. These are fed in to a Machine Learning algorithm to help it learn."

3. Model: It is the representation of what the ML algorithm has learnt during the training process from the data set and the features provided.

The machine learning methodology follows 5 major steps: data collection and preprocessing; feature selection and extraction; model selection; validation; fitting the model [38] [15].

Another subfield of machine learning is deep learning. Deep learning is based in a layered structure of algorithms called artificial neural network that makes intelligent decisions on its own. One important detail of these types of algorithms is that they need a large amount of data to become more efficient that the typical supervised algorithms. The larger the dataset, better the algorithm will surpass the traditional machine learning algorithms [10] [14].

## 2.5 Conclusion

In this chapter we could understand how CGI RMS works, their main tools and its portfolio. It was also presented all the basic terms and operation of a wind turbine, how downtimes are divided, which is a basis of the classification of the failures that we are going to study and how the key components work together in a wind turbine, allowing us to understand the complexity of a wind turbine, from the nacelle to the yaw, and the importance of monitoring and predicting a failure in a component, so that all the other components that are connected don't be affected and consequently damaged, and the energy production can carry on without losses due to downtimes. It was introduced the machine learning area that will be used to implement anomaly detection models. Considering all the machine learning types, we can understand that the machine learning algorithms that should be used are the classification ones because our goal is exactly to understand in which cases, we are going to have a failure, or set of failures, and when we will have a normal operation time, for which we have labelled data.

# 3

# Technologies

To accomplish our thesis goal of building a failure prediction tool in CGI RMS, the following main tools will be used.

## 3.1 Azure Machine Learning Studio

Cloud service used for implementing and managing machine learning projects. With this tool, you can create a model or use a model built from an open-source platform [30]. If needed, it can be used Python SDK to develop in a more specific way a more adjusted model to respond to our problem, not being attached only to the built-in classes of models that Azure Machine Learning Studio provide us [30]. This tool provides us a graphical user interface to check up all the outputs obtained according to the data that we provide and results metrics that we defined previously [30]. All the models created are made easily available through endpoints. CGI RMS have already prepared several web services to contact to new endpoints created by this tool, providing to the created models the necessary data (transformed directly from the database system to excel files), training and run these models and store the results back in the RMS database.

This will be the mainly used tool to make part of the feature engineering process, to build the machine learning models and to analyze the results of the machine trained models.

## 3.2 Visual Studio in .NET, REST API and Azure Functions

CGI RMS is built in a microservice architecture developed in .NET. The monitoring tool is based on a REST API interface, that provide a set of endpoints to the web portal. The other microservices are setup in azure service fabric clusters and are triggered, for example, with queue messages. Some RMS processes, like the predictive methods that are used to train and store the outputs of configured ML models, are setup in Azure Functions. These methods are triggered with time or queue activators.

We will use this technology mainly to make a console application to generate all the datasets needed to train and evaluate the machine learning models.

## 3.3 SQL Server Management Studio and Azure Blob Storage

CGI RMS is based on two storage providers: a relational database SQL server, that we explore by using SQL Server Management Studio and a non-relational database used to store unstructured data (like excel files), the Azure Blob Storage. The SQL Server is more used to store almost all the data used to feed the RMS web portal, all the client's static data, calculated data and the statistical non-raw data. The Azure Blob Storage is used to store blobs, meaning, the non-structured data like excel files and other data that have a bigger size. It is also used to store all the raw data that comes from the signals.

These are the tools that are used to achieve and store all the data needed.

# 4

# State of the Art

## 4.1 Introduction

In this chapter it will be presented a review on the articles that we consider as basis to support the methodology chosen. We will divide this section into several parts, according to the steps to develop a machine learning model. We will first introduce the references that were used to auxiliary the decision of our dataset and the pre-processing mechanisms that were decided. Then we will advance to the articles that support our feature engineering process. We advance to the machine learning algorithms that were chosen and conclude with the references that support the evaluation mechanisms that were decided to be used in order to rank our training models.

## 4.2 Literature Analysis

### 4.2.1 Features

In several articles like [39] [6], and as stated by CGI experts, vibration signals are one of the most important features that can be used in failure prediction in mechanical systems. In our particular case, we don't have access to vibration signals. In [44], for the prediction of "high temperature fault", it is presented in Table 1 several signals that it was used in this article, in which we can see that several temperature signals from several wind turbine components were chosen as well as signals that provide general information about the state of a wind turbine (like rotor speed, blade pitch angle and power). In [43] and [35], several sensors with generic information about a wind turbine were used for the fault detection. These sensors provide information like generated electrical power, rotor speed, generator speed and pitch angle. It is stated also that "the main challenges of the wind turbine fault detection lie in their non-linearity, unknown disturbances, and significant measurement noise at each sensor", providing an important feedback concerning the difficulty to detect standards between failures and specific component states. [42] studies two failures, one related to blade angle and another concerning generator. For each failure, the features selected were some general ones (like rotor speed, generator speed,

wind speed) and some specific of the component of the failure, like component sensors or sensors of related components (for the blade angle failure, blade and nacelle sensors; for the generator failure, generator, bearing and nacelle sensors). For the second failure there were used also used signals of the temperature of the generator and shaft bearing. In [27], it is considered as a parameter related to the fault the generator speed, power output and wind speed. Finally, in [11], for the prediction of generator and gearbox failures, mostly all the sensors concerning temperatures and oil temperatures were used as well as general ambient and wind turbine information like average wind speed, total production, ambient temperature.

### 4.2.2 Dataset Preparation and Feature Engineering

Classification problems, more particularly fault prediction problems, have several dataset preparation work that is make in order to improve the performance of the fault detection. In almost all of our literature that present some dataset information, we see some common approaches.

In [44], it is referred the removal of data outliers, by using Chebyshev inequality method, the scaling of the features, using normalization and after selecting a subset of signals that are related to the failure, it is performed a feature selection using a feature ranking using correlation coefficient. It also uses a sliding time window on the features.

In [43], it is pre-processed the features by "group scaling and feature transformation", reducing the dimensionality of the feature space using "multiway principal component analysis". The next step is to 10-fold cross-validation using SVM based classification, in order to assess if results will generalize to an independent dataset. This article also refer that "methods that focus on a specific part of the WT (Wind Turbine), require the choice of the most appropriate sensors, their advisable position in the sub-assembly".

[35] only refers the importance of reducing a large number of features and the use of feature extraction and feature selection techniques. It uses also the 10-fold cross-validation for data split.

[42] and [27] states that in a study like ours, of failure prediction, there exists an unbalanced dataset. The number of faults events is much lower that the number of normal state events. That is something that should be treated on the training dataset to balance the dataset that is used to train the model. [29] uses, for feature selection, Pearson correlation to eliminate the more correlated features and then use "PCA to identify the variables that exceeds the threshold".

[19] and [20] highlight one of the most important dataset transformations that can be used in machine learning problems such as the one of our master thesis. It refers for the dataset sliding window technique that allow the machine learning model to learn the behaviour of the wind turbine a set of time before the failure actually occurs.

Other important technique referred by [20], and more briefly by [31] is the running summaries. This consists in building new features from an statistical aggregation (for

example, average) of the last set of values of each feature. This allow us to compare the current value of the feature with the average behaviour of the feature, discovering values that are not considered as normal and thus indicating a possible failure.

In addition, [20] describes also some data pre-processing techniques that were also referred in the previous articles, like the removal of the null and duplicated rows, ways to deal with the split of data for the train/validation/test datasets.

### 4.2.3 Machine Learning Algorithm's

The machine learning algorithms that are mentioned by our literature to address our master thesis theme, are the same across all the literature.

We have to mention that not all of the articles use exactly the same techniques to predict failures as we use, and this have influence on the algorithms that are chosen. Another important factor is that we are restricted to use the algorithms that are available in Azure Machine Learning Studio, factor that influence the use of the algorithms and their parameters, thus affecting their performance.

Neural Network is introduced in [38], [4], [27] and [11]. Decision Tree algorithms are analyzed in [27], [35], [42] and [N7 (WIND)]. Logistic Regression ( [6], [31], [11] and K-Nearest Neighbors ( [35], [42], [29] are simpler algorithms that are also used but the one that is more mentioned in our literature is the Support Vector Machine algorithm, which is refereed in [27], [44], [35], [39], [43], [42] and [11].

Articles like [7] and [22] also discuss in detail some of these algorithms as the ones to be used for classification problems. Some of the algorithms mentioned by these two articles are K-Nearest Neighbor, Logistic Regression, Decision Tree Classifier, Random Forest Classifier, Support Vector Machine and Naive Bayes Classifier.

One algorithm that is also debated and that is said to be one of the most adequate for our problem in particular, is the Long Short Term Memory (LSTM) [19]. Although this analysis from our literature, we will not use it because it is not available in Azure Machine Learning Studio.

### 4.2.4 Evaluation Metrics

For binary classification problems, machine learning area documentation and our literature suggest a set of metrics to use. The metrics that are discussed in these documentation don not have any specific consideration in terms of business needs. In our particular case, we have a particular interest in reducing the number of false positives, even if that lead to a higher number of false negatives and a lower number of true positives. This idea is justified in more detail in the next chapters.

[40] mention the use of the accuracy, false positive rate, true positive rate, ROC curve and confusion matrix. [23] uses ROC curve and AUC curve. [42] uses ACC (classification accuracy), SEN (sensitivity) (also known as recall) and SPE (specifity), which are metrics that are derived from the confusion metrics and that provide us percentage values.

[27], [7], uses the most common metrics for evaluating binary classification problems: true positives, false negatives, true negatives and false positives, and percentage values that are calculated from the previous mention values, accuracy, recall (mentioned earlier as SEN), precision and F1 score.

In [20] and backed up by our CGI expert, we can't rely only on these metrics to evaluate a running algorithm when we are using the sliding time window technique. For this case, we want to build our own evaluation technique that should rely on "additional logic and business heuristics"[20] to understand if in a time window, the fault was predicted or not, and not only if the algorithm predicts all the timestamps that were flagged as belonging to the sliding time window of a failure. More on these heuristic will be explained in the next chapters.

## 4.3 Conclusion

Considering the literature previously mentioned and the advices from CGI experts, we built our methodology to address the problem of our master thesis with the technology that we have available.

In terms of the features that were selected for each fault, it was decided to use general data signals, like wind speed, external temperature, active power of the wind turbine for all the faults. Then, for specific signals of each fault, we selected the signals of the fault component or components that were neighbors of the fault components.

In terms of the dataset and feature engineering step, considering the limitations of the technology that we had to use, for the feature selection, we used the Filter Based Feature Selection module, with the Chisquared metric. The Principal Component Analysis was discarded due to limitations of the Azure Machine Learning Studio that doesn't have a built-in module to use the Principal Component Analysis (PCA). In the built of the dataset, we remove invalid rows (rows which the features are null) and features (when the feature have to much null values), select only data that are considered to have quality (meaning that have no noise) and build the running summaries of the features. Further one, we use the train-test split to train, validate and test our model.

In terms of the machine learning algorithms, and considering all the machine learning algorithms available in the Azure Machine Learning Studio, we selected the Two-Class Logistic Regression, Two-Class Support Vector Machine, Two-Class Neural Network, Two-Class Decision Tree and Two-Class Boosted Decision Tree.

Finally, to evaluate our models, we use as metrics Precision, Recall and F1 Score. Also considering the [20], we built a script to understand the real capacity of the model in predicting faults and run it into a dataset that was not used in the construction of the model.

# 5

# Methodology

## 5.1 Introduction

As presented before, our approach will use a data-driven based method that involves less restriction in knowing all the details of the underlying fault process and the technical knowledge of all the components [44]. With the large amount of historical data that is available in the CGI RMS, all the conditions are met to start developing the work with this method.

Since our goal is to build an generic experiment in Azure Machine Learning Studio that may be used as a foundation for the Failure Prediction tool, and due to the limitations of the signals that we have available and the built-in modules available in Azure Machine Learning Studio, we analyzed the literature and confronted it with the modules available in order to obtain the best experiment possible to build the best model to predict the failures of the wind turbines that are being studied.

## 5.2 Data Structure

Following the previous knowledge obtained with RMS CGI, the data model presented in Figure 5.1 and Figure 5.2, provides a global overview of the classification of an event into a failure of a certain type, and of how data from assets are obtained and stored.

Before all the data acquisition begins, the client (owner of the parks) provides all the information needed to monitor the complete wind park. The most important data entities for the development of the fault prediction model are:

1. Information about each wind park and their respective wind turbines, which we call assets of the park. Each asset has a determined manufacturer, model, and controller.

2. Components, which are the basic units that forms each wind turbine and that allow us to understand where a certain event occurs.

3. Signals, that exist in each asset or in the park. These signals provide us with different information, like the active power or the wind speed.

4. Faults, that allow us to understand the cause of a failure and the component where it occurs. It is going to be one of the main entities for the model.

5. Statuses, that allows us to associate to each asset their status and acknowledge the status associated with an event. The downtimes presented before may be associated to one or more statuses.

6. Budgets, that are used as monthly expectations that the park owners have on their assets in different type of measures (like production, ratio of performance). Allied with KPI, allow to better understand the performance of the assets.

7. Key Performance Indicators (KPI), that are calculations over signals that allow for a quicker understanding of the performance of the wind turbines. They may allow more than one source of information (like signals, budgets or constants) to provide a final value that directly indicates the performance of a wind turbine or park.

Other data that is provided to us, including some classifications and indicators that the system already have in its basis, are adapted to the specificities of each client.

In terms of the process of data acquisition, it involves all the entities represented in Figure 5.1. It begins with an event, registered with a certain state. Events are one of the most basic units registered in our system. RMS has a state machine, that uses these events and static data presented before like faults and states, allowing us to calculate the energy loss, duration, and number of events that occur in a certain time interval, and present the downtime of an asset and the classification of that downtime, meaning, the cause for that failure to occur. After starting, an event lasts until a change on the wind turbine occurs, meaning that a single event may last for days (for example, for an event that states that a wind turbine is running may last for days until something occurs that makes the wind turbine stops, closing the previous event that states the wind turbine is running and starting a new event stating that the wind turbine is stopped).

Data from assets, allows us to understand the values of a certain asset during a time interval. In figure 5.2, we have the representation of all the entities that are involved in the calculation and storage of the data that the RMS CGI acquires from the wind turbines. The data can have two origins: real-time data from field signals, or post processing data that uses the previous ones to make some calculations as soon as it is received (for example, expected power in wind turbines is calculated using wind speed, neighbors data, besides other data). The real-time data that is sent by the wind turbines are then received by a linked system that supports RMS CGI and that is responsible to aggregate them statistically in 10 minutes values (the number of minutes depends on each signal and client). This means, that in the statistical storage, we will have registered daily 144 values (one value per 10 minutes), each of these values represents the aggregation over the last 10 minutes, with the average value, minimum value, maximum of value and the standard deviation of the last 10 minutes of that signal for a specific timestamp. To

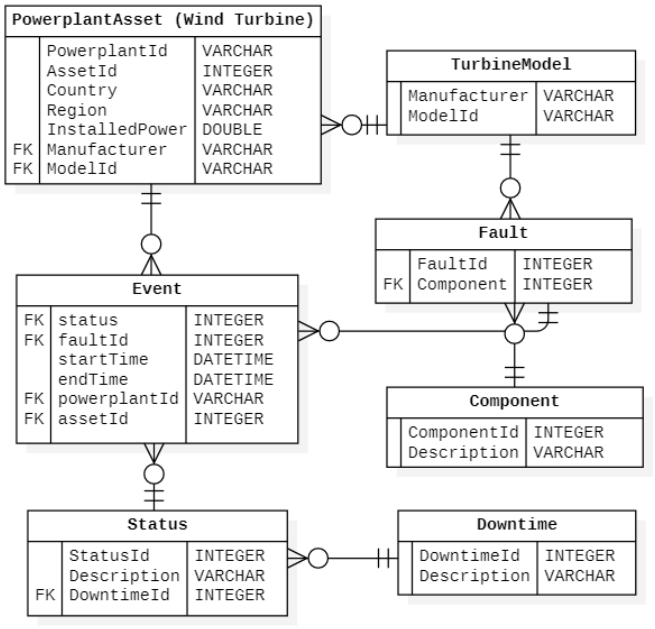



Figure 5.1: RMS Data Model of Data Acquisition Entities: It represents all the entities that are involved in the data acquisition process and the storage of failure events.

help in better structuring the signals comprehension, in RMS we use a term tag that works as a string mask of the signal. With this field, it becomes easier to search for all the signals of multiple assets that concern a certain measure (for example, using the tag 'GENACTIVEPOWER', that represents the active power of a wind turbine, we can obtain more easily all the signals ids that provide the information of this measure).

## 5.3 Dataset

This thesis will focus on the data from CGI RMS concerning different parks in different locations of all around Portugal, since the March 5th, 2021 until the end of that same year. This is the time interval of data that the current version of the RMS CGI has been running with all the configurations needed. Data that is stored in CGI RMS from previous years, came from a previous version of the CGI RMS system that was transitioned to the current system during this master thesis, and due to the fact that the processes of previous versions and current versions are not exactly the same, it has been decided that it shouldn't be used for the current work.

With the data available for those parks, a first analysis on the faults was made to understand two main questions in this thesis: 1) do we have enough quality data to develop the machine learning models? 2) how can we predict the failure with the available monitoring data?

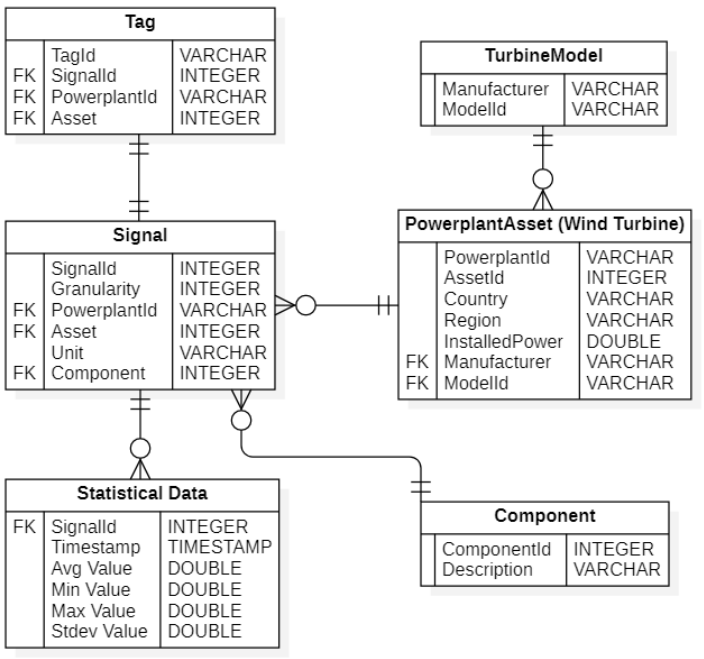



Figure 5.2: RMS Data Model of Storage of Data: It represents all the entities that are involved in the data acquisition, storage and statistical calculation of the data from the wind turbines.

### 5.3.1 Available Data

As stated previously, in terms of data available in CGI RMS and classified as quality data, we have one major client portfolio of data that has different parks in different locations in Portugal. This client provides data from 36 wind parks, with a total of 323 generators. These wind turbines have different manufacturers and models: a combination of 6 manufacturers and 13 turbine models compose these 323 generators. This client has configured 16,730 fault types and 31,970 different signals. Considering, for example, July 2021 and one of the faults and wind turbine presented in Table 5.3, we have registered 184 events that concerns to that particular failure and wind turbine.

As presented earlier in the Data Structure section and in image Figure 5.1, we see that we can map to each failure event the component where the failure occurs and the energy loss and duration associated to that failure event, allowing us to classify the impact that the failure has in terms of financial loss and time to fix it. By knowing the exact moment when the failure was recorded, we can correlate that data with the signals data from that component and asset, to understand how the wind turbine was behaving itself before the failure occurred, and to train the machine learning model to predict failures when those same or similar conditions occur.

### 5.3.2 Signals to use

To predict a failure, as any machine learning algorithm, we will provide samples of the dataset to train the algorithm. From all the signals available in the system, we selected a subset of some of the features that we will use as a starting point to train our model. The selection of each feature was supported by articles like [27] and by the opinion of CGI experts of renewable energy sources.

Each signal available depends on the manufacturer and model associated with the fault that is being studied. So, to support the study of each fault, the features for each one might not all be the same and so this might be a factor that affects the performance of the model for the fault being predicted.

Despite the particularities of each model, a set of general features is selected for each fault. These features, displayed in Table 5.2 and mentioned in [27], are general information about the environment and the wind turbine.

Table 5.1: Wind Turbine General Information Signals. This signals exists for every wind turbines in the system.

| Component | Unit | Description | Turbine Component |
|---|---|---|---|
| WNAC_WDSPD | M/S | Wind Speed | GENERAL / NACELLE |
| WGEN_W | KW | Active Power | GENERAL / GENERATOR |
| WNAC_WDDIR | DEG | Wind Direction | GENERAL / NACELLE |
| WNAC_EXTMP | DEGC | External Temperature / Temp outside / Temp. Ambient | GENERAL / NACELLE |
| AIR_DENSITY | KGM3 | Air Density | GENERAL |

We then selected some signals that are specific of each component of the fault itself. These features presented in Table 5.2, are features that exist for the majority of the manufacturers and models of the subset of faults and provide information specific of the component that they are connected to. Some of these signals are mentioned by some articles of our literature that studied failures that affect some of these components also.

In the last analysis on the features that we can use to provide more information about the wind turbines components, we made a specific analysis on the signals that we had available for each manufacturer and model. These signals are specific of some component of the wind turbine and are not available for all the wind turbines. Since these signals are available depending on turbine manufacturer and model, this may lead some machine learning models of failures of specific turbines to contain features that have key data for the prediction of a failure and, consequently, perform better in terms of predicting those failures. Some of the signals available are signals that have information about the current of the converter, temperature of several components and oil of these components, voltage of the transformer and generator. The complete list of signals is available in the Appendix 1.

Some signals that are mentioned in our literature and by CGI experts as important for the prediction of problems in wind turbines are vibration signals. Due to the fact that

Table 5.2: Wind Turbine Component Signals. These signals are specific for some manufacturers and models, and only exists for a subset of wind turbines.

| Component | Unit | Description | ManufId | ModelId |
|---|---|---|---|---|
| WGEN_SPD | RPM | Generator Speed | ALL | ALL |
| WTRM_HYOILTMP | DEGC | Hydraulic oil temperature | F01 / F02 | N60 / N90 / V90 |
| WTRM_COOLTMP2 | DEGC | Temp cooling water return | F01 | N60 / N90 |
| WTRM_HYOILPRES | BAR | Hydraulic oil pressure | F01 / F02 / F06 | N90 / V90 / MM100 |
| WTRM_HYOILTMP | DEGC | Hydraulic oil temperature | F01 / F02 | N60 / N90 / V90 |
| WNAC_INTLTMP | DEGC | Nacelle Internal Temperature | F01 / F02 / F06 | N60 / V90 / MM100 |
| WROT_PTAGVALBL | DEG | Blade 1, actual value | ALL | ALL |
| WROT_ROTSPD | RPM | Rotor Speed | ALL | ALL |

the wind turbine manufacturers don't provide them, we have to use other signals data that we have available, like temperature signals (gearbox oil temperature, etc.), that are considered to be one of the best alternatives to the vibration signals to predict failures in components, as said by CGI experts and in [44].

### 5.3.3 Dataset Structure

In terms of the dataset structure that will be introduced to the azure machine learning model, it will have the following columns. Some of these columns are only provided to have some metadata of the rows to be used for some intermediate steps (like the split of data).

1. TS (Datetime): provide information about the datetime of the row.
2. PowerplantId (String): the id of the wind turbine park.
3. AssetId (String): the id of the wind turbine.
4. Feature (one column per feature) (Double): the 10 min value of the feature for that particular timestamp. Some of the features column available are new variables created from others that are explained later in the Experiment Procedures-Feature Engineering chapter.
5. Fault (Boolean): indicates if the timestamp is in the sliding time window of failure. If it value is true, it means that the timestamp is inside the time window of one failure, and that the values of the feature of that same timestamp should be considered as indication that a failure will occur. Later, in the Experiment Setup chapter we will explain on how this column is built, as well as the size of the time window.

6. Failure Time (Boolean): indicates that in that timestamp a failure occurred. This column is specifically important in the evaluation of the final test dataset, to check if our machine learning model predicted with anticipation that failure time.

7. Fault Category (String): it was built to identify each failure event and to group them when each failure time is close to another failure time. In other words, indicates an id of a failure. The rows that do not correspond to a failure (indicating normal state of the wind turbine), will have an Fault Category value that indicates the next failure event. This means, that all the timestamps with a normal state of the wind turbine (Fault = 0) will have a Fault Category value equal to the next fault state of the wind turbine (Fault = 1).

Each row of the dataset represents one timestamp of the wind turbine, and the correspondent values of the features and the fault state for that particular timestamp. In Figure 5.3, we can see an example of the dataset, with a set of rows starting in the 1st of October, 2021 from 07:20 to 10:30, with the values of the features ('WNAC EXTMP avg' and 'WGEN W avg'), the fault column that have the sliding time window of 72h in this case, and the failure event that occurred at 08:50.

| ts | powerplant | assetId | WNAC_EXTMP_avg | WGEN_W_avg | fault | failureTime | faultCategory |
|---|---|---|---|---|---|---|---|
| 01/10/2021 07:20 | SCA | 34 | 19 | 1252.1 | 1 | 0 | 167 |
| 01/10/2021 07:30 | SCA | 34 | 19 | 1252.1 | 1 | 0 | 167 |
| 01/10/2021 07:40 | SCA | 34 | 19 | 1252.1 | 1 | 0 | 167 |
| 01/10/2021 07:50 | SCA | 34 | 19 | 1252.1 | 1 | 0 | 167 |
| 01/10/2021 08:00 | SCA | 34 | 19 | 1252.1 | 1 | 0 | 167 |
| 01/10/2021 08:10 | SCA | 34 | 19 | 1252.1 | 1 | 0 | 167 |
| 01/10/2021 08:20 | SCA | 34 | 19 | 1252.1 | 1 | 0 | 167 |
| 01/10/2021 08:30 | SCA | 34 | 19 | 1252.1 | 1 | 0 | 167 |
| 01/10/2021 08:40 | SCA | 34 | 19 | 1252.1 | 1 | 0 | 167 |
| 01/10/2021 08:50 | SCA | 34 | 19 | 1252.1 | 1 | 1 | 167 |
| 01/10/2021 09:00 | SCA | 34 | 15 | 2030.3747832 | 1 | 0 | 167 |
| 01/10/2021 09:10 | SCA | 34 | 15 | 1997.2377865 | 1 | 0 | 167 |
| 01/10/2021 09:20 | SCA | 34 | 15 | 2119.1185754 | 1 | 0 | 167 |
| 01/10/2021 09:30 | SCA | 34 | 15 | 2103.1287661 | 1 | 0 | 167 |
| 01/10/2021 09:40 | SCA | 34 | 15 | 2018.070706 | 1 | 0 | 167 |
| 01/10/2021 09:50 | SCA | 34 | 15.6859623 | 1918.0560848 | 1 | 0 | 167 |
| 01/10/2021 10:00 | SCA | 34 | 16 | 1896.097697 | 1 | 0 | 167 |
| 01/10/2021 10:10 | SCA | 34 | 16 | 1268.9130178 | 1 | 0 | 167 |
| 01/10/2021 10:20 | SCA | 34 | 16 | 1622.0667564 | 1 | 0 | 167 |
| 01/10/2021 10:30 | SCA | 34 | 15.7971793 | 1560.9872094 | 1 | 0 | 167 |

Figure 5.3: Example of the Dataset structure with all the columns: ts, powerplantId, assetId, two features ('WNAC EXTMP avg' and 'WGEN W avg'), fault, failure time and fault category. As explained previously in the fault category, rows prior to 07:20, that have the fault = 0, will have the same faultCategory, 167.

## 5.4 Faults to Predict

In terms of the faults that were selected to be predicted, we started our analysis by ordering them with a first criteria of the number of events, thus meaning the faults that have more occurrences and consequently have more fault data to feed to our models. The

second criteria was the total energy losses that the fault led to, thus meaning the direct impact in terms of lost of production and consequently the potential loss of revenues that the fault provoked to the client. We also filtered all the faults that are classified as alarms since these types of faults are not really faults, being alarms that the wind turbine provide to indicate a state of the wind turbine and that may be related to a maintenance or an alarm that is provoked by a component fault. These components faults are the ones that we will try to predict. The final selection criteria of the faults is to try to select the wider variety of manufacturers and model of faults and components, so that our study covers as many types of failures as possible. Our base number of top faults to study is around 10, but that number may vary according to the results of our analysis.

Initially when we started to analyze the best way to predict a specific failure in a portfolio of parks, we took the more generic approach possible. This means that we first considered about combining in our dataset all the data of all the wind turbines of the same manufacturer and model in the prediction of a fault of that same manufacturer and model. A second approach, restricts the model to the wind turbines of the same park and that share the same manufacturer and model. Then after our first tests and after talking with the CGI experts and according to the feedback that a client of CGI RMS as given previously, we understand that this generic approach is not correct mainly due to the following reason: even in the same park, each wind turbine varies from the others since each wind turbine is exposed to different external conditions (like wind direction), does not have always the same production (sometimes some wind turbines are limited in producing in comparison of wind turbines in the same park) and each component of the different wind turbines doesn't have the same level of degradation.

So considering this point of view and the opinion of a client of CGI RMS, our final approach in terms of the range of variety that each model should be built, and justified by what has been told previously, was to construct a different model to predict a single failure of a specific wind turbine. This way our model will be more fitted to a wind turbine and the particular fault, thus resulting a prediction that will direct the client maintenance team to the particular component of the failure and to a particular wind turbine.

So, according to the filter and order criteria specified before, we obtain the following results presented in Table 5.3, that represent the top failures of the portfolio of parks that are going to be studied in this master thesis, in order to understand the correct approach to predict a failure in a wind turbine and thus, being these models the base model for the implementation of the Failure Prediction tool in CGI RMS.

The column 'Park', represents the id of a wind park. The column 'Asset', represents the id of the wind turbine for that park and the column 'FaultId', represents an id for the fault.

After selecting the top faults, while the analysis was being made, we decided to include for the same faults, some wind turbines which have a lower number of occurrences of faults events in order to understand if our model can have a similar power of prediction

Table 5.3: Top Faults to be Analyzed

| Park | Asset | faultId | nrEvents | energyLosses [KWh] | duration [s] |
|---|---|---|---|---|---|
| SCA | 34 | 9112 | 573 | 28054 | 4561720 |
| SCA | 10 | 8487 | 325 | 348869 | 1449668 |
| BNE | 14 | 768 | 278 | 3476 | 1022380,828 |
| PAP | 35 | 8749 | 209 | 8510 | 1699280 |
| SCA | 28 | 8719 | 162 | 52329 | 1862142 |
| TMS | 6 | 14474 | 108 | 9564 | 1157738 |
| BNE | 18 | 768 | 102 | 801 | 277801 |
| CHF | 28 | 768 | 89 | 1760 | 1093046 |

in case of wind turbines that have a substantial lower number of fault events. This subset of faults are presented in Table 5.4.

Table 5.4: Subset Faults

| Park | Asset | faultId | nrEvents | energyLosses [KWh] | duration [s] |
|---|---|---|---|---|---|
| PLS | 9 | 140 | 45 | 636 | 556990 |
| CHF | 15 | 367 | 19 | 65773 | 478232,528 |
| PCA | 7 | 140 | 81 | 1805 | 733797 |

According to the information of the signals available, provided in the dataset section, and the faults selected previously, in the Appendix 1 we can see the complete set of features available for each fault. These features are based on the first analysis presented in Section Dataset.

For each of the faults presented in Table 5.3 and Table 5.4, we did an analysis on its distribution per month in order for us to understand the predictions that we should expect in each dataset used for training, validation and testing. That distribution information is presented in Table 5.5 and Table 5.6. In the Section Experimental Setup we will get into more detail in this step of distribution of the faults per each dataset.

## 5.5 Machine Learning Algorithms

In [38] [1] [25], it was studied, in a theoretical way, several possibilities of machine learning algorithms that are more adequate to achieve our main goal: the prediction and classification of failures that may occur in a wind turbine asset.

Using [38] [32] [1] [22], [21] and according to the recommendation from the CGI Machine Learning experts and the limitations of the algorithms that are already provided to us by the Azure Machine Learning Studio, the machine learning algorithms that we will focus this thesis are: Two-Class Logistic Regression, Two-Class Support Vector Machine (SVM), Two-Class Neural Network, Two-Class Boosted Decision Tree, Two-Class Decision Forest.

The machine learning algorithms will be used, with a dataset that contains the measures mentioned before for each fault and the other columns mentioned previously in

Table 5.5: Faults Distribution per Month (March to End of July 2021)

| Park | Asset | faultId | mar/21 | apr/21 | may/21 | jun/21 | jul/21 |
|---|---|---|---|---|---|---|---|
| SCA | 34 | 9112 | 0 | 0 | 27 | 99 | 184 |
| SCA | 10 | 8487 | 0 | 0 | 38 | 237 | 6 |
| BNE | 14 | 768 | 0 | 5 | 31 | 217 | 6 |
| PAP | 35 | 8749 | 0 | 0 | 3 | 43 | 16 |
| SCA | 28 | 8719 | 0 | 0 | 0 | 12 | 111 |
| TMS | 6 | 14474 | 0 | 0 | 4 | 2 | 58 |
| BNE | 18 | 768 | 0 | 0 | 0 | 0 | 0 |
| CHF | 28 | 768 | 0 | 0 | 0 | 0 | 0 |
| CHF | 24 | 434 | 1 | 4 | 2 | 18 | 4 |
| PAP | 22 | 8749 | 0 | 0 | 0 | 4 | 0 |
| PLS | 9 | 140 | 0 | 0 | 2 | 14 | 9 |
| CHF | 15 | 367 | 3 | 1 | 0 | 1 | 2 |
| PCA | 7 | 140 | 0 | 0 | 1 | 7 | 7 |

Table 5.6: Faults Distribution per Month (August to End of December 2021)

| Park | Asset | faultId | aug/21 | sep/21 | oct/21 | nov/21 | dec/21 |
|---|---|---|---|---|---|---|---|
| SCA | 34 | 9112 | 139 | 78 | 44 | 2 | 0 |
| SCA | 10 | 8487 | 2 | 4 | 10 | 8 | 0 |
| BNE | 14 | 768 | 4 | 2 | 0 | 13 | 0 |
| PAP | 35 | 8749 | 5 | 118 | 12 | 12 | 0 |
| SCA | 28 | 8719 | 26 | 13 | 0 | 0 | 0 |
| TMS | 6 | 14474 | 20 | 4 | 8 | 12 | 0 |
| BNE | 18 | 768 | 0 | 85 | 17 | 0 | 0 |
| CHF | 28 | 768 | 27 | 11 | 45 | 6 | 0 |
| CHF | 24 | 434 | 4 | 7 | 0 | 4 | 0 |
| PAP | 22 | 8749 | 0 | 16 | 26 | 0 | 0 |
| PLS | 9 | 140 | 2 | 12 | 2 | 4 | 0 |
| CHF | 15 | 367 | 4 | 3 | 5 | 0 | 0 |
| PCA | 7 | 140 | 2 | 32 | 12 | 20 | 0 |

the Dataset section. The target feature that will be used to fit our model is the column fault, mentioned earlier in the dataset section. The main idea with this master thesis is to investigate and implement several machine learning models, each one with the function of predicting a single fault type of a wind turbine and being fitted for particular sliding time window parameter (explained later in this section), so that it can provide to the client which component of the specific wind turbine will fail and anticipate a preventive maintenance to minor the damage of a potential failure. Another important goal of our models performance is to prioritize the minimization of false positives in comparison with predicting the maximum of true positives possible since we want our model to be reliable and not lead to maintenance's that are not needed.

### 5.5.1 Classification Algorithms Studied

#### 5.5.1.1 Two-Class Logistic Regression

The logistic regression is, at heart, a regression model. Logistic regression can be used as a classifier with the "purpose of obtaining a decision hyperplane that separates different classes"[26]. The logistic regression, uses a function, denominated as a cost function, that takes value between 0 and 1 and that tell us the probability of a certain value being from one class or another. Thus, it is more applicable to binary problems. This classification algorithm is the simplest to implement of all four and so, and because it is more advised to be used in binary classifiers, it is a good candidate to be an algorithm used to investigate single failures instead of a combination of failures from the same component.

In terms of its implementation in Azure Machine Learning Studio, we have available two parameters:

1. Optimization tolerance "Specify a threshold value to use when optimizing the model. If the improvement between iterations falls below the specified threshold, the algorithm is considered to have converged on a solution, and training stops"[30]

2. L2 Regularization Weight "Regularization is a method for preventing overfitting by penalizing models with extreme coefficient values. Regularization works by adding the penalty that is associated with coefficient values to the error of the hypothesis"[30]

#### 5.5.1.2 Two-Class SVM (Support Vector Machine)

It is a binary classification learning algorithm where each data item is a point in a n-dimensional space (where n is the number of features provided) and the classification is performed by finding the hyperplane that ideally linearly separates the two classes. The support vectors are data points that are closer to the hyperplane and influence the position and orientation of this hyperplane. Using the support vectors, we maximize the margin of the classifier. [18]

One of the key parameters of the SVM classifier is the kernel function that is used. The kernel function is a function that "give us the inner product of the transformed vectors as a function of the original vectors"[26], meaning that it is the function that converts the input data space into a higher-dimensional space. The kernel function "takes two data points and calculates a distance score between those. This score is higher for closer data points and vice versa."[18]. The more popular and the kernel functions that we are going to study are Linear Function, Polynomial Function and Radial Basic Function (RBF).

The key parameters that should be studied in order to find out the values that are more adequate to our dataset is gamma, that defines how far the influence of a single training example reaches and C, that behaves as a regularization parameter in the SVM, in terms of margin of acceptance for the decision surface (lower C values lead to allowing

more errors). It is a good algorithm when we have a high number of features having a disadvantage of being slow. It is suited for extreme case binary classification and when the classes are separable. Due to all this presented earlier, it seems to be a good algorithm for cases of component failures, where we have a lot of signals from that component, and when the amount of data is not high, because of being slow when the data set has a high number of samples. If these component failures don't happen at the same time, meaning that they are separable, it is also a good candidate.

In terms of its implementation in Azure Machine Learning Studio, we don not have available the specification of which kernel function to use neither some parameters mentioned earlier. In the Microsoft documentation it is not specified which kernel function is used, so we cannot have the complete information of how this module is implemented in the built-in module made available in Azure Machine Learning Studio. For configuration of the hyperparameters of this module, we have available two parameters:

1. Number of Iterations "Denotes the number of iterations used when building the model. This parameter can be used to control trade-off between training speed and accuracy."[30]

2. Lambda "Value to use as the weight for L1 regularization. This regularization coefficient can be used to tune the model. Larger values penalize more complex models."[30]

3. Normalize Features "Before training, data points are centered at 0 and scaled to have one unit of standard deviation."[30]

#### 5.5.1.3 Two-Class Neural Network

A neural network, is an algorithm based in deep learning, as presented before in the background section. As presented in the Azure Machine Learning Studio documentation [30], a neural network is a set of interconnected layers. The inputs are the first layer, and are connected to an output layer by an acyclic graph comprised of weighted edges and nodes. Between the input and output layers you can insert multiple hidden layers. (...) The relationship between inputs and outputs is learned from training the neural network on the input data. The direction of the graph proceeds from the inputs through the hidden layer and to the output layer. All nodes in a layer are connected by the weighted edges to nodes in the next layer. To compute the output of the network for a particular input, a value is calculated at each node in the hidden layers and in the output layer. The value is set by calculating the weighted sum of the values of the nodes from the previous layer.

In terms of its implementation in Azure Machine Learning Studio, we have available two parameters:

1. Hidden Layer Specification "Type of network architecture to create."[30]. It value can only be a Fully connected layer, where we have one hidden layer, being connected with the output layer and the input layer.

2. Learning Rate "Define the size of the step taken at each iteration, before correction. A larger value for learning rate can cause the model to converge faster, but it can overshoot local minimal."[30]

3. Number of learning iterations "specify the maximum number of times the algorithm should process the training cases."[30]

4. The momentum "specify a weight to apply during learning to nodes from previous iterations"[30]

5. Shuffle Examples "shuffle cases between iterations"[30]

#### 5.5.1.4 Two-Class Decision Forest Classifier

Decision Forest algorithm is based on Decision Tree. Decision Tree is a type of supervised algorithm that starts with a node that receives all the input features and that splits recursively based on those features. It predicts the value of a variable by extracting simple rules from data properties and learning those rules. Each split at a node is chosen to maximize information gain minimize entropy.

In terms of the decision forest, it is an ensemble learning method, thus meaning, that relies on multiple models, fixing in each iteration the mistakes from the previous build model. "This particular implementation of a decision forest works by building multiple decision trees and then voting on the most popular output class"[30]. This module implementation, use in each iteration the same complete dataset but with different starting points.

In terms of its implementation in Azure Machine Learning Studio, we have available two parameters:

1. Resampling Method "Method used to create the individual trees". The methods available are Bagging, in which "each tree is grown on a new sample, created by randomly sampling the original dataset with replacement until you have a dataset the size of the original"and Replicate in which "each tree is trained on exactly the same input data. The determination of which split predicate is used for each tree node remains random and the trees will be diverse"[30].

2. Number of Decision Trees "Maximum number of decision trees that can be created in the ensemble"[30].

3. Maximum depth of the decision trees "Number to limit the maximum depth of any decision tree". It allows to increase the precision of the model, with the risk of leading to overfit. [30]

4. Minimum number of samples per leaf node "Indicate the minimum number of cases that are required to create any terminal node (leaf) in a tree". This way, it allows to control the threshold for creating new rules. [30].

#### 5.5.1.5 Two-Class Boosted Decision Tree

As the two-class decision forest classifier, boosted decision tree is also a type of decision tree classifier. Boosting, is a method that combines many weak learners tree, by each tree learning from the previous one, thus making a strong classifier. "Predictions are based on the entire ensemble of trees together that makes the prediction"[30].

In terms of its implementation in Azure Machine Learning Studio, we have available two parameters:

1. Maximum number of leaves per tree "Indicate the maximum number of terminal nodes (leaves) that can be created in any tree". It allows to increase the precision of the model, with the risk of leading to overfit. [30]

2. Minimum number of samples per leaf node "Indicate the number of cases required to create any terminal node (leaf) in a tree". This way, it allows to control the threshold for creating new rules [30].

3. Learning Rate "Number between 0 and 1 that defines the step size while learning. (...) Determines how fast or slow the learner converges on the optimal solution"[30].

4. Number of trees constructed "Indicate the total number of decision trees to create in the ensemble". [30]

#### 5.5.1.6 Configurable Parameters

For the previous presented algorithms, as enumerated we have a serious of parameters to configure. In the Table 5.7, we present all the parameters that were configured for the algorithms. Some parameters presented are a subset of values. These parameters are the ones that are going to be analyzed using the Tune Model Hyperparameter module, that correspond to the hyperparameters of the algorithms. The static parameter values were chosen by running in an initial version of the experiment, and choosing from the available values, the parameter value that lead to the best results.

## 5.6 Experimental Setup

As presented previously, our main goal is to analyze the features transformations that should be made and build an experiment that can be used for the prediction of the failures of wind turbines. So for that, and according to the modules available in the Azure Machine Learning Studio, we designed an experiment that builds several models, each with the same procedures only differing the machine learning algorithm, so that for each

Table 5.7: Machine Learning Algorithms Set of Parameters Configured in the Azure Machine Learning Studio Experiment

| Two-Class Logistic Regression | |
|---|---|
| Optimization Tolerance | 0.00001; 0.00000001 |
| L2 Regularization Weight | 0.01; 0.1; 1.0 |
| Two-Class SVM | |
| Number of Iterations | 1; 10; 100; 250; 500; 750 |
| Lambda | 0.00001; 0.0001; 0.001; 0.01; 0.1 |
| Normalize Features | False |
| Two-Class Neural Network | |
| Hidden Layer Specification | Fully-connected case |
| Learning Rate | 0.1; 0.2; 0.4; 0.6; 0.8 |
| Number of learning iterations | 25,50,100,175,250 |
| The momentum | 0 |
| Shuffle Examples | True |
| Two-Class Decision Forest Classifier | |
| Resampling Method | Bagging Resampling |
| Number of Decision Trees | 1; 8; 16; 32 |
| Maximum depth of the decision trees | 1; 16; 32; 64 |
| Minimum number of samples per leaf node | 1; 4; 8; 16 |
| Two-Class Boosted Decision Tree | |
| Maximum number of leaves per tree | 2; 8; 32; 64; 128; |
| Minimum number of samples per leaf node | 1; 10; 25; 50 |
| Learning Rate | 0.025; 0.05; 0.1; 0.2; 0.4; 0.6 |
| Number of trees constructed | 20; 50; 100; 250; 500 |

fault and wind turbine we can evaluate each model result and then choosing the best fit for the scope that is being predicted.

### 5.6.1 Data Pre-Processing Procedures

In order to prepare our data, some pre-processing on the measures data that we have available from the signals described earlier in Table 5.1 and 5.2 must be done. The procedures that we selected to build our experience were based in our literature like [21] [35] [42] [27] [29] [20] [31] and according to the modules available in the Azure Machine Learning Studio [30].

First, in the data acquisition step, we analyze if a feature has a high percentage of null values for the entire dataset. If that is the case, we remove the feature because it means that it doesn't have enough quality data and the machine learning algorithm can't have null rows. The next step is to remove the remaining rows that contain any feature with a null value. These steps guarantee that we have a dataset with no invalid rows or features so that the process of training and fitting the model have no errors.

Then we proceed to the feature engineering step, by "create new features from the given dataset or tweak existing features to extract more valuable information"[21], as

stated also in [31] [20]. This procedure is called running summaries as named in [20].

Other step that was already mentioned before is the use of a sliding time window. This is a procedure that is made "to expand the failure or target window. That is, make the dependent variable, not just the day the equipment failed but the (...) appropriate interval leading up to the failure"[20]. This mechanism is also mentioned in Table 2 of [31], with the column Failure and in [44] and [19] were is explained how this new column is generated. As stated in this last article, "one usually does not need to predict the lifetime very accurate, far in the future. Often the maintenance team only needs to know if the machine will fail 'soon'". So in order to do this and as presented in next Figure 5.4, we will use a sliding time window (presented in the figure as 'Z' in the x axis) and we will going to label it as 1 to represent a failure. This way, our model will be fitted to learn that Z time before a failure occur, the values of each feature may represent a behaviour that will lead to the failure of the wind turbine.

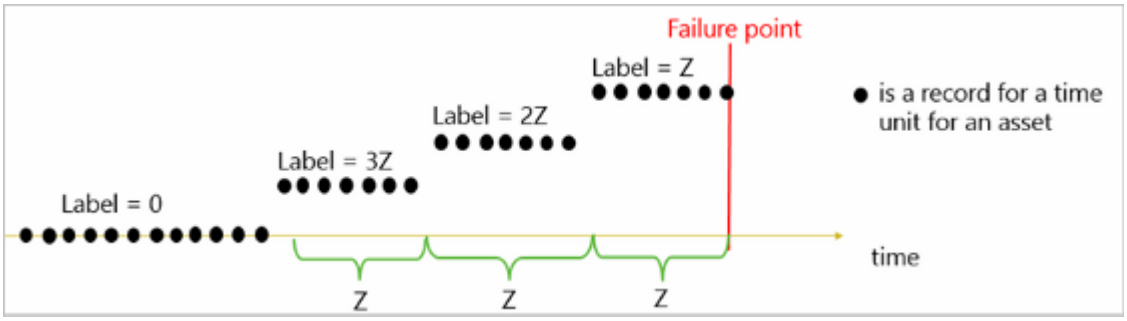



Figure 5.4: Sliding Time Window ... [19]

The next step refereed in [21], [29] and [35] is the feature selection process where we extract only the useful and relevant features. This step "remove the problem of overfitting from your classification model". To do this procedure, we will use the Filter Based Feature Selection module available in Azure Machine Learning Studio, using the two-way chi-squared test. This is a "statistical method that measures how close expected values are to actual results (...) and indicates how far results are from the expected (random) result"[30]. "The (...) higher the Chi-Square value the feature is more dependent on the response and it can be selected for model training"[36]. The number of features that we choose to be selected was 30, and it was chosen by selecting on some of the best faults, several number of features and analyzing the results to check which number of features we obtained the best results of the model.

The next procedure is to split our dataset into the train, validation and test dataset, as mentioned in [37] [2]. In other articles in our literature, like [9] [21] [29] [**41_WIND**] only use two-splits, the train and test dataset. In our case, we will use a more custom split. In the beginning of the experiment we will split our dataset from March 2021 to the 1st of October, 2021 and from that day to the end of the year of 2021. The first split will be used to train, fit and do a first evaluation of the model and the second split will be

to get a total new dataset and simulate the behaviour of the model after being train and fitted, to understand it real score and real capacity for predicting faults. In our first split (March 2021 to 1st of October 2021), we will apply the Split Data module available in Azure Machine Learning Studio twice, so that we can achieve the three datasets that we intended. We expect to obtain 60% for train, 30% for validation and 10% for testing. To obtain that, since we have to use the Split Data module twice, first we split with a fraction of 60%, obtaining the train dataset, and on the remaining 40%, we will split it into 70%, thus giving approximately the percentages mentioned earlier for validation and test.

In terms of the type of split that we choose, we tried two approaches: first we did a randomized split, so that our data is completely unbiased in the rows chosen. Our second approach, by using a stratified split using the Fault Category column mentioned earlier, lead to best results and so was the approach selected. As stated earlier, the failure event, has an id represented by the column fault category (presented earlier in dataset section). This failure event is multiplied by our sliding time window, as stated earlier. By splitting by the fault category, each dataset (the train, validation and test) contains part of each failure event and so our model is trained and evaluated using all the fault events available. This way, we guarantee that our models will use all the failure events in that timestamp, and thus learn the maximum number of patterns of the fault being study as possible. Our validation and test dataset will still evaluate if the model was correctly fitted to all the failure events since in these datasets it is present a portion of each failure event. For example, if we have a sliding time window of 1 hour, our failure event is represented with 6 rows. With this split, our failure event is represented in the train dataset with 3 rows (approximately 60%), that represents 3 timestamps, the validation will contain 2 rows of that failure (approximately 70% of the remaining) and the test will contain 1 row of that failure (approximately the last portion). Since we still have a second not used dataset to obtain our final results (data since October 2021 to the end of that same year), we still have available a complete new dataset to simulate our model behaviour and thus providing us a correct evaluation of our model.

If we perform an analysis on the fault column for one of our faults that have the more number of fault events, and with the higher sliding time window of 72 hours (meaning that for each failure event, we will have 72*6 = 432 rows with fault = 1), for the column fault we have 9663 negatives (represented as 0, that means that the wind turbine was working correctly in that timestamp) and only 1996 positives, demonstrating that we have class imbalance since even with the sliding time window of 72 hours, only approximately 20% of our rows represent a failure condition of the wind turbine. "Class imbalance means one class is dominating and there are very few instances of the other class"[21]. This is normal in problems like the one that is being studied in this master thesis, since a wind turbine should stay most of it time running correctly and not with a specific type of failure. In order to address this problem in our train dataset, we will synthesize new examples of the minority class, than in our case is the fault event. To do that, we will use the SMOTE (Synthetic Minority Oversampling Technique) module that exists in Azure

Machine Learning Studio. SMOTE uses the k-nearest neighbor logic to build examples: "A randomly selected neighbor is chosen and a synthetic example is created at a randomly selected point between the two examples in feature space."[SMOTE].

### 5.6.2 Train and Fit of the Model

After all the pre-processing of the data, we are ready to feed to our machine learning algorithms our dataset, thus creating our trained models. Each algorithm is available in Azure Machine Learning Studio and receives as input the train dataset. These modules configurations receive a parameter range, so that each parameter of each algorithm can receive a list of possible values that are then analyzed, using the Tune Model Hyperparameters module, and the validation dataset. The values selected for each parameter that were introduced were a varied list of values so that we can understand, for all the faults, for each algorithm which parameter values perform better on the overall of the faults. In the previous presented Table 5.7, we can see for each algorithm which values were introduced for each parameter.

### 5.6.3 Evaluation of the Model

In terms of evaluating the model we based our evaluation in some standard metrics that are used in classification problems and that are mentioned in [**41_WIND**] [29] [20] [Classification]. These metrics are the precision, recall and F1 Score. To evaluate the dataset that simulates the behaviour of the dataset with a complete new dataset (with data since October 1st 2021 until the end of December 2021) which allow us to understand the true prediction power of our model, we use the metrics introduced earlier and we build a python script that evaluates the model and if the failure events were correctly predicted inside the time window. This built metric is based according to [20] and the opinion of the CGI experts and is going to be explained later on.

Our standard metrics will be available by combining the "Score Model"module available at Azure Machine Learning Studio, which provides an table with the score of the model in each row, and the Evaluate Model module, that analysis the results of the "Score Model"module and provide several metrics, in which are included all the ones mentioned. The standard metrics will compare the Fault column from our dataset (the column that contains the failure event and the sliding time window rows) and the prediction output from our model.

#### 5.6.3.1 Standard Metrics

The standard metrics mentioned earlier (Confusion Matrix, precision, recall and F1 Score) that we are going to use are common in evaluations of problems of binary classification.

1. Confusion Matrix Confusion matrix is a matrix that provide us four important terms: true positives (when the model predicts YES and the output is YES); true

negatives (when the model predicts NO and the output is NO); false positives (when the model predicts YES and the output is NO); false negatives (when the model predicts NO and the output is YES). As stated previously in this work, we have the priority of minimizing as much possible the false positives and try to have some true positives. The confusion matrix terms forms the basis for the other types of metrics.

2. Precision: it is the number of correct positive results divided by the number of positive results predicted by the classifier.

3. Recall: it is the number of correct positive results divided by the number of all relevant samples (all samples that should have been identified as positive).

4. F1 Score: is calculated from the precision and recall. It is obtained by the Harmonic Mean between the two and tries to find the balance between precision and recall, meaning that will tell how precise your classifier is, as well as how robust it is.

#### 5.6.3.2 Final Evaluation Metric

As presented in [20] and explained by the CGI experts, by using the column 'Fault', created in the data pre-processing step, to evaluate the performance of our model does not provide the correct result. This column reflects the failure event represented the Sliding Time Window multiplied by 6 (because each hour have 6*10 minute rows). For example, for a Sliding Time Window of 12 hours, we have each failure event represented 72 times. This may lead to cases where our model does not predict every single 1's in the fault column, but in the sliding time window interval, detects the amount of 1's enough to consider that in that interval it was detected the pattern that will lead to a failure of that fault type in that wind turbine. To understand and evaluate this capacity of the model, the most important column is the Failure Time column, that really indicates the timestamp where a fault occurred and the output from the python script that takes also into consideration the sliding time window interval of the dataset.

Our evaluation metrics, and considering our final evaluation dataset which contains the structure presented in the first step of Figure 5.5, with the score of the model column, named Scored Model, is build following the presented steps:

1. Considering the table with the score model column, we will consider the sliding time window parameter. In this example, represented in the second step of Figure 5.5 we will consider 0.5 hours (30 minutes, that match with 3 timestamps).

2. On the dataset we build a new column, called 'Failure Prediction'. This column will be a Boolean (1 or 0), and it is going to indicate the real prediction of the model according to the interval considered in the previous step and the percentage threshold parameter. The percentage threshold parameter is a percentage value, and indicates for a certain interval of analysis (the interval presented before), the

percentage of predictions (1 values) in the interval that have to exists in order to consider that the end time of that interval represents a timestamp in which in the last 0.5 hours (our sliding time window parameter) the wind turbine is in failure conditions. The fill of that column is represented in the third step of Figure 5.5.

3. The last step is an iterative process that runs in all the rows. Each row evaluate the sliding time window before the timestamp in that row.

4. After the 'Failure Prediction' column is filled, we analyze the number of intervals that consist of consecutive true values (represented as 1). Each of these intervals represents a interval of prediction of the model. This means that from the start of that interval, to the end of that interval, the model predicted a failure event that might occur since the start timestamp of that interval to the end of that interval. These intervals are represented in Figure 5.6.

5. After these intervals are built up, we compare them to the timestamps in which we have a failure event, represented in our dataset with the column 'Failure Time'. Then we evaluate with the same metrics as the Confusion Matrix: if the Failure Time is contained in one of these intervals, we classified as a True Positive; if the Failure Time is not contained in any of these intervals, it is considered a miss in prediction and so it counts to the False Negative; if the interval predicted does not contain any failure event, is a bad prediction and is considered an False Positive.

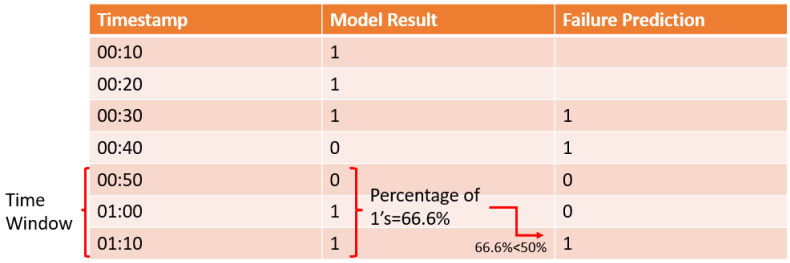



Figure 5.5: Evaluation Script Example. The first column represents the timestamp, the second the prediction of the model and the third is the new column, created by the script that indicates, according to the interval and percentage threshold, which is the prediction of the model.

## 5.7 Conclusion

In this chapter, we present all the information, structures and methods that we are going to use to build our experiment and perform its evaluation. We started by presenting our data information that was used to build our dataset, the dataset structure, the scope of features that were used and then proceed to the scope of faults and wind turbines that was studied. Then, we advance to indicate which machine learning algorithms were

| Timestamp | Model Result | Failure Prediction |
|---|---|---|
| 00:10 | 1 | |
| 00:20 | 1 | |
| 00:30 | 1 | 1 |
| 00:40 | 0 | 1 |
| 00:50 | 0 | 0 |
| 01:00 | 1 | 0 |
| 01:10 | 1 | 1 |

Failure predicted on interval ]00:20,00:40]

Failure predicted on interval ]01:00,01:10]

Figure 5.6: Evaluation Script Example 2. In this image and after the prediction of the model column ('Failure Prediction') is created, we have the predicted failure intervals represented.

used and present all the methods that are going to be used in our experiment. All these procedures and methods were selected according to the existing built-in modules in our main technology, the Azure Machine Learning Studio.

# 6

# Experiments and Results

## 6.1 Introduction

The experiments that were built during this work, as mentioned before, started by having a more generic and simple approach, in terms of the scopes of the datasets and in terms of the dataset pre-processing steps. While we were studying the experimental results by using the evaluation metrics mentioned in the previous Methodology section, we learned more about the problems and considering the clients, CGI experts feedback and our literature, we improved our experiment to its final form.

In this section we will present the final form of the experiment that is going to be implemented as a base for the failure prediction tool. We will present all the dataset pre-processing procedures that were implemented in .NET, using Visual Studio, to acquire all the data from the CGI RMS database and to execute some of the pre-processing procedures needed in order to build the dataset. This dataset is then provided to the experiment build inside the Azure Machine Learning Studio, that is the main tool that receive the dataset and build the machine learning models that are then evaluated.

## 6.2 Experiment

In the presentation of our experiment, we will separate it into two steps, according to the technologies that were used. Firstly we will introduce the construction of our dataset, that contains some data pre-processing and feature engineering methods, which results in the dataset that is then provided to the next step, in the Azure Machine Learning Studio experiment.

### 6.2.1 Dataset Acquisition and Build Up

This first step was build in .NET, using Visual Studio, and with SQL Server, the database system where the CGI RMS data is stored.

In this first experimental phase, we are not going to refer all the study work of the signals that should be associated with a fault in a wind turbine is done, presented previously

in the Methodology section.

Our dataset acquisition parameters, will have a start time, defined as the March 6th 2021, and an end date, the December 31st 2021 at 23:59:59. The sliding time windows that are going to be studied are 12 hours, 24 hours and 72 hours. For each wind turbine fault, we will generate a dataset according to different sliding time windows.

1. Firstly we acquire the data from the signals that have good quality. For each feature, it is generated the running summaries, mentioned earlier in the Methodology section, with the generation of these features considering the average, standard deviation, minimum and maximum of each signal for the last 6 hours. The parameter number of hours defined in this step was fixed since this technique was applied later in this master thesis work and from the knowledge of the CGI experts, we consider that looking 6 hours before it is good to compare mainly abrupt changes in a wind turbine behavior. It is mentioned in the Future Work section, an improvement that we consider that should be studied after this master thesis work.

2. After the acquisition of the signals data, with the running summaries features, it is removed from the features column, all the features that have more than 50% of null rows. We have to do this step, since if we keep these features, we will have to remove the null rows and that will lead to a big amount of timestamps being removed only because of one signal. Then, after the analysis of the invalid features is concluded, the remaining null rows are removed.

3. Then we step up to the failure events data acquisition, that it is generated according to the current sliding time window. In this step, besides the acquisition of the timestamps that have a failure event, it is also generated the fault column, mentioned earlier in the Methodology section.

4. After the acquisition of the timestamps that correspond to a failure event and the construction of the fault column, it is built an auxiliar column, the fault category.

5. Our final step, is a simple merge of the signals data and the faults data, so that we have only the corresponding timestamps that exist in both tables obtained earlier. To finalize the dataset, it is done a last check to delete rows with null features, whetever they exist.

After this process of the dataset generation, we have generated all datasets for each fault and wind turbine, one .csv file per sliding time window, meaning that for each wind turbine fault we have 3 .csv files, one for the sliding time window of 12h, one for the 24h and another for the 72h.

### 6.2.2 Azure Machine Learning Studio Experiment

Having the datasets available to provide to our Azure Machine Learning Studio Experiment, we can start running our final experiment. In this subsection, we will introduce each module that is going to run in our experiment, providing a brief explanation on what each module is going to do and justifying our decisions according to what was introduced earlier in the methodology section. In Figure 6.1 it is briefly represented in a diagram the complete experiment that is going to be explained later in this section. Our experiment is divided into three parts that are equal to each other. In each part, we will provide a different dataset for each fault with each file (.csv) varying by the time window of that dataset, as presented before.

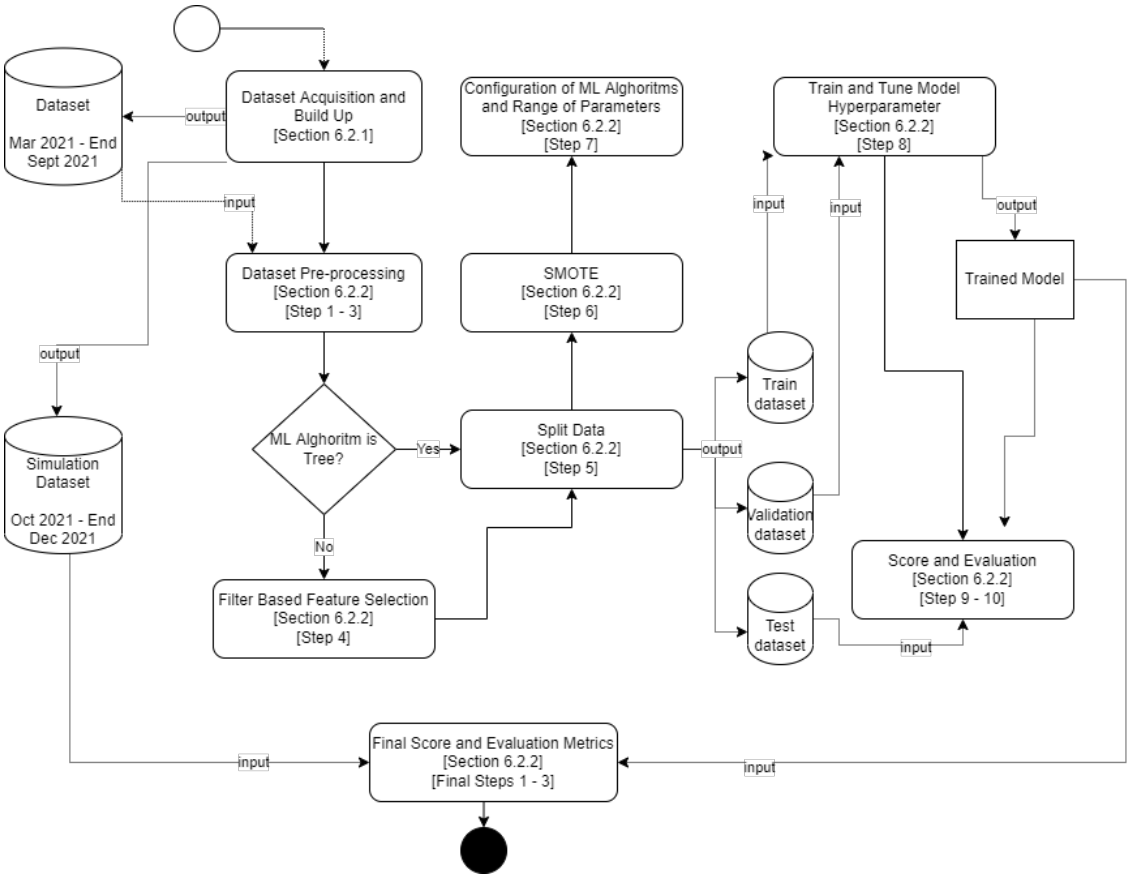



Figure 6.1: Diagram representing the experiment explained in the previous section.

1. Import Data: Component that imports data. In our case, we will import data from the Azure Storage that contains all the .csv files with the dataset. The data that comes from the .csv file contains all the available data for that particular fault and time window, this means since March 2021 to the end of December 2021.

2. Separate the Training Dataset from the Final Evaluation Dataset: Using the 'Apply SQL Transformation' module, we will split our import data into the first split

dataset, that we call Training Dataset and that will contain the data from March 2021 to the end of September 2021, and the second split dataset, that we call Final Evaluation Dataset and that will contain the data from October 2021 to the end of December 2021.

The next steps uses the first split dataset, the Training dataset. The second split dataset, called Final Evaluation Dataset, is used in the steps presented at the end of this subsection.

3. Edit Metadata: This component allows us to set to each column some metadata info that is useful to improve the performance of the models.

4. Filter Based Feature Selection: This component goal is to identify and select the "columns in the input dataset that have the greatest predict power"[30]. This module provide the option to choose the correlation method to classify the prediction power of each feature. The available methods are Pearson correlation and Chi-squared values.

   We selected Chi-squared for the correlation method since it is refereed in [27] and [8]. In the last article, it is refereed that this correlation method is the more adequate for our problem, a classification predictive problem with categorical output, despite the fact that our inputs are numerical. Since in this article, for an numerical input and categorical output, the correlation method advised is not available, we decided to select the one that seems more fitted to our problem. The article refers that Pearson Correlation is more adequate for regression problems.

   This component was only used for three algorithms: Two-Class Logistic Regression, Two-Class Support Vector Machine and Two-Class Neural Network. In the Decision Tree based algorithms (Two-Class Decision Forest and Two-Class Boosted Decision Tree), we don't use this module since the base of these algorithms already analyzes the best predictive features [3].

   Before the use of this component is important to refer that it can only be provided to the module the features and target column ('fault'). The other auxiliar columns (like 'ts', 'powerplantId', 'assetId', 'faultCategory', 'failureTime') are not provided to this module.

5. Split Data: Build the Train, Validation and Test dataset: We will split the provided dataset into three datasets, as already explained in methodology. Our first dataset will be the training dataset and that will have 60% of the input dataset. This dataset will be used to train and fit our the different machine learning models. Then the remaining 40%, will be divided into 70% for the validation dataset, that is going to be used to tune the model hyperparameters and the remaining 30% to the test dataset.

For the split data module, we will configure it to use a stratified split using the 'Fault Category' column, so that all the output datasets contain a representative sample of all the faults. The other configuration missing is the Randomized Split, that will set as True, so that the stratified split selects the rows of each group randomly.

6. SMOTE: As presented earlier in the Methodology section, since our dataset is unbalanced (meaning that the number of failure events is much lower that the number of events representing the machine running without any issues), we use the SMOTE module on the training dataset so that this dataset becomes a balanced dataset in the training of our model.

7. Machine Learning Algorithms: In the next step is the selection and configuration of all the previously presented algorithms, with the parameters presented earlier as well. This component will be used as an input to the 'Tune Model Hyperparameter' to train and fit the model.

8. Tune Model Hyperparameter: This module has the goal to "determine the optimum hyperparameters for a machine learning model"[Azure Machine Learning]. The configurations that were settled to this component was to perform a Random Sweep on the provided algorithms parameters, and to have a maximum of 50 runs when performing a random sweep. The metric that was used for measuring the performance of classification was the F1-Score, so that we can have the best balance between precision and recall, and try to build a model that doesn't have the false positives that we don't want to have, but also gives importance to the prediction of faults.

   This component receives as input a validation dataset and an untrained model and provides as an output the trained and fitted model. It also provides the performance metrics of the model for all the combination of the algorithm parameters.

9. Score Model: After having the fitted model, we will run this component that runs in the model a new dataset. We will provide to this component the test dataset. It provides as an output the complete dataset with the scored labels and scored probabilities for each row.

10. Evaluation Model: This module provides the metrics of evaluation of the module using as input the Score Model output. This is the component that allow us to evaluate the performance of the model. As stated earlier in the methodology, we will use as evaluation metrics, and for this dataset, the Precision, Recall and F1 Score.

11. Permutation Feature Importance: This component receives the trained model and a test dataset and evaluates the importance of each feature in the prediction of the target column. It is important for our study, in order to understand for each fault

and wind turbine, and considering the model that provides the best result, which features are considered the most important and so consider them to be used as default features for the Failure Prediction tool of CGI RMS.

After these steps, and having the trained model, we will perform our final evaluation of the build model and will evaluate the model with a total new data. Let's recall that this data is from the 1st October 2021 to the end of December 2021.

1. Score Model (with the new dataset): The next steps will use as a dataset the second split dataset, called Final Evaluation Dataset, mentioned in the previous enumerate. This module is the same as explained earlier, with the difference that is evaluating the dataset explained earlier.

2. Evaluate Model (with the new dataset): This module is the same as explained earlier, with the difference that is evaluating the dataset explained earlier.

3. Execute the Evaluation Script (in Python): This module, will run the Evaluation Script explained earlier in the Methodology, that allow us to obtain the intervals of fault that the model predicted. The output of this script is the evaluation if the intervals that the model predicted contains failure events. This comparison will lead to the confusion matrix for each percentage threshold (the parameter used in the evaluation script) that allow us to understand how the model performs with a complete new set of failure events.

## 6.3 Results

As stated earlier in our methodology, we will have two results per failure and asset to evaluate: the first one, that will evaluate our first test dataset and that contain the data from March 2021 to the final of September of 2021, and that will have represented all the failure events by using the stratified split on the fault category; the second one is a simulation of a real test of the model with a complete new dataset, that contains the data from October 2021 to the final of December 2021, and consequently, a complete set of failure events.

In the next subsections we will present for each fault and wind turbine, the model for each algorithm that presented the best result, with the parameters information that lead to that result.

Each scope is identified by three terms: the fault id, that indicates a identifier of the fault, the wind park id, with a three-letter string and the asset/wind turbine id, that is an identifier of the wind turbine of the park. These scopes are the ones presented in the Methodology section, in Table 5.3 and Table 5.4.

The selection of the best model was made by evaluating their metrics. As mentioned before, we prioritize the minimum value of false positives, but we also want that our

model is fitted to predict some failures. So we will consider the F1 Score, as mentioned before in the Tune Model Hyperparameter module, as it evaluate the best balance between the precision and recall.

In each subsection we will have three tables with the evaluations:

1. The first table will have the results of the first dataset, from 5th March 2021 to the final of September 2021, that was used to train, fit and tune the model. The results presented are from the test dataset and using the evaluate model module. We will use the evaluation metrics presented earlier in the methodology: precision, recall and F1 score.

2. The second table, presents the result of the simulation of running the already existing machine learning model with a new dataset, from October 2021 to the end of December 2021. In this table we will present the same metrics as in the previous step, also from the evaluate model module.

3. The final table, will present the results of our evaluation script, presented earlier in the methodology, and that tell us the real performance of the model by analyzing the confusion matrix metrics of the intervals of prediction that our model has predicted, as explained in the methodology section.

In these subsections we will also present the best hyperparameters selected from the tune model hyperparameter module for each model and algorithm studied. The time window of these models are the ones presented in the third table presented earlier.

The number of fault events that exist for each of the fault and scope presented bellow was presented earlier in the subsection 5.4.

### 6.3.1 Fault 9112 - SCA 34

Table 6.1: Fault 9112 - SCA 34: First Split (March 2021 - End September 2021) - Test Dataset - Evaluate Model Results

| Algorithm | Time Window | Precision | Recall | F1 Score |
|---|---|---|---|---|
| Two-Class Logistic Regression | 72 | 0.814 | 0.641 | 0.717 |
| Two-Class Support Vector Machine | 24 | 0.553 | 0.698 | 0.617 |
| Two-Class Neural Network | 72 | 0.849 | 0.857 | 0.853 |
| Two-Class Decision Forest | 72 | 0.992 | 0.992 | 0.985 |
| Two-Class Boosted Decision Tree | 72 | 0.99 | 0.995 | 0.992 |

In terms of the machine learning model built for each algorithm that provided the best result on the final evaluation dataset, the best hyperparameters selected from the Tune Model Hyperparameter module are:

1. Two-Class Logistic Regression

    a) Optimization Tolerance: 0

Table 6.2: Fault 9112 - SCA 34: Second Split (October 2021 - End December 2021) - Evaluate Model Results

| Algorithm | Time Window | Precision | Recall | F1 Score |
|---|---|---|---|---|
| Two-Class Logistic Regression | 12 | 0.341 | 0.279 | 0.307 |
| Two-Class Support Vector Machine | 12 | 0.338 | 0.307 | 0.322 |
| Two-Class Neural Network | 72 | 0.352 | 0.461 | 0.399 |
| Two-Class Decision Forest | 72 | 0.391 | 0.45 | 0.418 |
| Two-Class Boosted Decision Tree | 72 | 0.389 | 0.552 | 0.445 |

Table 6.3: Fault 9112 - SCA 34: Second Split (October 2021 - End December 2021) - Evaluation Script Predictions

| Algorithm | Time Window | % Threshold | FP | TP | FN |
|---|---|---|---|---|---|
| Two-Class Logistic Regression | 24 | 0.2 | 0 | 15 | 3 |
| Two-Class Support Vector Machine | 24 | 0.2 | 0 | 15 | 3 |
| Two-Class Neural Network | 24 | 0.2 ; 0.3 | 0 | 16 | 2 |
| Two-Class Decision Forest | 24 | 0.2 | 0 | 15 | 3 |
| Two-Class Boosted Decision Tree | 24 | 0.2 | 0 | 15 | 3 |

   b) Regularization Weight: 1

2. Two-Class Support Vector Machine

   a) Number of Iterations: 750

   b) Lambda:[0.01, 0.001, 0.0001, 0.00001]

3. Two-Class Neural Network

   a) Number of Learning Iterations: 500

   b) Learning Rate: 0.4

4. Two-Class Decision Forest

   a) Number of Decision Trees: 32

   b) Minimum number of samples per leaf node: 1

   c) Maximum depth of the decision tree: 64

5. Two-Class Boosted Decision Tree

   a) Maximum number of leaves per tree: 128

   b) Number of trees constructed: 250

   c) Minimum number of samples per leaf node: 25

   d) Learning Rate: 0.6

Table 6.4: Fault 8749 - PAP 35: First Split (March 2021 - End September 2021) - Test Dataset - Evaluate Model Results

| Algorithm | Time Window | Precision | Recall | F1 Score |
|---|---|---|---|---|
| Two-Class Logistic Regression | 72 | 0.587 | 0.567 | 0.577 |
| Two-Class Support Vector Machine | 72 | 0.515 | 0.465 | 0.489 |
| Two-Class Neural Network | 72 | 0.975 | 0.985 | 0.98 |
| Two-Class Decision Forest | 72 | 0.996 | 0.994 | 0.995 |
| Two-Class Boosted Decision Tree | 72 | 0.999 | 0.994 | 0.997 |

Table 6.5: Fault 8749 - PAP 35: Second Split (October 2021 - End December 2021) - Evaluate Model Results

| Algorithm | Time Window | Precision | Recall | F1 Score |
|---|---|---|---|---|
| Two-Class Logistic Regression | 72 | 0.288 | 0.16 | 0.206 |
| Two-Class Support Vector Machine | 72 | 0.167 | 0.099 | 0.124 |
| Two-Class Neural Network | 72 | 0.259 | 0.237 | 0.248 |
| Two-Class Decision Forest | 72 | 0.223 | 0.369 | 0.278 |
| Two-Class Boosted Decision Tree | 72 | 0.43 | 0.295 | 0.35 |

Table 6.6: Fault 8749 - PAP 35: Second Split (October 2021 - End December 2021) - Evaluation Script Predictions

| Algorithm | Time Window | % Threshold | FP | TP | FN |
|---|---|---|---|---|---|
| Two-Class Logistic Regression | 24 | 0.2 ; 0.3 | 0 | 2 | 6 |
| Two-Class Support Vector Machine | 72 | 0.2 ; 0.3 | 0 | 1 | 7 |
| Two-Class Neural Network | 72 | 0.2 | 0 | 2 | 6 |
| Two-Class Decision Forest | 72 | 0.4 ; 0.5 ; 0.6 | 0 | 1 | 7 |
| Two-Class Boosted Decision Tree | 72 | 0.5 ; 0.6 ; 0.7 | 0 | 1 | 7 |

### 6.3.2 Fault 8749 - PAP 35

In terms of the machine learning model built for each algorithm, the best hyperparameters selected from the Tune Model Hyperparameter module are:

1. Two-Class Logistic Regression

    a) Optimization Tolerance: 0

    b) Regularization Weight: 0.01

2. Two-Class Support Vector Machine

    a) Number of Iterations: 750

    b) Lambda: All

3. Two-Class Neural Network

    a) Number of Learning Iterations: 500

b) Learning Rate: 0.6

4. Two-Class Decision Forest

   a) Number of Decision Trees: 32

   b) Minimum number of samples per leaf node: 1

   c) Maximum depth of the decision tree: [32, 64]

5. Two-Class Boosted Decision Tree

   a) Maximum number of leaves per tree: [64,128]

   b) Number of trees constructed: 500

   c) Minimum number of samples per leaf node: 10,50

   d) Learning Rate: [0.2, 0.1]

### 6.3.3 Fault 768 - BNE 18

Table 6.7: Fault 768 - BNE 18: First Split (March 2021 - End September 2021) - Test Dataset - Evaluate Model Results

| Algorithm | Time Window | Precision | Recall | F1 Score |
|---|---|---|---|---|
| Two-Class Logistic Regression | 72 | 0.447 | 0.224 | 0.298 |
| Two-Class Support Vector Machine | 72 | 0.875 | 0.082 | 0.151 |
| Two-Class Neural Network | 24 | 0.966 | 1 | 0.983 |
| Two-Class Decision Forest | 72 | 0.998 | 0.993 | 0.995 |
| Two-Class Boosted Decision Tree | 72 | 1 | 0.993 | 0.996 |

Table 6.8: Fault 768 - BNE 18: Second Split (October 2021 - End December 2021) - Test Dataset - Evaluate Model Results

| Algorithm | Time Window | Precision | Recall | F1 Score |
|---|---|---|---|---|
| Two-Class Logistic Regression | 72 | 0.126 | 0.02 | 0.035 |
| Two-Class Support Vector Machine | All | 1 | 0 | 0 |
| Two-Class Neural Network | 72 | 0.291 | 0.066 | 0.107 |
| Two-Class Decision Forest | 72 | 0.481 | 0.116 | 0.187 |
| Two-Class Boosted Decision Tree | 72 | 0.42 | 0.114 | 0.179 |

In terms of the machine learning model built for each algorithm, the best hyperparameters selected from the Tune Model Hyperparameter module are:

1. Two-Class Logistic Regression

   a) Optimization Tolerance: 0

   b) Regularization Weight: 0.01

Table 6.9: Fault 768 - BNE 18: Second Split (October 2021 - End December 2021) - Evaluation Script Predictions

| Algorithm | Time Window | % Threshold | FP | TP | FN |
|---|---|---|---|---|---|
| Two-Class Logistic Regression | All | All | 0 | 0 | 8 |
| Two-Class Support Vector Machine | All | All | 0 | 0 | 8 |
| Two-Class Neural Network | 72 | >= 0.4 | 0 | 0 | 8 |
| Two-Class Decision Forest | 72 | 0.2 | 0 | 1 | 7 |
| Two-Class Boosted Decision Tree | 72 | 0.2 | 0 | 1 | 7 |

2. Two-Class Support Vector Machine

    a) Number of Iterations: 750

    b) Lambda: All

3. Two-Class Neural Network

    a) Number of Learning Iterations: 160

    b) Learning Rate: 0.8

4. Two-Class Decision Forest

    a) Number of Decision Trees: 16

    b) Minimum number of samples per leaf node: 1

    c) Maximum depth of the decision tree: 32

5. Two-Class Boosted Decision Tree For this model, the Tune Model Hyperparameter module return almost all the parameter range values as classified with the same ranking. So it is not possible to take conclusions on the hyperparameters that provided the best results

### 6.3.4 Fault 768 - CHF 28

Table 6.10: Fault 768 - CHF 28: First Split (March 2021 - End September 2021) - Test Dataset - Evaluate Model Results

| Algorithm | Time Window | Precision | Recall | F1 Score |
|---|---|---|---|---|
| Two-Class Logistic Regression | 12 | 0.561 | 0.217 | 0.313 |
| Two-Class Support Vector Machine | 72 | 0.561 | 0.107 | 0.18 |
| Two-Class Neural Network | 12 | 0.991 | 0.991 | 0.991 |
| Two-Class Decision Forest | 72 | 0.999 | 1 | 0.999 |
| Two-Class Boosted Decision Tree | 72 | 0.999 | 1 | 0.999 |

In terms of the machine learning model built for each algorithm, the best hyperparameters selected from the Tune Model Hyperparameter module are:

Table 6.11: Fault 768 - CHF 28: Second Split (October 2021 - End December 2021) - Evaluate Model Results - Test Dataset - Evaluate Model Results

| Algorithm | Time Window | Precision | Recall | F1 Score |
|---|---|---|---|---|
| Two-Class Logistic Regression | 72 | 0.7 | 0.161 | 0.262 |
| Two-Class Support Vector Machine | 72 | 0.81 | 0.074 | 0.136 |
| Two-Class Neural Network | 72 | 0.444 | 0.123 | 0.193 |
| Two-Class Decision Forest | 72 | 0.681 | 0.11 | 0.189 |
| Two-Class Boosted Decision Tree | 72 | 0.651 | 0.084 | 0.148 |

Table 6.12: Fault 768 - CHF 28: Second Split (October 2021 - End December 2021) - Evaluation Script Predictions

| Algorithm | Time Window | % Threshold | FP | TP | FN |
|---|---|---|---|---|---|
| Two-Class Logistic Regression | 72 | 0.2 | 0 | 21 | 3 |
| Two-Class Support Vector Machine | 72 | 0.2 | 0 | 10 | 14 |
| Two-Class Neural Network | 12, 72 | 0.2 | 0 | 2 | 22 |
| Two-Class Decision Forest | 72 | 0.2 | 0 | 8 | 16 |
| Two-Class Boosted Decision Tree | 72 | 0.2 | 0 | 1 | 23 |

1. Two-Class Logistic Regression
    a) Optimization Tolerance: 0
    b) Regularization Weight: 0.1
2. Two-Class Support Vector Machine
    a) Number of Iterations: 750
    b) Lambda: All
3. Two-Class Neural Network
    a) Number of Learning Iterations: 500
    b) Learning Rate: 0.6
4. Two-Class Decision Forest
    a) Number of Decision Trees: 32
    b) Minimum number of samples per leaf node: 1
    c) Maximum depth of the decision tree: [32, 64]
5. Two-Class Boosted Decision Tree
    a) Maximum number of leaves per tree: [128,64]
    b) Number of trees constructed: [50, 500, 250]
    c) Minimum number of samples per leaf node: [10,1]
    d) Learning Rate: All

### 6.3.5 Fault 140 - PCA 7

Table 6.13: Fault 140 - PCA 7: First Split (March 2021 - End September 2021) - Test Dataset - Evaluate Model Results

| Algorithm | Time Window | Precision | Recall | F1 Score |
|---|---|---|---|---|
| Two-Class Logistic Regression | 72 | 0.587 | 0.556 | 0.571 |
| Two-Class Support Vector Machine | 72 | 0.598 | 0.542 | 0.569 |
| Two-Class Neural Network | 72 | 0.978 | 0.978 | 0.978 |
| Two-Class Decision Forest | 72 | 0.996 | 0.995 | 0.996 |
| Two-Class Boosted Decision Tree | 24 | 0.999 | 0.997 | 0.998 |

Table 6.14: Fault 140 - PCA 7: Second Split (October 2021 - End December 2021) - Evaluate Model Results

| Algorithm | Time Window | Precision | Recall | F1 Score |
|---|---|---|---|---|
| Two-Class Logistic Regression | 72 | 0.139 | 0.571 | 0.223 |
| Two-Class Support Vector Machine | 72 | 0.141 | 0.579 | 0.227 |
| Two-Class Neural Network | 72 | 0.124 | 0.546 | 0.202 |
| Two-Class Decision Forest | 72 | 0.129 | 0.42 | 0.197 |
| Two-Class Boosted Decision Tree | 72 | 0.146 | 0.262 | 0.188 |

Table 6.15: Fault 140 - PCA 7: Second Split (October 2021 - End December 2021) - Evaluation Script Predictions

| Algorithm | Time Window | % Threshold | FP | TP | FN |
|---|---|---|---|---|---|
| Two-Class Logistic Regression | 72 | 0.3, 0.4, 0.5 | 0 | 11 | 2 |
| Two-Class Support Vector Machine | 72 | 0.3, 0.4, 0.5 | 0 | 11 | 2 |
| Two-Class Neural Network | 72 | 0.3 | 0 | 11 | 2 |
| Two-Class Decision Forest | 72 | 0.2 | 0 | 10 | 3 |
| Two-Class Boosted Decision Tree | 12 | <= 0.6 | 0 | 2 | 11 |

In terms of the machine learning model built for each algorithm, the best hyperparameters selected from the Tune Model Hyperparameter module are:

1. Two-Class Logistic Regression

    a) Optimization Tolerance: 0

    b) Regularization Weight: 0.1

2. Two-Class Support Vector Machine

    a) Number of Iterations: 100

    b) Lambda: All

3. Two-Class Neural Network

a) Number of Learning Iterations: 500

b) Learning Rate: 0.4

4. Two-Class Decision Forest

a) Number of Decision Trees: 16

b) Minimum number of samples per leaf node: 1

c) Maximum depth of the decision tree: 32

5. Two-Class Boosted Decision Tree

a) Maximum number of leaves per tree: 128

b) Number of trees constructed: [50, 100]

c) Minimum number of samples per leaf node: 10

d) Learning Rate: 0.4

### 6.3.6 Fault 14474 - TMS 6

Table 6.16: Fault 14474 - TMS 6: First Split (March 2021 - End September 2021) - Test Dataset - Evaluate Model Results

| Algorithm | Time Window | Precision | Recall | F1 Score |
|---|---|---|---|---|
| Two-Class Logistic Regression | 72 | 0.414 | 0.317 | 0.359 |
| Two-Class Support Vector Machine | 72 | 0.574 | 0.14 | 0.224 |
| Two-Class Neural Network | 72 | 0.939 | 0.943 | 0.941 |
| Two-Class Decision Forest | 72 | 0.998 | 0.99 | 0.994 |
| Two-Class Boosted Decision Tree | 72 | 0.999 | 0.993 | 0.996 |

Table 6.17: Fault 14474 - TMS 6: Second Split (October 2021 - End December 2021) - Evaluate Model Results

| Algorithm | Time Window | Precision | Recall | F1 Score |
|---|---|---|---|---|
| Two-Class Logistic Regression | 72 | 0.399 | 0.073 | 0.124 |
| Two-Class Support Vector Machine | 72 | 0.265 | 0.018 | 0.033 |
| Two-Class Neural Network | 72 | 0.173 | 0.083 | 0.112 |
| Two-Class Decision Forest | 24 | 0.62 | 0.228 | 0.333 |
| Two-Class Boosted Decision Tree | 72 | 0.05 | 0.009 | 0.015 |

In terms of the machine learning model built for each algorithm, the best hyperparameters selected from the Tune Model Hyperparameter module are:

1. Two-Class Logistic Regression

a) Optimization Tolerance: 0

Table 6.18: Fault 14474 - TMS 6: Second Split (October 2021 - End December 2021) - Evaluation Script Predictions

| Algorithm | Time Window | % Threshold | FP | TP | FN |
|---|---|---|---|---|---|
| Two-Class Logistic Regression | 12, 72 | >= 0.3 | 0 | 0 | 10 |
| Two-Class Support Vector Machine | All | All | 0 | 0 | 10 |
| Two-Class Neural Network | 72 | >= 0.3 | 0 | 0 | 10 |
| Two-Class Decision Forest | 12, 24 | All | 0 | 0 | 10 |
| Two-Class Boosted Decision Tree | 12, 24 | All | 0 | 0 | 10 |

   b) Regularization Weight: 0.01

2. Two-Class Support Vector Machine

   a) Number of Iterations: 750

   b) Lambda: All

3. Two-Class Neural Network

   a) Number of Learning Iterations: 500

   b) Learning Rate: 0.4

4. Two-Class Decision Forest

   a) Number of Decision Trees: 16

   b) Minimum number of samples per leaf node: 1

   c) Maximum depth of the decision tree: 16

5. Two-Class Boosted Decision Tree

   a) Maximum number of leaves per tree: 32

   b) Number of trees constructed: 500

   c) Minimum number of samples per leaf node: 1

   d) Learning Rate: 0.05

### 6.3.7 Fault 768 - BNE 14

In terms of the machine learning model built for each algorithm, the best hyperparameters selected from the Tune Model Hyperparameter module are:

1. Two-Class Logistic Regression

   a) Optimization Tolerance: 0

   b) Regularization Weight: 0.01

Table 6.19: Fault 768 - BNE 14: First Split (March 2021 - End September 2021) - Test Dataset - Evaluate Model Results

| Algorithm | Time Window | Precision | Recall | F1 Score |
|---|---|---|---|---|
| Two-Class Logistic Regression | 24 | 0.555 | 0.233 | 0.328 |
| Two-Class Support Vector Machine | 24 | 0.636 | 0.119 | 0.2 |
| Two-Class Neural Network | 72 | 0.986 | 0.99 | 0.988 |
| Two-Class Decision Forest | 72 | 1 | 0.991 | 0.996 |
| Two-Class Boosted Decision Tree | 72 | 1 | 0.995 | 0.998 |

Table 6.20: Fault 768 - BNE 14: Second Split (October 2021 - End December 2021) - Evaluate Model Results

| Algorithm | Time Window | Precision | Recall | F1 Score |
|---|---|---|---|---|
| Two-Class Logistic Regression | All | 0 | 0 | 0 |
| Two-Class Support Vector Machine | All | 0 | 0 | 0 |
| Two-Class Neural Network | All | 0 | 0 | 0 |
| Two-Class Decision Forest | All | 0 | 0 | 0 |
| Two-Class Boosted Decision Tree | All | 0 | 0 | 0 |

Table 6.21: Fault 768 - BNE 14: Second Split (October 2021 - End December 2021) - Evaluation Script Predictions

| Algorithm | Time Window | % Threshold | FP | TP | FN |
|---|---|---|---|---|---|
| Two-Class Logistic Regression | All | All | 0 | 0 | 6 |
| Two-Class Support Vector Machine | 12, 72 | All | 0 | 0 | 6 |
| Two-Class Neural Network | All | All | 0 | 0 | 6 |
| Two-Class Decision Forest | All | All | 0 | 0 | 6 |
| Two-Class Boosted Decision Tree | All | All | 0 | 0 | 6 |

2. Two-Class Support Vector Machine

    a) Number of Iterations: 750

    b) Lambda: All

3. Two-Class Neural Network

    a) Number of Learning Iterations: 500

    b) Learning Rate: 0.4

4. Two-Class Decision Forest

    a) Number of Decision Trees: 32

    b) Minimum number of samples per leaf node: 1

    c) Maximum depth of the decision tree: [32,64]

5. Two-Class Boosted Decision Tree

a) Maximum number of leaves per tree: [32, 64, 128]

b) Number of trees constructed: [250,500]

c) Minimum number of samples per leaf node: All

d) Learning Rate: All

### 6.3.8 Fault 367 - CHF 15

Table 6.22: Fault 367 - CHF 15: First Split (March 2021 - End September 2021) - Test Dataset - Evaluate Model Results

| Algorithm | Time Window | Precision | Recall | F1 Score |
|---|---|---|---|---|
| Two-Class Logistic Regression | 72 | 0.32 | 0.068 | 0.111 |
| Two-Class Support Vector Machine | 72 | 0.471 | 0.034 | 0.063 |
| Two-Class Neural Network | 24 | 0.979 | 0.979 | 0.979 |
| Two-Class Decision Forest | 72 | 1 | 0.992 | 0.996 |
| Two-Class Boosted Decision Tree | 72 | 0.998 | 0.994 | 0.996 |

Table 6.23: Fault 367 - CHF 15: Second Split (October 2021 - End December 2021) - Evaluate Model Results

| Algorithm | Time Window | Precision | Recall | F1 Score |
|---|---|---|---|---|
| Two-Class Logistic Regression | 72 | 0.6 | 0.052 | 0.095 |
| Two-Class Support Vector Machine | All | 1 | 0 | 0 |
| Two-Class Neural Network | 72 | 0.154 | 0.063 | 0.089 |
| Two-Class Decision Forest | 72 | 0.12 | 0.029 | 0.046 |
| Two-Class Boosted Decision Tree | 72 | 0.104 | 0.031 | 0.048 |

Table 6.24: Fault 367 - CHF 15: Second Split (October 2021 - End December 2021) - Evaluation Script Predictions

| Algorithm | Time Window | % Threshold | FP | TP | FN |
|---|---|---|---|---|---|
| Two-Class Logistic Regression | 24, 72 | All | 0 | 0 | 4 |
| Two-Class Support Vector Machine | All | All | 0 | 0 | 4 |
| Two-Class Neural Network | 24, 72 | All | 0 | 0 | 4 |
| Two-Class Decision Forest | All | All | 0 | 0 | 4 |
| Two-Class Boosted Decision Tree | 12, 24 | All | 0 | 0 | 4 |

In terms of the machine learning model built for each algorithm, the best hyperparameters selected from the Tune Model Hyperparameter module are:

1. Two-Class Logistic Regression

   a) Optimization Tolerance: 0

   b) Regularization Weight: 0.1

2. Two-Class Support Vector Machine

a) Number of Iterations: 750

b) Lambda: All

3. Two-Class Neural Network

a) Number of Learning Iterations: 500

b) Learning Rate: 0.2

4. Two-Class Decision Forest

a) Number of Decision Trees: 32

b) Minimum number of samples per leaf node: 1

c) Maximum depth of the decision tree: [32,64]

5. Two-Class Boosted Decision Tree

a) Maximum number of leaves per tree: 8

b) Number of trees constructed: 250

c) Minimum number of samples per leaf node: 25

d) Learning Rate: 0.2

### 6.3.9 Fault 140 - PLS 9

Table 6.25: Fault 140 - PLS 9: First Split (March 2021 - End September 2021) - Test Dataset - Evaluate Model Results

| Algorithm | Time Window | Precision | Recall | F1 Score |
|---|---|---|---|---|
| Two-Class Logistic Regression | 72 | 0.469 | 0.374 | 0.416 |
| Two-Class Support Vector Machine | 72 | 0.48 | 0.359 | 0.411 |
| Two-Class Neural Network | 72 | 0.964 | 0.98 | 0.972 |
| Two-Class Decision Forest | 72 | 0.993 | 0.992 | 0.993 |
| Two-Class Boosted Decision Tree | 72 | 0.996 | 0.997 | 0.997 |

Table 6.26: Fault 140 - PLS 9: Second Split (October 2021 - End December 2021) - Evaluate Model Results

| Algorithm | Time Window | Precision | Recall | F1 Score |
|---|---|---|---|---|
| Two-Class Logistic Regression | 72 | 0.098 | 0.394 | 0.156 |
| Two-Class Support Vector Machine | 72 | 0.096 | 0.398 | 0.155 |
| Two-Class Neural Network | 72 | 0.077 | 0.194 | 0.111 |
| Two-Class Decision Forest | 72 | 0.064 | 0.17 | 0.093 |
| Two-Class Boosted Decision Tree | 72 | 0.066 | 0.17 | 0.095 |

In terms of the machine learning model built for each algorithm, the best hyperparameters selected from the Tune Model Hyperparameter module are:

Table 6.27: Fault 140 - PLS 9: Second Split (October 2021 - End December 2021) - Evaluation Script Predictions

| Algorithm | Time Window | % Threshold | FP | TP | FN |
|---|---|---|---|---|---|
| Two-Class Logistic Regression | 12, 24 | All | 0 | 0 | 3 |
| Two-Class Support Vector Machine | 24 | All | 0 | 0 | 3 |
| Two-Class Neural Network | 12 | <= 0.4 | 0 | 1 | 2 |
| Two-Class Decision Forest | 12, 24 | All | 0 | 0 | 3 |
| Two-Class Boosted Decision Tree | 12 | All | 0 | 0 | 3 |

1. Two-Class Logistic Regression
    a) Optimization Tolerance: 0
    b) Regularization Weight: 0.01
2. Two-Class Support Vector Machine
    a) Number of Iterations: 500
    b) Lambda: All
3. Two-Class Neural Network
    a) Number of Learning Iterations: 500
    b) Learning Rate: 0.6
4. Two-Class Decision Forest
    a) Number of Decision Trees: 16
    b) Minimum number of samples per leaf node: 1
    c) Maximum depth of the decision tree: 32
5. Two-Class Boosted Decision Tree1
    a) Maximum number of leaves per tree: 8
    b) Number of trees constructed: 500
    c) Minimum number of samples per leaf node: 10
    d) Learning Rate: 0.4

### 6.3.10 Fault 8487 - SCA 10

In terms of the machine learning model built for each algorithm, the best hyperparameters selected from the Tune Model Hyperparameter module are:

1. Two-Class Logistic Regression24

Table 6.28: Fault 8487 - SCA 10: First Split (March 2021 - End September 2021) - Test Dataset - Evaluate Model Results

| Algorithm | Time Window | Precision | Recall | F1 Score |
|---|---|---|---|---|
| Two-Class Logistic Regression | 24 | 0.688 | 0.543 | 0.607 |
| Two-Class Support Vector Machine | 72 | 0.619 | 0.548 | 0.581 |
| Two-Class Neural Network | 72 | 0.982 | 0.989 | 0.986 |
| Two-Class Decision Forest | 24 | 0.99 | 0.996 | 0.993 |
| Two-Class Boosted Decision Tree | 24 | 0.995 | 0.999 | 0.997 |

Table 6.29: Fault 8487 - SCA 10: Second Split (October 2021 - End December 2021) - Evaluate Model Results

| Algorithm | Time Window | Precision | Recall | F1 Score |
|---|---|---|---|---|
| Two-Class Logistic Regression | All | 0 | 0 | 0 |
| Two-Class Support Vector Machine | All | 0 | 0 | 0 |
| Two-Class Neural Network | All | 0 | 0 | 0 |
| Two-Class Decision Forest | All | 0 | 0 | 0 |
| Two-Class Boosted Decision Tree | All | 0 | 0 | 0 |

Table 6.30: Fault 8487 - SCA 10: Second Split (October 2021 - End December 2021) - Evaluation Script Predictions

| Algorithm | Time Window | % Threshold | FP | TP | FN |
|---|---|---|---|---|---|
| Two-Class Logistic Regression | All | All | 0 | 0 | 0 |
| Two-Class Support Vector Machine | All | All | 0 | 0 | 0 |
| Two-Class Neural Network | All | All | 0 | 0 | 0 |
| Two-Class Decision Forest | All | All | 0 | 0 | 0 |
| Two-Class Boosted Decision Tree | All | All | 0 | 0 | 0 |

   a) Optimization Tolerance: 0

   b) Regularization Weight: 0.1

2. Two-Class Support Vector Machine

   a) Number of Iterations: 750

   b) Lambda: All

3. Two-Class Neural Network

   a) Number of Learning Iterations: 500

   b) Learning Rate: 0.6

4. Two-Class Decision Forest

   a) Number of Decision Trees: 32

   b) Minimum number of samples per leaf node: 1

c) Maximum depth of the decision tree: [32,64]

5. Two-Class Boosted Decision Tree

a) Maximum number of leaves per tree: 32

b) Number of trees constructed: 500

c) Minimum number of samples per leaf node: 1

d) Learning Rate: 0.05

### 6.3.11 Fault 8719 - SCA 28

Table 6.31: Fault 8719 - SCA 28: First Split (March 2021 - End September 2021) - Test Dataset - Evaluate Model Results

| Algorithm | Time Window | Precision | Recall | F1 Score |
|---|---|---|---|---|
| Two-Class Logistic Regression | 72 | 0.414 | 0.317 | 0.359 |
| Two-Class Support Vector Machine | 72 | 0.475 | 0.251 | 0.328 |
| Two-Class Neural Network | 24 | 0.692 | 0.911 | 0.786 |
| Two-Class Decision Forest | 72 | 0.996 | 0.997 | 0.996 |
| Two-Class Boosted Decision Tree | 72 | 0.999 | 0.998 | 0.998 |

Table 6.32: Fault 8719 - SCA 28: Second Split (October 2021 - End December 2021) - Evaluate Model Results

| Algorithm | Time Window | Precision | Recall | F1 Score |
|---|---|---|---|---|
| Two-Class Logistic Regression | All | 0 | 0 | 0 |
| Two-Class Support Vector Machine | All | 0 | 0 | 0 |
| Two-Class Neural Network | All | 0 | 0 | 0 |
| Two-Class Decision Forest | All | 0 | 0 | 0 |
| Two-Class Boosted Decision Tree | All | 0 | 0 | 0 |

Table 6.33: Fault 8719 - SCA 28: Second Split (October 2021 - End December 2021) - Evaluation Script Predictions

| Algorithm | Time Window | % Threshold | FP | TP | FN |
|---|---|---|---|---|---|
| Two-Class Logistic Regression | All | All | 0 | 0 | 0 |
| Two-Class Support Vector Machine | All | All | 0 | 0 | 0 |
| Two-Class Neural Network | All | All | 0 | 0 | 0 |
| Two-Class Decision Forest | All | All | 0 | 0 | 0 |
| Two-Class Boosted Decision Tree | All | All | 0 | 0 | 0 |

In terms of the machine learning model built for each algorithm, the best hyperparameters selected from the Tune Model Hyperparameter module are:

1. Two-Class Logistic Regression

a) Optimization Tolerance: 0

b) Regularization Weight: 0.01

2. Two-Class Support Vector Machine

a) Number of Iterations: 750

b) Lambda: 0.01x

3. Two-Class Neural Network24

a) Number of Learning Iterations: 500

b) Learning Rate: 0.6

4. Two-Class Decision Forest

a) Number of Decision Trees: 32

b) Minimum number of samples per leaf node: 1

c) Maximum depth of the decision tree: [34,64]

5. Two-Class Boosted Decision Tree

a) Maximum number of leaves per tree: 128

b) Number of trees constructed: 500

c) Minimum number of samples per leaf node: 1

d) Learning Rate: 0.1

# 7

# Results Discussion

## 7.1 Introduction

After presenting in the Experiments and Results section our experiment and results, we will now analyze the presented metrics, do some analysis on results obtained per fault and considering the dataset presented for each fault and take our conclusions.

## 7.2 Machine Learning Models Performance

As explained in the Methodology and Experiments and Result section, to understand the real performance of our machine learning models we will dedicate our analysis to the final evaluation metrics results because it is our final test dataset and will match to a simulation of the behavior of the model in the real world. This will tell us from the prediction interval that the model was built, and according to a set of percentage threshold parameter, the false positives, true positives and false negatives in comparison to the failure events that occurred.

During this section, we will present the scopes using the following formatting: (Wind Park, Asset/Wind Turbine, Fault Id).

As mentioned in the Experiments and Result section, for the scopes (SCA 10, Fault 8487) and (SCA 28, Fault 8719), in this test dataset (from October 2021 to the end of December 2021) we don't have registered any failure events. So, for the models that were built for these two scopes, we will not analyze the prediction of a fault but yes the model performance in not predicting false positives. Since the beginning of this study, we mentioned that one of our main goals is that our models don't have a high number of false positives, so this analysis is also important to guarantee the reliability of these models and experiment. As we can see from Table 6.30 and 6.31, our models don't have any prediction of failure event for these two scopes, for any time window and percentage threshold. So, we can conclude that the models performed well in a case when it was submitted to a dataset that do not have any failure events.

Focusing on the remaining 9 scopes of wind turbines faults that have failure events in

the final evaluation dataset, we can see from the analysis of the Final Evaluation Tables (the tables with the confusion matrix metrics that is an output of the evaluation script), presented in Tables 6.3, 6.6, 6.9, 6.12, 6.15, 6.18, 6.21, 6.24, 6.27 that:

1. 6 of them have at least, one machine learning model that predicted 1 interval that contained a failure events, and in all of them there was no registered any predicted intervals that contains false positives. These scopes are: (SCA 34 Fault 9112), (PAP 35 Fault 8749), (BNE 18 Fault 768), (CHF 18 Fault 768), (PCA 7 Fault 140), (PLS 9 Fault 140).

2. The remaining 3 scopes that were not predicted (meaning that only have registered false negative scores), have the positive conclusion that did not lead the any false positives, which goes in the direction of this study goal.

3. Considering only the scopes with predictions and that have at least 5 fault events, which are (PCA 7 Fault 140), (CHF 18 Fault 768), (BNE 18 Fault 768), (PAP 35 Fault 8749), (SCA 34 Fault 9112), we present in the Table 7.1 the recall of the best model, that indicate us in all the failure events in the dataset, the percentage of success of the model in terms of its prediction. By looking at the results of Table 7.1, we can see that 3 scopes (SCA 34 Fault 9112), (CHF 18 Fault 768), (PCA 7 Fault 140) have a great performance, with a recall of over 85%

4. The models have the best results when the percentage threshold parameter (that is configured on the evaluation python script), has its lower levels (from 0.2 or 0.3). This indicates that for each time window interval, the model can only flag it as a failure interval when it detects a low percentage of flagged rows (rows that have the column 'Score Model' = 1). When a high percentage of flagged rows are needed to consider the interval as a failure, it is not considered as a failure interval and thus the number of true positives decreases (and the false negatives increases). Which meets the low results from the Evaluate Model Metrics (Precision, Recall, F1 Score), that only consider the prediction of each row separately against the 'Fault' column.

Table 7.1: Best Models and the Recall Results. In the Model Info column, we will name our algorithms by their abbreviation (NN: Neural Network; DT: Decision Tree; BDT: Boosted Decision Tree; LogReg: Logistic Regression; SVM: Support Vector Machine

| Scope | Fault | Model Info | % Threshold | TP | FN | Recall |
|---|---|---|---|---|---|---|
| SCA 34 | 9112 | NN TW 24h | 0.2 ; 0.3 | 16 | 2 | 89% |
| PAP 35 | 8749 | NN TW 72h | 0.2 | 2 | 6 | 25% |
| BNE 18 | 768 | DF/BDT TW 72h | 0.2 | 1 | 7 | 13% |
| CHF 18 | 768 | LogReg TW 72h | 0.2 | 21 | 3 | 88% |
| PCA 7 | 140 | LogReg/SVM TW 72h | 0.3, 0.4, 0.5 | 11 | 2 | 85% |

In terms of the performance of each algorithm in all the fault scopes, by analyzing mainly the first split test dataset, presented by Tables 6.1, 6.4, 6.7, 6.10, 6.13, 6.16, 6.19,

6.22, 6.25 and the second split test dataset, presented by Tables 6.2, 6.5, 6.8, 6.11, 6.14, 6.17, 6.20, 6.23, 6.26 we can take the following conclusions:

1. Support Vector Machine(SVM) This algorithm indicates that it is the more reliable in terms of their predictions. It have the lower number of false positives of all the other models algorithms and the evaluation metrics from the first split are more reliable in comparison with other algorithms, that seem to be overfitted. In other hand, their true positive and recall metrics, indicates a worst performance in predict all the failure intervals, compensating with the fact that is more reliable leading to none failure positives.

2. Logistic Regression Have a prediction behaviour similar to the SVM, gaining in the prediction of intervals, having the same or best numbers in terms of recall and true positives, but losing in terms of its reliability.

3. Two-Class Neural Network, Two-Class Decision Forest and Two-Class Boosted Decision Tree These three algorithms seem to have similar results. In the first split dataset, they same to be overfitted to this dataset, since they have results of precision, recall and F1 score close to 1, but when a new dataset (the second split dataset) is presented, their results decrease. When analyzing their performance with the confusion matrix metrics presented after the evaluation script analyzes the models predicted behavior, we can see that the models of these algorithms have lower reliability, having a higher number of false positives, and a higher number of true positives and predictions.

In terms of the time window parameter, by analyzing the final evaluation results, in Table 6.3, 6.6, 6.9, 6.12, 6.15, 6.18, 6.21, 6.24, 6.27, we can see that for the fault scopes that have the highest number of failure events in the dataset and that are predicted, the models with the highest time window (72 hours) has the best performance. This might be related to the fact that with a higher time window, the number of rows with the 'Fault Column' with true value is higher and thus the model detects more failure patterns.

## 7.3 Conclusion

With this analysis, we can conclude that our experiment works in predicting most of the fault scopes that we studied. According to the results presented earlier, we conclude that the Two-Class Support Vector Machine give us the predictions that we want, without any false positives which is one of our main goals for the machine learning models. The sliding time window that provide us best results is the 72 hour, so we advice for a future work to study more days as a sliding time window and study if the results improve. We also leave as future work, the possibility of generating for each signal, different running summaries time windows.

# 8

# Work Conclusions

## 8.1 Work Conclusions

This master thesis work allow us to reach the goal initially created by CGI: to understand the major steps and procedures to do in order to create a general experiment that allow us to construct a machine learning model that could predict a failure in a wind turbine. Although it was not possible to create a model that can predict all the fault scopes studied, for 3 of these scopes, (SCA 34 Fault 9112), (CHF 18 Fault 768) and (PCA 7 Fault 140), the result was good and in 5 other, (PCA 7 Fault 140), (CHF 18 Fault 768), (BNE 18 Fault 768), (PAP 35 Fault 8749), (SCA 34 Fault 9112), we had some predictions also. This conclusions give us good perspectives for the use of this experiment in the Failure Prediction tool in CGI RMS that will be use to predict these failures and to try and predict other failures that were not studied during this master thesis work.

## 8.2 Future Work

After the conclusions presented in the previous section, we provide the following items to be studied later by CGI in order to improve the performance of the experiment presented:

1. Feature Importance: Azure Machine Learning Studio presents a module named "Permutation Feature Importance"that have as an output a value associated with each feature representing the importance of that feature in the correct prediction of a failure. This allows to understand which features were more important in the correct prediction of the results. Doing an analysis on the outputs of this module will allow for each failure scope studied, to understand features that are not important and might be removed from the dataset and the features that are considered important and that might be more explored to improve the model performance.

2. Dataset Size: Despite the positive results that were obtained with the dataset that we had available, a dataset with more than a year will be important in order to improve the performance of our model.

3. Running Summaries Features with multiple Windows: In this master thesis work, in the feature engineering step we create the so called running summaries features with a single window. It will be important to do the same but with multiple windows (for example 6h, 12h, 24h, 72h). This way the models algorithms will be able to compare the behaviour of the feature not only in comparison with the value last 6h but also with higher windows, thus allowing to understand possible major changes in the feature.

# Bibliography


[1] A. Agasthian, R. Pamula, and L. A. Kumaraswamidhas. "Fault classification and detection in wind turbine using Cuckoo-optimized support vector machine". In: *Neural Computing and Applications* 31.5 (May 2019), pp. 1503–1511. ISSN: 09410643. DOI: 10.1007/s00521-018-3690-z (cit. on p. 29).

[2] S. Agrawal. *Towards Data Science - How to split data into three sets (train, validation, and test) And why?* May 2021. URL: https://towardsdatascience.com/how-to-split-data-into-three-sets-train-validation-and-test-and-why-e50d22d3e54c (cit. on p. 36).

[3] S. Asaithambi. *Towards Data Science - Why, How and When to apply Feature Selection*. Jan. 2018. URL: https://towardsdatascience.com/why-how-and-when-to-apply-feature-selection-e9c69adfabf2 (cit. on p. 45).

[4] P. Bangalore and M. Patriksson. "Analysis of SCADA data for early fault detection, with application to the maintenance management of wind turbines". In: *Renewable Energy* 115 (2018), pp. 521–532. ISSN: 18790682. DOI: 10.1016/j.renene.2017.08.073 (cit. on p. 19).

[5] A. Bennouk and A. Nejmi. "Wind turbine failures analysis based on performances study and FMECA". In: *Proceedings of the 2018 International Conference on Optimization and Applications, ICOA 2018*. Institute of Electrical and Electronics Engineers Inc., May 2018, pp. 1–6. ISBN: 9781538642252. DOI: 10.1109/ICOA.2018.8370570 (cit. on p. 2).

[6] M. K. Bodla et al. "Logistic regression and feature extraction based fault diagnosis of main bearing of wind turbines". In: *Proceedings of the 2016 IEEE 11th Conference on Industrial Electronics and Applications, ICIEA 2016*. Institute of Electrical and Electronics Engineers Inc., Oct. 2016, pp. 1628–1633. ISBN: 9781509026050. DOI: 10.1109/ICIEA.2016.7603846 (cit. on pp. 17, 19).

[7] E. Bozkurt. *Medium - Machine Learning Classification Algorithms with Codes*. Mar. 2021. URL: https://medium.com/analytics-vidhya/machine-learning-classification-algorithms-with-codes-5a8af4491fcb (cit. on pp. 19, 20).

[8] J. Brownlee. *Machine Learning Mistery - How to Choose a Feature Selection Method For Machine Learning*. Aug. 2020. URL: https://machinelearningmastery.com/feature-selection-with-real-and-categorical-data/ (cit. on p. 45).

[9] J. Brownlee. *Machine Learning Mistery - Train-Test Split for Evaluating Machine Learning Algorithms*. Aug. 2020. URL: https://machinelearningmastery.com/train-test-split-for-evaluating-machine-learning-algorithms/ (cit. on p. 36).

[10] J. Brownlee. *Machine Learning Mistery - What is Deep Learning?* Aug. 2019. URL: https://machinelearningmastery.com/what-is-deep-learning/ (cit. on p. 14).

[11] J. Carroll et al. "Wind turbine gearbox failure and remaining useful life prediction using machine learning techniques". In: (). DOI: 10.1002/we.2290 (cit. on pp. 18, 19).

[12] CGI IT Portugal. *CGI RMS*. Lisbon, Portugal (cit. on pp. 7, 8).

[13] CGI IT Portugal. *CGI RMS Website*. URL: https://www.cgi.com/en/solutions/renewables-management-system (cit. on pp. 3, 6).

[14] Data Science Brigade. *Medium - A Diferença Entre Inteligência Artificial, Machine Learning e Deep Learning*. Aug. 2016. URL: https://medium.com/data-science-brigade/a-diferen%C3%A7a-entre-intelig%C3%AAncia-artificial-machine-learning-e-deep-learning-930b5cc2aa42 (cit. on p. 14).

[15] G. Edwards. *Towards Data Science - Machine Learning | An Introduction*. Nov. 2018. URL: https://towardsdatascience.com/machine-learning-an-introduction-23b84d51e6d0 (cit. on pp. 12–14).

[16] *Energy Gov - Wind*. June 2014. URL: https://www.energy.gov/articles/how-wind-turbine-works (cit. on pp. 9, 12).

[17] S. Faulstich, B. Hahn, and P. J. Tavner. "Wind turbine downtime and its importance for offshore deployment". In: *Wind Energy* 14.3 (Apr. 2011), pp. 327–337. ISSN: 10954244. DOI: 10.1002/we.421 (cit. on p. 2).

[18] R. Gandhi. *Towards Data Science - Support Vector Machine — Introduction to Machine Learning Algorithms*. June 2018. URL: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47 (cit. on p. 31).

[19] A. Gonfalonieri. *Towards Data Science - How to Implement Machine Learning For Predictive Maintenance*. Nov. 2019. URL: https://towardsdatascience.com/how-to-implement-machine-learning-for-predictive-maintenance-4633cdbe4860 (cit. on pp. 18, 19, 36).

[20] S. Griffin. *Medium - Machine Learning for Equipment Failure Prediction and Predictive Maintenance (PM)*. July 2020. URL: https://medium.com/swlh/machine-learning-for-equipment-failure-prediction-and-predictive-maintenance-pm-e72b1ce42da1 (cit. on pp. 18–20, 35, 36, 38, 39).

[21] S. Gupta. *Towards Data Science - How to tackle any classification problem end to end & choose the right classification ML algorithm.* Feb. 2020. URL: https://towardsdatascience.com/how-to-tackle-any-classification-problem-end-to-end-choose-the-right-classification-ml-algorithm-4d0becc6a295 (cit. on pp. 29, 35–37).

[22] S. Gupta. *Towards Data Science - Pros and cons of various Machine Learning algorithms*. Feb. 2020. URL: https://towardsdatascience.com/pros-and-cons-of-various-classification-ml-algorithms-3b5bfb3c87d6 (cit. on pp. 19, 29).

[23] G. Helbing and M. Ritter. *Deep Learning for fault detection in wind turbines*. Dec. 2018. DOI: 10.1016/j.rser.2018.09.012 (cit. on pp. 1, 2, 19).

[24] *IFS Solutions - BoP Definition*. URL: https://ifsolutions.com/what-is-balance-of-plant-in-power-plants/ (cit. on p. xvi).

[25] Jouf University and Institute of Electrical and Electronics Engineers. *2019 International Conference on Computer and Information Sciences (ICCIS) : Jouf University - Aljouf - kingdom of Saudi Arabia, 03 - 04 April 2019*. 2019. ISBN: 9781538681251 (cit. on p. 29).

[26] L. Krippahl. *Theorical Classes from "Aprendizagem Automática- FCT-UNL*. 2019 (cit. on pp. 12, 13, 31).

[27] A. Kusiak and W. Li. "The prediction and diagnosis of wind turbine faults". In: *Renewable Energy* 36.1 (Jan. 2011), pp. 16–23. ISSN: 09601481. DOI: 10.1016/j.renene.2010.05.014 (cit. on pp. 18–20, 25, 35, 45).

[28] S. R. Madeti and S. N. Singh. *A comprehensive study on different types of faults and detection techniques for solar photovoltaic system*. 2017. DOI: 10.1016/j.solener.2017.08.069 (cit. on p. 1).

[29] P. Marti-Puig et al. "Feature selection algorithms for wind turbine failure prediction". In: *Energies* 12.3 (Jan. 2019). ISSN: 19961073. DOI: 10.3390/en12030453 (cit. on pp. 18, 19, 35, 36, 38).

[30] Microsoft. *Azure Machine Learning*. URL: https://docs.microsoft.com/en-us/azure/machine-learning/overview-what-is-azure-machine-learning (cit. on pp. 15, 31–36, 45).

[31] E. Ouda, M. Maalouf, and A. Sleptchenko. "Machine learning and optimization for predictive maintenance based on predicting failure in the next five days". In: *ICORES 2021 - Proceedings of the 10th International Conference on Operations Research and Enterprise Systems*. SciTePress, 2021, pp. 192–199. ISBN: 9789897584855. DOI: 10.5220/0010247401920199 (cit. on pp. 18, 19, 35, 36).

[32] S. Ozturk and V. Fthenakis. "Predicting frequency, time-to-repair and costs of wind turbine failures". In: *Energies* 13.5 (Mar. 2020). ISSN: 19961073. DOI: 10.3390/en13051149 (cit. on pp. 3, 29).

[33] *Power Technology - Top10 Biggest Wind Farms*. URL: https://www.power-technology.com/features/feature-biggest-wind-farms-in-the-world-texas/ (cit. on p. 9).

[34] Ritchie Hannah and M. Roser. *OurWorldInData - Renewable Energy*. 2020. URL: https://ourworldindata.org/renewable-energy (cit. on pp. 7, 8, 10, 11).

[35] M. Ruiz et al. "Wind turbine fault detection and classification by means of image texture analysis". In: *Mechanical Systems and Signal Processing* 107 (July 2018), pp. 149–167. ISSN: 10961216. DOI: 10.1016/j.ymssp.2017.12.035 (cit. on pp. 17–19, 35, 36).

[36] s. kumar gajawada sampath. *Towards Data Science - Chi-Square Test for Feature Selection in Machine learning*. Oct. 2019. URL: https://towardsdatascience.com/chi-square-test-for-feature-selection-in-machine-learning-206b1f0b8223 (cit. on p. 36).

[37] T. Shah. *Towards Data Science - About Train, Validation and Test Sets in Machine Learning*. Dec. 2017. URL: https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7 (cit. on p. 36).

[38] A. Stetco et al. *Machine learning methods for wind turbine condition monitoring: A review*. Apr. 2019. DOI: 10.1016/j.renene.2018.10.047 (cit. on pp. 1, 12, 14, 19, 29).

[39] B. Tang et al. "Fault diagnosis for a wind turbine transmission system based on manifold learning and Shannon wavelet support vector machine". In: *Renewable Energy* 62 (Feb. 2014), pp. 1–9. ISSN: 09601481. DOI: 10.1016/j.renene.2013.06.025 (cit. on pp. 17, 19).

[40] *TutorialsPoint - Renewable Energy Wind*. URL: https://www.tutorialspoint.com/renewable_energy/wind_energy_basic_theory.htm (cit. on pp. 9, 19).

[41] G. B. Upton and B. F. Snyder. "Funding renewable energy: An analysis of renewable portfolio standards". In: *Energy Economics* 66 (Aug. 2017), pp. 205–216. ISSN: 01409883. DOI: 10.1016/j.eneco.2017.06.003 (cit. on p. 1).

[42] A. Verma and A. Kusiak. *PREDICTIVE ANALYSIS OF WIND TURBINE FAULTS: A DATA MINING APPROACH*. Tech. rep. (cit. on pp. 17–19, 35).

[43] Y. Vidal, F. Pozo, and C. Tutivén. "Wind turbine multi-fault detection and classification based on SCADA data". In: *Energies* 11.11 (Nov. 2018). ISSN: 19961073. DOI: 10.3390/en11113018 (cit. on pp. 17–19).

[44] C. Yang et al. "Real-time condition monitoring and fault detection of components based on machine-learning reconstruction model". In: *Renewable Energy* 133 (Apr. 2019), pp. 433–441. ISSN: 18790682. DOI: 10.1016/j.renene.2018.10.062 (cit. on pp. 17–19, 21, 26, 36).

# A

# Appendix 1

In this appendix we will present all the signals that are used as features for each one of the fault scopes studied. As detailed earlier during the document, the scope will be presented in the caption of each table with the following string format: fault id, wind park id, wind turbine/asset id.

Table A.1: Signals selected for Fault 140 PCA 7

| component | sub_component | description |
|---|---|---|
| WGEN_SPD | RPM | Speed generator |
| WGEN_VA | KVA | Apparent power |
| WGEN_W | KW | Active power |
| WNAC_EXTMP | DEGC | Temp outside |
| WNAC_INTLTMP | DEGC | Temp Nacelle |
| WNAC_WDDIR | DEG | Wind direction |
| WNAC_WDSPD | M/S | Wind speed |
| WROT_PTAGVALBL | DEG | Phase angle |
| WROT_ROTSPD | RPM | Speed rotor |
| WTUR_GCTRVEL | RPM | Filtered speed breaking |
| WTUR_PRESFREN | BAR | Brake pressure |
| WYAW_YAWANG | DEG | Nacelle direction |

Table A.2: Signals selected for Fault 367 CHF 15

| component | sub_component | description |
|---|---|---|
| WCNV_GRIA | A | Current Phase L2 |
| WCNV_GRIA1 | A | Current Phase L1 |
| WCNV_GRIA3 | A | Current Phase L3 |
| WGEN_SPD | RPM | Generator speed |
| WGEN_W | KW | Active power |
| WNAC_EXTMP | DEGC | Ambient temperature |
| WNAC_WDDIR | DEG | Wind direction |
| WNAC_WDSPD | M/S | Wind speed |
| WTRF_VSTARS | V | Voltage Phase L1-L2 |
| WTRF_VSTAST | V | Voltage Phase L2-L3 |
| WTRF_VSTATR | V | Voltage Phase L1-L3 |
| WTRM_COOLTMP2 | DEGC | Temperature of Generator's cooling water |
| WTRM_HYOILPRES | BAR | Hydraulic pressure |
| WTRM_HYOILTMP | DEGC | Hydraulic oil temperature |

Table A.3: Signals selected for Fault 768 BNE 14

| component | sub_component | description |
|---|---|---|
| WCNV_GRIA | A | Current Phase L2 |
| WCNV_GRIA1 | A | Current Phase L1 |
| WCNV_GRIA3 | A | Current Phase L3 |
| WGEN_SPD | RPM | Generator speed |
| WGEN_W | KW | Active power |
| WNAC_EXTMP | DEGC | Ambient temperature |
| WNAC_WDDIR | DEG | Wind direction |
| WNAC_WDSPD | M/S | Wind speed |
| WTRF_VSTARS | V | Voltage Phase L1-L2 |
| WTRF_VSTAST | V | Voltage Phase L2-L3 |
| WTRF_VSTATR | V | Voltage Phase L1-L3 |
| WTRM_COOLTMP2 | DEGC | Temperature of Generator's cooling water |
| WTRM_HYOILPRES | BAR | Hydraulic pressure |
| WTRM_HYOILTMP | DEGC | Hydraulic oil temperature |

Table A.4: Signals selected for Fault 768 BNE 18

| component | sub_component | description |
|---|---|---|
| WCNV_GRIA | A | Current Phase L2 |
| WCNV_GRIA1 | A | Current Phase L1 |
| WCNV_GRIA3 | A | Current Phase L3 |
| WGEN_SPD | RPM | Generator speed |
| WGEN_W | KW | Active power |
| WNAC_EXTMP | DEGC | Ambient temperature |
| WNAC_WDDIR | DEG | Wind direction |
| WNAC_WDSPD | M/S | Wind speed |
| WTRF_VSTARS | V | Voltage Phase L1-L2 |
| WTRF_VSTAST | V | Voltage Phase L2-L3 |
| WTRF_VSTATR | V | Voltage Phase L1-L3 |
| WTRM_COOLTMP2 | DEGC | Temperature of Generator's cooling water |
| WTRM_HYOILPRES | BAR | Hydraulic pressure |
| WTRM_HYOILTMP | DEGC | Hydraulic oil temperature |

Table A.5: Signals selected for Fault 8487 SCA 10

| component | sub_component | description |
|---|---|---|
| WGEN_W | KW | Grid Power |
| WNAC_EXTMP | DEGC | Ambient Temperature |
| WNAC_INTLTMP | DEGC | Nacelle Internal Temperature |
| WNAC_WDDIR | DEG | Wind direction |
| WNAC_WDSPD | M/S | Wind Speed |
| WROT_HUBTMP | DEGC | Controller hub temperature |
| WROT_INVTMP | DEGC | Inverter temperature phase 2 |
| WROT_PTAGVALBL | DEG | Blades pitch angle |
| WROT_ROTSPD | RPM | Rotor Speed |
| WROT_TOPTMP | DEGC | Controller top temperature |
| WTRM_TMPGBXOIL | DEGC | Gearbox oil temperature |
| WYAW_YAWANG | DEG | Nacelle direction |

Table A.6: Signals selected for Fault 8719 SCA 28

| component | sub_component | description |
|---|---|---|
| WGEN_GNTMPSTA | DEGC | Phase 2 temperature |
| WGEN_GRIA | A | Current phase 2 |
| WGEN_HZ | HZ | Frequency |
| WGEN_SLRTMP | DEGC | Split ring temperature |
| WGEN_SPINTMP | DEGC | Spinner temperature |
| WGEN_STAPPV | KV | Voltage phase 2 |
| WGEN_W | KW | Grid Power |
| WNAC_EXTMP | DEGC | Ambient Temperature |
| WNAC_WDDIR | DEG | Wind direction |
| WNAC_WDSPD | M/S | Wind Speed |
| WTRF_HVTMP | DEGC | Transformer temperature phase 2 |
| WTRF_INVTMP | DEGC | Inverter temperature phase 1 |
| WTRM_HYOILPRES | BAR | Hydraulic oil pressure |
| WTRM_HYOILTMP | DEGC | Hydraulic oil temperature |
| WTRM_TMPGBXOIL | DEGC | Gearbox oil temperature |

Table A.7: Signals selected for Fault 8749 PAP 35

| component | sub_component | description |
|---|---|---|
| WGEN_W | KW | Grid Power |
| WNAC_EXTMP | DEGC | Ambient Temperature |
| WNAC_INTLTMP | DEGC | Nacelle Internal Temperature |
| WNAC_WDDIR | DEG | Wind direction |
| WNAC_WDSPD | M/S | Wind Speed |
| WTRM_HYOILPRES | BAR | Hydraulic oil pressure |
| WTRM_HYOILTMP | DEGC | Hydraulic oil temperature |
| WTRM_TMPGBXOIL | DEGC | Gearbox oil temperature |
| WYAW_YAWANG | DEG | Nacelle direction |

Table A.8: Signals selected for Fault 9112 SCA 34

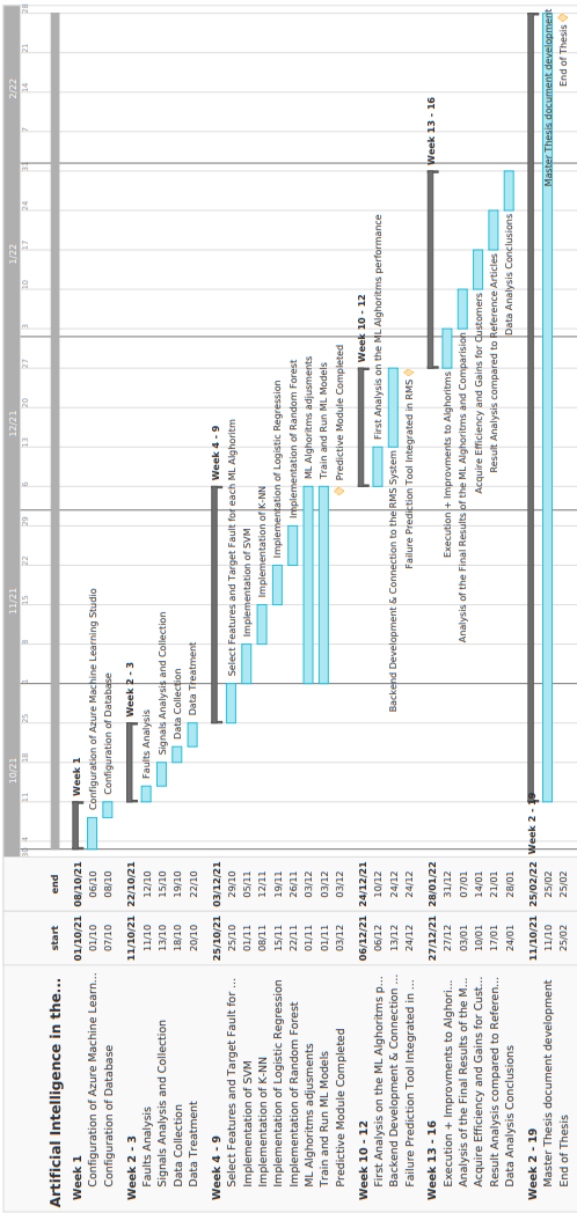
| component | sub_component | description |
|---|---|---|
| WGEN_W | KW | Grid Power |
| WNAC_EXTMP | DEGC | Ambient Temperature |
| WNAC_INTLTMP | DEGC | Nacelle Internal Temperature |
| WNAC_WDDIR | DEG | Wind direction |
| WNAC_WDSPD | M/S | Wind Speed |
| WROT_HUBTMP | DEGC | Controller hub temperature |
| WROT_INVTMP | DEGC | Inverter temperature phase 2 |
| WROT_PTAGVALBL | DEG | Blades pitch angle |
| WROT_ROTSPD | RPM | Rotor Speed |
| WROT_TOPTMP | DEGC | Controller top temperature |
| WTRM_TMPGBXOIL | DEGC | Gearbox oil temperature |
| WYAW_YAWANG | DEG | Nacelle direction |

Table A.9: Signals selected for Fault 14474 TMS 6

| Component | Unit | Description |
|---|---|---|
| WGEN_BRGFTMP2 | DEGC | Blades pitch angle |
| WGEN_SPD | RPM | Generador speed |
| WGEN_W | KW | Active Power |
| WNAC_EXTMP | DEGC | Temp. Ambient |
| WNAC_INTLTMP | DEGC | Temp. Góndola |
| WNAC_WDDIR | DEG | Wind direction |
| WNAC_WDSPD | M/S | Wind Speed |
| WROT_PTAGVALBL | DEG | Blade 1, actual value A |
| WROT_ROTSPD | RPM | Rotor Speed |
| WTRM_TMPGBXOIL | DEGC | Gearbox oil temperature |
| WYAW_YAWANG | DEG | Nacelle direction |

# I

# Work Plan


Artificial Intelligence in the...
Week 1
Configuration of Azure Machine Learn...
Configuration of Database
Week 2 - 3
Faults Analysis
Signals Analysis and Collection
Data Collection
Data Treatment
Week 4 - 9
Select Features and Target Fault for ...
Implementation of SVM
Implementation of K-NN
Implementation of Logistic Regression
Implementation of Random Forest
ML Aghoritms adjusments
Train and Run ML Models
Predictive Module Completed
Week 10 - 12
First Analysis on the ML Aghoritms p...
Backend Development & Connection ...
Failure Prediction Tool Integrated in ...
Week 13 - 16
Execution + Improvments to Aghori...
Analysis of the Final Results of the M...
Acquire Efficiency and Gains for Cust...
Result Analysis compared to Referen...
Data Analysis Conclusions
Week 2 - 19
Master Thesis document development
End of Thesis
start
end
10/21
11/21
12/21
1/22
2/22
Week 1
Configuration of Azure Machine Learning Studio
Configuration of Database
Week 2 - 3
Faults Analysis
Signals Analysis and Collection
Data Collection
Data Treatment
Week 4 - 9
Select Features and Target Fault for each ML Aghoritm
Implementation of SVM
Implementation of K-NN
Implementation of Logistic Regression
Implementation of Random Forest
ML Aghoritms adjusments
Train and Run ML Models
Predictive Module Completed
Week 10 - 12
First Analysis on the ML Aghoritms performance
Backend Development & Connection to the RMS System
Failure Prediction Tool Integrated in RMS
Week 13 - 16
Execution + Improvments to Aghoritms
Analysis of the Final Results of the ML Aghoritms and Comparision
Acquire Efficiency and Gains for Customers
Result Analysis compared to Reference Articles
Data Analysis Conclusions
Master Thesis document development
End of Thesis