POLITECNICO DI TORINO

College of Computer Engineering, Cinema and Mechatronics

Master's Degree Thesis

A data-driven approach for modeling a digital twin of a wind turbine under ideal conditions

Supervisors
prof. Bartolomeo Montrucchio
dr. Antonio Costantino Marceddu

Candidate
Matteo Ferrenti

April 2023

Summary

Wind turbine generators (WTGs) are one of the most widely used sources of renewable energy currently available. To accurately predict their production and quickly notice any anomalies, it is important to analyze the data produced by these turbines to understand their behavior and patterns. The purpose of this thesis is to create a data-driven digital twin of a wind turbine generator capable of simulating its ideal behavior.

To carry out this task, the model receives input data of environmental variables, including wind speed and ambient temperature, and produces output values of parameters that a turbine should have under ideal conditions, including produced power, rotor speed, and more.
The digital twin serves as a reference model that can be used as a comparison metric for the real turbines to evaluate their real-time performance and verify that the turbine is working properly by comparing the parameters of the internal components.
The work was carried out in collaboration with the Turin-based company Sirius s.r.l. and exploits data provided by the company itself and collected at some wind farms in southern Italy. The data is acquired through Supervisory Control And Data Acquisition (SCADA) systems installed on the turbines with the aim of monitoring and collecting data both on the environment and on the internal components of the turbine.

The work is structured in several parts: the initial part is characterized by data extraction and dataset creation. The data is in the form of a ten-minute average and is taken from turbines of the same model belonging to the same wind farm.
Subsequently, a significant work was done on filtering the data with the goal of keeping only the data related to moments in which the behavior of the turbine can be defined as ideal. In this way, the algorithms can be trained only with ideal data and can adequately learn its trends without being misled by other non-ideal data.
To obtain this result, multiple filters have been used considering both environmental and turbine variables, also using algorithms such as the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) for outlier removal.

Finally, several models have been created with the filtered data using different algorithms from both machine learning and deep learning, trying many combinations of inputs and outputs. Specifically, the Feedforward Neural Network (FNN), Support Vector Regression (SVR), and Gated Recurrent Unit (GRU) were tested.
The tests can be divided into two parts: the first part is characterized by the fact that the algorithms are trained on multiple turbines and tested on others never previously seen by the algorithm. Instead, the second part concerns some variables

representing the temperature of the internal components of the turbine, whose behavior is highly variable from turbine to turbine, even if they belong to the same model. In these cases, further studies were needed, leading to alternative solutions. In both cases, satisfactory results were achieved.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In recent decades, the total amount of energy produced has been steadily increasing as a result of the continuous increase in its demand. From 1971 to 2019, global production increased from 230 EJ (Exajoule) to 606 EJ [1], an increase of more than 250%. Figure 1.0.1 shows a graph of the trend of energy production divided by source from 1971 to 2019. More than 75% of energy production comes from coal, oil or natural gas, which are resources present in large quantities on the planet and are easy to use. However, they do not regenerate over time, not in human-scale times. This leads to the fact that sooner or later these resources are bound to run out, and given their very large use in a context of continuously increasing energy demand they are being depleted rapidly. Furthermore, they provide energy through combustion, which produces very high amounts of pollution and greenhouse gases, adding to the problems associated with the current climate crisis.

Given this background, it is easy to see how important it is to find alternative solutions to produce energy as soon as possible. These solutions consist of the use of renewable energy, which is that set of energy sources that are derived from the use of renewable sources. A renewable source is defined as a source that regenerates itself, either naturally or as a result of production processes, fast enough to always replace the amount used in a human-scale amount of time. Some of these sources are called perpetual resources because they are unlikely to ever run out given their speed of regeneration. Sources such as solar energy, wind, water movement and geothermal heat are perpetual sources of energy. Moreover, many of the renewable resources are also sustainable resources, meaning that their use does not compromise the environment or increase the climate crisis.

To avoid the complete depletion of resources and for greater sustainability of energy production in recent decades, the use of renewable sources, especially wind and solar energy, has been growing greatly. Like all high-growth sectors, many studies are being carried out on the subject in the renewables sector. And it is precisely in this context that this work was born, the purpose of which is to build a digital twin capable of simulating the behavior of a wind turbine generator in ideal conditions. This is done in collaboration with the Turin-based company Sirius s.r.l., which provides all the data needed for the study, collected from several wind farms in southern Italy. The data is collected through the use of Supervisory Control and Data Acquisition (SCADA) sensors placed directly on the individual

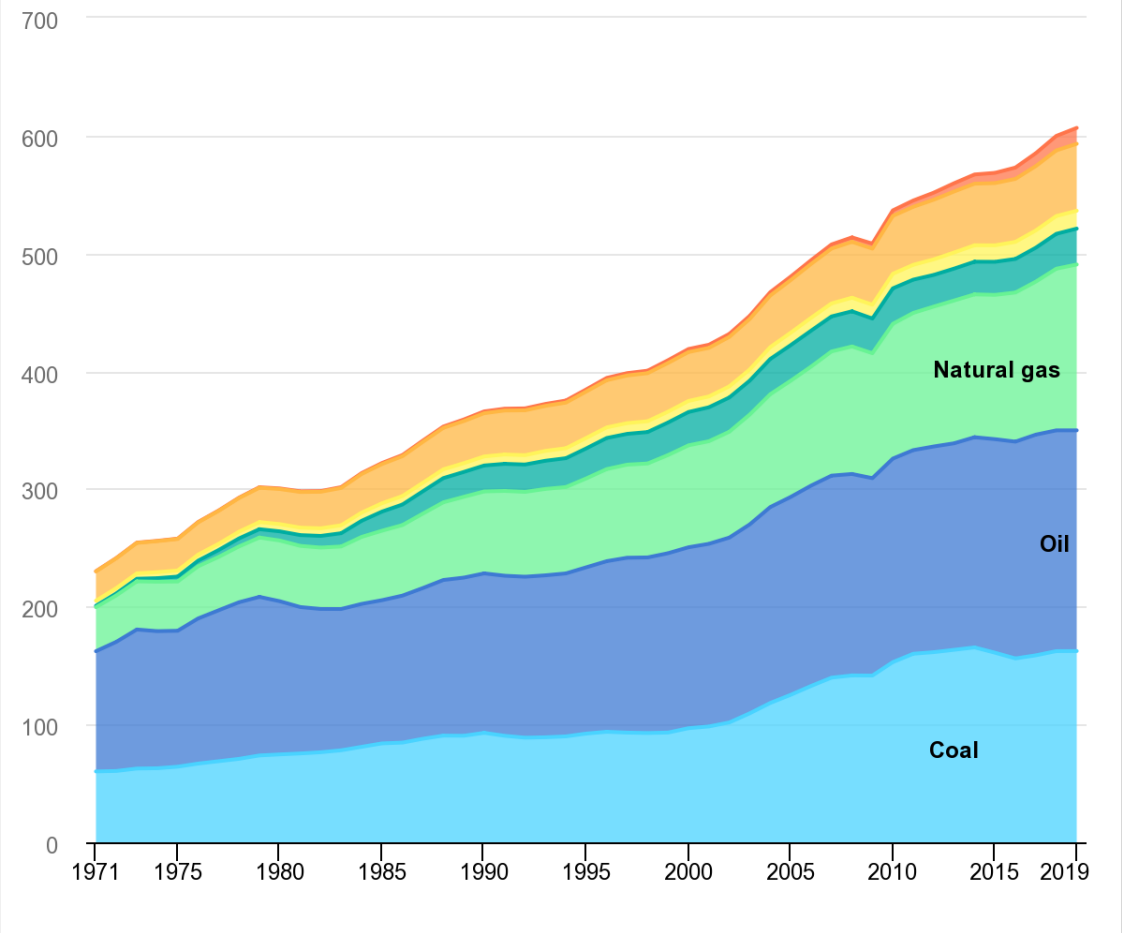

Figure 1.0.1. World total energy supply divided by source, from 1971 to 2019 [1]. The chart already contains which the three main sources are, while the other sources in order from bottom to top are: nuclear, hydro, biofuels and waste, other. Other includes the geothermal, solar, wind, tide/wave/ocean, heat and other sources.

turbines in order to collect data from each of them. The data collected concern both environmental conditions such as wind speed or ambient temperature and data concerning the turbine itself, such as power production, rotor rotation speed or temperature of the various internal components.

Simulating the operation of a Wind Turbine Generator (WTG) means being able to predict fairly accurately how the turbine will behave, using only what a real turbine has at its disposal, i.e., current and eventually previous atmospheric conditions. The behavior of the turbine is represented by the data values collected by the sensors concerning the turbine and its components, while the data concerning the conditions of the surrounding environment provide the information about the atmospheric conditions.
This simulation should be carried out under ideal operating conditions, which means that the model should always represent the turbine in a situation where no malfunctions or problems of any kind are present. The constraint "under ideal conditions" does not refer to atmospheric conditions but to the fact that for each situation that comes up the model is able to provide what the correct operation of the turbine

should be. This involves a great deal of cleaning of the dataset to remove all samples that do not meet ideal conditions so that the training phase is carried out in an environment as ideal as possible.

The development of this work is structured in several distinct parts, each with its own objective and different methods of resolution.
The first phase consists of developing through the use of the Application Programming Interface (API) provided by the company a program capable of downloading the required data and transforming it in such a way as to make it easily usable in the next steps. Immediately after the first phase takes place the second one, which consists of data preprocessing. It includes several sets of filters that are intended to eliminate sets of samples that are invalid or do not meet the requirement of ideal conditions. This part is very important because not filtering properly will result in training the model on non-ideal data, risking creating a model that does not always return the correct operation of a turbine. Samples are removed in which the amount of power produced does not match (with a margin of error) that of the theoretical curves, or samples in which the temperature of an internal component is markedly different from what the temperature of the same component is in similar situations. In addition, studies are also done on the data to calculate and add information not originally present to try to improve the learning process when it does not achieve the desired results. This was especially necessary in the phase of studying internal component temperatures.

Once the data has been properly processed and divided into the train, evaluation and test sets these are used to train the different algorithms and models that have been chosen for the creation of the digital twin. One machine learning model, namely Support Vector Regression (SVR), and two deep learning models, namely Feedforward Neural Network (FNN) and Gated Recurrent Unit (GRU), were chosen as models. In this way, three very different models are tested that operate following different logics. SVRs are an adaptation of Support Vector Machines (SVM) which are able to perform regression and work very similarly to them. FNNs are one of the most classical and basic of the deep learning models, while GRUs are recurrent models that can process sequential data by generating internal memory states that take into account what is contained in the previous data in the sequence.
All of these models require a tuning phase in which, for each of them, many tests must be performed with different hyperparameters in order to choose the ones that obtain better results. In this phase, the results obtained with each test are reported and an attempt is made to motivate the results obtained and choose the most appropriate ones. In addition, tests are also carried out with the additional data produced in the previous stage. The evaluation set is used to perform these tests and make the decisions.

In the final part of the work, once all the models have been tested and the ideal hyperparameters have been chosen, the best ones are used on the test set to perform a more detailed final analysis of their results, providing considerations of their performance and examples of how they work.

Chapter 2

Background

2.1 Thermodynamics effects on atmospheric air: wind

The natural movement of air in relation to the surface of the earth is referred to as wind. When two places have different atmospheric pressures, the air is forced to travel from the high-pressure area to the low-pressure area, which is the major source of wind production [2]. The temperature difference between two geographic regions is what determines the pressure difference; this variation may be either large-scale or small-scale. The intensity of the wind is given by the speed with which the wind makes this movement, in meters per second (m/s).

An example of a large-scale difference is that between the poles and the equator. Due to the varied inclination between the sun rays and the planet surface, the heat that travels to the earth via solar radiation is not uniform, which results in this type of discrepancy. As a result of the sun beams low inclination near the equator, almost perpendicular to the surface of the planet, a lot of heat is produced there, whereas the poles high inclination causes less heat. This is mainly due to the fact that the same irradiation covers an area inversely proportional to the inclination: the area at the equator will be smaller and the heat will therefore be concentrated in that area, whereas the area at the poles will be larger and the heat will therefore be spread over the entire area [3].

The breezes are an illustration of a small-scale variation; these are weak local winds with daily regularity. They are brought on by particular local geography, such as the coastal areas. Earth and seawater have extremely different specific heat capacity; while land warms and cools more quickly, water does so more slowly. These differences in capacities affect the air masses above. As a result, the land quickly warms up throughout the day and produces regions of low-pressure hot air, while the sea produces regions of high-pressure cold air. Due to this reason, air coming from the water moves towards the land. The converse occurs at night, though: the earth rapidly cools, creating high-pressure, while the water maintains its warmth from the day heat, creating low-pressure. Because of this, air moves from the land to the sea in the opposite direction from how it does during the day [4].

Figure 2.1.1 shows an example of how a pressure difference, and consequently a wind, can be generated.

Finally, there are other sources of wind not related to the pressure difference, such as those due to the Coriolis effect caused by the Earth rotation.

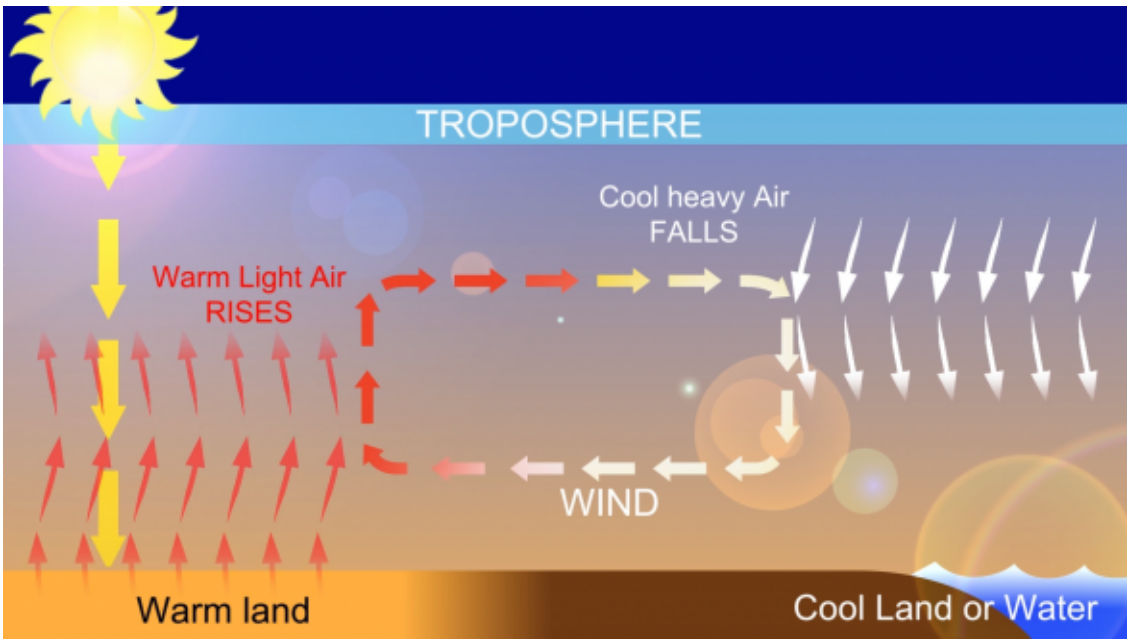

Figure 2.1.1. The cycle bottom part, which is depicted in the illustration, is the wind that blows over the earth surface; the cycle higher component takes place at high altitudes and is irrelevant to our needs [5].

2.1.1 Weibull Distribution

The Weibull distribution, which is a probability density function, can be used to statistically represent the distribution of wind speeds at a given location. Although there are several distributions that can be used to simulate wind dispersion, this one is the most popular in the industry because it most closely approximates it [6]. An integral of this distribution over an interval returns the probability that the wind will be within that interval.
The following formula defines it [7]:

$$f(V) = k(\frac{V^{k-1}}{C^k})e^{-(\frac{V}{C})^k}$$

where V is the wind speed and k is the Weibull form parameter, which determines the distribution shape and has a range of values from 1 to 3. Higher values indicate constant winds, while smaller values indicate variable winds. Scale parameter C is equal to mean wind speed.
An example with real data is shown in Figure 2.1.2.

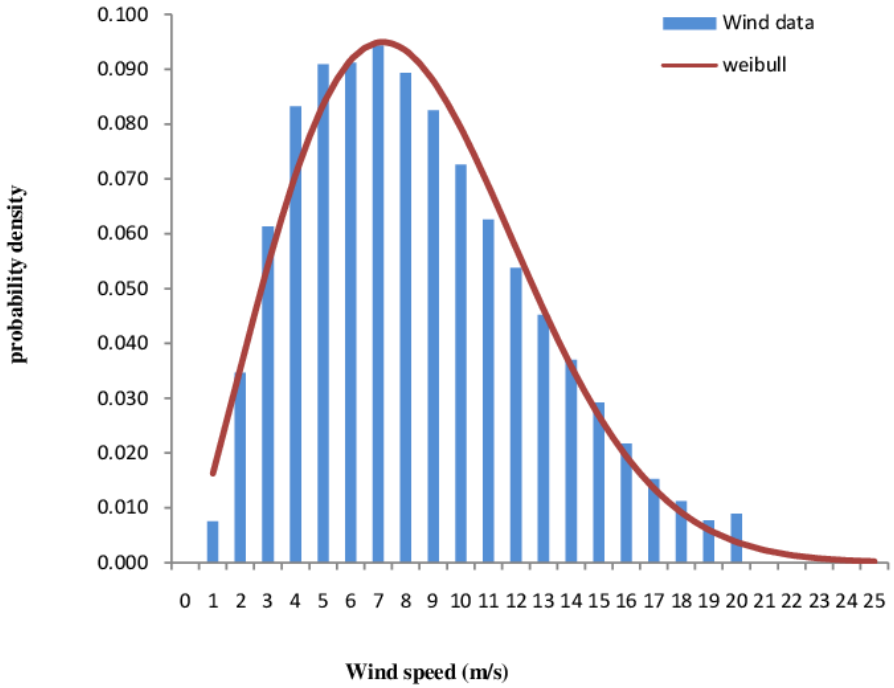

Figure 2.1.2. Comparison between a Weibull distribution and real wind data in Col de Touahar Taza, as shown in [6].

2.1.2 Energy Contained in the Wind

The wind has a finite amount of energy, and its value can easily be calculated. It is needed to know the speed and density of the wind, and decide on the size of the area in which this quantity needs to be computed.
Given a circular area of section S_r, density ρ and velocity V_0, the energy contained in the flow is [8]:

$$E = \frac{1}{2}\rho S_r V_0^3 \text{ [J]}$$

In this formula, it is important to note that the energy contained in the wind is directly proportional to the density of the air. This means that the ambient temperature influences the amount of energy since the density in turn depends on the air temperature.
This formula is especially important when applied to the section of area covered by the blades of a wind turbine: Figure 2.1.3 shows the shape of the flux tube tangential to the tip of the blades. Refer to the next section for more information on the names of the components of a turbine.

The flow tube has the same energy at every point; this means that where the section is smaller than the section of the area covered by the blades, the air contains more energy for the same section (it is faster and/or denser) while where the section is greater the air contains less energy section parity (it is slower and/or less dense). As can be seen, the flow has a smaller section than the blade area before reaching the turbine. In the vicinity of the turbine, the flow widens; once it has passed the turbine it has a larger cross-section than the area of the blades. This behavior

occurs because the turbine subtracts energy from the wind, which therefore has a lower speed and/or density than before crossing it.

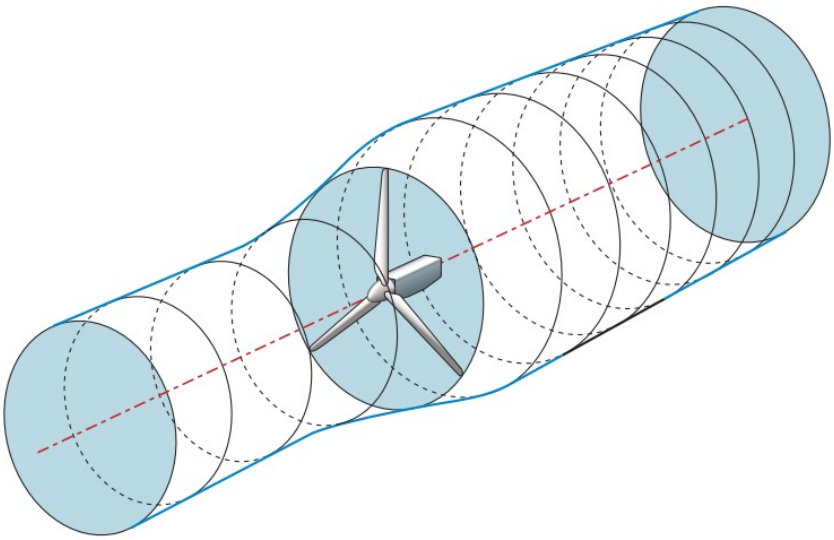

Figure 2.1.3. How the section of a constant-energy air flow changes in passing through a turbine [8].

Since the turbine subtracts energy from the wind, a constant C_p can be defined which determines the amount of energy that the turbine subtracts from the wind:

$$P = C_p E$$

where P is the power absorbed by the turbine and E the energy contained in the wind.
It is possible to obtain the maximum theoretical value of C_p, which represents the theoretical limit of energy that can be extracted from the wind with a wind turbine [8]. This limit is known as the Betz limit, and is equivalent to:

$$C_{p_{max}} = 0.593$$

Modern turbines are capable of reaching 70-80% of this theoretical limit [8].

2.2 Wind Turbine Generator

Wind turbines are devices specifically designed to convert the energy contained in the wind into electricity. The basic functioning is the opposite of that of a fan: instead of using electricity to spin a motor to which blades that move the air are connected, it is the air that spins the blades which are however connected to a generator, which produces electricity.

There are mainly two types of turbines, which differ in the direction in which their axis of rotation is oriented [9]:

- **Horizontal axis**: These are the most common ones, they have the axis of rotation parallel to the ground and the direction of the blades perpendicular to it. The direction of the axis of rotation must be facing that of the wind to obtain maximum productivity. It is necessary to place the generator and gearbox at the height of the rotor, which is placed on the top of the tower. Being large and heavy components, placing them at the top involves higher costs. The shape, material, and number of blades is variable and there are many types, but the modern wind turbine model is the most efficient and reliable due to its three-bladed shape and is what is considered in this study. An example of a modern wind turbine is shown in Figure 2.2.1.

Figure 2.2.1. Horizotal axis wind turbine [10].

- **Vertical axis**: Vertical axis turbines, on the other hand, have the rotation axis perpendicular to the ground, which brings a series of advantages such as not needing to be oriented according to the wind and being able to position the generator and gearbox on the ground, being possible to carry the rotation from the top directly to the bottom. The biggest disadvantage, however, is the reduced efficiency, which is why they are not used in large wind farms in favor of their horizontal axis counterpart. An example of a vertical wind turbine is shown in Figure 2.2.2

Modern horizontal-axis turbines provide the best performance [12], and are therefore the ones used in large wind farms such as those considered in this study. For this reason, they are the type of turbine that will be analyzed in the most detail.

Figure 2.2.2. Vertical axis wind turbine [11].

2.2.1 Wind Turbine Components

All the most important components that make up a horizontal-axis turbine and what their functionalities is are explained below [8]:

- **Tower**: The tower is a circular metal structure with variable height, generally one and a half times the length of the blade. Its job is to stably and safely support all components resting on it, which need to be placed at high heights because the farther we are from the ground, the faster the wind. It must be able to withstand continuous oscillation due to the rotation of the system above and the pressure generated by the wind, even in the case of windstorms.
- **Rotor**: The rotor is the rotating part of the turbine. It is located at the front of the nacelle. The rotor includes the hub and the blades.
- **Hub**: The hub is the rotating component at the tip of the nacelle, whose job is to keep the blades attached.
- **Blades**: The rotor blades are the components of the wind turbine that are most noticeable. Typically, they are constructed of a lightweight material

resistant to the bending stress caused by the angle of incidence between the wind direction and the blade cross section. Materials with these characteristics are for example carbon fiber or fiberglass.
The wind is captured by the rotor blades, which then transform its kinetic energy into rotary energy. Modern turbine blades are shaped to maximize aerodynamic efficiency; close to the hub, the blade section is circular, and the further away the blade is from the hub, the more the thickness decreases. It is necessary to have a shape that ensures good lift and low resistance, especially towards the ends because as the distance from the center increases, the relative speed increases quickly. The blade also wraps around itself at an angle of about 25 degrees. It is possible to see the shape of the blade and its individual sections in Figure 2.2.3. Through a system for adjusting

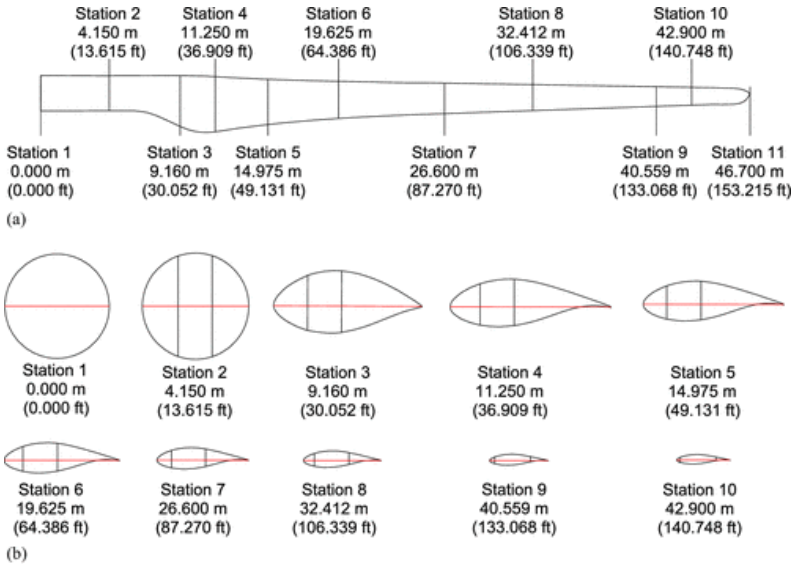

Figure 2.2.3. Blade shape and sections at different distances from the hub [13].

blade pitch, the blades can also rotate around themselves. The angle created between the plane of the blade segment and the plane of the rotor is known as the blade pitch angle. In order to control the rotor rotational speed and the amount of energy stored, this angle is adjusted in accordance with the wind speed. In situations where wind speeds are too high, the angle is adjusted so that the leading edge faces the wind, reducing the aerodynamic load on the blades and the force applied to the tower.

- **Nacelle**: The gearbox, generator, and other mechanical and electrical parts of the wind rotor are accommodated in the nacelle. It is built to turn and face the wind and stands atop the tower.

- **Shaft**: The shaft is the part that transmits the rotation of the component in which it is generated, i.e. the rotor, up to the generator. The shaft can be divided into two distinct parts: the first is the one that goes from the rotor to the gearbox and is the one that spins at low speed, i.e. the rotational speed of the rotor. The second is the one that goes from the gearbox to the generator and runs at high speed.

- **Gearbox**: The gearbox is a vital component for the turbine, which allows converting the low rotation speed of the shaft produced by the rotor to a high rotation speed, required by the generator.
- **Generator**: The generator, which performs the task of converting rotational mechanical energy into electrical energy, is a crucial component of the turbine. Generators come in two varieties: synchronous and asynchronous. The primary distinction between the two is that asynchronous generators produce energy at a constant rotational speed; in order to maintain this constant speed, a gearbox is required. On the other hand, the synchronous ones can still generate power despite variable shaft rotation speed because they generate energy at variable frequency, directly proportional to rotation speed, which is then converted into fixed frequency by an electronic rectifier. Although larger and more expensive, these generators do not require a gearbox. The asynchronous generator with gearbox is a feature of the turbine models used in this study.
- **Brake**: The turbine brake is the component that serves to stop the rotation of the rotor and blades in case of need, or when the turbine does not have to produce. This situation can arise for various reasons, such as wind speeds that are too low to produce energy or too high and therefore dangerous, or due to breakdowns or repairs, or due to limitations in energy production imposed from outside.
- **Yaw**: The yaw system is the device that allows the turbine to turn and follow the direction of the wind so that the rotor can always face the direction of the wind. The rotation includes the entire upper part of the turbine, i.e. the nacelle. The tower remains fixed. The system consists of two components, the yaw motor which generates the mechanical energy required for rotation and the yaw drive which applies rotation to the nacelle.
- **Anemometer**: The anemometer is the instrument placed above the nacelle which has the purpose of measuring the speed of the wind.
- **Wind Vane**: The wind vane is the instrument placed above the nacelle which has the purpose of measuring the direction of the wind. Knowing the direction of the wind is useful for orienting the turbine using the yaw system.

2.3 Statistics

In the following Subsections, all the notions of statistics necessary to fully understand what is done during this work are provided.

2.3.1 Moving Average

A particular kind of mean used for data series is the moving average. It is determined by creating a series of averages, one for each point in the dataset. In order

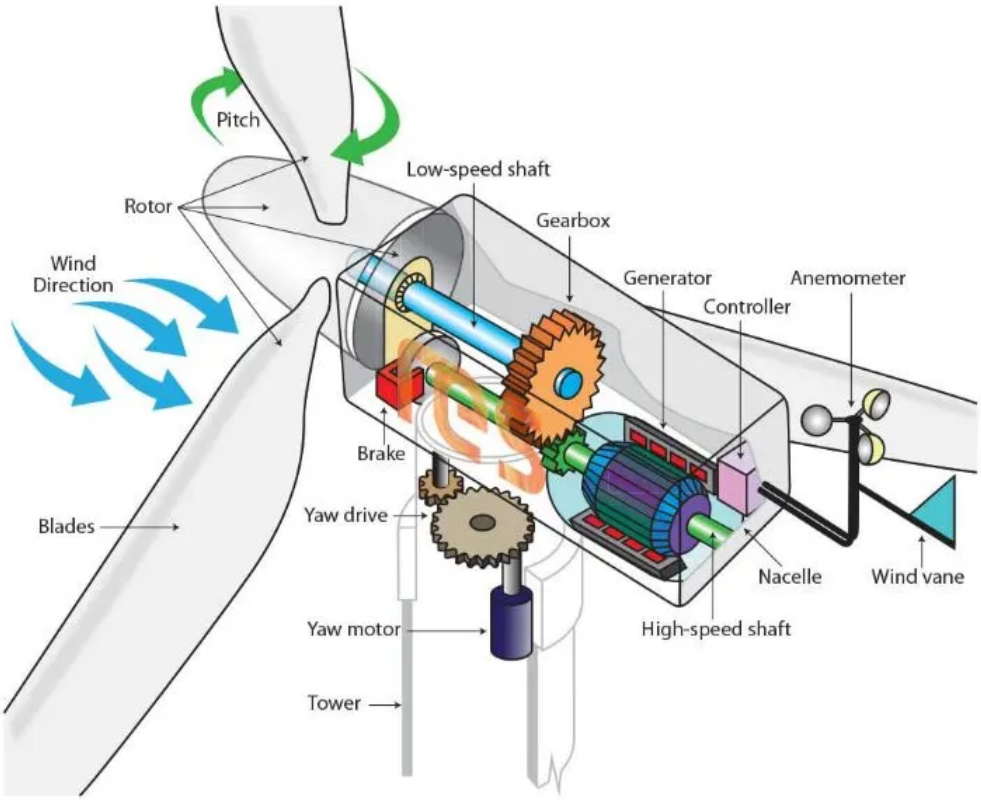

Figure 2.2.4. Arrangement of components in a horizontal-axis wind turbine with asynchronous motor and gearbox [14].

to provide information on the local average, each average is only calculated on a subset of the points, i.e. on the k points closest to the point where the average is calculated.
A variation of the moving average that uses the k points prior to the point where the average is computed is called simple moving average. The simple moving average can be determined as follows for the nth point p_n in the dataset [15]:

$$SMA_k = \frac{p_{n-k+1} + p_{n-k+2} + ... + p_n}{k} = \frac{1}{k} \sum_{i=n-k+1}^{n} p_i$$

2.3.2 Standardization

The standard score, also referred to as the z-score, shows how much the observed value is above or below the average of the dataset. Standard scores are positive for values above the mean and negative for values below the mean. The standard score value reveals how many standard deviations from the mean the observed value deviates. It is determined as follows [16]:

$$z = \frac{x - \mu}{\sigma}$$

Determining the standard score for each sample is the same thing as standardizing a dataset. This produces a new dataset with some peculiar properties, including

zero mean and unitary variance. The new mean is zero because the dataset mean has been subtracted from each sample, and the new variance is one because each sample has been divided by the dataset variance.

2.3.3 Correlation Matrix

Given a group of variables, a matrix known as a correlation matrix is used to find out the relationships between each of them. When two variables are linearly correlated, they are said to be related to one another.
The most popular method for determining correlation is the Pearson correlation coefficient, which is a normalized measure of correlation with a range from -1 to 1. This metric ignores all other forms of correlation and only considers the linear correlation between variables. A correlation value of 1 corresponds to a perfect linear relationship, i.e. the two variables have exactly the same increasing trend. Conversely, a correlation of -1 indicates the opposite, ie that they have exactly the opposite trend. A correlation of 0 means that there is no relationship whatsoever between the two variables and that they are independent of each other. Numbers between 0 and 1 indicate a partial positive correlation, increasing as the correlation increases. The same is true for values between 0 and -1, but with partial negative correlation [17].
The correlation between two random variables X and Y must be calculated by dividing the covariance of the two variables by the product of their standard deviations as follows [18]:

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y} = \frac{\mathbb{E}\left[(X-\mu_X)(Y-\mu_Y)\right]}{\sigma_X \sigma_Y}$$

The correlation matrix contains the variables to be analyzed on both the columns and the rows, and each element corresponds to the correlation between the variables of the row and of the column. This implies that each element of the diagonal of the matrix is compared with itself and therefore has a maximum correlation, i.e. 1. The part above the diagonal mirrors the one below, thus leading to an exclusion of the pairs on the diagonal in double copy. An example of correlation matrix is shown in Figure 2.3.1.

2.4 Outlier Detection

According to the science of statistics, an outlier is a point in a dataset that differs significantly from all other observations and only occasionally occurs [20]. Figure 2.4.1 shows an example of outliers.
In the field of data analysis and machine learning, these samples are frequently taken out of the dataset because anomalies are likely to cause issues, such as inaccurate mean and standard deviation values or a reduction in the accuracy of the final results in the case of supervised learning in case the dataset goal is to train an algorithm for a specific purpose [21].
For the purpose of removing outliers from the dataset, particular automatic techniques have been created. The ideal approach would be to examine every single

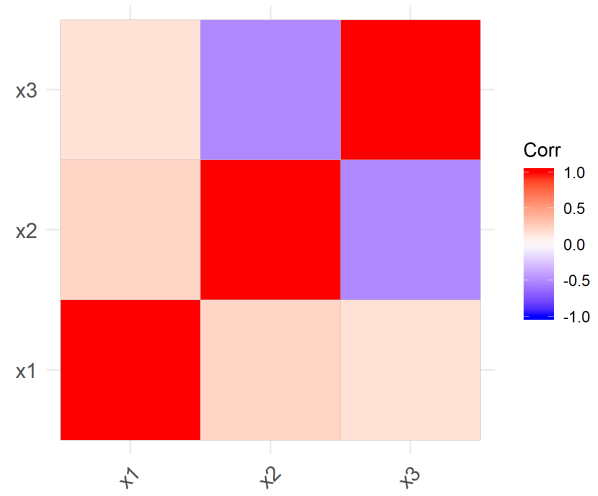

Figure 2.3.1. Example of correlation matrix taken from [19]. It is commonly used to color the boxes according to the correlation, attributing two distinct colors for positive and negative correlation and blending them according to the degree of correlation, until reaching a correlation of 0 represented by white. In this way, even without reading the individual values (which are not shown in this example matrix) it is possible to understand how the dataset is structured very quickly.

point to determine whether it should be classified as an outlier or not, but this is not practically feasible, so algorithms are used instead. These algorithms generally work well for most samples, but they may also eliminate some high-quality samples or leave some outliers in the dataset.

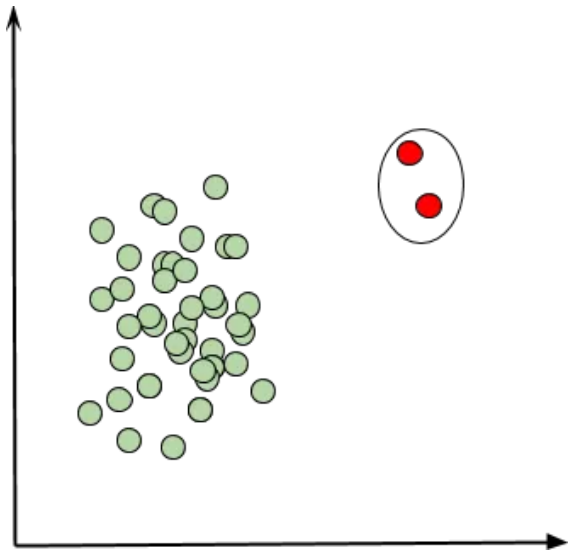

Figure 2.4.1. A graph from [22] that provides a visual example of what outliers look like. The set of green dots represents the samples of the dataset that are similar to each other and respect a certain distribution, while the two circled red dots are outliers, and as can be seen they are detached from the rest of the points and do not respect their distribution.

2.5 Digital Twin

A digital twin is a representation of a complex system, process, or actual item in the digital domain. The items or systems that are duplicated might be quite diverse, with various sizes and levels of complexity. A full system, a section of it, or a component of it may all be duplicated. It replicates the real object behavior and performance by using a virtual, interactive version of it [23].
A wide range of industries, including manufacturing, building, wind energy, agriculture, healthcare, and many more, may employ the digital twin. The digital twin may assist discover any issues or inefficiencies, forecast future behavior, improve performance, and determine the time for routine maintenance since it can offer a real-time digital version of the physical thing. Moreover, the digital twin may be utilized to try out situations and solutions in a virtual setting, lowering the costs and dangers of in-person testing and making it useful for design.

The digital twin often adopts a data-driven methodology, where data is continuously gathered through sensors, sensing equipment, and other data sources. The behavior of the model under study is then determined using this data, which is subsequently analyzed using artificial intelligence and machine learning techniques. Although it doesn't require much system expertise, creating an accurate model does demand a lot of data and cleaning effort.
This strategy contrasts with the model-based strategy, which creates a model of the system under examination using theoretical, mathematical, physical, or statistical information. The model, which is predicated on a number of presumptions, explains how the actual system works. The biggest drawback is that in order to effectively characterize a system that is being modeled, one must have a very thorough grasp of it [24].

2.6 Interpolation

Interpolation is an estimation method developed in mathematics called numerical analysis. It consists in finding new data starting from the ones we already have, which are generally data obtained experimentally. To do this, it is necessary to find the values that the original function assumes among the known points, in order to be able to estimate new points. The interpolation can be calculated on one dimension (univariate interpolation) or on several dimensions (multivariate interpolation).

The most common methods for calculating univariate interpolation are:

- **Linear**: it consists of connecting each point with the next with a straight segment.
- **Polinomial**: it consists in finding a polynomial of high degree that adequately approximates the trend of all points
- **Spline**: also in this case polynomials are used, but instead of using a single one of high degree to approximate all the points in one go, multiple ones of low degree are used to approximate subsets of points.

An example of spline univariate interpolation is shown in Figure 2.6.1.

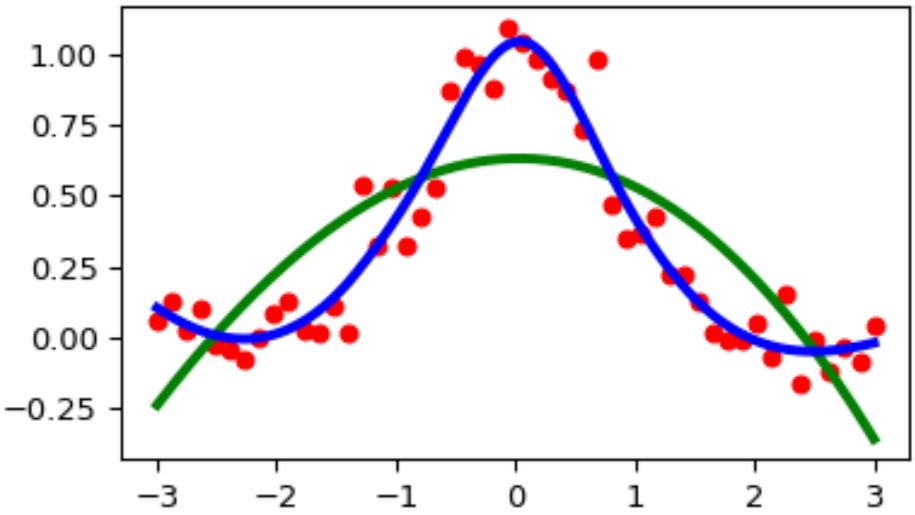

Figure 2.6.1. Interpolation example taken from [25], with two different smoothing factors.

2.7 Machine Learning

Machine learning (ML) is a branch of artificial intelligence that deals with creating models and algorithms that let computers learn from data without having to be explicitly programmed to carry out every action [26]. In other words, machine learning enables computers to examine vast volumes of data, find patterns, and identify relationships within them, then use that understanding to forecast or decide on fresh data.

In conventional programming, inputs and outputs are handled explicitly by the programmer, who must write the code to develop the input data and produce accurate output data. The software must be designed to execute a specific operation on a single input and cannot generalize the solution to other inputs without changing the code. Machine learning, on the other hand, trains the model on a collection of labeled input and output data in order to learn to generalize the relationship between the input and output data.
In this manner, the model may be utilized to automatically process new input data and provide accurate output data without the need for further source code alterations 2.7.1.

Machine learning comes in several forms, and they may be identified by the nature of the input and output data as well as the learning techniques applied. Machine learning may be divided into three primary categories [27]:

- **Supervised Learning**: In this type of machine learning, the model is trained on a dataset of pre-labeled inputs and outputs in order to learn a mathematical function that maps the inputs to the correct outputs. Once trained, the

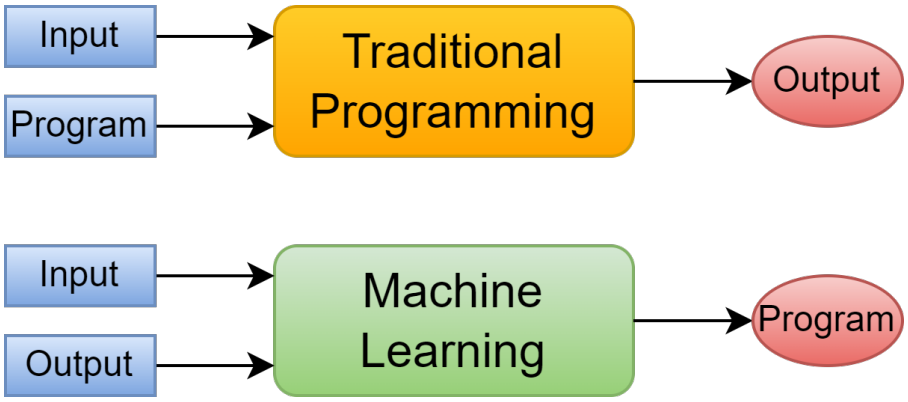

Figure 2.7.1. A basic idea of the structural difference between regular program and machine learning.

model can be used to make predictions on new input data.

- **Unsupervised learning**: In this type of machine learning, the model is trained on a set of unlabeled input data in order to find patterns or clusters within the data. It is often used for data analysis and data clustering.
- **Reinforcement Learning**: In this type of machine learning, the model learns through interaction with its surroundings. The model receives feedback based on the actions it takes and learns how to maximize a certain reward or goal. It is often used for creating intelligent agents and for robotics.

Furthermore, deep learning should be mentioned since it is a well-known subclass of machine learning. Deep learning uses artificial neural networks with numerous hidden layers and is capable of learning intricate patterns from unstructured input data.
The complexity of the models that can be learned is the primary distinction between deep learning and machine learning. While deep learning allows to build complicated models and learn from unstructured data much more effectively, machine learning focuses on creating comparatively simple models [28].

2.7.1 Density-Based Spatial Clustering of Applications with Noise DBSCAN

The goal of the clustering analysis is to divide a set of data into various groups (referred to as clusters), each of which comprises samples that are similar to one another and distinct from the samples of the other groups for the same feature. There are numerous definitions of clusters and clustering algorithms [29], and one of the best known and most used algorithms is known as DBSCAN, with its density-based cluster definition.

DBSCAN is a density-based unsupervised clustering technique that may divide data into groups according to density. A point neighborhood is defined as a circle with a radius of ε and a center at the point. For each point p, we count the number of points (also counting p) that are present in its neighborhood. The point p is

considered as a core point if there are at least $minPts$ points in its neighborhood. The parameters ε and $minPts$ are hyperparameters and can be adjusted as needed. All points that aren't designated as core points can be:

- **Directly reachable** from a point q if q is a core point and they're at most ε away from q.
- **Reachable** from a point q if a path of directly reachable points runs between them. By this definition, the point q and all connecting points are required to be core points.
- **Unreachable** from a point q if they are neither directly reachable nor reachable from that point.

The clusters are created immediately after all the points have been classified: each core point creates a cluster with any non-core or core points that can be reached from it. Different clusters are made up of groups of distinct core points that are not linked to one another. Outliers are any points that cannot be reached from any other point and are not a part of any cluster [30]. Figure 2.7.2 shows an example of a DBSCAN result.

The main benefit of DBSCAN is that it can identify clusters regardless of their shape, which is something that many other clustering algorithms cannot do. This is because the clusters are assigned based on the density of the points, which eliminates the need to specify the number of clusters to search for, which is frequently unknown beforehand. Another benefit is that the algorithm ability to recognize and distinguish outliers makes the DBSCAN robust since it is not impacted by any outliers.

However, assigning clusters based on density has a certain number of drawbacks. For example, due to the curse of dimensionality, which makes it extremely challenging to find suitable ε and $minPts$ values in datasets with high dimensionality, the distance metric used for distance calculation, the Euclidean distance, becomes useless in these situations. Furthermore, it is challenging to distinguish between clusters if they have drastically different densities since not all clusters can have an adequate selection of hyperparameters [32].

Although it is not its primary function, DBSCAN can be used as an outlier detection algorithm in addition to clustering. This can be achieved by eliminating the outliers that the algorithm discovers automatically or, in some circumstances, by classifying specific clusters as outliers that the algorithm recognizes but that, in the given case study, need to be removed. The method for deciding which clusters to keep and which to discard differs from standard practice because it is only an alternative application of the algorithm and must be defined in on the basis of each unique case study.

2.7.2 Support Vector Regression (SVR)

Support vector regression (SVR) [33] is a machine learning technique used for regression, that is, to predict a continuous numeric value rather than a class or category. It aims to find a function that approximates the input data while minimizing

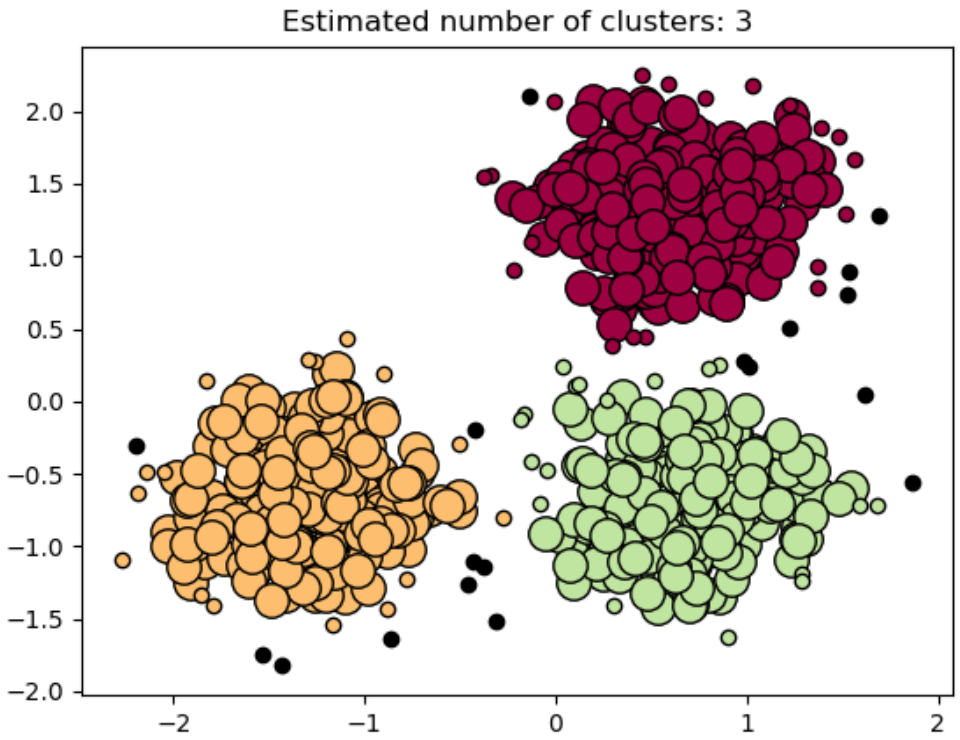

Figure 2.7.2. This graph [31] displays the outcome of using the DBSCAN algorithm on a set of data. As can be seen, the data are clearly split into three distinct groups, and there are some outliers within each group. The algorithm successfully distinguished between the three clusters and gave each one a distinct color. It also correctly identified the outliers, which can be distinguished from other points because they are not colored.

the prediction error and maximizing distance between the function and the nearest points.

The SVR functions similarly to the support vector machine (SVM) [34], but rather than classifying the data into two groups, it seeks a function that both fits the data and doesn't fit it too much. This means that the function must be adaptable enough to change in response to inputs while not being overly adaptable to noise.
The SVR evaluates the function included in the input data before making a judgment. The prediction is deemed accurate if the function delivers a result that is reasonably near to what was anticipated given an input. Among other benefits, SVR has the ability to handle both linear and nonlinear data, is resilient against noise in input data, and is adaptable to the data.
To obtain such results, the objective function is minimized while subject to certain constraints [35]:

- **Objective Function**:

$$\underset{w}{\text{minimize}} \ \frac{1}{2}\|w\|^2$$

- **Contraints**:

$$|y_i - w_i x_i| \leq \varepsilon$$

The constraint dictates that each point must have a maximum absolute error ε, which can be adjusted. This creates a distance ε from the hyperplane, and the points residing on it are called support vectors.
However, this condition cannot always be respected and for this reason a further term ξ is introduced, called slack variable, with the function of adding a weight to all those points that do not fall within the constraint. For all points that are already inside the margin, this variable is 0, reverting the problem for those points to the one that has already been seen. Instead, the value of ξ at points outside the margin is greater than 0, inversely proportional to the distance from the margin. Since the goal is to have the least possible number of points that do not respect the constraints, this weight must be minimized together with the function. For those points that do not respect the constraints, their distance from the margin must be minimized.
Using the slack variable and the new constraints, the new objective function is as follows [35]:

- **Objective Function**:

$$\underset{w}{\text{minimize}} \ \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}|\xi_i|$$

- **Contraints**:

$$|y_i - w_i x_i| \leq \varepsilon + |\xi_i|$$

The constant C, which stands for how much importance is given to the points outside the margin, controls the new variable. A problem known as overfitting occurs when the algorithm learns to predict only the input dataset and does not generalize well to other datasets. This prevents the algorithm from making accurate predictions on other datasets.
The example in Figure 2.7.3 illustrates the outcome of using the SVR; the regression line is indicated in red, and the top and lower borders are indicated in gray, both at eps distance. The slack variable is bigger than 0 at points outside the margin.

2.7.3 Feedforward Neural Networks (FNN)

Artificial intelligence algorithms, also known as neural networks, were influenced by the design and operation of the human brain. Similar to it, an artificial neural network is made up of several interconnected neurons, which process information via their connections and activation functions. They are modeled in an artificial neural network as interconnected nodes that receive input from external sources and produce output through activation functions [37].
Neural networks have the capacity to learn from data and form conclusions and predictions on their own. They are utilized in a variety of fields, such as banking, marketing, speech recognition, machine vision, picture categorization, and many

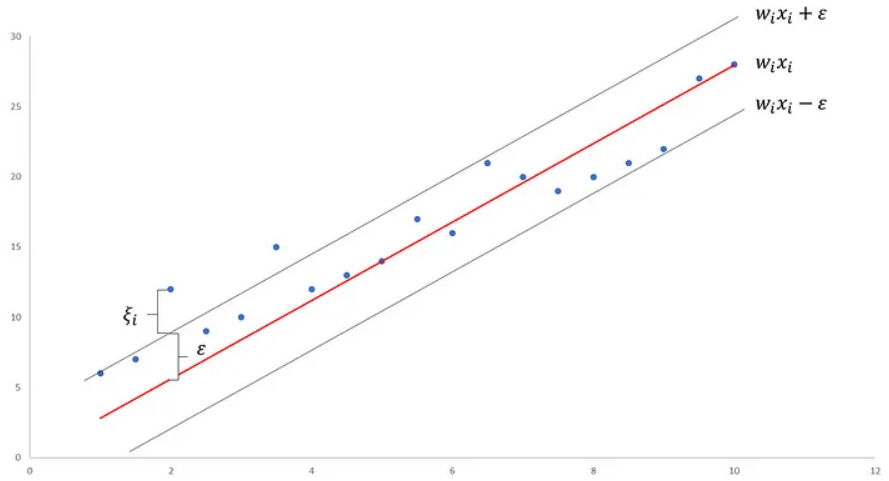

Figure 2.7.3. Example graph of SVR, taken from [35]. In this case a Linear SVR has been used, but it is also possible to use the kernel trick [36] as in the SVM to obtain non-linear results.

more. When compared to other machine learning or statistical analysis techniques, neural networks have demonstrated a unique ability to address issues that would otherwise be challenging or impossible to do so. They have grown more potent and adaptable with the introduction of new deep learning techniques, creating new possibilities for artificial intelligence and its use in a variety of industries [28].

The term feedforward neural network (FNN) refers to a specific type of artificial neural network in which signals are fed forward from input to output through one or more hidden layers of neurons that do not use feedback or recursion. This kind of neural network is utilized for continuous value classification, regression, and prediction. The number of hidden layers in a feedforward neural network determines its internal structure. Three different layer types make up a typical feedforward neural network [38]:

- **Input layer**: The network first layer, known as the input layer, is where data enters the system. In this layer, each neuron corresponds to a certain aspect of the input. For instance, each neuron of this layer represents one pixel of the image if the network has been trained to identify photos of animals.
- **Hidden layers**: Between the input and output layers of a neural network are what are known as the hidden layers. The number of neurons that make up each buried layer varies depending on how difficult the problem is. Each neuron in a hidden layer is linked to every other neuron in the layer above and below it. The nonlinear interactions between the input features are captured by hidden layers.
- **Output layer**: The output of the network is represented by the output layer, which is the final layer of the network. In this layer, each neuron stands for a class or an output value. For instance, if the neural network was taught to recognize photos of animals (dog, cat, horse, etc.). they could indicates the likelihood that an image belongs to one of the aforementioned classes.

In a feedforward neural network, every neuron is linked to every other neuron of the previous and following layers. A synaptic weight, which reflects the significance of a connection for calculating output, is linked to each neuronal connection [39]. A structure of a FNN with 3 hidden layer of 5 neurons each is shown in Figure 2.7.4.

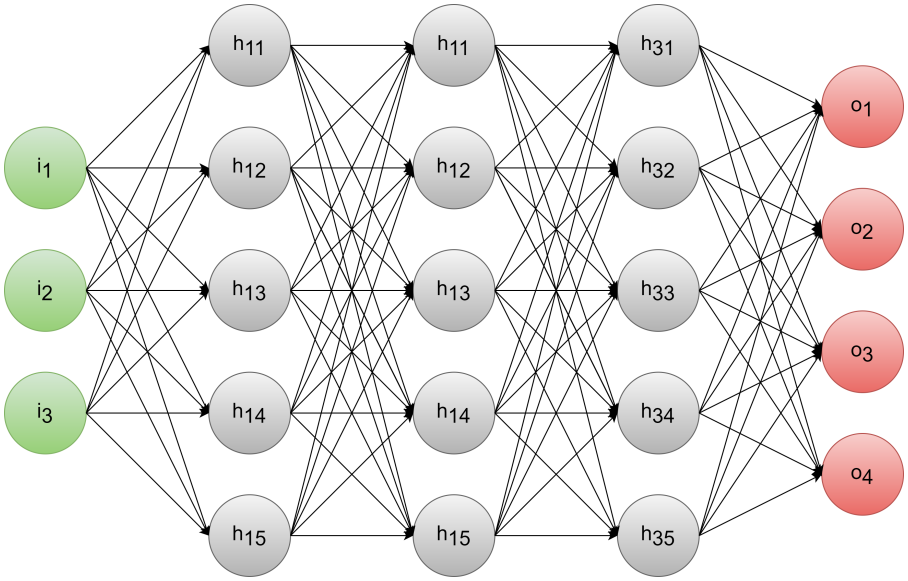

Figure 2.7.4. Structure of a FNN with 3 hidden layer of 5 neurons each: the neurons related to the input layer are colored in green and are as many as the inputs. The hidden layers are hued in gray, and can be in variable quantities. Not only the number of hidden layers can be chosen, but also the number of neurons of each hidden layer, without necessarily having to keep the same number for each hidden layer. In this example, there are 3 hidden layers of 5 neurons each. Finally, in red, the output layer with the number of neurons equal to the number of outputs. Each of the arcs that connect the neurons has its own weight and each neuron ha its own bias.

A feedforward neural network bases its computation on the mathematical function feedforward, in which a neuron output is determined depending on its inputs and associated synaptic weights. The input is first provided to the network first layer of neurons, where the feedforward process starts. These neurons generate an output by activating their inputs, which is subsequently sent to the network next layer by these neurons. The feedforward process keeps on until the final output is generated by the last layer of neurons of the network.
Mathematically, this is what happens during the forward step for each neuron and for each hidden layer [38], as is also shown in Figure 2.7.5:

$$y_n = f(\sum_{i=0}^{m}(h_{n-1,i}w) + b)$$

where n is the number of the layer on which the forward is being made, m is the number of neurons of the previous hidden layer, w is the synaptic weight that connects the neurons of layer n-1 to those of the n^{th} layer (a different weight for each single connection), b is the bias (a bias for each neuron of the nth hidden layer), and f is the activation function.

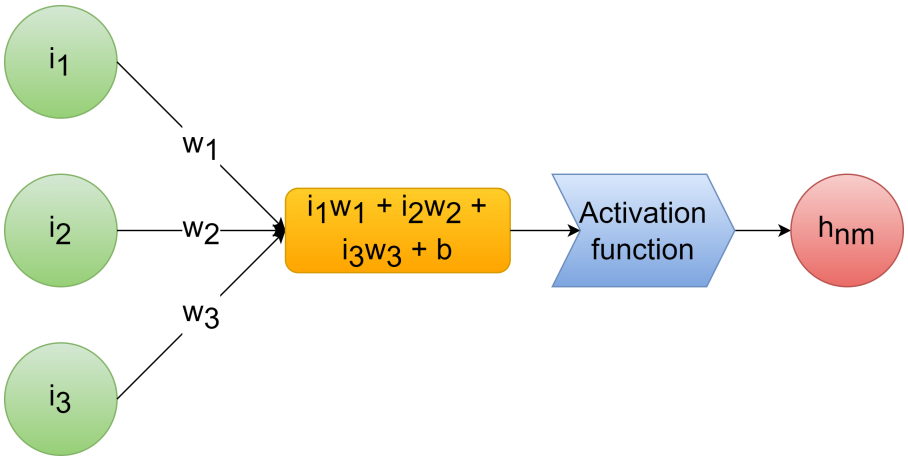

Figure 2.7.5. Graphical example of what happens for each single neuron (except the input ones that receive the value from the outside) during forwarding.

The activation function is a non-linear function through which the data are passed, having the purpose of introducing non-linearity in the network. This allows to respect the universal approximation theorem, which says that any continuous function can be approximated by a suitable set of combinations of linear and non-linear functions.
The backpropagation algorithm is used by FNNs to learn from the data. It involves calculating the discrepancy between the desired output and the actual output produced by the network, using this error to modify the weights of the connections between neurons in order to minimize the error going forward [40].

The advantages of FNNs include their propensity to process massive amounts of data, discover intricate patterns, be applied to a variety of problems, and generalize from training data to fresh data. Nevertheless, drawbacks include the requirement for a lot of training data to prevent overfitting, the computational complexity needed for training, and the challenge of deciphering the neural network inner workings.

2.7.4 Recurrent Neural Networks (RNN)

Recurrent Neural Networks (RNNs) are a class of artificial neural networks that are created to handle data in sequences, such as time series, natural language, etc. RNNs, in contrast to conventional neural networks, feature one or more layers of memory units that enable the network to keep some type of memory of prior data. This indicates that the RNN can analyze each element in a sequence based on its prior values as well as its relationship to the overall context of the sequence, in addition to the current value of the element [41].
The RNNs have a clear structure that allows them to achieve this outcome. Each layer is made up of a memory cell that stores the current state of the sequence, which is determined by the state of the previous element of the sequence and the current element as input. Once the current state has been determined, an output can be made and the state can be sent to the following layer. A graphical overview is shown in Figure 2.7.6.

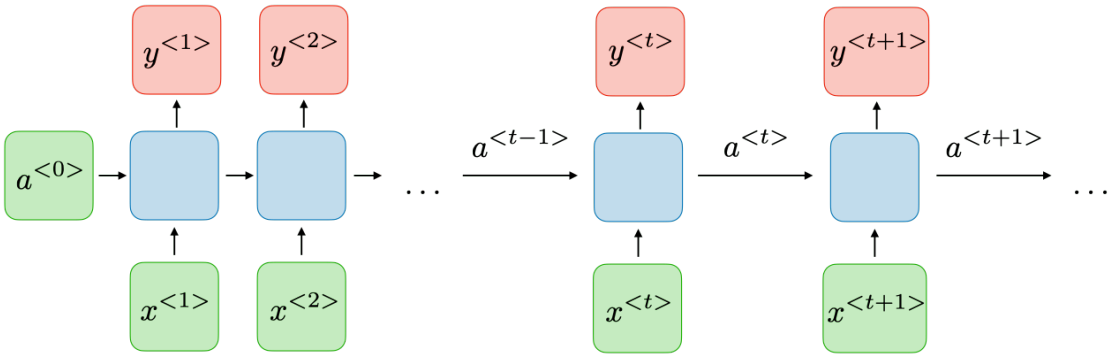

Figure 2.7.6. Structure of a many-to-many RNN with the same number of inputs and outputs [42].

Furthermore, it is possible to create RNNs composed of several layers stacked vertically as shown in Figure 2.7.7, thus having several internal states to represent each element of the sequence. This type of architecture is called Deep-RNN (DRNN).

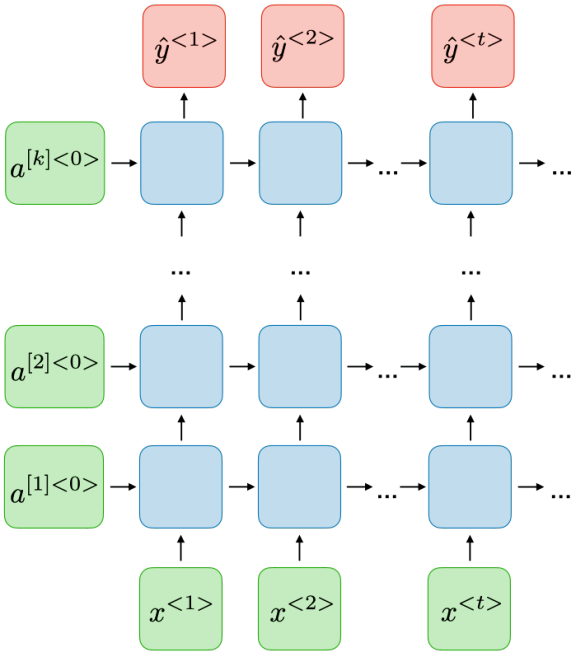

Figure 2.7.7. Structure of DRNN [42].

RNNs may also be built in a variety of ways, depending on what they will be used for. These include "one-to-one" RNNs, which process non-sequential data, "one-to-many" RNNs, which process a single input to produce a series of outputs, "many-to-one" RNNs, which process a series of inputs to produce a single output, and "many-to-many" RNNs, which process a series of inputs to produce a series of outputs [42].

The fundamental issue with RNNs is caused by the vanishing gradient and exploding gradient problems, two related issues that frequently arise in RNNs that

might impair the network capacity to train and generate reliable results.
The vanishing gradient happens when the gradient calculated during the backpropagation phase significantly decreases as it moves farther away into the network, getting smaller and smaller until it is almost zero. As a result, the network is unable to understand the long-term associations between the input data since the learning algorithm cannot properly update the connection weights [43].
On the other side, an exploding gradient happens when the gradient increases too quickly as it moves away into the network, growing larger and larger until it becomes unstable. As a result, the network may become unstable during learning as the connection weights may grow too large.

The RNN design, which allows for error propagation over connections at each time step during the backpropagation phase, may be the root source of both of these issues. This means that the gradient of the present storage unit is influenced by the gradients of the storage units from earlier steps. As a result, the gradient may swiftly increase or rapidly fall.

2.7.5 Gated Recurrent Unit (GRU)

The GRU is a variation of the RNN architecture that was created to overcome the problem of vanishing gradients in conventional RNNs and is extensively employed in numerous speech recognition and natural language processing tasks [44].

The primary distinction between a GRU and a conventional RNN is that a GRU contains two internal gates, an update gate and a reset gate, which control the information flow within the network. Whereas the reset gate decides how much of the previous hidden state to forget, the update gate governs how much of the previous hidden state is mixed with the current input [45].
Hence, the internal structure of the memory units differs between RNN and GRU, but the exterior structure of RNNs and GRUs is the same. Figure **??** illustrates the internal structure of an RNN and a GRU memory cell. As can be observed, the GRU now includes the aforementioned ports.
As can be seen from Figure 2.7.8, the GRU has a much more complex internal structure. The hidden state h_t in an RNN is calculated as [46]:

$$h_t = \tanh\left(Wh_{t-1} + Ux_t\right)$$

where W is the weight that is applied to the previous input state and U the weight that is applied to the input x. Once the state h_t is produced, an output y can be generated by applying another weight V, and then using the softmax function.

Instead, the hidden state in a GRU is compured as [47]:

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tanh\left(Ux_t + r_t \odot Wh_t - 1\right)$$

The notations are the same as the previous formula, with the addition of z which is the update gate, and r which is the reset gate, whose formulas are:

$$z_t = \sigma(U^{(z)}x_t + W^{(z)}h_{t-1})$$

$$r_t = \sigma(U^{(r)}x_t + W^{(r)}h_{t-1})$$

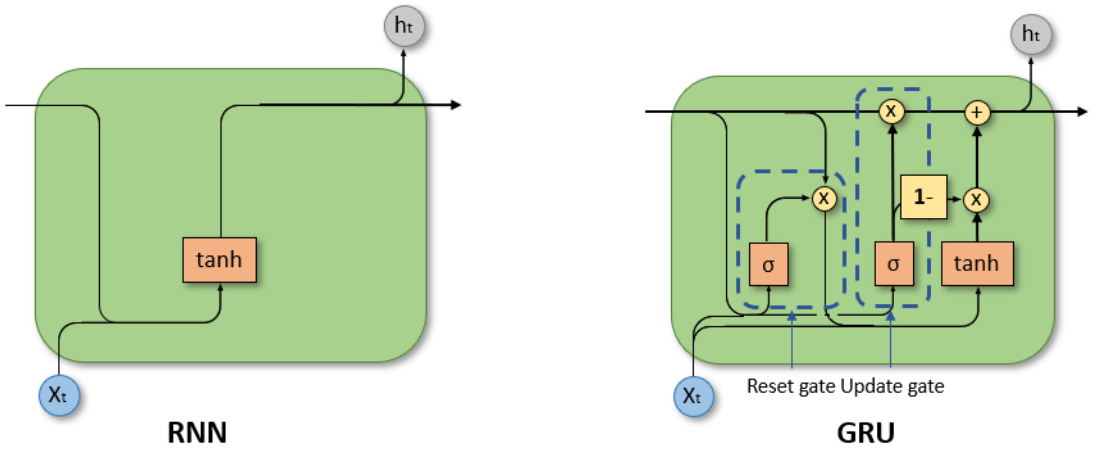

Figure 2.7.8. RNN memory cell architecture(on left) vs GRU memory cell architecture (on right) [45].

Also in this case it is then possible to produce the output y by applying a further weight V to the state and the softmax function.
It is important to note that the notation does not include the biases, which are added at the same time as the weights.
In calculating the state of the GRU, it can be seen that the formula is divided in two by a sum: the part on the left represents how much to keep from the previous state, while the one on the right how much to keep from the current state (already influenced by input x), which is in turn obtained through the reset gate which decides how much to delete from the previous state. To decide how much to maintain of each of the two parts the update gate is used, which being the result of a sigmoid is a number between 0 and 1, and is interpreted as the fraction of the previous state to maintain, while the remainder $(1 - z_t)$ is taken from the current state.

2.7.6 Other useful notions

Mean Squared Error

Mean Squared Error (MSE) is a metric used in machine learning to measure the severity of errors made in predictions. To calculate it, it is necessary to calculate, as the name implies, the mean squared error, that is [48]:

$$MSE = \frac{1}{N}\sum_{i}^{N}(y_i - \hat{y}_i)^2$$

where y_i is the observed value and $\hat{y}_i$ is the predicted value

Coefficient of determination R^2

The coefficient of determination, better known as R^2 (R squared) is a metric used to measure the goodness of predictions made by a predictor. It indicates how good

the model is at replicating observed values, and to do so it relates the predictions of the predictor under analysis to those of a dumb predictor that always predicts the mean value of the measure under analysis.
It is calculated as follows [49]:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

where

$$SS_{res} = \sum_{i}^{N} (y_i - \hat{y}_i)^2$$

is the residual sum of squares, with y_i observed value and $\hat{y}_i$ predicted value.
And

$$SS_{tot} = \sum_{i}^{N} (y_i - \bar{y})^2$$

is the total sum of squares, with y_i observed value and $\bar{y}$ mean value for the observed data.

R^2 values can range from minus infinity to 1. The value of 0 is an important value, as it indicates that the predictions have the same level of accuracy of the dumb predictor just mentioned.
Values greater than 0 indicate that the predictor performs better than the dumb predictor, up to an upper limit in 1 indicating that the predictions are identical to the observed values.
Finally, negative values thus indicate worse performance than the dumb predictor, with virtually no lower bound.

Chapter 3

Dataset and Data Preprocessing

This chapter provides a brief introduction of the dataset that is used for the development of this thesis, going to explain what the different features represent and how they are collected, how the data is selected and used, and all the process they undergo before being used for training, also known as preprocessing.

3.1 Dataset

The Turin-based company Sirius s.r.l. provided the dataset used for this thesis. It offers a substantial amount of information about numerous wind farms located throughout southern Italy. These wind farms contain different models of turbines; from the data it is possible to see that some of them behave quite similarly, while others diverge visibly. The dimensions of the turbine, from which all the other various characteristics are derived, are the primary distinction.

Since it is very difficult to create a single digital twin that generalizes multiple turbine models at once, especially if the models are not similar to each other, it has been decided to use data that only refers to one turbine model. Additionally, only the turbines from the same wind farm were chosen in order to ensure greater data coherence since they are all in the same location, are exposed to similar environmental conditions, and likely were installed at the same time.

The data is gathered directly through SCADA systems with sensors that are physically installed on the turbines. These sensors gather pertinent information like wind speed, power generated, internal component temperature, and more. They are able to produce data at a rate of seconds, but this data is saved as a ten-minute average, which is calculated every ten minutes from the data for that period. This allows for the storage of long and complex data for numerous turbines without taking up excessive amounts of space, and it also allows for a broader perspective of the trend of the variables. However, the detailed view of what actually happens is lost, leaving out important details.

The dataset as a whole consists of data that was provided by 18 turbines over the course of a full year. This provides data for all seasons and avoids to make it biased.

As shown in Figure 3.1.1, it was divided into 3 smaller datasets for model training purposes : train, validation and test.

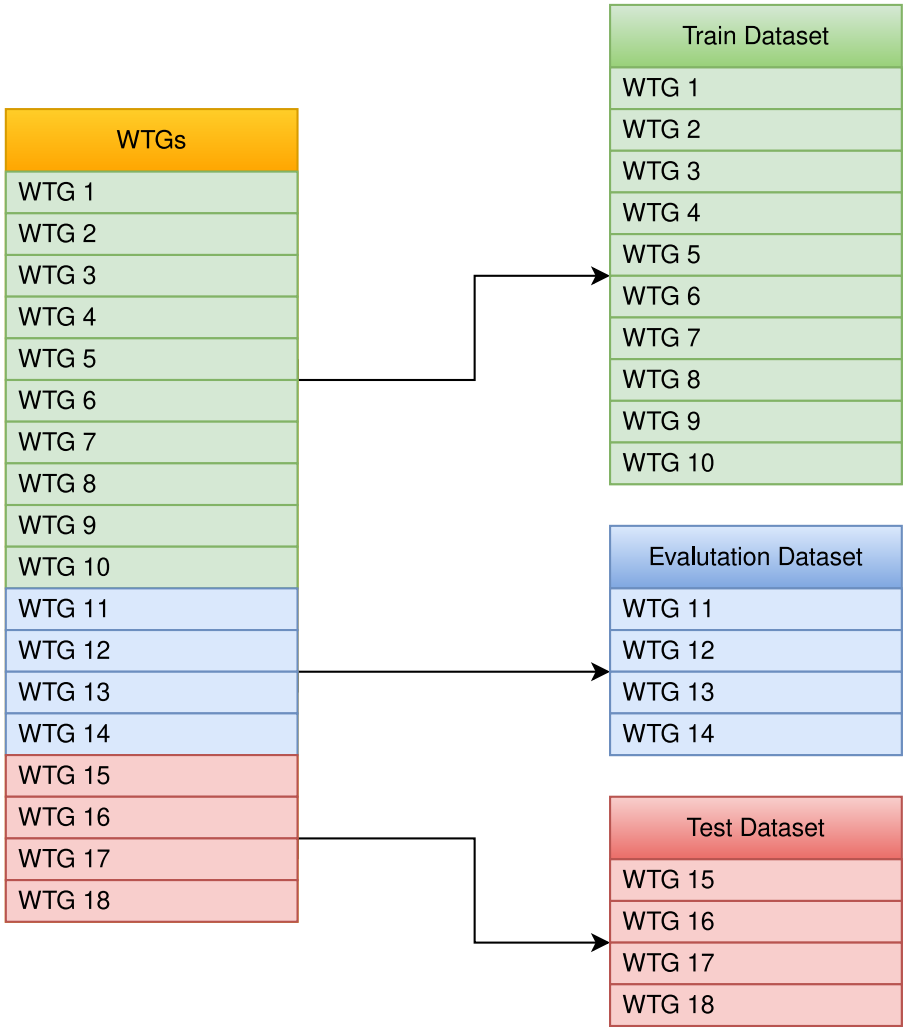

Figure 3.1.1. Dataset division in train, validation, and test datasets.

The samples are not divided completely randomly, but those belonging to a turbine remain in the same dataset, so that in each dataset only data relating to turbines that are not present in the other two are available. Their dimensions are not equal: that of train is larger as it is the one on which the network is trained, and is composed of 10 of the total 18 turbines. The other two datasets consist of 4 turbines each, and are used for different purposes. Validation dataset is used to test and evaluate models, to choose which parameters are best, and generally make any necessary choices. Test dataset instead is used to carry out the final tests. It is not used for any other purpose and no choices are made based on the test results, so as not to have a model with bias towards the data present in the dataset on which the final results are calculated. This is because test formally represents real data in a real use scenario, so they must be data that has never been seen before and that it is not possible to use for anything except to carry out the final tests.

3.1.1 Measures

This Subsection explains which are the main measures present in the dataset and what they refer to. In the original dataset there were measures that were not considered useful for carrying out this work and for this reason they were omitted. Here are the measures used:

- **Timestamp**: Indicates the instant of time all other values refer to. It is saved as a Unix timestamp, which is the amount of milliseconds that have passed since 00:00:00 on January 1, 1970.
- **Wind Speed** [m/s]: Wind speed at the top of the turbine, regardless of its direction.
- **Wind Vane** [°]: Direction of the wind with respect to the orientation of the turbine, calculated in degrees.
- **Wind Standard Deviation**: Since there are only data in the form of ten-minute averages, it is not possible to know how the data vary, if they are always close to the average or if they fluctuate a lot. This information is particularly important for wind speed as it represents its turbulence. The reason that the standard deviation of the wind represents its turbulence is that the latter is defined as a rapid fluctuation in wind speed, whereas the standard deviation indicates for the measurement on which it is calculated by how much it deviates from its mean. This means that the higher the standard deviation of the wind speed is, the further its values are from the sample mean, which consequently implies that the wind speed is not stable and therefore turbulence is present. For this reason, it is averaged over a ten minute period in order to have a measure of its turbulence.
- **Ambient Temperature** [°C]: Temperature of the environment where the turbine is placed.
- **Blade Pitch Angle** [°]: Inclination of the blades with respect to their own axis, as explained in Subsection 2.2.1.
- **Active Power** [W]: Power generated by the turbine.
- **Rotor RPM** [RPM]: Number of rotations per minute made by the hub.
- **Gearbox Bearing Temp** [°C]: Temperature of the bearings inside the gearbox. The reason why the temperature of the bearings is recorded is that they are very important for the operation of the gearbox and their role leads them to generate a lot of heat. Due to this reason, they are the main source of failure for this component [50]. Consequently monitoring these temperatures makes it possible to quickly notice when they exceed a minimum or maximum temperature threshold and issue alarm signals and possibly stop the turbine, or to notice when there are suboptimal conditions even if the temperatures meet the alarm thresholds.

- **Gearbox Oil Temp** [°C]: Temperature of the gearbox oil, used for lubrication and friction heat displacement. It is closely linked to the previous measure as the two components are in direct contact.
- **Generator Bearing Temp** [°C]: Temperature of the generator bearings. As in the case of the gearbox, they are a source of heat and therefore it is advisable to control their temperature.

For what regards the digital twin, it is critical to distinguish between features that will be input and those that will be output. The distinction is quite intuitive: all input variables are regarded as environmental variables, i.e., those that report information obtained from the environment and are independent of the turbine. All other variables that are directly related to the turbine are used as output since the digital twin main function is to forecast these values.

There are two exceptions: the first are the timestamps, which are neither inputs nor outputs but only serve to link the other variables to the same instant. The other exception is the blade pitch angle which, despite being a turbine variable, is still regarded as an input because it is directly related to wind speed in a known and predictable manner: the higher it is, the more the blades of the turbine will rotate.

Before starting to work with a dataset, it is always advisable to do some study on it to obtain useful information. The first thing to do is calculate the mean and variance of each feature, which are given in Table 3.1, to see how the dataset is distributed. In the later stages of the study, knowing these values can be helpful to

Feature Name	Mean	Standard Deviation
Wind Speed [m/s]	4.62	3.59
Wind Vane [°]	0.08	40.4
Wind Standard Deviation [m/s]	0.8	0.53
Ambient Temperature [°C]	15.02	7.6
Blade Pitch Angle [°]	20.56	27.84
Active Power [W]	315.94	538.92
Rotor RPM [rpm]	8.18	5.95
Gearbox Bearing Temp [°C]	56.6	11.64
Gearbox Oil Temp [°C]	54.15	8.66
Generator Bearing Temp [°C]	47.31	13.16

Table 3.1. Mean and Standard Deviation for each measure.

better understand why certain things happen in order to look for possible solutions.

3.1.2 Correlation Matrix

Another important measure to monitor is correlation between all pairs of variables. This can be made by calculating the correlation matrix.

From it, shown in Figure 3.1.2 it can be observed that many of the variables are positively correlated with each other, which makes sense: as the wind speed increases, the rotor RPM, generator RPM and consequently the energy produced increase. The increase in the rotation speed leads to a greater internal heat production, and consequently the temperatures also rise.

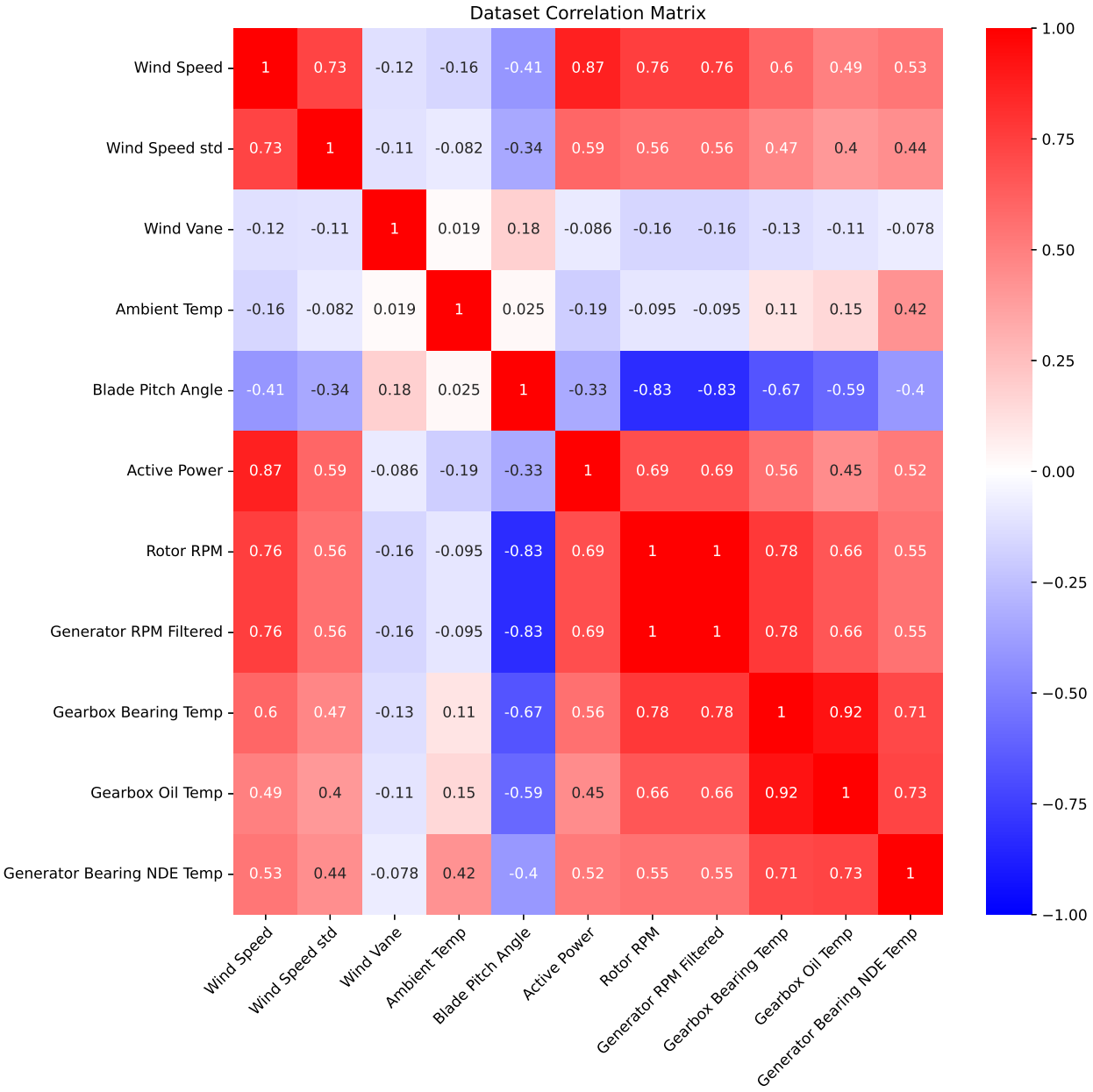

Figure 3.1.2. Correlation Matrix of the dataset.

However, there are some obvious exceptions: ambient temperature has almost no correlation with many of the variables and low with others. This is because it has no influence on the rotor and generator RPM which depend on the wind speed.

It has a low influence on energy production since, as seen previously, wind energy depends on its density, which in turn depends on temperature. The only correlation of note is with generator bearing temperature.

Another interesting exception is that of the blade pitch angle, which has negative correlations with the other variables. This is because the blades rotate (inwards, therefore increasing the blade pitch angle) as wind speed decreases in order to reduce the load on the turbine tower, which with too high wind speeds could lead to problems.
However, this involves a slowdown in the rotation speed of the rotor, as the amount of energy absorbed by the wind is decreasing; this explains the negative correlation with the turbine-related variables. Finally, it should be noted that the rotation speeds of the generator and of the rotor have a correlation of 1: this means that they have exactly the same trend and that there are redundant data. For this reason, in subsequent studies only one of the two variables will be taken into consideration, specifically the rotor RPM.

Finally, the wind vane has virtually zero correlation with all other variables, which is why it will be ignored as well.

3.2 Preprocessing

Data preprocessing consists of applying a series of transformations and manipulations to data with the aim of improving the performance of the models that must be built using this data.
The operations carried out allow unnecessary data to be removed through a process called data cleaning: using statistical and heuristic methods it is possible to remove all those points that can be considered incorrect, inconsistent, irrelevant or misleading due to some of their characteristics.
In this phase, it is important to keep in mind that the final aim of this thesis is the construction of a model under ideal conditions: this implies that the data cleaning process will also include the removal of data acceptable for a generic purpose but not belonging to ideal conditions.

This section will show the filters and transformations applied to the dataset, divided by category.
There are many measures in the dataset and displaying them all is difficult and can be confusing. One of the most important measurements to take into account, especially in this preprocessing section, is active power. If observed in relation to the wind speed, it provides an excellent indicator of the level of cleanliness of the data since generally the amount of energy produced based on the wind speed is quite stable and it is therefore easy to observe samples that do not respect this ratio.

In addition, the manufacturers of wind turbines provide the theoretical wind-power curve that the turbine should respect, so there are theoretical values to refer to. For this reason, after each filtering step, this scatter will be displayed accompanied by the number of total points represented, in order to be able to understand how much and how each filter affects the dataset.

Figure 3.2.1 shows what the data looks like before applying filters. The red curve is the theoretical curve provided by the manufacturer for the turbine model under analysis.

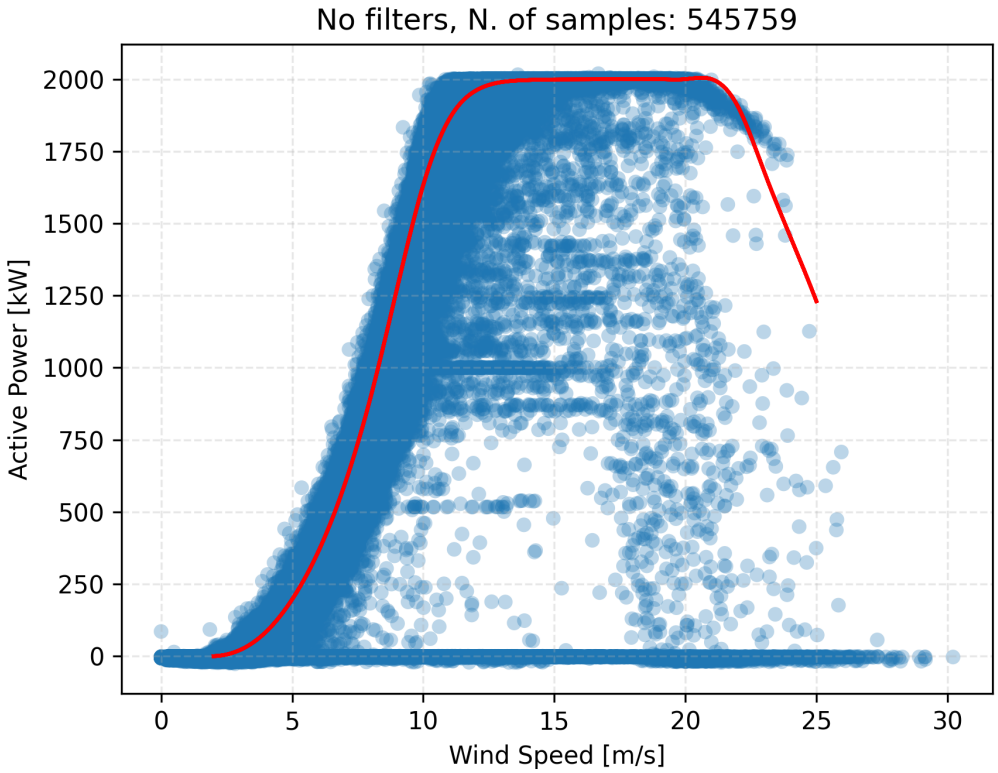

Figure 3.2.1. Raw data.

3.2.1 Generic Filters

The first set of filters used is the broadest one possible and is applied to all studies. These filters simply discard a set of invalid samples because they contain missing values or originate from situations where the turbine was not correctly active, unlike other filters that will be presented later.
It is very easy to remove samples with missing values because regardless of the cause, if the data is missing it is replaced with a null value (nan). As a result, eliminating samples that have at least one null value is sufficient.

There are two factors to take into account when determining when the turbine is not functioning properly: the Transmission System Operator (TSO) limitations and the status of the turbine.

The former relates to moments when the turbine is working correctly but is unable to produce due to some restrictions set by Terna, the organization responsible for managing the electricity networks in Italy. These restrictions are typically put

in place because production must be reduced or stopped due to an energy surplus. Since the production is much lower than it should be but the turbine is formally active, it is better to remove these moments from the data.

Instead, the latter, based on the value it assumes, indicates the state in which the turbine is operating. Based on this value, it is possible to determine whether the turbine is operational or not and possibly discard the values. The values of the state can be divided into three groups:

- **Ready**: turbine correctly functioning and ready, but it is not producing because the conditions are not respected, as in the case of no or too weak wind.
- **Active**: The turbine is working properly and is producing.
- **Error**: All the other status refer to different reasons why the turbine is stopped, they can include malfunctions of various kinds, communication errors, ordinary and extraordinary maintenance operations and more.

Unlike the measurement values, the values of TSO limitations and the status are not collected with ten-minute frequency: the limitations are saved as pairs of instants in time that represent the start and end of the limitation, while the exact moment is saved for each change in the status each time it takes place. Then, a conversion must be performed in order to make these values work with the 10 minute steps. In order to achieve this, each state change should be interpreted as the conclusion of the previous state and the start of a new one, and each pair of state changes should be viewed as the duration of an active state. This concept is illustrated in Figure 3.2.2.

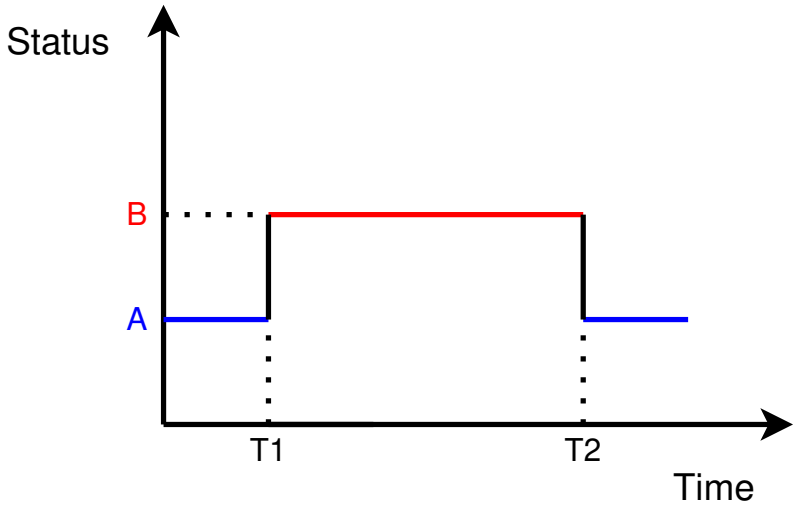

Figure 3.2.2. In this example, a transition from state A to state B can be seen at the instant of time T1 and the reverse transition at the instant of time T2. The state before T1 and after T2 is A, while the state between T1 and T2 is B. So in this case the pair of time instants T1 and T2 indicate the beginning and the end of state B.

Once the pairs of state changes have been obtained, it is necessary to remove those having an error state. These pairs and those related to TSO limitations can be treated in the same way: all 10-minute timesteps contained in the period of any one of those pairs must be removed, even if they are only partially contained.

The results of the two filters are shown in Figure 3.2.3. The filter for removing missing values removes fewer samples than the filter applied on the status of the turbine and TSO limitations (about 14,000 against 40,000). This is because usually, even in the event of malfunctions or stops, the sensors positioned on the turbines continue to supply values even if they relate to the stopped turbine. There can be missing values only for some faults or in the event of communication errors.

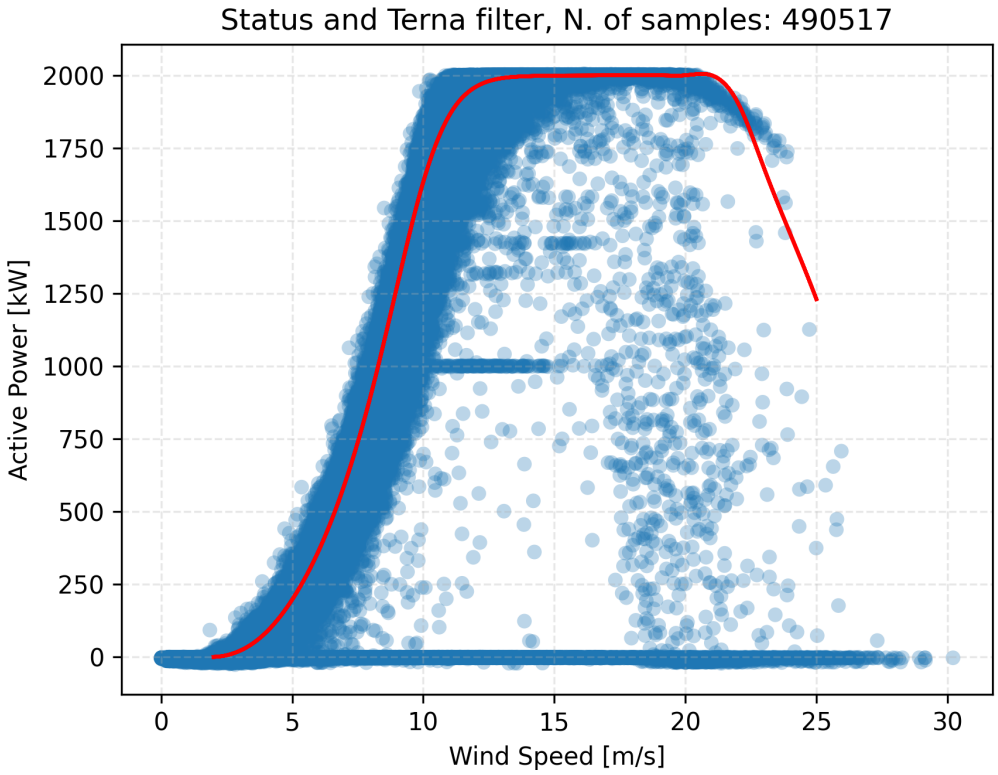

Figure 3.2.3. Filters over status and Terna limitations.

Furthermore, the samples removed based on the missing values do not help to clean up the curve, unlike those on the status and TSO, which instead remove many values among the misleading ones. This is because the missing values are not connected to the turbine production but are random moments, and being that the points close to the theoretical curve have an extremely higher density than the noise points, if random points are taken, they will likely belong to that area.

Status-based filters only remove error status, leaving ready status. If ready status were also removed, it would be possible to clean up the curve better, however losing a lot of data especially for low wind speeds. For this reason, they are not removed in this step but other filters will be used to adequately clean up the wind-power curve.

3.2.2 Power curve filters

After using the first set of generic filters, it is easy to realize that these help remove inappropriate samples but on their own they are not sufficient. For this reason, it is necessary to develop a second set that work more directly on the wind-power curve.

The first two filters of this new set are conceptually very simple, but also very effective. What they do is remove all samples with values below a certain threshold, which in the first filter concerns the active power while in the second the wind speed. In details:

- Samples that have an active power lower than or equal to 0 W are removed as this indicates that the turbine is completely stopped and is not producing.

$$\text{Active Power} > 0 \text{ W}$$

- Samples that have a wind speed lower than or equal to 3 m/s are removed because the turbine model under analysis is activated with a minimum wind speed of 3 m/s, and therefore lower speeds indicate a lack of production.

$$\text{Wind Speed} < 3 \text{ m/s}$$

The reason why the data relating to the periods in which the turbine is not producing is discarded is that the purpose of this work is to study the operation of the turbine during its operation, and therefore these data are considered superfluous and misleading.
The results of this pair of filters is shown in Figure 3.2.4.

They helps to remove the samples relating to useless periods, but do not improve the situation of the wind-power curves: for this purpose, a more complex filter has been developed.

It is based directly on the ideal value of the theoretical curve provided by the manufacturer, and was made to remove any samples that stray too far from it. The definition of how far is too far is defined as follows: two values called ratio and offset need to be chosen, both of which represent the upper limit of acceptable error.
Ratio is a value generally between 0 and 1 representing the fraction of how many ideal values the current value can deviate from the ideal. Mathematically:

$$(1 - \text{ratio}) * \text{ideal value} \leq \text{actual value} \leq (1 + \text{ratio}) * \text{ideal value}$$

Offset instead represents how many Watts from the theoretical circle it is possible to move away from the ideal, imposing a limit not in relation to the ideal value:

$$\text{ideal value} - \text{offset} \leq \text{actual value} \leq \text{ideal value} + \text{offset}$$

These two thresholds must both be respected, otherwise the value is discarded. In this way, it is possible to create a dynamic filter with which the curve can be

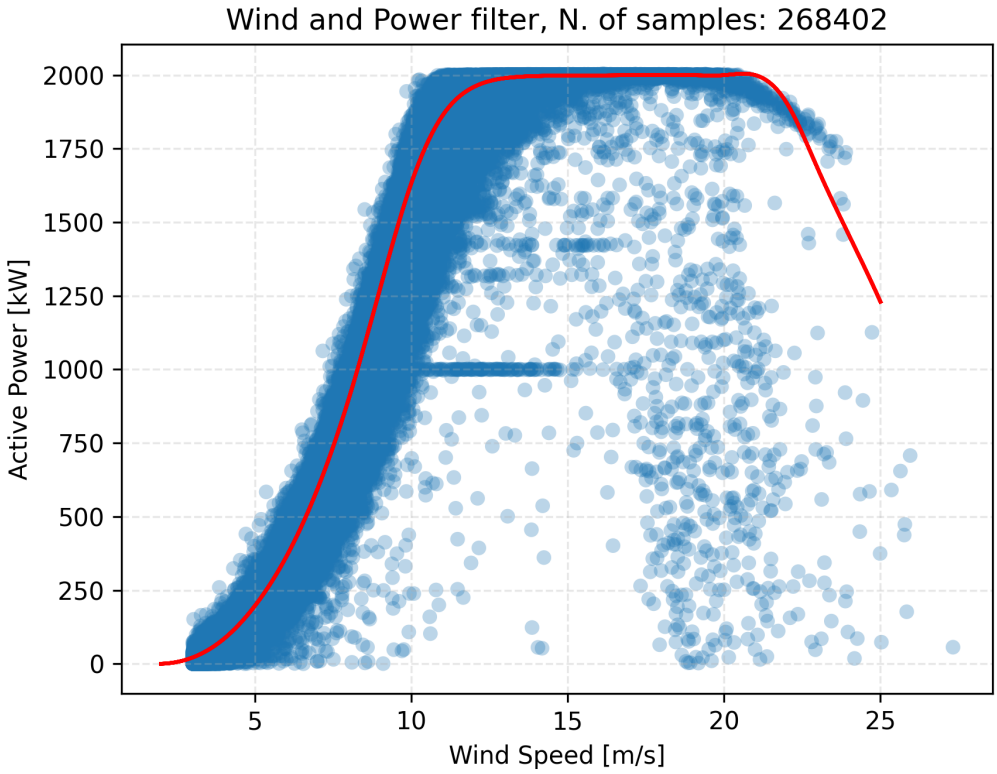

Figure 3.2.4. Basic wind and power filters.

cleaned properly.
Since the power values vary a lot, even using the double filtering parameter it is advisable to use this filter several times with different parameters on different wind speed ranges.
This is because by choosing a single pair of parameters to be used on the whole curve, for low power values the ratio parameter is too limiting as it depends on the ideal value, while for high powers the threshold is exaggerated. Conversely, the offset parameter imposes a threshold that is too high for low power values and too low for high values.

The set of filters used is shown in Table 3.2, and the results are shown in Figure 3.2.5.

This filter correctly removes all misleading samples without affecting the correct samples close to the ideal curve.

3.2.3 Standardization

The last step of the preprocessing is to apply standardization: this step is extremely important because it allows features that generally have very different values to be made comparable. As described in Chapter 2, this is done by making the mean of each feature 0 and the variance 1.

Wind Speed Start	Wind Speed End	Ratio	Offset
0 m/s	8 m/s	0.6	350 kW
8 m/s	13 m/s	0.35	350 kW
13 m/s	15 m/s	0.35	300 kW
15 m/s	22 m/s	0.1	150 kW
22 m/s	25 m/s	0.25	200 kW

Table 3.2. Set of filter used for the curve cleaning.

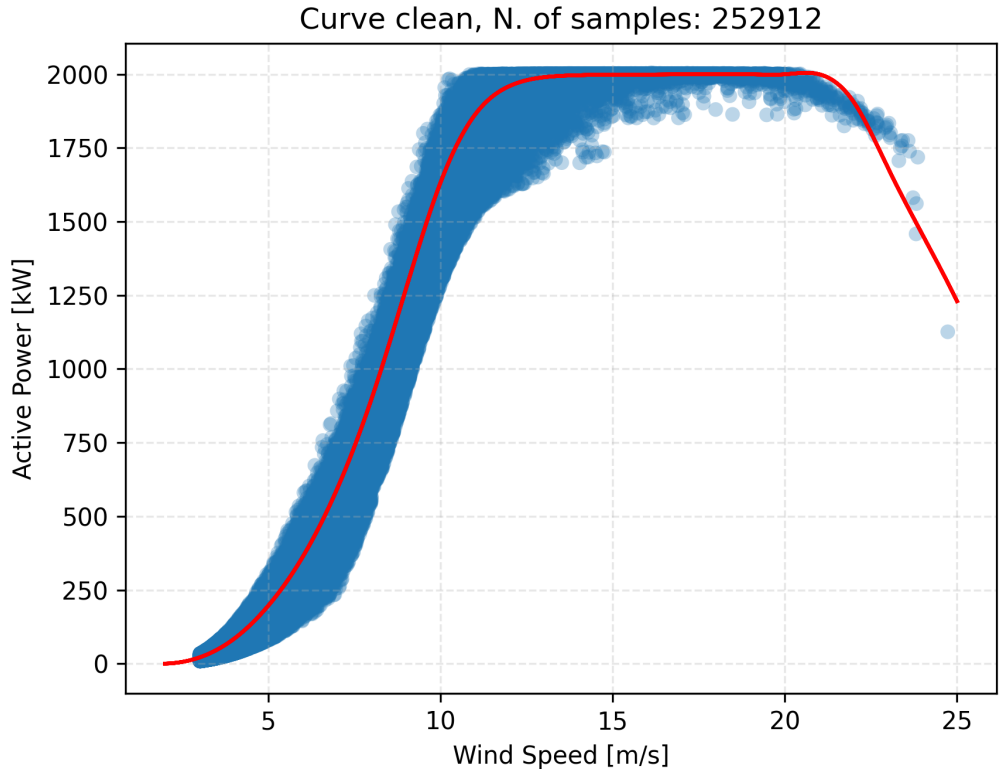

Figure 3.2.5. Final results of filtering after curve clean.

These features allow algorithms to perform better and/or achieve convergence faster.

When applying standardization, it is crucial to take into account the division of the dataset: to apply it, the mean and variance of the dataset must first be calculated, and this must be done strictly on the train dataset. Once this information is obtained, standardization can be applied to the train dataset itself and also to the val and test datasets, using mean and variance obtained on the train dataset. Recalculating mean and variance on the other two parts of the dataset is not correct and would lead to worse results since the digital twin is trained with standardized data using mean and variance from the train dataset, and using different ones would

compromise the proper functioning of the model. It would also be incorrect from a logical point of view, since each dataset has its own functionality and neither the evaluation dataset nor the test dataset should be used for training.

3.3 Temperature Preprocessing

During the temperature study phase, it became necessary to study additional preprocessing methods specifically for temperatures because the previous ones were not sufficient. This section of preprocessing is applied only when referring to models that have to do with temperatures, while for models that do not include temperatures the preprocessing previously presented is sufficient.

3.3.1 Time Series Model

The first thing realized during the study of temperature is that it is not only the data for the present instant that are important, but it is also important to know the past trend of the features. This is because the temperature of any heat-producing object depends not only on the heat produced at the current moment but also on the heat produced and accumulated previously.

Although the data are divided into samples of 10 minutes each, the components involved are very large and the temperature changes slow, so a fairly wide time range should be taken into account. For this study it was decided to use a range of two hours (twelve samples) as it is a good compromise between a not too long sequence and a good amount of information.
This method of using the data as a time series is a concept called rolling window, in which for each sample the previous n samples are also taken, and the prediction is made on the next sample, as is shown in Figure 3.3.1. The length of the sequence for this work is kept fixed at two hours, namely twelve samples: this is because it is a good compromise between a sequence long enough to contain relevant information on the temporal trend and one short enough not to create difficulties in finding all within the dataset a sufficient number of consecutive samples to form the sequences.

The previously applied filters remain almost unchanged, but at the end of the filtering phase it is necessary to check for each sample whether the eleven samples in the previous eleven timesteps are present in order to declare that sample valid. The only other difference is in the wind-power curve cleaning filter, which is slightly changed to make it more flexible: instead of directly removing samples that do not meet the parameters for a given measure, it calculates the moving average over the twelve samples in the time sequence for that measure. Only if the average does not meet the parameters then the sequence is considered not valid.

3.3.2 Temperature filters

With the filters introduced so far it is possible to obtain excellent results on the wind-power curve; it is therefore possible to state that the remaining samples are

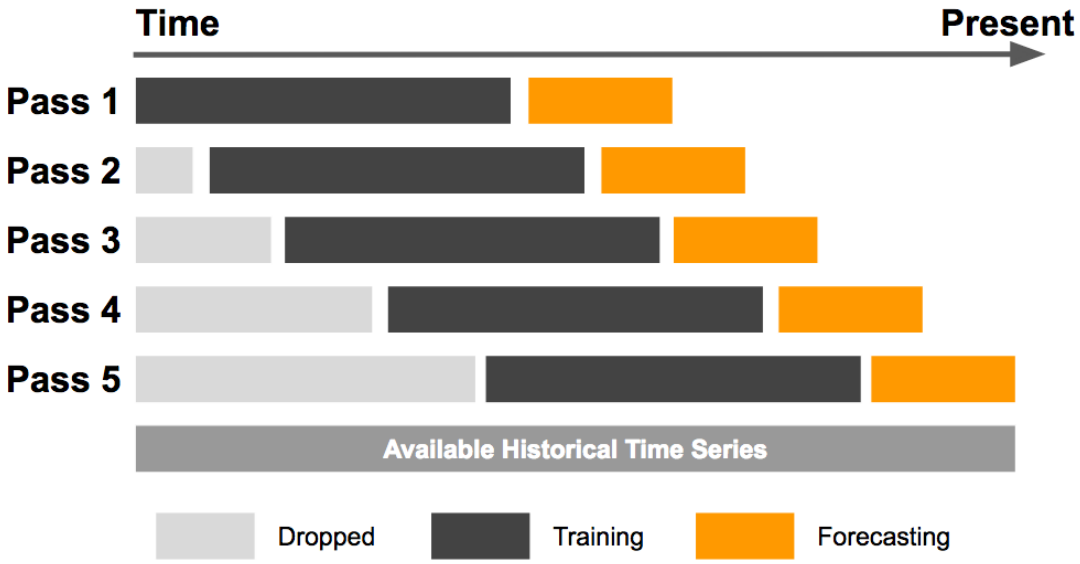

Figure 3.3.1. Image taken from [51].

all correct and respect the ideal conditions regarding active power.

As for the other measurements, the filters applied may not be sufficient to clean the relative wind-measure curves, therefore it is necessary to check each measurement independently.
As regards the rotor RPM, the filters applied for the power are sufficient to obtain excellent results, producing a clean curve without outliers.

The same cannot be said for temperature measurements: power filters correctly remove many inadequate samples, but cannot remove them all. This is due to the fact that, although power and temperature are related, not all moments in which temperature is non-ideal correspond to moments in which also power is non-ideal. This reasoning is particularly valid regarding too high temperatures that often occur during intense production, while too low temperatures are removed quite well by the power filters since they usually occur when the turbine underperforms.
For this reason, it is necessary to develop additional special filters for temperatures, which are applied only during their study. These filters are used in conjunction with those for cleaning the wind-power curve, as well as with the generic filters that are always applied.

By carefully observing the graphs of the wind-temperature curves for different turbines of the same model shown in Figure 3.3.2, it is possible to notice that the shapes that the points create are different from one turbine to another: in some cases the difference is large, while in other cases it is smaller. The reason why the turbines do not all behave in the same way is not well defined, but it is probably caused by a set of factors: first of all, they do not have a theoretical curve to respect as in the case of power, therefore already in the production phase it is not so important to obtain aligned values between different turbines. Furthermore, the turbines have years of life behind them, which involve wear, maintenance, breakdowns and repairs. These events do not depend on the model of the turbine but on the life of the turbine itself, therefore over time they can lead to the creation of

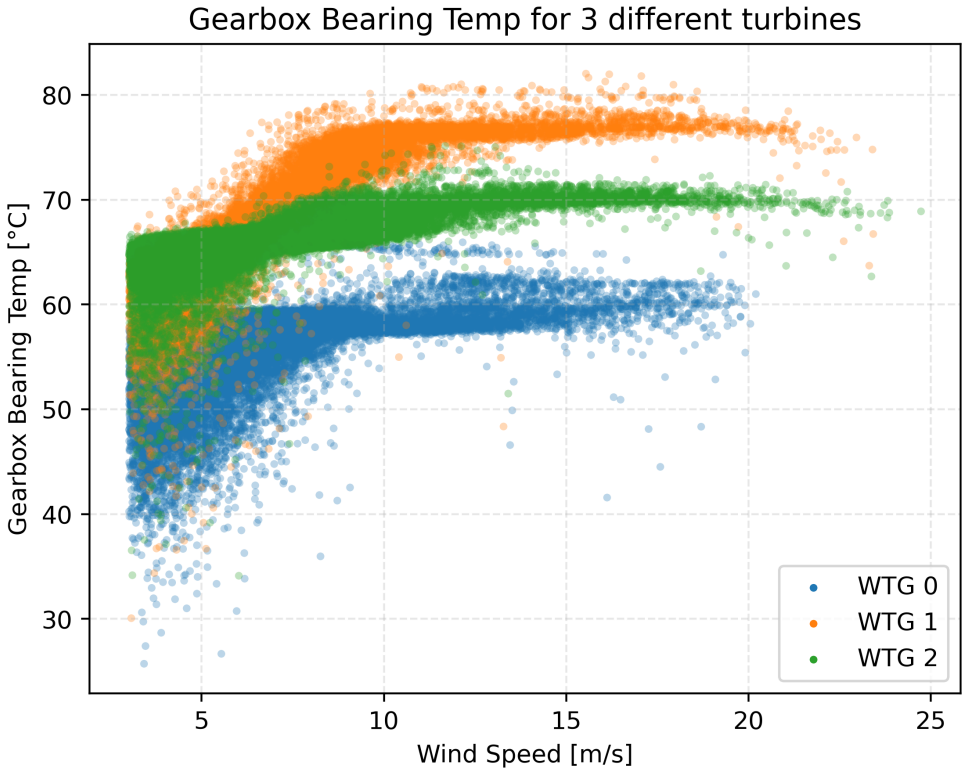

Figure 3.3.2. Gearbox bearing temperature of different WTGs.

differences between turbines of the same model. Finally, even if they are all located in the same geographical area, each is located in a precise point which may have slightly different characteristics from the others (exposure to the sun, wind speed and turbulence) and these differences may contribute over time to having different probabilities of breakdowns, which, as already mentioned, lead the turbines to differ from each other.
This difference between the graphs creates a great difficulty for the data cleaning work, as it is very difficult to heuristically determine methods to remove unwanted samples since what is correct and an ideal behavior for a turbine is completely wrong for another. This implies that it is not possible to apply filters to the entire dataset, but filters must be built to be applied separately to each turbine, so as to be able to analyze the behavior of each turbine and remove those samples which are not adequate for it.

To determine which samples must be removed, different filters are needed from those applied to the power as in this case there is no theoretical curve, therefore it is not possible to know in advance what the ideal values are but they must be deduced based on what is the standard behavior of the turbine and then remove the points that deviate from it. To carry out this task, the DBSCAN algorithm is used to divide the points into clusters (groups of points) and outliers.
In this way, the outliers are separated immediately and it is also possible to identify groups of points which deviate from normal operation but which are close to each other, thus not appearing as outliers at the beginning. To also remove all these

point clusters, only the cluster with the highest number of points is kept, while all points from all other minor clusters are discarded.

It is necessary to choose the hyperparameters such that the dataset is not partitioned into a multitude of small clusters, but that the larger set of points is not broken up. To obtain these results, the values used are $minsamples = 25$ and $eps = 0.6$ for gearbox temperatures and $minsamples = 30$ and $eps = 1$ for generator temperatures; for different datasets it may be necessary to choose different ones, since the DBSCAN is based on density and therefore the number of samples in the dataset influences the choice.

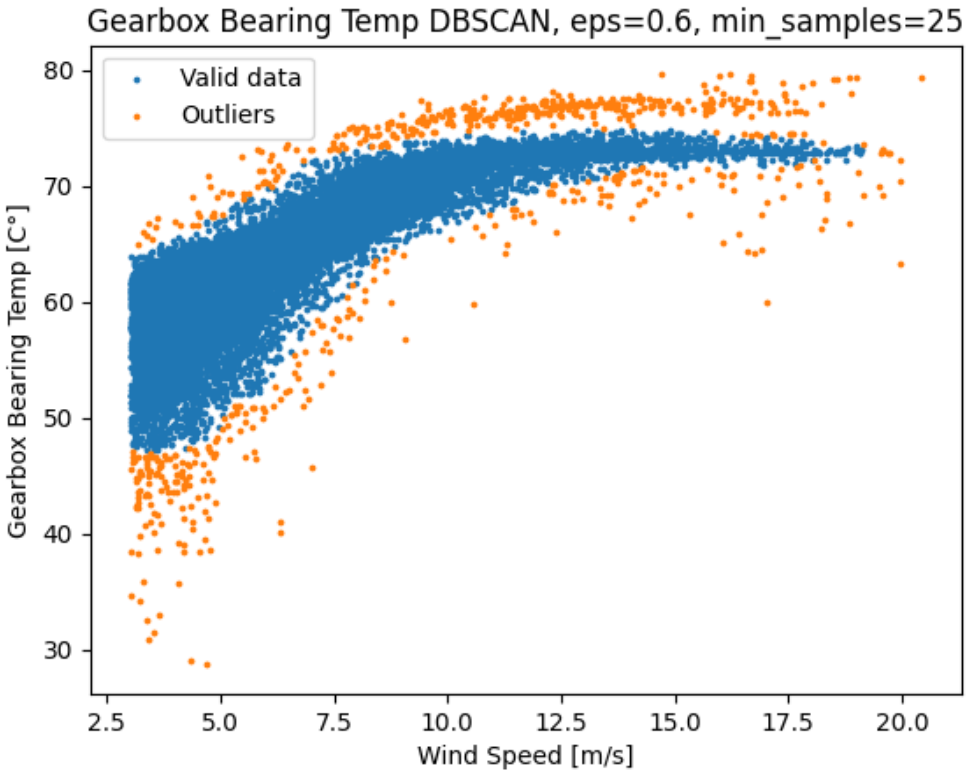

Figure 3.3.3. Effects of DBSCAN on gearbox bearing temp.

Figure 3.3.3 shows the results of the DBSCAN applied on a turbine. For the sake of brevity, the results coming from other turbines are missing. However, the filters were applied on all of them and the obtained results are almost the same.

3.3.3 Features Expansion

The diversity of the wind-temperature curve between different turbines makes it very difficult to predict results on turbines never seen before, even by analyzing multiple turbines belonging to the same model during the training phase. To remedy this difficulty, the dataset is expanded in order to provide the algorithms used with more information. Specifically, two columns (features) are added for each

temperature taken into consideration, one connected to the interpolation of the wind-temperature curve of the single turbine and one connected to the temperature in question but in the previous time step. This Section explains how the two added features are formed and why they were chosen.

Interpolation

The first of the two added features concerns the interpolation of the wind-temperature curve. The feature value for each sample corresponds to that of the interpolation of the wind-temperature curve for the sample wind speed, given the interpolation of the turbine from which the sample was taken.

To obtain this value, it is therefore necessary to calculate the interpolation of each individual turbine, even in the case of a turbine never seen before. This leads to one of the main disadvantages of this strategy, namely the fact that in order to use the digital twin on new turbines, it is required to have enough data available to be able to calculate the interpolations even before starting. If this requirement is satisfied, the problem is solved as there is no need to do new training or anything, but simply calculate the value of the interpolation for each sample and add the new feature to the dataset before having it analyzed by the digital twin.
The motivation behind this choice is simple: since different turbines of the same model have different behaviors, this new feature tries to introduce knowledge about each wind-temperature curve directly into the data, something previously not contained in the inputs (but only derivable from the outputs). In this way, even without having previously seen a turbine, it is possible to have information on what is the standard value for that specific turbine for that wind speed. An example of interpolation is given in Figure 3.3.4

Temperature Shift

The second feature added concerns the temperature in the previous time step. The value assumed by this column for each sample corresponds to the temperature value assumed by the turbine itself in the previous ten minutes, therefore the one of the previous sample.

Unlike the previous feature, this is not connected to the single turbine and therefore does not require having a fair amount of data available before being able to calculate the values of the new column, as the value is measured directly in the turbine. On the other hand, by including among the inputs what was an output up to ten minutes before, it is no longer possible to state that only environmental variables are used as inputs. This feature can be generated through real-time use or by having a dataset available.
In the case of real-time use, it is sufficient to save the recorded temperature value and insert it in the next sample, in the field of the new feature. As for the dataset already created, to add the feature it is needed to duplicate the temperature column and shift it one step forward in time, so that each sample at instant t has the

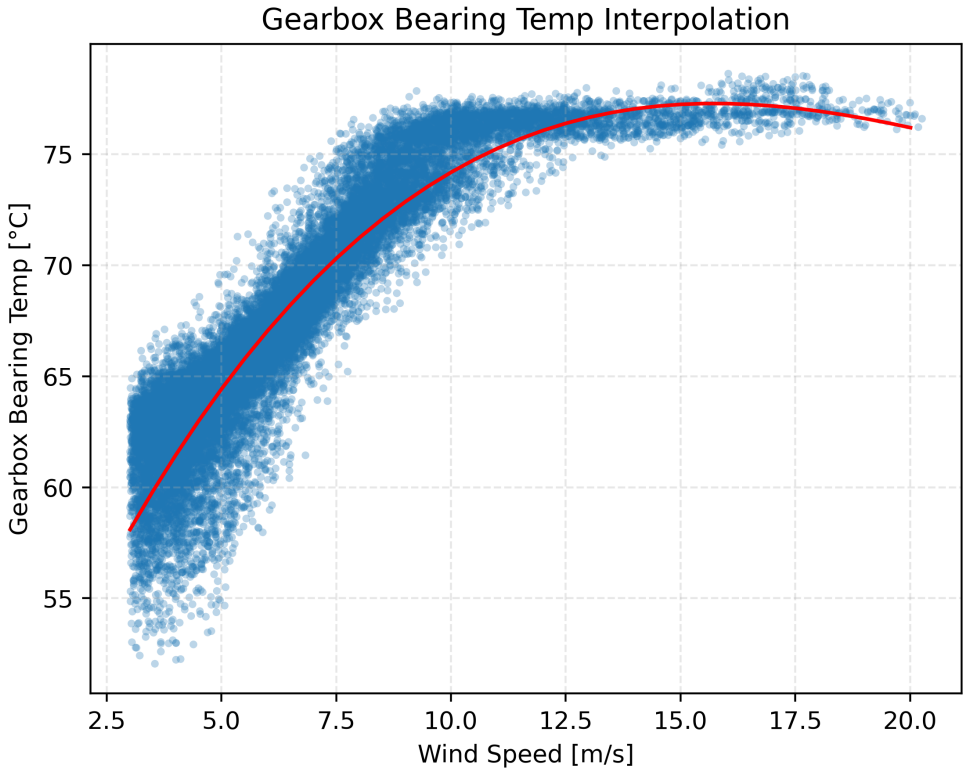

Figure 3.3.4. Interpolation of gearbox bearing temperature.

Timestep	Original Temperature	Shifted Temperature
0	T_0	/
1	T_1	T_0
...	...	...
t	T_t	T_{t-1}
...	...	...
n	T_n	T_{n-1}

Table 3.3. Example of how temperature shift works.

temperature of instant t (to be used as output) plus the temperature at instant t-1 (column added, to be used as input).

As can be seen from Table 3.3, which shows the result of the operation described above, the value of the temperature shifted in the first timestep is null as there is no previous temperature data. This implies that it is necessary to remove the first timestep for each continuous sequence of data, since it contains a null value. This is not a problem because this operation is performed before all other filters, therefore the sequences are few and long since they have not already been broken up by the filters themselves by removing the values.

Chapter 4

Methods and Experiments

This chapter discusses in detail all the methods used to carry out this study and what results are obtained for each method. Therefore, all the models and algorithms used to create the digital twin and the various parameter sets tried are analyzed, and the results obtained for each are reported and discussed.

All work and development is written in Python [52], relying on libraries known and renowned in their field. For data handling and manipulation NumPy [53] and Pandas [54] are used, for machine learning Scikit-Learn [55] is used, and for deep learning PyTorch [56], which also allows the GPU to be leveraged to dramatically speed up execution times.
The development phase of the models takes place in several steps, following the trial and error scheme: starting with a basic set of tests chosen at the beginning of the work as the starting point of testing, subsequent modifications and attempts are chosen based on the results obtained previously. This pattern is repeated multiple times until satisfactory goals are achieved, performing numerous tests designed to modify various aspects of the digital twin, from the data to the algorithms tried.
For the sake of clarity of exposition and brevity, not all the tests performed are analyzed in detail, studying the results. Only those with the best results or some significant ones are reported, while the others are only mentioned or omitted.

The task that the digital twin has to perform is regression, and the metric used for evaluating and comparing all the results obtained from the various models is R^2, one of the most well-known and widely used metrics for regression. It has an upper bound in 1, where R^2 equal to 1 indicates predictions that are equal and identical to the original data, and it has no lower bound, so it can take values up to minus infinity. An important value, however, is 0: an R^2 of 0 indicates that the predictions have the same level of accuracy as would be obtained using a model that always chooses as the value to be predicted only the mean of the measure on which the predictions are being made. Negative values thus indicate worse performance than the model that predicts only the mean of the measure, while values greater than 0 indicate better performance.

The chapter is divided into sections and is developed as follows: the first section details the models and algorithms used, going on to analyze the different configurations tried. In each of the following sections, the algorithms shown in the first

section are tested, applied to a specific topic with different values of their hyperparameters, in order to chose the best ones. They include a recap of inputs, outputs, preprocessing, models, parameters and configurations used followed by the related results obtained and a brief discussion of them.

4.1 Algorithms and Configurations

This section shows in detail the algorithms and models that are used to create the digital twin, discussing the reasons why they were chosen. The configurations that are tested are also shown, including the hyperparameters chosen and for neural networks the number and size of layers.

4.1.1 Support Vector Regression

SVR (Support Vector Regression) is an algorithm derived from SVM (Support Vector Machine), which is adapted to perform regression tasks instead of classification. The reason this algorithm was chosen is that it is considered good because it is derived from the SVM, which is one of the best algorithms for classification. Also, this is a model that does not use deep learning, so a wider variety of models are tested. Models that make use of deep learning are known for its excellent results, but they are also uninterpretable. It is always a good idea to try at least one model that does not require the use of deep learning, so as to try different approaches and possibly have more easily interpretable models.
One of the major disadvantages of the SVR is that it is not very suitable for overly large datasets, as it is not very scalable. For this reason following trials it was decided that only 20000 randomly selected samples from the training set are used for all tests that make use of SVRs.
Details on the structure and functioning of an SVR are given in Chapter 2.

There are two parameters to be optimized for SVR: C and ε.
ε dictates what the maximum absolute error threshold is, going to form a fixed distance space within which samples have a zero penalty in the training phase. Samples with distance greater than ε have a penalty dependent on their distance from the threshold.
C is a regularization term that is used to choose how much importance to give to the samples outside the threshold. Its strength of regularization is inversely proportional to its value. Values of C that are too high can lead to the phenomenon of underfitting in that samples that do not meet the constraints are almost ignored, and for this reason the algorithm does not learn adequately. Conversely, C values that are too low can impose too strong an adjustment that leads to the phenomenon of overfitting, in which too much importance is given to samples that do not meet the constraints and the model learns by relying too much on the training dataset, losing the ability to generalize to data that have never been seen.

When it is necessary to choose hyperparameters, two different strategies can be followed to perform the tests [57]: the first is called the grid search, in which a set of values is chosen for each parameter and all possible combinations of values

are tried, thus forming an n-dimensional grid, with n the number of parameters. This method is exhaustive but requires a great many tries, thus becoming very time-consuming to perform.
The second possible strategy is random search, whose main difference from grid search is that the search is not exhaustive, but stops when the chosen number of attempts is reached, regardless of the number of possible parameter combinations to be tried. For each attempt, parameter values are randomly drawn from a previously provided set of values or distribution.

Using random search allows for a more diverse set of parameters to be tested because it is not necessary to try every single combination, which requires using each parameter value multiple times, but different values for all parameters are tried on each attempt.
This can be particularly advantageous in cases where one of the parameters to be tested is particularly more relevant than the others, so that more values can be tried and a larger space explored, as shown in Figure 4.1.1.
In contrast, grid search requires trying all possible combinations of values, thus requiring many more attempts, or for the same number of attempts testing many fewer different values. If the parameters tested, however, are all equally important and/or influence each other, then it is especially important to try all combinations, even at the cost of trying fewer different values.

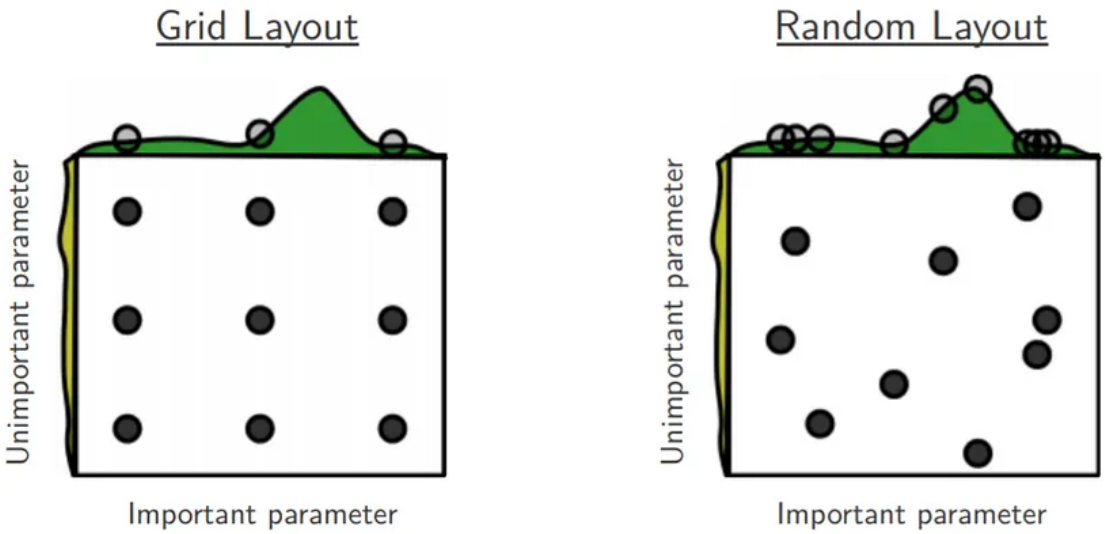

Figure 4.1.1. In the image taken from [57], it can be seen that more different values are tried for each of the two parameters in the random search than in the grid search. In this example, one of the two parameters has much influence on the final results, while the other parameter does not have much influence. The influence of each parameter on the results is shown on the relative axis with a graph, where the higher the value, the greater the influence. By trying several different values, random search succeeds in obtaining better results than grid search for a low number of attempts.

Since the main SVR parameters both play an important role and it is not easy to understand how they influence each other, it is best to use a grid search type approach so as to have an exhaustive analysis.

Table 4.1 provides all the values that are given to the grid search for each hyperparameter.
For the implementation of the SVR is provided by Scikit-Learn [58].

C	ε
0.1	0.001
0.5	0.005
1	0.01
5	0.05
10	0.1
50	0.5
100	1

Table 4.1. Possible values of C and epsilon.

4.1.2 Feedforward Neural Networks

A typical form of artificial neural network utilized in deep learning and machine learning applications is the Feedforward Neural Network (FNN). These networks are made up of linked neurons that process and transfer data across layers. The output of one layer in a FNN, as opposed to a recurrent neural network, is only dependent on the inputs and weights of the preceding layer since feedforward neural networks lack loops.
An input layer, one or more hidden layers, and an output layer make up the fundamental components of a FNN. The input layer accepts data and transmits it to the hidden levels for processing and transformation. The output layer then generates the final outcome, such as a prediction or classification.
More technical and detailed information on the functioning of FNNs is provided in Chapter 2.

FNNs are well known because they can be used to solve a wide range of complex problems. They can be trained to spot intricate patterns in data, which is helpful for jobs that need for sophisticated analysis and prediction abilities. Moreover, FNNs are built using supervised learning techniques and for this reason are often simple to train. They may thus be used to train on enormous volumes of data and increase their accuracy over time, making them a popular choice for many applications.
The reason for choosing to use FNNs is therefore because they are a simple model, quick to train but also very high-performance and customisable. Depending on the type of loss used, it is possible to train them for both classification and regression purposes. Since regression work is required in this study, the loss used is the Mean Squared Error (MSE), which is the most commonly used for this purpose.

In FNNs, there are many parameters to be configured, and the structure of the network itself can be modified by changing the number and size of hidden layers.

It would therefore be possible to carry out many different tests, but as time and resources are limited, it is necessary to restrict the field to a few models to be tested. For this reason, instead of testing all possible combinations of hyperparameters, they are tested one at a time, since having several parameters to tune the total number of combinations is not sustainable. Therefore, starting with those that are the most relevant parameters, they are tested one at a time while keeping all others fixed. Standard values are used for the initial parameter values, and as tests are carried out these standard values are replaced with the best ones obtained during the tuning.
Before testing the parameters, however, it is necessary to choose the model architecture, i.e., the size and number of layers. This is the first test that is performed, and thereafter all the tests regarding the various hyperparameters will be performed.

They are chosen so as to try to create some variety in testing but at the same time so as to understand trends for improvement: grid search is used to do this, unlike SVR where random search is used. Specifically, two different numbers of layers are tested, each with four different dimensions. The two architectures with different numbers of layers are shown in Figures 4.1.2 and 4.1.3 without showing the precise layer size, but showing "hidden size" instead. The latter corresponds to the number of neurons in the layer, and is specified below, along with all the other information of the eight architectures tested. For each architecture, the list of individual layers and their size is given, in the order in which they are arranged in the network.

FNN 1, shown in Figure 4.1.2:

- **Input layer**
- **Hidden layer 1**: 4 neurons
- **Output layer**: number of neurons equal to the number of outputs

FNN 2, shown in Figure 4.1.2:

- **Input layer**
- **Hidden layer 1**: 16 neurons
- **Output layer**: number of neurons equal to the number of outputs

FNN 3, shown in Figure 4.1.2:

- **Input layer**
- **Hidden layer 1**: 64 neurons
- **Output layer**: number of neurons equal to the number of outputs

FNN 4, shown in Figure 4.1.2:

- **Input layer**

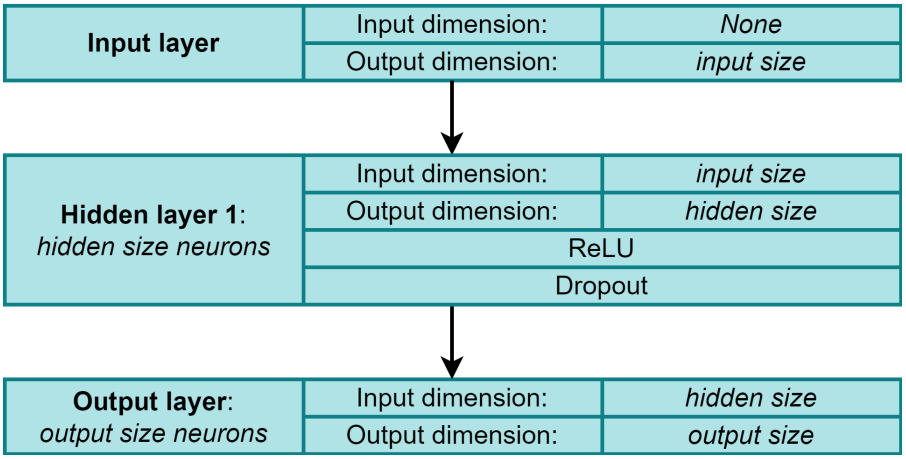

Figure 4.1.2. FNN 1 layer.

- **Hidden layer 1**: 256 neurons
- **Output layer**: number of neurons equal to the number of outputs

FNN 5, shown in Figure 4.1.3:

- **Input layer**
- **Hidden layer 1**: 4 neurons
- **Hidden layer 2**: 4 neurons
- **Hidden layer 3**: 4 neurons
- **Output layer**: number of neurons equal to the number of outputs

FNN 6, shown in Figure 4.1.3:

- **Input layer**
- **Hidden layer 1**: 16 neurons
- **Hidden layer 2**: 16 neurons
- **Hidden layer 3**: 16 neurons
- **Output layer**: number of neurons equal to the number of outputs

FNN 7, shown in Figure 4.1.3:

- **Input layer**
- **Hidden layer 1**: 64 neurons
- **Hidden layer 2**: 64 neurons

- **Hidden layer 3**: 64 neurons
- **Output layer**: number of neurons equal to the number of outputs

FNN 8, shown in Figure 4.1.3:

- **Input layer**
- **Hidden layer 1**: 256 neurons
- **Hidden layer 2**: 256 neurons
- **Hidden layer 3**: 256 neurons
- **Output layer**: number of neurons equal to the number of outputs

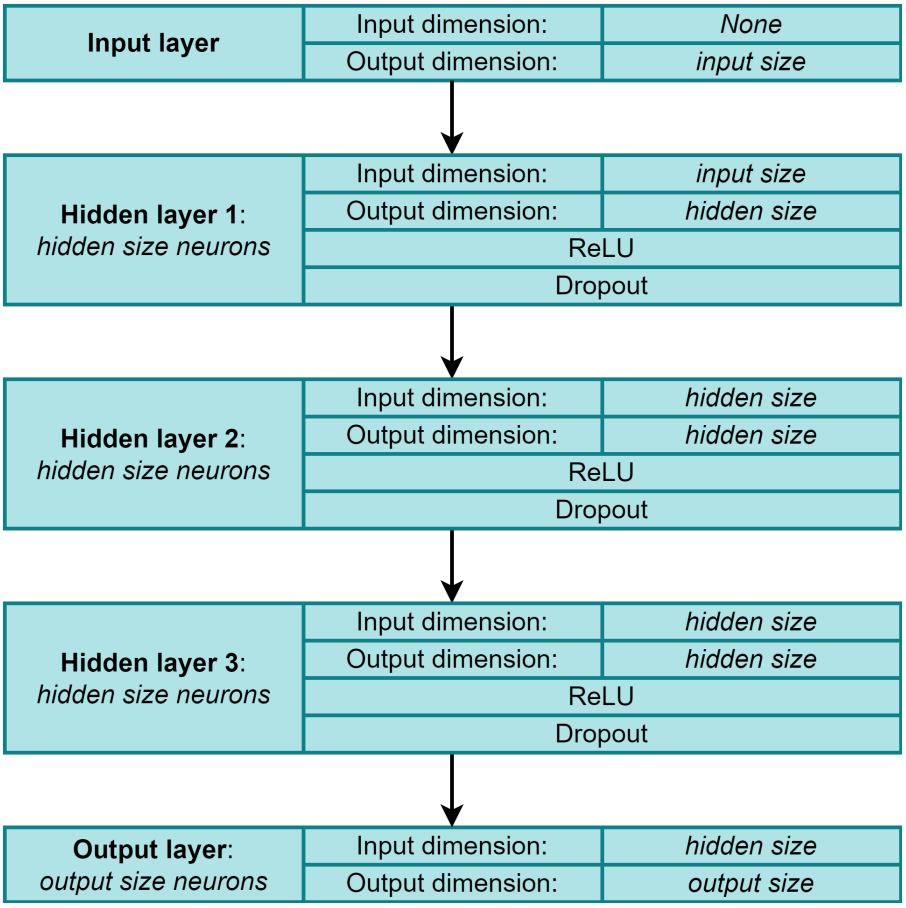

Figure 4.1.3. FNN 3 layers.

Each hidden layer is internally made up of three elements in sequence:

- **Linear layer**: It is the set of neurons of the layer, with their weights and biases
- **ReLU**: It applies the activation function to the outputs of the layer's neurons, so as to introduce non-linearity

- **Dropout Layer**: It is a regularisation function, each output of the layer has a given probability of being taken to 0. The probability is a hyperparameter that in this work is kept fixed

It should also be noted that the input layer is depicted for clarity of presentation, but in practice has no neurons as the input values are fed directly to the first Hidden layer. The output layer is composed only of the linear part that includes neurons, and has neither ReLU nor dropout.

The following are the various hyperparameters used in the order in which they are tested, giving for each the list of values to be tested and which is the default one used before testing that parameter.

- **Learning Rate**:
  - **Default**: 0.001
  - 1
  - 0.1
  - 0.01
  - 0.0001
  - 0.00001
- **Dropout**:
  - **Default**: 0.2
  - 0
  - 0.4
  - 0.6
  - 0.8

Finally, the following are other useful parameters and values, but they are not tested.

- **Loss Type**: MSE
- **Epochs**: 30
- **Batch Size**: 256
- **Evaluation Metric**: R^2
- **Optimizer**: Adam
  - **Weight Decay**: 0.001
- **ReduceLROnPlateau**:
  - **Patience**: 5
  - **Factor**: 0.1

4.1.3 Gated Recurrent Unit

In the area of deep learning, a particular kind of Recurrent Neural Network (RNN) called Gated Recurrent Unit (GRU), has become well-known for its capacity to simulate long-term relationships in sequential data.
In order to analyze sequential data, such as time series or natural language, RNNs were developed. They have the capacity to maintain a hidden state that encodes knowledge about previous inputs, in contrast to FNNs that operate on fixed-size inputs and have no memory of previous inputs, allowing them to model temporal dependencies and learn to make predictions based on context. A new hidden state is created by iteratively applying a set of weights to the current input and the previous hidden state. This new hidden state is then sent through an output layer to provide a prediction. The network can gather data about the whole sequence and use it to create a prediction since this procedure is repeated for each input in the sequence.

GRUs modify the core design of RNNs by adding gating mechanisms that control the information flow across the network, enabling it to selectively remember or forget the past depending on the input at hand. Two gates are used to do this: the reset gate, which chooses how much of the prior data to erase, and the update gate, which chooses how much of the fresh data to retain. GRUs can efficiently capture both short-term and long-term relationships, making them a valuable tool for modeling sequential data. Further technical information on the operation of a GRU can be found in Chapter 2.
The main reason for using GRUs is because of their ability to analyze sequential data. This is important since in this way it is possible to do tests that are very different from those done with the other models previously described that work with nonsequential inputs. Studying the data sequentially can be very important in this study, because the data to be analyzed include temperatures, which therefore have a dependence on the previous state and not just the current inputs.

In order to build a GRU-based model, there are a number of parameters to tune and it is necessary to choose how to compose the architecture. The basic structure of a single GRU is that it receives inputs in the form of a sequence of data, analyzes each data item in the sequence by creating for each one an internal representation called a hidden state, and finally returns as output the set of hidden states. The last hidden state is fed to a set of linear layers that produce the final output of the model. It is also possible to feed directly the whole output into the linear layers, but this introduces a lot of data. Tests were carried out in this regard, the results of which are not reported to avoid showing an exaggerated amount of evidence and making the study confusing. Those tests always led to worse results, which is why in the tests that are carried out in the following sections only the last hidden state in the sequence is always used.

The hidden states produced by the GRU can be fed to another GRU as inputs before they are fed to the linear layers. This process can be repeated multiple times by using each time the output of the last GRU as input for a new one, creating an architecture that is called a stacked GRU. The number of stacked GRUs is an important parameter to choose, as is the size of the hidden state, which can be chosen as desired and represents the number of features used to describe the hidden

state.

The size and number of linear layers can also be chosen as desired. In GRUs, as in the case of FNNs, there are many parameters to be configured besides the structure of the network itself. It would therefore be possible to carry out many different tests, but as time and resources are limited, it is necessary to restrict the field to a few models to be tested. For this reason, instead of testing all possible combinations of hyperparameters, they are tested one at a time, since having several parameters to tune the total number of combinations is not sustainable. Therefore, starting with those that are the most relevant parameters, they are tested one at a time while keeping all others fixed. Standard values are used for the initial parameter values, and as tests are carried out these standard values are replaced with the best ones obtained during the tuning.
Before testing the parameters, however, it is necessary to choose the model architecture, i.e., the size and number of layers. This is the first test that is performed, and thereafter all the tests regarding the various hyperparameters will be performed.

They are chosen so as to try to create some variety in testing but at the same time so as to understand trends for improvement: grid search is used to do this, unlike SVR where random search is used. Specifically, two different numbers of layers are tested, each with five different dimensions. The two architectures with different numbers of layers are shown in Figures 4.1.4 and 4.1.5 without showing the precise layer size, but showing "hidden size" instead. The latter corresponds both to the hidden state dimension and to the number of neurons in the linear layer, and is specified below, along with all the other information of the ten architectures tested. For each architecture, the list of individual layers and their size is given, in the order in which they are arranged in the network.

GRU 1, shown in Figure 4.1.4:

- **Input layer**
- **GRU layer 1**: Hidden state dimension of 1
- **Linear layer 1**: 1 neurons
- **Output layer**: number of neurons equal to the number of outputs

GRU 2, shown in Figure 4.1.4:

- **Input layer**
- **GRU layer 1**: Hidden state dimension of 4
- **Linear layer 1**: 4 neurons
- **Output layer**: number of neurons equal to the number of outputs

GRU 3, shown in Figure 4.1.4:

- **Input layer**

- **GRU layer 1**: Hidden state dimension of 16
- **Linear layer 1**: 16 neurons
- **Output layer**: number of neurons equal to the number of outputs

GRU 4, shown in Figure 4.1.4:

- **Input layer**
- **GRU layer 1**: Hidden state dimension of 64
- **Linear layer 1**: 64 neurons
- **Output layer**: number of neurons equal to the number of outputs

GRU 5, shown in Figure 4.1.4:

- **Input layer**
- **GRU layer 1**: Hidden state dimension of 256
- **Linear layer 1**: 256 neurons
- **Output layer**: number of neurons equal to the number of outputs

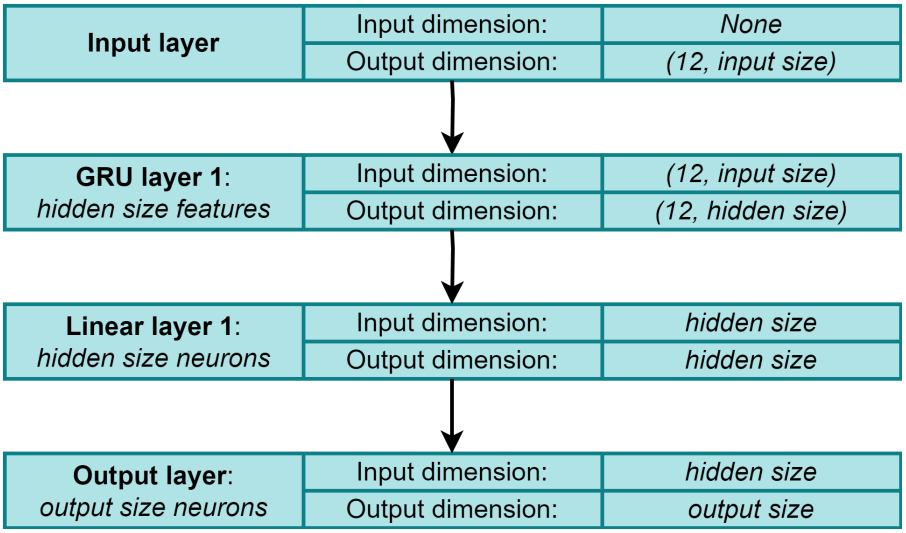

Figure 4.1.4. GRU with 1 GRU layer and 1 linear layer. The difference in data size between the output of the GRU layer and the input of the linear layer is due to the fact that only the last of the hidden states is used, as explained above.

GRU 6, shown in Figure 4.1.5:

- **Input layer**
- **GRU layer 1**: Hidden state dimension of 1

- **GRU layer 2**: Hidden state dimension of 1
- **GRU layer 3**: Hidden state dimension of 1
- **Linear layer 1**: 1 neurons
- **Linear layer 2**: 1 neurons
- **Linear layer 3**: 1 neurons
- **Output layer**: number of neurons equal to the number of outputs

GRU 7, shown in Figure 4.1.5:

- **Input layer**
- **GRU layer 1**: Hidden state dimension of 4
- **GRU layer 2**: Hidden state dimension of 4
- **GRU layer 3**: Hidden state dimension of 4
- **Linear layer 1**: 4 neurons
- **Linear layer 2**: 4 neurons
- **Linear layer 3**: 4 neurons
- **Output layer**: number of neurons equal to the number of outputs

GRU 8, shown in Figure 4.1.5:

- **Input layer**
- **GRU layer 1**: Hidden state dimension of 16
- **GRU layer 2**: Hidden state dimension of 16
- **GRU layer 3**: Hidden state dimension of 16
- **Linear layer 1**: 16 neurons
- **Linear layer 2**: 16 neurons
- **Linear layer 3**: 16 neurons
- **Output layer**: number of neurons equal to the number of outputs

GRU 9, shown in Figure 4.1.5:

- **Input layer**
- **GRU layer 1**: Hidden state dimension of 64
- **GRU layer 2**: Hidden state dimension of 64

- **GRU layer 3**: Hidden state dimension of 64
- **Linear layer 1**: 64 neurons
- **Linear layer 2**: 64 neurons
- **Linear layer 3**: 64 neurons
- **Output layer**: number of neurons equal to the number of outputs

GRU 10, shown in Figure 4.1.5:

- **Input layer**
- **GRU layer 1**: Hidden state dimension of 256
- **GRU layer 2**: Hidden state dimension of 256
- **GRU layer 3**: Hidden state dimension of 256
- **Linear layer 1**: 256 neurons
- **Linear layer 2**: 256 neurons
- **Linear layer 3**: 256 neurons
- **Output layer**: number of neurons equal to the number of outputs

The input layer is not a true layer, but represents only the inputs that are introduced into the network.
Regarding dropout in GRUs, this is applied on the output of the various GRU layers except the last one: this means that dropout can only be used when there are multiple stacked GRUs.
The following are the various hyperparameters used in the order in which they are tested, giving for each the list of values to be tested and which is the default one used before testing that parameter.

- **Learning Rate**:
  - **Default**: 0.001
  - 1
  - 0.1
  - 0.01
  - 0.0001
  - 0.00001
- **Dropout**:
  - **Default**: 0.2 if there is more than one GRU layer, 0 otherwise
  - 0

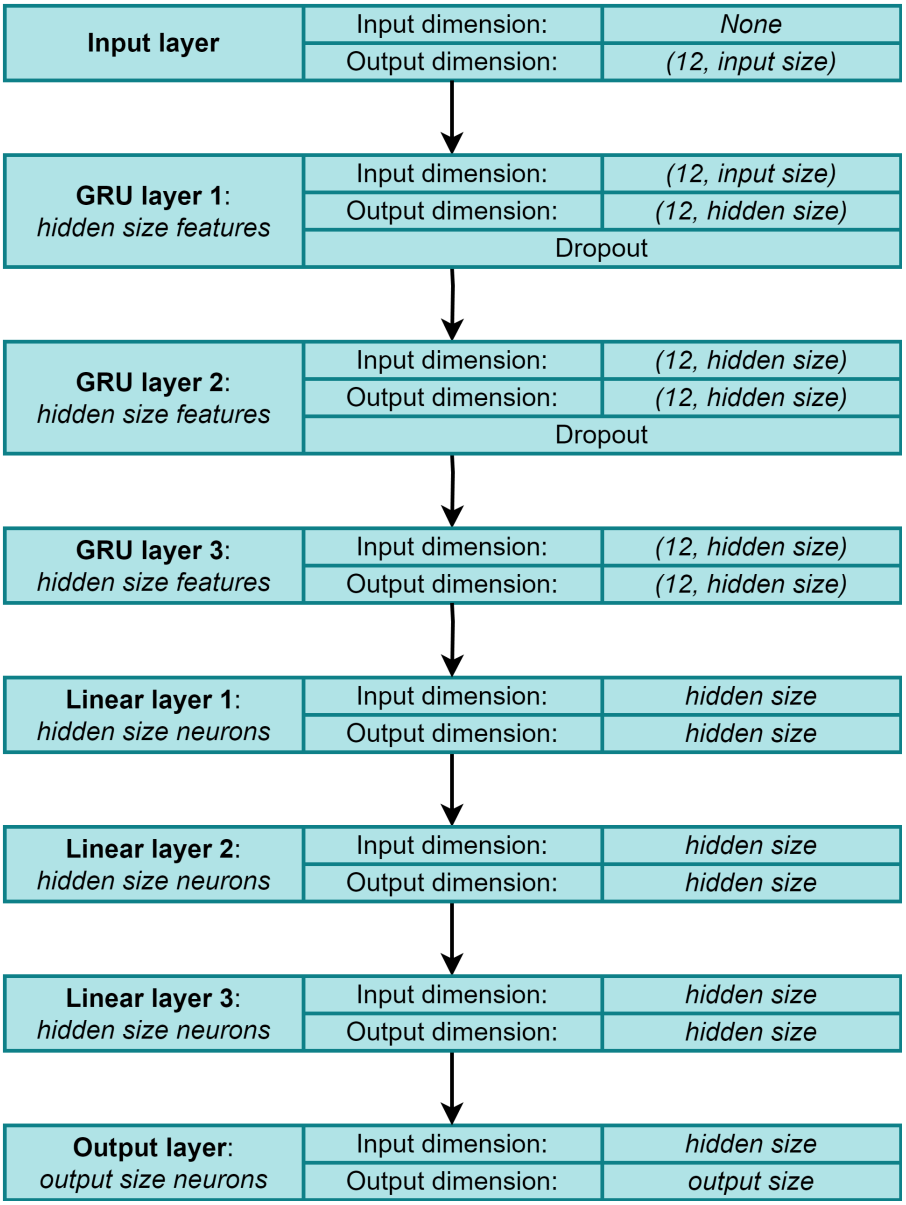

Figure 4.1.5. GRU with 3 GRU layers and 3 linear layers. The difference in data size between the output of the GRU layer and the input of the linear layer is due to the fact that only the last of the hidden states is used, as explained above.

- 0.4
- 0.6
- 0.8

Finally, the following are other useful parameters and values, but they are not tested.

- **Loss Type**: MSE
- **Epochs**: 30

- **Batch Size**: 256
- **Evaluation Metric**: R^2
- **Optimizer**: Adam
  - **Weight Decay**: 0.001
- **ReduceLROnPlateau**:
  - **Patience**: 5
  - **Factor**: 0.1

4.2 Active Power and Rotor RPM Model

In this section, a model is tested with the purpose of simultaneously predicting active power and rotor RPM, since these are two closely related measures with the same trend and the same dependencies. Therefore, the model receives environmental data as input and produces two outputs.
Once all the tests for the choice of hyperparameters have been performed, a final training will be carried out for each output. In this way the tuning is made jointly but the final model can focus on a single output at a time.
It is worth mentioning that because of their structure SVRs cannot make predictions on multiple outputs at once, so each output will have a training of its own from the beginning.

The set of inputs includes the environmental variables, namely:

- **Wind Speed**
- **Wind Speed standard deviation**
- **Ambient Temperature**
- **Blade Pitch Angle**

While that of outputs includes:

- **Active Power**
- **Rotor RPM**

4.2.1 Support Vector Regression

Tests of the hyperparameters of SVRs occur all at once, trying the different possible combinations. Table 4.2 shows the active power results, and Table 4.3 those of rotor RPM.

The values obtained as a result of the tests report slightly different results between active power and rotor RPM: active power is easier to predict and for this

C \ ε	0.001	0.005	0.01	0.05	0.1	0.5	1
0.1	0.99	0.99	0.99	0.99	0.989	0.928	0.426
0.5	**0.991**	**0.991**	**0.991**	**0.991**	0.99	0.952	0.432
1	**0.991**	**0.991**	**0.991**	**0.991**	**0.991**	0.952	0.444
5	**0.991**	**0.991**	**0.991**	**0.991**	**0.991**	0.959	0.444
10	**0.991**	**0.991**	**0.991**	**0.991**	**0.991**	0.959	0.444
50	**0.991**	**0.991**	**0.991**	**0.991**	0.99	0.959	0.444
100	**0.991**	**0.991**	**0.991**	0.99	0.99	0.959	0.444

Table 4.2. Hyperparameters tuning for active power in terms of R^2.

C \ ε	0.001	0.005	0.01	0.05	0.1	0.5	1
0.1	0.972	0.972	0.972	0.971	0.97	0.93	0.596
0.5	0.979	0.979	0.979	0.976	0.972	0.95	0.647
1	**0.98**	**0.98**	0.979	0.976	0.971	0.954	0.672
5	**0.98**	**0.98**	0.979	0.975	0.969	0.959	0.713
10	0.979	0.979	0.979	0.974	0.968	0.959	0.713
50	0.977	0.977	0.977	0.97	0.967	0.96	0.713
100	0.976	0.975	0.975	0.968	0.966	0.959	0.713

Table 4.3. Hyperparameters tuning for rotor RPM in terms of R^2.

reason the results obtained are all very similar and no real trend of improvement or deterioration can be seen except for too high values of ε leading to significantly worse results. In contrast for rotor RPM the results obtained are equally good but it is possible to see a small trend of improvement as C increases up to a peak at $C = 1$ and then worsening.
In both cases, at the same C the first ε values obtain virtually identical results, and then worsen at it exceeds the threshold of 0.01. The reason the results with low ε values are all similar is that this parameter imposes an error threshold below which predictions are considered correct. Predictions considered to be outside the threshold have a cost penalty in the loss function used during training. When this threshold is too low, no or very few predictions meet it. By using different values of very low ε, the tests get similar results because most of the samples do not meet the threshold and therefore all of them are considered in the same way among different instances of SVR, which is as incorrects. A cost is then applied to all of them, which will be very similar as it depends on the magnitude of the prediction error. The results with $\varepsilon = 1$ are in all cases very bad compared to the others.

The parameters selected as best, as there are many equal results, are chosen by considering not only the pair of values in question, but also the results that each

value obtained in its entire row or column, giving priority to low C values as they decrease training time. According to these criteria, the best values for active power are $C = 5$ $\varepsilon = 0.01$, and for rotor RPM are $C = 1$ and $\varepsilon = 0.001$

4.2.2 Feedforward Neural Networks

Architecture

The first test is done on the architecture, the results in terms of R^2 can be found in Table 4.4.

Architecture	Active Power	Rotor RPM
FNN 1	0.951	0.915
FNN 2	0.965	0.949
FNN 3	0.982	0.963
FNN 4	0.989	0.972
FNN 5	0.899	0.904
FNN 6	0.979	0.972
FNN 7	**0.99**	**0.98**
FNN 8	0.989	0.979

Table 4.4. Architecture tests for FNN active power and rotor RPM model in terms of R^2.

The tested architectures achieved similar results, showing generally improved results as the size of the layers increased. The number of layers also leads to an improvement in results, but less obvious. In both cases, a stalemate point is reached where the improvement stops. Overall, the network that performs best is FNN 7, which is the one that is chosen to continue testing.

Learning Rate

Table 4.5 shows the results of training with different learning rates.

The results obtained from the learning rate tests show how important it is to find the right value, which in this case is 0.001. Both by increasing and decreasing the learning rate starting from this heat, progressively worse results are obtained.

Dropout

Table 4.6 shows the results of training with different dropouts.

Dropout tests show that values that are too high spoil the results, as using very high probabilities leads to having most of the neurons shut down during training.

Learning Rate	Active Power	Rotor RPM
1	-0.0	-0.0
0.1	0.981	0.969
0.01	0.984	0.976
0.001	**0.99**	**0.98**
0.0001	0.989	0.979
0.00001	0.981	0.958

Table 4.5. Learning Rate tests for FNN active power and rotor RPM model in terms of R^2.

Dropout	Active Power	Rotor RPM
0	**0.993**	0.978
0.2	0.99	**0.98**
0.4	0.988	0.979
0.6	0.973	0.965
0.8	0.932	0.931

Table 4.6. Dropout tests for FNN active power and rotor RPM model in terms of R^2.

This means that it is difficult for the network to learn anything, since neurons are often turned off. Lower values, on the other hand, help the network to generalize better, adding some difficulty in the training phase. The results with Dropout at 0 and at 0.2 are virtually identical, in the former case we have better results on active power and worse results on rotor RPM, in the latter case we have the opposite situation. Since it is always appropriate to have a minimum of regularization, the model with Dropout at 0.2 is retained.

Single output

Now that the best values for the hyperparameters have been found, a final training is carried out for each output, to see if training on only one measure at a time yields better results. The results are in Table 4.7.

Test results with single outputs yielded identical results for active power but higher results for rotor RPM.

4.2.3 Gated Recurrent Unit

Architecture

The first test is done on the architecture and the results in terms of R^2 can be found in Table 4.8.

Active Power	Rotor RPM
0.993	0.988

Table 4.7. Final test over single outputs in terms of R^2.

Architecture	Active Power	Rotor RPM
GRU 1	0.962	0.937
GRU 2	0.99	0.983
GRU 3	**0.991**	0.984
GRU 4	**0.991**	**0.985**
GRU 5	**0.991**	0.984
GRU 6	0.924	0.884
GRU 7	0.987	0.979
GRU 8	0.99	0.983
GRU 9	**0.991**	0.984
GRU 10	**0.991**	0.983

Table 4.8. Architecture tests for GRU active power and rotor RPM model in terms of R^2.

Unlike FNNs which show a clear trend of improvement, GRUs are much more stable, and apart from GRU 1 and GRU 6 which have hidden size to 1, the others all perform very well. When it is needed to choose between multiple networks with similar results, choosing a smaller network is advantageous if the performance is equal compared to a larger one, as it is faster and lighter by having fewer parameters, and for the same reason it is also easier to train. GRU 4 performs slightly better than the other and its dimension is quite small, so it is the one used in the following tests.

Learning Rate

Table 4.9 shows the results of training with different learning rates.

The results obtained from the learning rate tests show how important it is to find the right value, which in this case is 0.001. Both by increasing and decreasing the learning rate starting from this heat, progressively worse results are obtained. In this case the task is quite easy so the results are still good.

Dropout

Since the chosen architecture is GRU 4 and it is composed of a single GRU layer, it is not possible by the very definition of GRU to have Dropout greater than 0.

Learning Rate	Active Power	Rotor RPM
1	0.908	0.707
0.1	0.99	0.982
0.01	0.991	0.984
0.001	**0.991**	**0.985**
0.0001	0.99	0.981
0.00001	0.975	0.941

Table 4.9. Learning Rate tests for GRU active power and rotor RPM model in terms of R^2.

Single output

Now that the best values for the hyperparameters have been found, a final training is carried out for each output, to see if training on only one measure at a time yields better results. The results are in Table 4.10.

Active Power	Rotor RPM
0.991	0.984

Table 4.10. Final test over single outputs in terms of R^2.

Test results with single outputs yielded identical results for active power and very slightly lower for rotor RPM.

4.3 Temperatures Model

After testing active power and rotor RPM, temperatures are tested simultaneously, following the same principle as before: they have similar trends and dependencies, so it makes sense to train them together.
Once all the tests for the choice of hyperparameters have been performed, a final training will be carried out for each output. In this way the tuning is made jointly but the final model can focus on a single output at a time.
Again, because of the structure of SVR, it is not possible to produce the outputs simultaneously, so separate models will be trained for each output from the beginning.

The set of inputs includes the environmental variables, namely:

- **Wind Speed**
- **Wind Speed standard deviation**
- **Ambient Temperature**

- **Blade Pitch Angle**

While that of outputs includes:

- **Gearbox Bearing Temperature**
- **Gearbox Oil Temperature**
- **Generator Bearing Temperature**

It is important to note that for completeness all three temperatures in the dataset have been included, but little importance will be given to the Gearbox Oil Temperature as this is the temperature of an oil that is specially cooled to keep it within certain temperature ranges, which is why it is very difficult to predict as well as not very useful.

4.3.1 Support Vector Regression

Tests of the hyperparameters of SVRs occur all at once, trying the different possible combinations. Table 4.11 shows the gearbox bearing temperature results, Table 4.12 those of gearbox oil temperature and Table 4.13 those of generator bearing temperature.

C \ ε	0.001	0.005	0.01	0.05	0.1	0.5	1
0.1	0.568	0.567	0.568	0.569	0.573	**0.626**	0.536
0.5	0.57	0.57	0.571	0.572	0.576	**0.626**	0.54
1	0.57	0.57	0.57	0.572	0.574	0.621	0.539
5	0.56	0.56	0.56	0.561	0.564	0.605	0.531
10	0.555	0.555	0.555	0.554	0.561	0.599	0.529
50	0.543	0.543	0.543	0.546	0.551	0.583	0.51
100	0.533	0.533	0.533	0.539	0.541	0.569	0.5

Table 4.11. Hyperparameters tuning for gearbox bearing temperature in terms of R^2.

What was said during the analysis of active power and rotor RPM also applies to temperatures: for ε values that are too low, the results are very similar for the same reason. Unlike the previous case, however, temperatures get slightly better results for higher ε values, and even with $\varepsilon = 1$ no drastically worse values are obtained as is the case in the other tests.
The trend that is observed for C values is similar to the previous one, in which the best results are obtained for low C values, although in this case there is not a trend of initial improvement in all tests. However, the ε trend is different in that instead of having a worsening of the results as the parameter increases, there is an

C \ ε	0.001	0.005	0.01	0.05	0.1	0.5	1
0.1	0.273	0.272	0.272	0.273	0.278	0.319	0.301
0.5	0.279	0.278	0.278	0.28	0.286	**0.324**	0.302
1	0.28	0.28	0.28	0.282	0.287	**0.324**	0.301
5	0.28	0.28	0.28	0.281	0.287	**0.324**	0.296
10	0.28	0.28	0.28	0.281	0.286	0.322	0.297
50	0.28	0.28	0.279	0.281	0.285	0.318	0.285
100	0.277	0.277	0.278	0.278	0.282	0.311	0.282

Table 4.12. Hyperparameters tuning for gearbox bearing temperature in terms of R^2.

C \ ε	0.001	0.005	0.01	0.05	0.1	0.5	1
0.1	0.62	0.62	0.62	0.621	0.621	**0.628**	0.622
0.5	0.611	0.611	0.611	0.611	0.612	0.625	0.625
1	0.607	0.607	0.607	0.607	0.607	0.622	0.624
5	0.595	0.595	0.595	0.594	0.597	0.614	0.617
10	0.59	0.59	0.59	0.59	0.593	0.612	0.612
50	0.585	0.585	0.584	0.584	0.584	0.604	0.606
100	0.579	0.58	0.58	0.579	0.579	0.601	0.603

Table 4.13. Hyperparameters tuning for gearbox bearing temperature in terms of R^2.

improvement as it increases until almost $\varepsilon = 0.5$, beyond which the results begin to worsen.

The best results are all obtained for values of $\varepsilon = 0.5$, while for the choice of C since more equal results are present other results obtained in the same row are also considered by prioritizing low C values since they have shorter training times. The values of C selected are 0.5 for gearbox bearing temperature, 1 for gearbox oil temperature and finally 0.1 for generator bearing temperature.

4.3.2 Feedforward Neural Network

Architecture

For FNNs, the first test is done on the architecture, the results in terms of R^2 can be found in Table 4.14. The tested architectures achieved similar results, showing generally improved results as the size of the layers increased. The number of layers also leads to an improvement in results, but less obvious. In both cases, a stalemate point is reached where the improvement stops. Overall, the network that performs best is FNN 8, which is the one that is chosen to continue testing.

Architecture	Gearbox Bearing Temp	Gearbox Oil Temp	Generator Bearing Temp
FNN 1	0.577	0.331	0.584
FNN 2	0.614	0.344	0.617
FNN 3	0.631	0.354	**0.628**
FNN 4	0.632	0.364	**0.628**
FNN 5	0.465	0.278	0.456
FNN 6	0.616	0.345	0.613
FNN 7	**0.639**	0.362	0.623
FNN 8	0.638	**0.367**	**0.628**

Table 4.14. Architecture tests for FNN temperature model in terms of R^2.

Learning Rate

Table 4.15 shows the results of training with different learning rates.

Learning Rate	Gearbox Bearing Temp	Gearbox Oil Temp	Generator Bearing Temp
1	-0.0	-0.0	-0.0
0.1	0.601	0.338	0.601
0.01	0.636	0.366	0.628
0.001	**0.638**	**0.367**	0.628
0.0001	0.63	0.366	**0.631**
0.00001	0.628	0.355	**0.631**

Table 4.15. Learning Rate tests for FNN temperature model in terms of R^2.

The results obtained from the learning rate tests show how important it is to find the right value, which in this case is 0.001. Both by increasing and decreasing the learning rate starting from this heat, progressively worse results are obtained.

Dropout

Table 4.16 shows the results of training with different dropouts.

Dropout tests show that values that are too high spoil the results, as using very high probabilities leads to having most of the neurons shut down during training. This means that it is difficult for the network to learn anything, since neurons are often turned off. Lower values, on the other hand, help the network to generalize

Dropout	Gearbox Bearing Temp	Gearbox Oil Temp	Generator Bearing Temp
0	**0.638**	**0.369**	**0.629**
0.2	**0.638**	0.367	0.628
0.4	0.636	0.359	0.629
0.6	0.623	0.36	0.632
0.8	0.602	0.352	0.61

Table 4.16. Dropout tests for FNN temperature model in terms of R^2.

better, adding some difficulty in the training phase. In this case, however, the model with dropout at 0 performs better on all three temperatures, albeit by a very small amount. For this reason, it is the one that is selected as best.

Single output

Now that the best values for the hyperparameters have been found, a final training is carried out for each output, to see if training on only one measure at a time yields better results. The results are in Table 4.17.

Gearbox Bearing Temp	Gearbox Oil Temp	Generator Bearing Temp
0.662	0.371	0.665

Table 4.17. Final test over single outputs in terms of R^2.

Test results with individual outputs led to better results in all three cases. This indicates that it is more advantageous for FNNs to have a small number of outputs.

4.3.3 Gated Recurrent Unit

Architecture

For GRUs, the first test is done on the architecture, the results in terms of R^2 can be found in Table 4.18.

In this case GRUs show a clear trend of improved results as the size of the layers increases, while increasing the number of GRU and linear layers brings very marginal improvements. The model that performs best is GRU 9, which is used in subsequent tests.

Architecture	Gearbox Bearing Temp	Gearbox Oil Temp	Generator Bearing Temp
GRU 1	0.544	0.341	0.683
GRU 2	0.685	0.405	0.777
GRU 3	0.713	0.439	0.778
GRU 4	0.719	0.438	0.779
GRU 5	0.727	0.436	0.777
GRU 6	0.511	0.329	0.682
GRU 7	0.691	0.39	0.775
GRU 8	0.714	0.441	**0.785**
GRU 9	**0.73**	**0.454**	0.779
GRU 10	0.728	0.451	0.777

Table 4.18. Architecture tests for GRU temperature model in terms of R^2.

Learning Rate

Table 4.19 shows the results of training with different learning rates.

Learning Rate	Gearbox Bearing Temp	Gearbox Oil Temp	Generator Bearing Temp
1	-29676.576	-50608.435	-14962.218
0.1	0.702	0.38	0.761
0.01	**0.733**	0.444	0.766
0.001	0.731	**0.456**	**0.78**
0.0001	0.698	0.442	0.775
0.00001	0.663	0.379	0.769

Table 4.19. Learning Rate tests for GRU temperature model in terms of R^2.

The results obtained from the learning rate tests show how important it is to find the right value, which in this case is 0.001. Both by increasing and decreasing the learning rate starting from this heat, progressively worse results are obtained.

Dropout

Table 4.20 shows the results of training with different dropouts.
In this case, the best results are obtained with a dropout of 0.4.

Dropout	Gearbox Bearing Temp	Gearbox Oil Temp	Generator Bearing Temp
0	0.729	0.451	0.777
0.2	0.711	0.45	0.778
0.4	**0.73**	**0.456**	0.774
0.6	0.721	0.45	**0.779**
0.8	0.705	0.434	0.78

Table 4.20. Dropout tests for GRU temperature model in terms of R^2.

Single output

Now that the best values for the hyperparameters have been found, a final training is carried out for each output, to see if training on only one measure at a time yields better results. The results are in Table 4.21.

Gearbox Bearing Temp	Gearbox Oil Temp	Generator Bearing Temp
0.708	0.435	0.783

Table 4.21. Final test over single outputs in terms of R^2.

Test results with single outputs resulted in worse results in two out of three cases and slightly better results in the third case. This indicates that GRUs unlike FNNs are not disadvantaged in having multiple outputs to compute simultaneously.

4.4 Further Temperatures Tests

As can be seen from the test results, while temperature prediction can produce good results, these are not as good as those for active power or rotor RPM. For this reason, more tests need to be conducted to try to obtain more reliable results. These tests do not focus on the choice of parameters or architecture since they have already been extensively tested, but look for alternative ways. Then for each test the architecture and the best parameters found in previous temperature tests will be used.

All tests in this section study one temperature at a time, thus training one model for each temperature. This is for two reasons: the first is that in this way the algorithm performs one task at a time and this may help in obtaining better results, and the second is that some of the following tests increase the dimensionality of the dataset and the inputs in particular, and in this way it is possible not to provide too many inputs to the models at once.

4.4.1 Introduction of Additional Information into the Dataset

The two methods presented below are based on expanding the dataset with additional columns that aim to add data useful for learning and prediction purposes. Details on these data are given in Chapter 3.

Temperature Interpolation

Interpolation is a curve that approximates the trend of a data distribution. By calculating the interpolation of the distribution between each temperature and wind speed, it is possible to obtain a curve that provides information about the trend of the distribution. This comes in particularly handy in view of the fact that different turbines have different distributions, as shown in Chapter 3. So what is done is to calculate the interpolation of the temperatures for each turbine and add a column for each temperature to the dataset that reports the value of the interpolation based on the wind speed present in each sample. This adds information about the individual turbine that is not otherwise present in the data. The results are shown in Table 4.22.

Architecture	**Gearbox Bearing Temp**	**Gearbox Oil Temp**	**Generator Bearing Temp**
SVR	0.77	0.403	0.722
FNN	0.786	0.432	0.732
GRU	0.889	0.558	0.877

Table 4.22. Interpolation tests in terms of R^2.

The results of adding interpolation bring obvious improvements in the results on all models, while still maintaining performance differences between models. The GRU-based model is thus confirmed as the one that performs best on temperatures.

Temperature Shift

Shifting temperatures involves adding a column for each temperature. It contains the temperature value being analyzed but from the previous timestep, that is, from ten minutes earlier. It is called a temperature shift because to obtain this new column it is sufficient to copy the temperature column and translate (shift) it forward in time by one timestep, so that at each timestamp t in the new column there is the temperature of timestamp $t - 1$.

In this way, an attempt is made to provide information regarding the previous temperature situation, so that the starting value is a real value since it is not estimated but is measured ten minutes earlier. In this way, it should be easier to estimate the current temperature because it is no longer necessary to calculate it

from scratch but only to figure out how much it may have varied since the last timestep. The results are shown in Table 4.23.

Architecture	Gearbox Bearing Temp	Gearbox Oil Temp	Generator Bearing Temp
SVR	0.938	0.917	0.971
FNN	0.97	0.976	0.996
GRU	0.976	0.948	0.996

Table 4.23. Shift tests in terms of R^2.

Adding the previous timestep temperature leads to even greater improvements than adding the interpolation, achieving results comparable to those obtained for active power and rotor RPM. Even differences previously present between the different models become irrelevant, making them comparable.

Interpolation and Shift

After trying the two methods individually, they are tested together. The results are shown in Table 4.24.

Architecture	Gearbox Bearing Temp	Gearbox Oil Temp	Generator Bearing Temp
SVR	0.948	0.92	0.965
FNN	0.981	0.976	0.997
GRU	0.979	0.949	0.996

Table 4.24. Interpolation and shift tests in terms of R^2.

Using interpolation and previous timestep temperature simultaneously does not bring particularly better results than using temperature shift alone. This is because the two columns provide very similar data, the difference being that the interpolation is a generic value that approximates a distribution, while the shift is more precise in that it is the temperature of the component ten minutes earlier, so it is certainly much more accurate for that particular situation.

4.4.2 Single Turbine Models

Since different turbines have very different trends, another approach that is completely different from the previous ones is to study each turbine individually: this

does not require introducing additional non-environmental data into the dataset however on the other hand requires training for each turbine individually. The purpose of this test is simple: since it is difficult to generalize to multiple turbines, in this test can be found out if it is possible to do so on at least one at a time.

In order to do this it is necessary to create a new dataset from scratch, since the old one doesn't work for this purpose because each turbine that is present in one of the three partitions of the dataset, is not in the other two. For this test, it is necessary to create a different dataset for each turbine, dividing into the three partitions of train, val, and test the data available in each dataset. For this new dataset, all data for a period of just under one year and four months are used, and are divided as follows:

- Train: almost 10 months
- Val: two and a half months
- Test: two and a half months

Figure 4.4.1 shows a diagram of the division for an example turbine.

This implies that the trained models will have much less data available, since in the previous dataset the train partition included data from a full year of ten turbines, whereas in this new dataset the training will be done on less than a year of data from a single turbine. The results obtaind from the test on single devices are in Table 4.25 for the SVR-based model, in Table 4.26 for the FNN-based model and in Table 4.27 for the GRU-based model, in which each lines shows the results over a different Wind Turbine Generator.

In this part of the tests there are no additional input dimensions so it might be considered to use for the GRU-based model a single model that makes predictions on all three temperatures at the same time, since it worked best in the tests in the previous section. For brevity, numerical results are not reported, but due to the smaller amount of data in the datasets on the individual turbines the single model in this case performs worse than the three separate models for all WTGs. For this reason, the single output models are used in this training as well.

Results from all three models show that on average the results obtained on individual devices are better than those obtained on the general dataset.

The fact that all models perform better when trained on a single turbine confirms the theory that part of the difficulty of the task is hidden in the difference between Wind Speed-Temperature curve distributions of different turbines. At the same time, however, it can be seen that the accuracy of the results is not constant across all turbines, but rather is highly variable. On some it is very high, on others it is good, and finally there are some that get poor results. This indicates that the difference in behavior is not the only difficulty in this task, but it certainly has a great significance.

It is important to try to understand the reasons for these differences, and to do so, further studies have been conducted trying to find what the turbines with poor

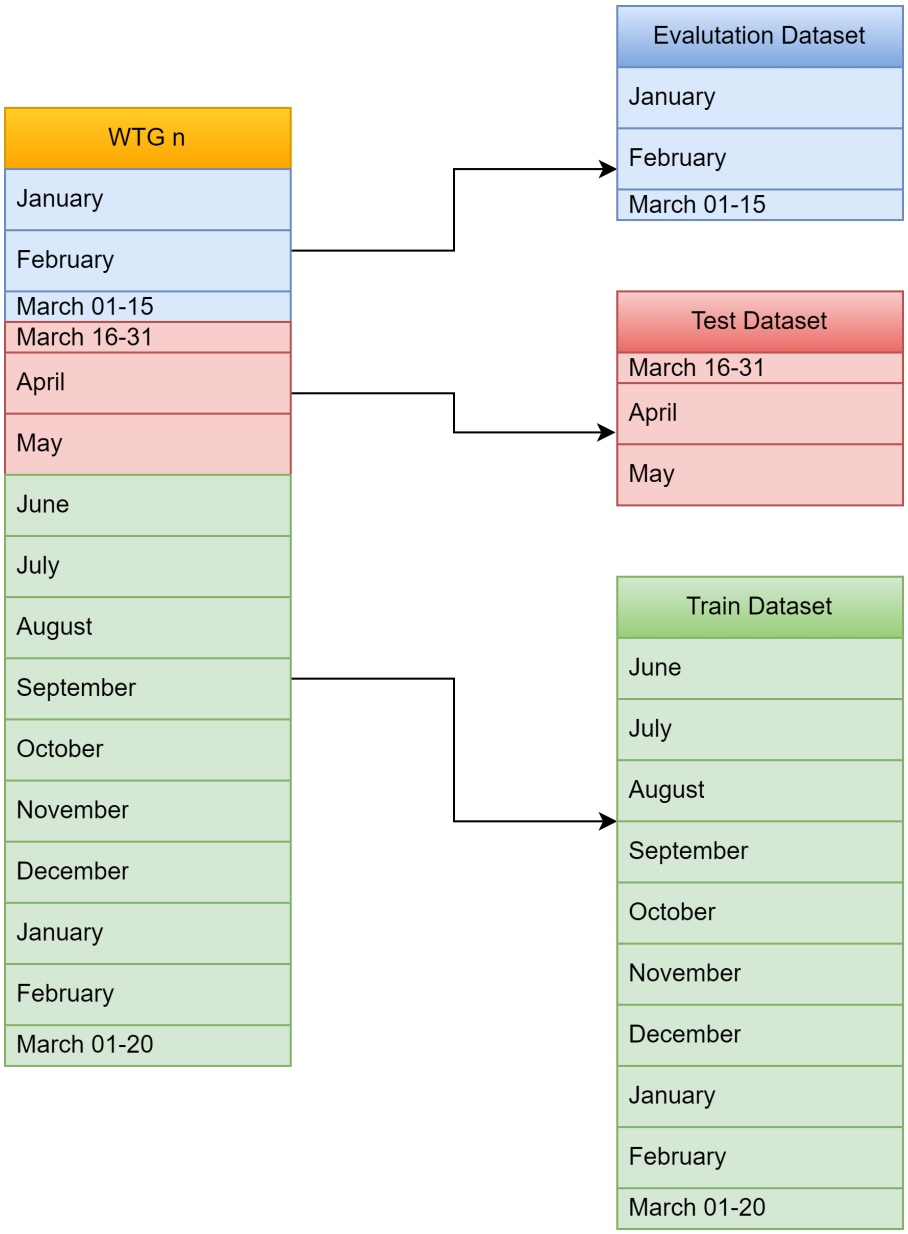

Figure 4.4.1. The data split for a generic turbine. The data starts from January 1, 2022 and goes to March 20, 2023.

performance have in common and how they differ from the others. To do this, the distributions of the Wind Speed-Temperature curve are analyzed through the use of the boxplot graphs shown in Figures 4.4.2 and 4.4.3, belonging to WTG 10 and WTG 14, which are the worst and the best performing ones in terms of gearbox bearing temp, respectively. What can be seen is that a general trend is present whereby the best performing turbines often have a curve that sweeps a lot over the temperatures, thus taking on very different values for different wind speeds. In contrast, the worst performing turbines have the same curve much flatter, thus taking on more similar values for different wind speeds. This is just a general trend and is not always observed, however, it is a very good help in understanding whether a turbine can be more or less easy to predict.

Wind Turbine	Gearbox Bearing Temp	Gearbox Oil Temp	Generator Bearing Temp
WTG 1	0.633	0.234	0.533
WTG 2	0.654	0.131	0.097
WTG 3	0.773	0.212	0.333
WTG 4	0.728	0.424	0.593
WTG 5	0.778	0.275	0.441
WTG 6	0.807	0.376	0.711
WTG 7	0.7	0.278	0.523
WTG 8	0.626	0.617	0.548
WTG 9	0.802	0.489	0.441
WTG 10	0.448	0.291	0.638
WTG 11	0.744	0.203	0.541
WTG 12	0.776	0.262	0.445
WTG 13	0.703	0.283	0.557
WTG 14	0.821	0.321	0.599

Table 4.25. Tests on single devices in terms of R^2 for SVR models.

This trend is particularly valid for Gearbox Bearing Temperature, which has a more compact, curve-like distribution. As for Generator Bearing Temperature, the problem is more complex since the distribution is very broad, not forming a real curve but simply going over an area of varying shape. This is confirmed by the fact that the improvements for this measure are smaller than for the others. Finally, Gearbox Oil Temperature is not given particular importance for the reasons explained above, but in general it respects what was said for Gearbox Bearing Temperature.

Wind Turbine	Gearbox Bearing Temp	Gearbox Oil Temp	Generator Bearing Temp
WTG 1	0.66	0.299	0.541
WTG 2	0.689	0.129	0.142
WTG 3	0.792	0.266	0.363
WTG 4	0.73	0.424	0.597
WTG 5	0.792	0.302	0.437
WTG 6	0.819	0.415	0.731
WTG 7	0.701	0.274	0.453
WTG 8	0.661	0.637	0.518
WTG 9	0.809	0.474	0.42
WTG 10	0.495	0.362	0.645
WTG 11	0.752	0.209	0.563
WTG 12	0.794	0.301	0.469
WTG 13	0.734	0.323	0.544
WTG 14	0.835	0.343	0.595

Table 4.26. Tests on single devices in terms of R^2 for FNN models.

Wind Turbine	Gearbox Bearing Temp	Gearbox Oil Temp	Generator Bearing Temp
WTG 1	0.816	0.522	0.737
WTG 2	0.879	0.201	0.24
WTG 3	0.93	0.218	0.332
WTG 4	0.81	0.629	0.835
WTG 5	0.925	0.279	0.726
WTG 6	0.939	0.538	0.893
WTG 7	0.848	0.473	0.715
WTG 8	0.793	0.772	0.76
WTG 9	0.924	0.743	0.672
WTG 10	0.672	0.42	0.833
WTG 11	0.848	0.216	0.824
WTG 12	0.854	0.338	0.367
WTG 13	0.823	0.427	0.783
WTG 14	0.935	0.372	0.838

Table 4.27. Tests on single devices in terms of R^2 for GRU models.

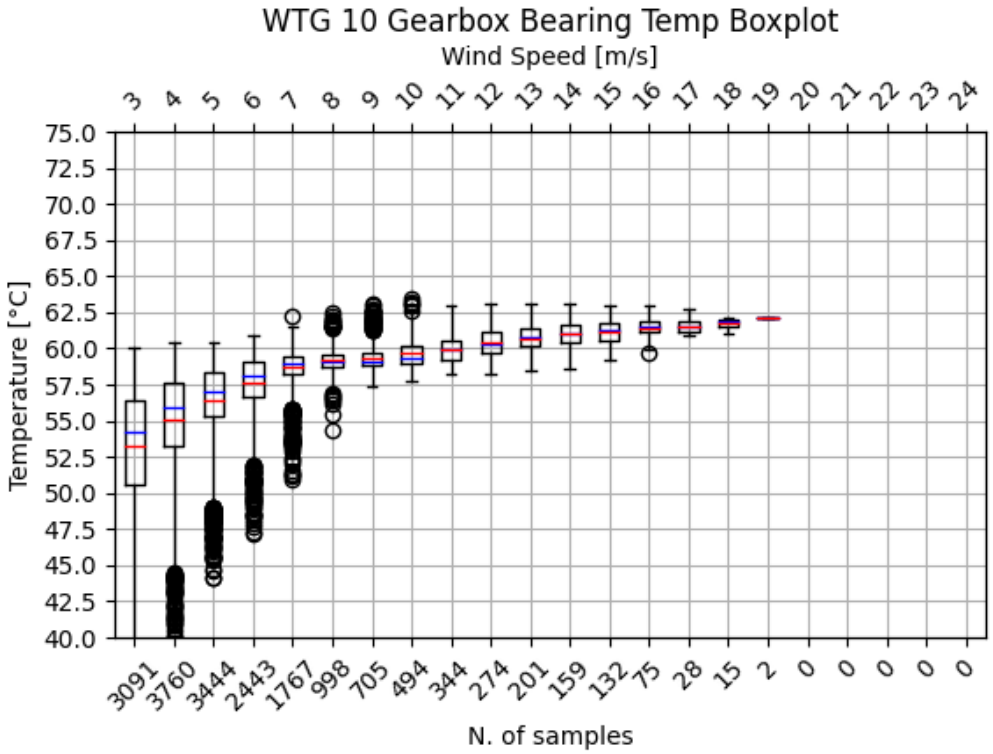

Figure 4.4.2. WTG 10 wind - gearbox bearing temp boxplot.

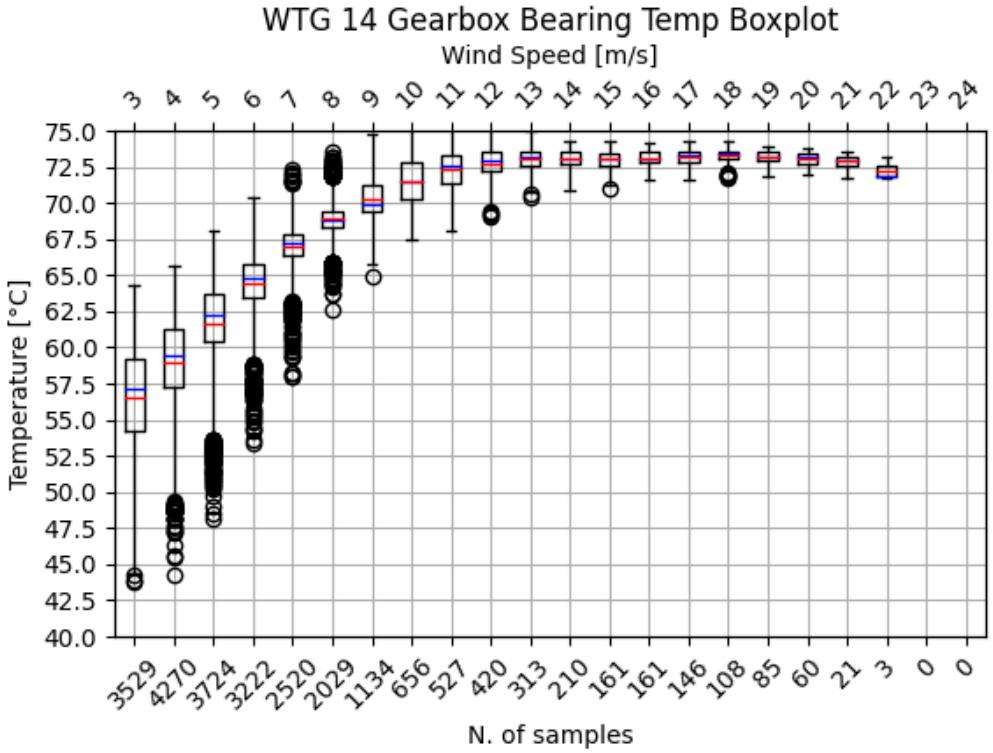

Figure 4.4.3. WTG 14 wind - gearbox bearing temp boxplot.

Chapter 5

Discussion

In this chapter, the models that performed best in the tests of the previous chapter are analyzed in more detail on the test dataset. For each model there are running examples comparing predicted values with observed values showing the magnitude of prediction errors, and other problems more specific to individual models are analyzed.
The chapter is divided into sections, and each section reports the results for a set of measures obtained with the model that performed best for them. Each section is internally divided into subsections, one for each measure analyzed. For the sections referring to temperatures, several models are considered without choosing only the best one since different solutions were used among them that are worth analyzing.

5.1 Active Power and Rotor RPM

The first set of measures studied is that of active power and rotor RPM. They are quite similar to each other, which is why the tests for tuning the hyperparameters were conducted together. This affinity comes from the fact that active power represents the output of the turbine while rotor RPM represents the rotational speed of the rotor. The turbine generator converts rotational energy into electrical energy, and from this comes the fact that they are closely related in that as the rotor RPM increases, the power production also increases and vice versa.

Although all models performed well and similarly, the one that performed best of all is the one based on FNNs. Specifically, the parameters chosen for this FNN are:

- **Hidden layers**: 3
- **Neurons per hidden layer**: 64
- **Learning rate**: 0.001
- **Dropout**: 0.2

Since in the final tuning test the FNNs towed on a single output performed better, again the models will be trained on one output at a time.

5.1.1 Active Power

Active power is the first measure under analysis and is also the easiest to model, as could be seen from the excellent results obtained in almost all the tests in the previous chapter.

The results that the model obtains on the test dataset are shown in Table 5.1.

Active Power
0.993

Table 5.1. Results of the FNN active power model on the test set in terms of R^2.

The results obtained on the test dataset are consistent with those obtained during training. Figure 5.1.1 shows a comparison between the predictions made by the digital twin and the actual data produced by the turbine over the course of any given day. As can be seen, the predictions are extremely accurate, deviating very little from the actual data.

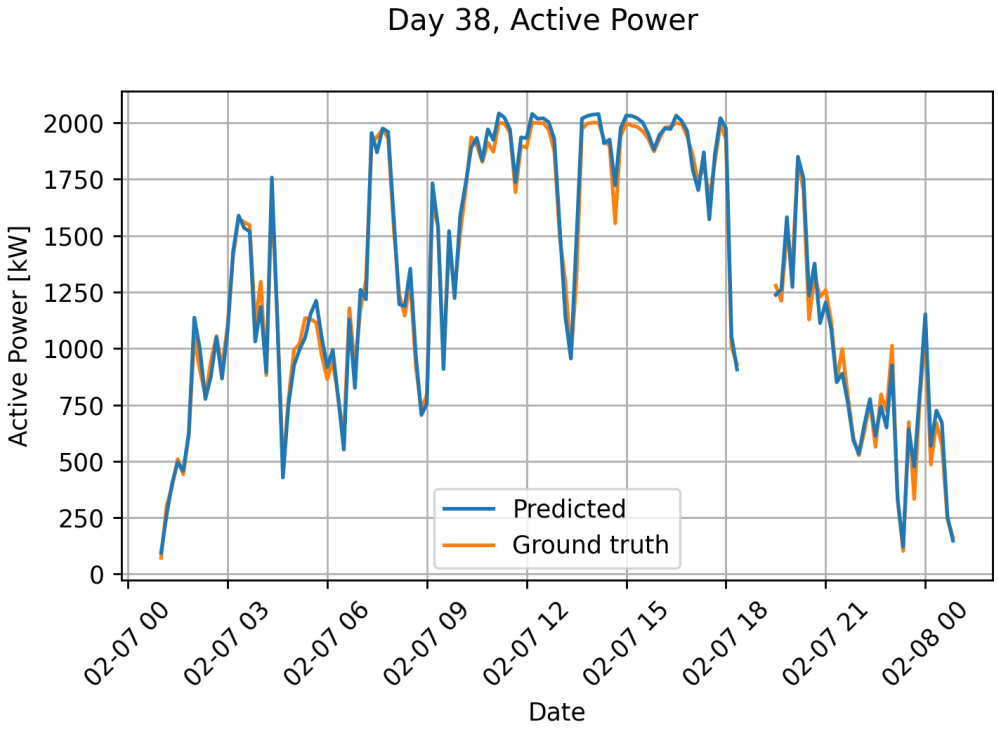

Figure 5.1.1. Active power prediction over test set.

Figure 5.1.2, shows a histogram showing what the error distribution is, obtained by calculating the difference between the actual and predicted value for each sample. Most of the errors are within -40 kW and +60 kW from zero, and considering that

the power goes up to 2000 kW they are very small. Only a small part reaches higher magnitudes, thus committing more significant errors.

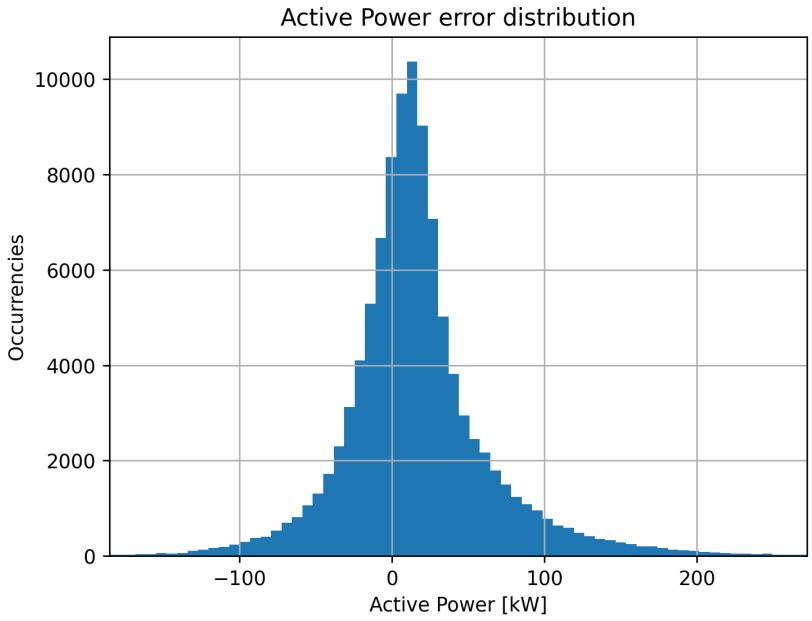

Figure 5.1.2. Active power error distribution over test set.

5.1.2 Rotor RPM

The second measure under analysis is Rotor RPM, and it too performed excellently in most tests.

The results that the model obtains on the test dataset are shown in Table 5.2.

Rotor RPM
0.984

Table 5.2. Results of the FNN rotor RPM model on the test set in terms of R^2.

Also in this case the results obtained on the test dataset are consistent with those obtained during training. Figure 5.1.3 shows a comparison between the predictions made by the digital twin and the actual data produced by the turbine over the course of any given day. As can be seen, the predictions are extremely accurate, deviating very little from the actual data.

Figure 5.1.4, shows a histogram showing what the error distribution is, obtained by calculating the difference between the actual and predicted value for each sample. Most of the errors are within -0.3 rpm and +0.3 rpm from zero, and considering that the rotor RPM goes up to 17 rpm they are very small. Only a small part reaches higher magnitudes, thus committing more significant errors.

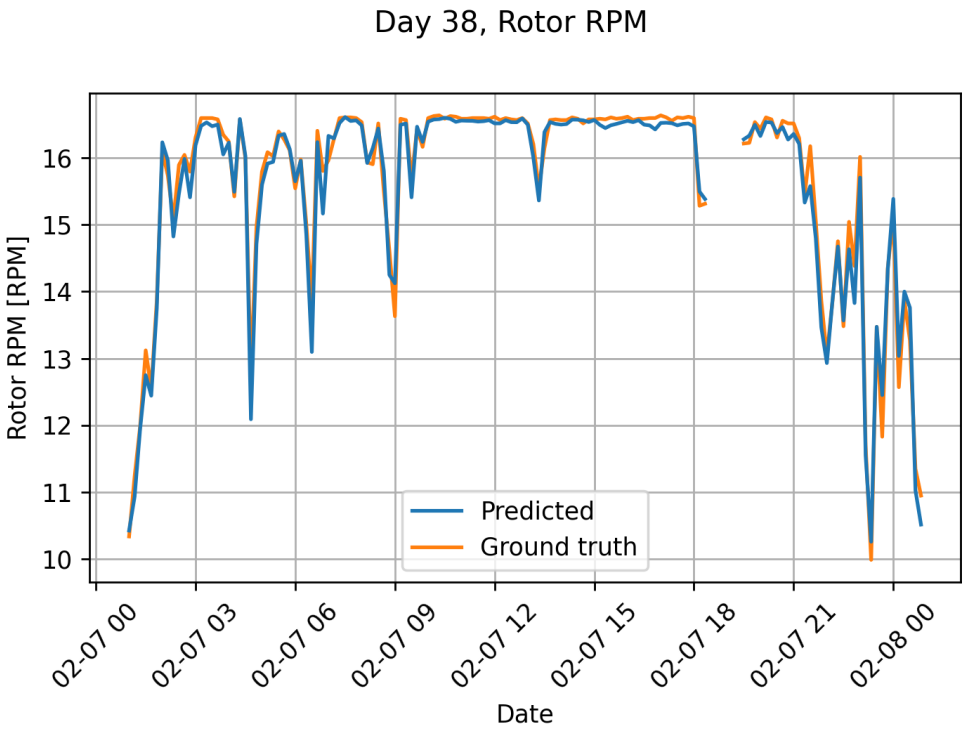

Figure 5.1.3. Rotor RPM prediction over test set.

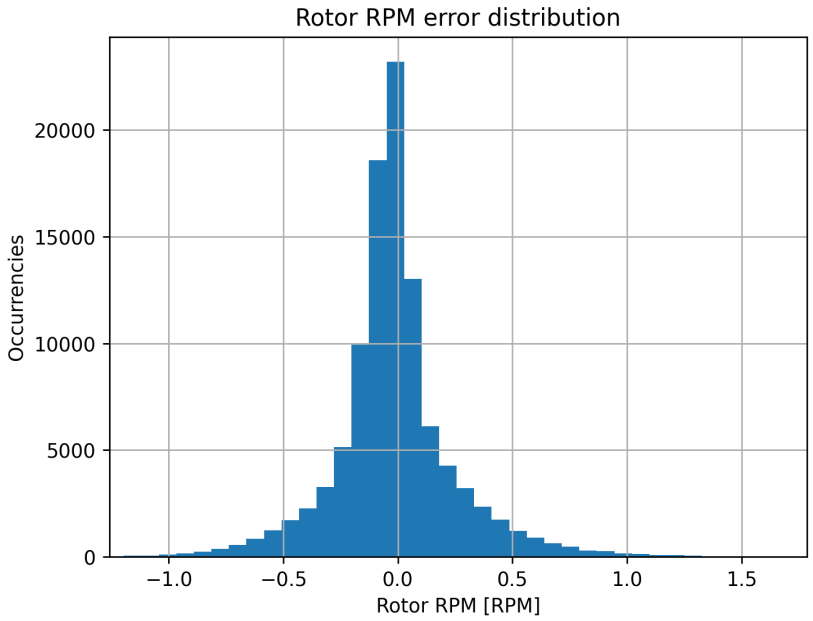

Figure 5.1.4. Rotor RPM error distribution over test set.

5.2 Temperatures

In contrast to active power and rotor RPM in which all models had obtained very good and similar results, for temperatures there were important differences: models based on GRU obtained significantly better results than models based on SVR and FNN. This is because temperatures have an important time dependence that other measurements do not. The temperature of an object depends not only on the heat that is supplied during the last time period under analysis, but also on the temperature it has at the beginning of that period. To get an estimate of the initial temperature, it is therefore important to know the heat previously supplied, something that only GRU-based models can take into account. Specifically, the parameters chosen for the best of these models are:

- **Hidden layers**: 3
- **Neurons per hidden layer**: 64
- **Learning rate**: 0.001
- **Dropout**: 0.4

Since very different models have been tried for temperatures, it is not possible to define one as the best, because each has advantages and disadvantages. Therefore three different ones are analyzed and compared.

The models that obtained the best results of all are those that use interpolation and temperature shift together, so they are the first to be taken into analysis. As can be deduced from the test results in the previous chapter, what makes the results so good is the shift. This can be understood by looking at the models trained with interpolation only and shift only, respectively: the former obtained significantly lower results than the latter. Moreover, while in the models that do not use shift the gap in performance between the GRU-based models and the others is maintained, in those that do use shift this gap is lost and similar results are obtained between them due to the predominance of the information contained in shift. And it is precisely because of this predominance that it makes no sense to analyze the model that uses only the shift, since it has very similar but slightly worse results to the one that uses shift and interpolation together.
Instead, the model that makes use of interpolation alone is analyzed since it has results that are still good although not as good as the model that also uses the shift, but by not introducing among the inputs the output of the previous sample is a very different model and is able to generalize better.
Finally, the model trained on single turbines is analyzed as it also obtained good results and is the only one that does not introduce in any form among the inputs the temperatures to be predicted.

5.2.1 Gearbox Bearing Temperature

The gearbox bearing temperature is the first temperature taken in the analysis and is also the most important to study, because as mentioned in Chapter 3 the gearbox is the component most prone to high temperature and failures.

Interpolation and Shift

The first type of model taken into analysis is one that introduces interpolation of the wind-temperature distribution and temperature shift between inputs. This model is the one that obtained the best results in terms of R^2. The results that the model obtains on the test dataset are shown in Table 5.3.

Gearbox Bearing Temp
0.968

Table 5.3. Results of the GRU model using interpolation and shift on the test set in terms of R^2.

Figure 5.2.1 shows a comparison between the predictions made by the digital twin and the actual data produced by the turbine over the course of any given day. Although the predictions may appear to be quite accurate, close observation shows

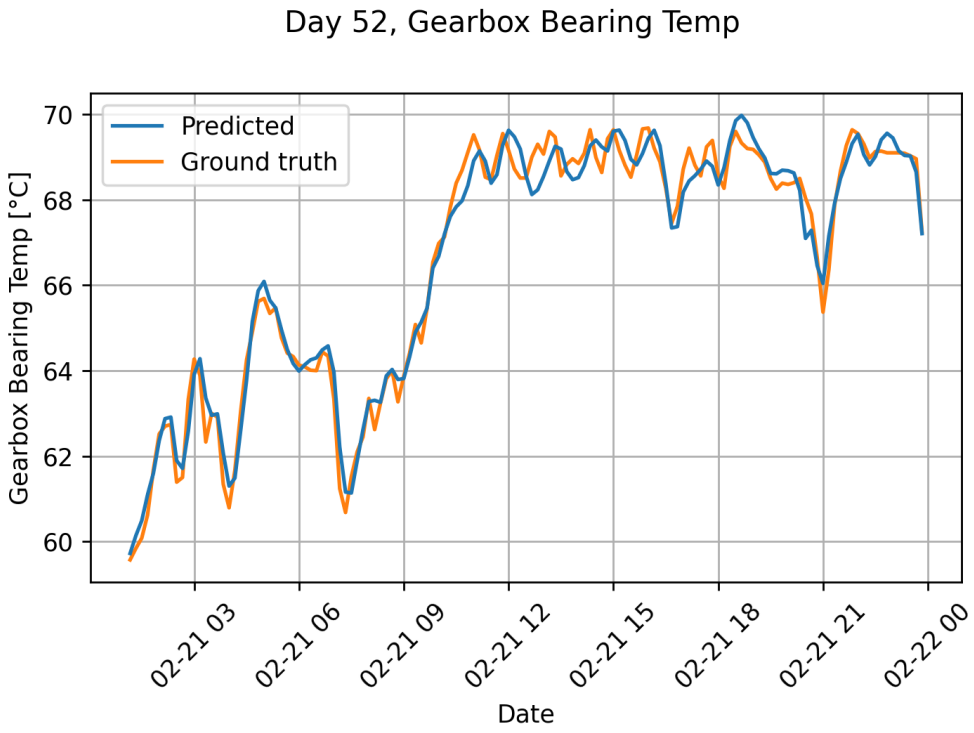

Figure 5.2.1. Gearbox bearing temp interpolation and shift model prediction over test set.

that accuracy of the prediction depends greatly on the trend of the data at the time being analyzed. If the data has a constant trend, whether it is increasing or decreasing, the prediction is accurate. Conversely, if the trend is not constant but

has many unexpected variations, the predictions often turn out to be inaccurate. This is due to the fact that by providing previous temperatures as input, the model tends to focus mainly on them, learning to estimate the trend and predict how it continues. For this reason, in highly unstable situations where the current temperature takes values that cause the local trend to change (as in the case of local maxima and minima) frequently, the predictions are not accurate, and tend to re-establish themselves after the trend changes, once a constancy is re-established. When this happens, the prediction graph takes the shape of the real one but slightly out of phase, due precisely to the fact that it cannot predict these changes and therefore notices them with a delay, when they appear among the inputs in the form of the temperature of previous samples. Therefore, the prediction error remains small, and that is why the R^2 values are high, but this is because a value close to that of the temperature of the previous sample is predicted, which, however, never strays too far from the next one. In contrast, when temperatures are stable or do not have too many sudden changes, predictions are very accurate.

The main problem with relying primarily on previous temperatures and learning to predict the trend is that if non-ideal conditions such as temperatures that are too high or too low are among the previous data, the predictions will continue the trend of the previous temperatures by predicting temperatures that do not deviate too much from the previous one, actually producing non-ideal data and thus going against the basic purpose of the digital twin.
This is the main reason why using previous temperatures as input is not recommended.

Figure 5.2.2, shows a histogram showing the error distribution, obtained by calculating the difference between the actual and predicted value for each sample. Most of the errors are within -1 °C and +1 °C from zero, which is a very small error. Only a small part reaches higher magnitudes, thus committing more significant errors.

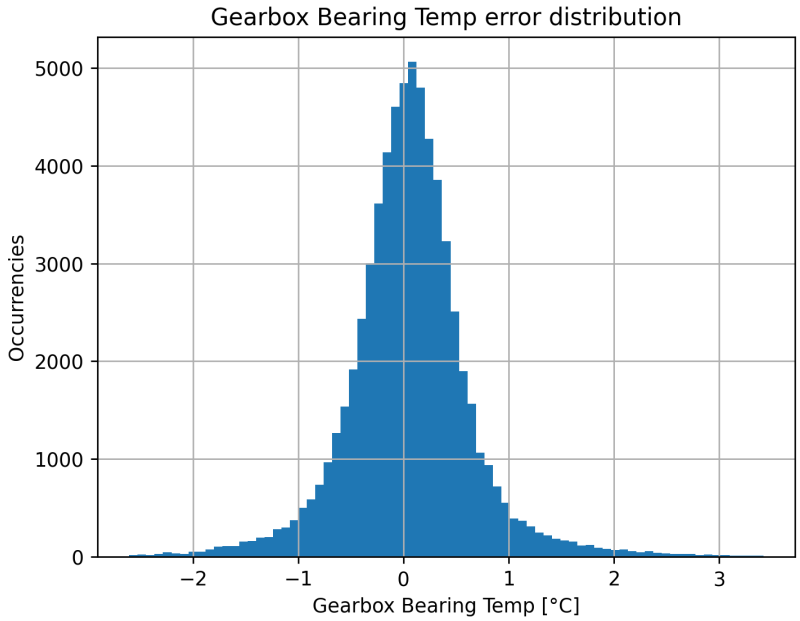

Figure 5.2.2. Gearbox bearing temperature interpolation and shift model error distribution over test set.

Interpolation

The second model analyzed is the one without the temperature shift, so as to see if the interpolation is sufficient to build a single model that can predict the temperatures of several different WTGs. The results that the model obtains on the test dataset are shown in Table 5.4.

Gearbox Bearing Temp
0.861

Table 5.4. Results of the GRU model using interpolation on the test set in terms of R^2.

It is already clear from R^2 that this model is less accurate than the previous one, as is confirmed in Figure 5.2.3, which shows a comparison between the predictions made by the digital twin and the actual data produced by the turbine over the course of any given day.

Although the predictions are worse than those in the previous model, it can be seen that the type of errors is different. They are not only concomitant with trend changes, which indeed are predicted correctly when not too frequent, but most are temperature prediction errors: that is, the trend is understood and predicted correctly, but perhaps the predicted temperature is incorrect and thus the predictions are shifted up or down. In addition, the predictions tend not to be too far away from the interpolation value. This sometimes leads to prediction errors like the shift

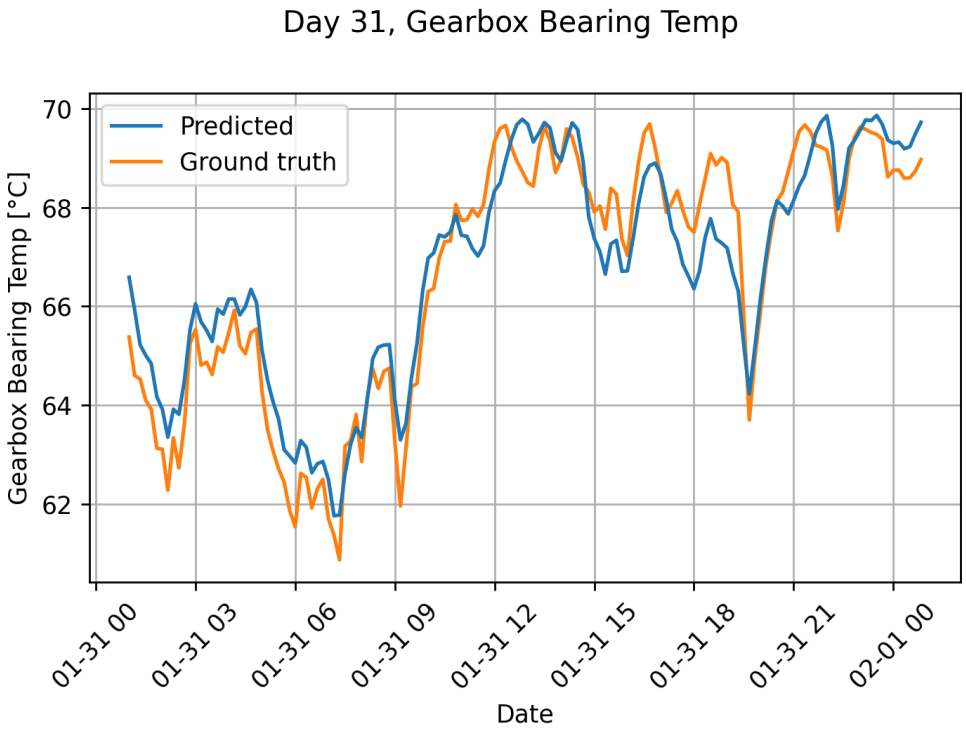

Figure 5.2.3. Gearbox bearing temperature interpolation model prediction over test set.

just mentioned, but at the same time it prevents the prediction of temperatures that are too far out of the standard and thus would not be ideal.

Figure 5.2.4, shows a histogram showing the error distribution, obtained by calculating the difference between the actual and predicted value for each sample. Most of the errors are within -2 °C and +2 °C from zero, which is a quite small error. Only a small part reaches higher magnitudes, thus committing more significant errors.

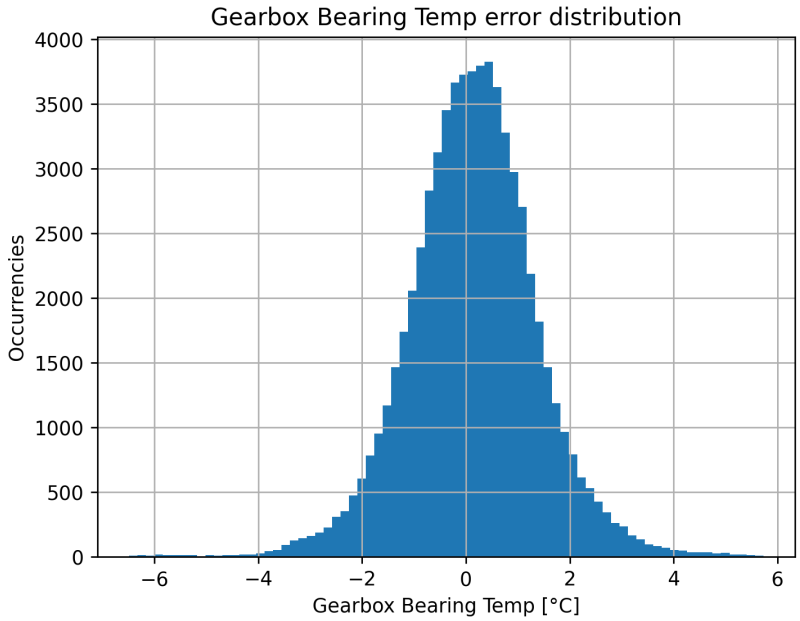

Figure 5.2.4. Gearbox bearing temp interpolation model error distribution over test set.

Single devices

The last model analyzed is the one trained individually on each turbine, designed to learn to understand and predict the temperature trend of the individual turbine since it is different for each one. The results that the model obtains on the test dataset are shown in Table 5.5.

The first thing that can be noticed is that the results are not constant among all turbines but rather vary quite a bit. The reasons are related to the distribution of the wind-temperature curve and are explained in Chapter 4. However, most of the turbines obtained satisfactory results and only a minority obtained results that were not very good.
Figure 5.2.5, which shows a comparison between the predictions made by the digital twin and the actual data produced by one of the turbines with good results over the course of any given day.

The predictions are not as accurate as in the shift model, however, as with interpolation the trend is generally predicted correctly but it may occur that the predicted temperature is not accurate, so the predictions have the correct trend but are shifted up or down from the observed values. The magnitude of this shift depends very much from turbine to turbine, it is generally contained but sometimes it is quite high.
However, non-ideal values are not predicted because in training the model has never seen any and has no input temperatures that would cause it to deviate from the standard.

Figure 5.2.6, shows a histogram showing the error distribution of one of the turbines with good results, obtained by calculating the difference between the actual

Turbine	Gearbox Bearing Temp
WTG 1	0.813
WTG 2	0.91
WTG 3	0.942
WTG 4	0.795
WTG 5	0.903
WTG 6	0.924
WTG 7	0.876
WTG 8	0.764
WTG 9	0.878
WTG 10	0.513
WTG 11	0.767
WTG 12	0.876
WTG 13	0.772
WTG 14	0.919

Table 5.5. Results of the GRU model using the single device model on the test set in terms of R^2. The results are divided by turbine.

and predicted value for each sample. In most of the turbines, most of the errors are between -2 °C and +2 °C or so from zero, which is a quite small error. Only a small part reaches higher magnitudes, thus committing more significant errors.

In conclusion, this is a very good model assuming, however, that the turbine to be analyzed is among the turbines that are predicted well (which are most turbines) and also assuming that the necessary data to train the model for each turbine is available. If these assumptions are met, the model is also better than the model with interpolation. Since the training time is very rapid, having to do one per turbine does not create any particular slowdown.

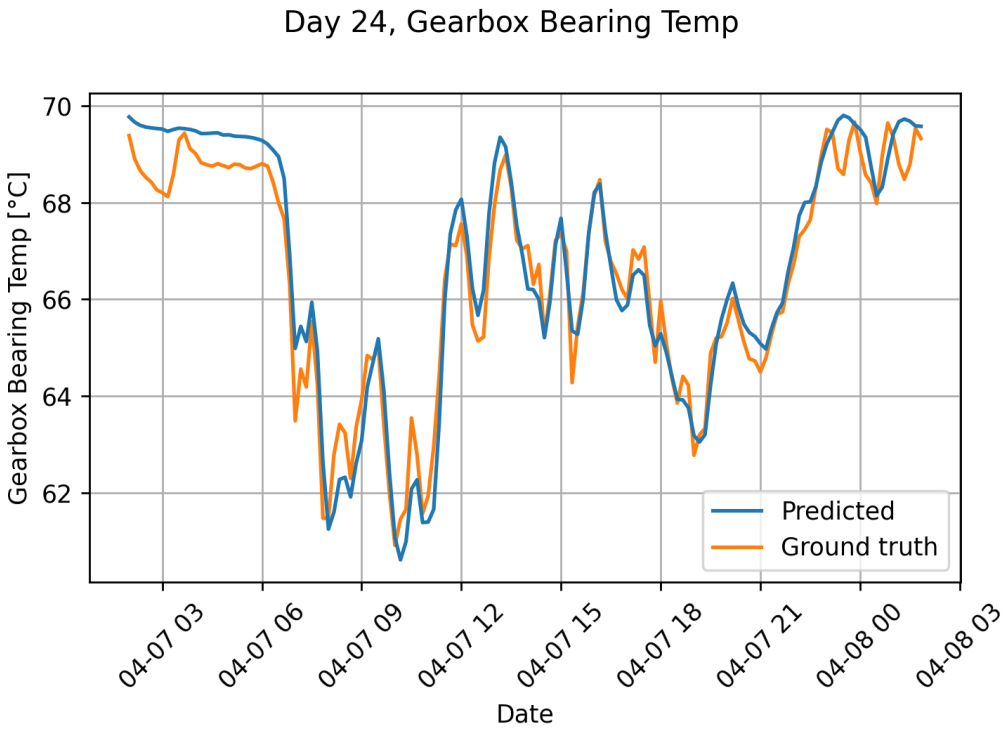

Figure 5.2.5. Gearbox bearing temperature single device model prediction over test set.

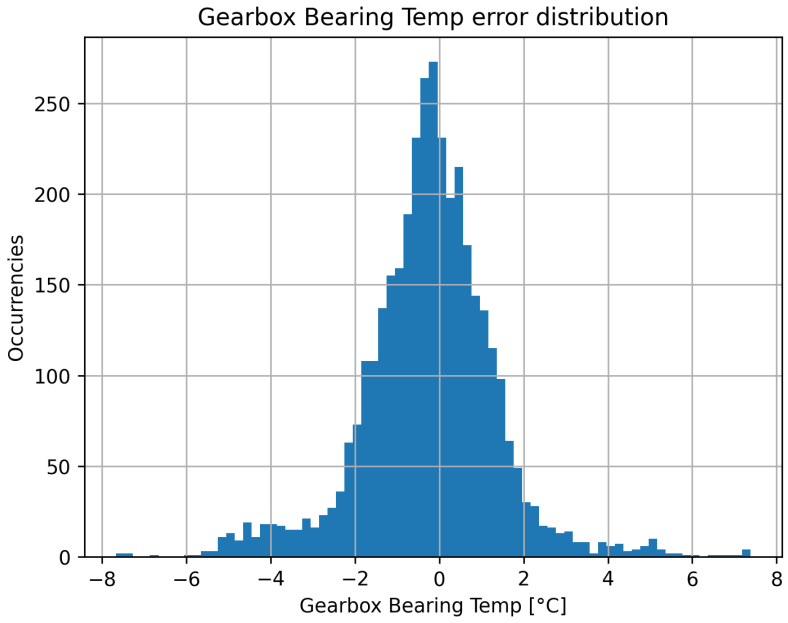

Figure 5.2.6. Gearbox bearing temperature single device model error distribution over test set.

5.2.2 Generator Bearing Temperature

The generator bearing temperature is the second temperature taken in the analysis and is important to analyze because as with the gearbox the generator is prone to high temperatures and failures.

Interpolation and Shift

The first type of model taken into analysis is one that introduces interpolation of the wind-temperature distribution and temperature shift between inputs. This model is the one that obtained the best results in terms of R^2. The results that the model obtains on the test dataset are shown in Table 5.6.

Generator Bearing Temp
0.997

Table 5.6. Results of the GRU model using interpolation and shift on the test set in terms of R^2.

Figure 5.2.7 shows a comparison between the predictions made by the digital twin and the actual data produced by the turbine over the course of any given day. As explained extensively in the previous subsection for the gearbox bearing temperature, the model has problems in predictions where the current temperature changes trend very frequently, presenting predictions that appear out of phase with the observed data, due to the fact that trend changes are noticed only when they appear among the inputs in the form of the temperature of previous samples. In contrast, when temperatures are stable or do not have too many sudden changes, predictions are very accurate.
The results are better than those obtained on the gearbox bearing temperature although it is more difficult to predict because the changes in generator bearing temperature are slower and less frequent and the graph appears smoother.

As also explained in the previous subsection, the main problem with this model is related to the fact that relying mainly on the temperatures of previous samples, if these are not ideal the model can produce non-ideal results. This is the main reason why using previous temperatures as input is not recommended.

Figure 5.2.8, shows a histogram showing the error distribution, obtained by calculating the difference between the actual and predicted value for each sample. Most of the errors are within -1 °C and +1 °C from zero, which is a very small error. Only a small part reaches higher magnitudes, thus committing more significant errors.

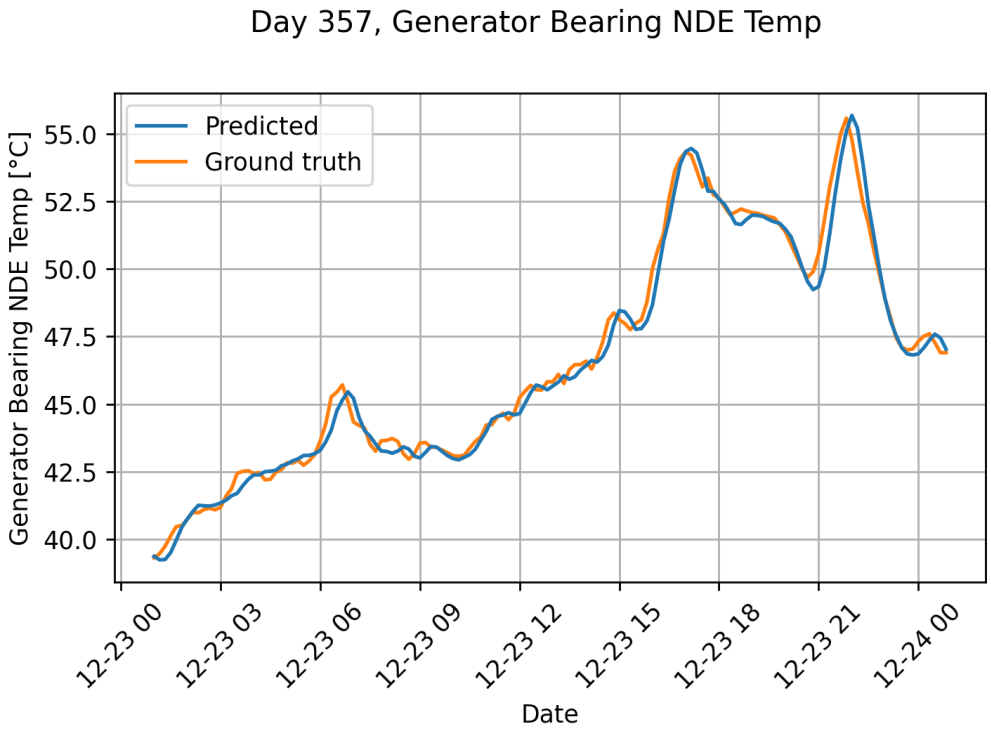

Figure 5.2.7. Generator bearing temp interpolation and shift model prediction over test set.

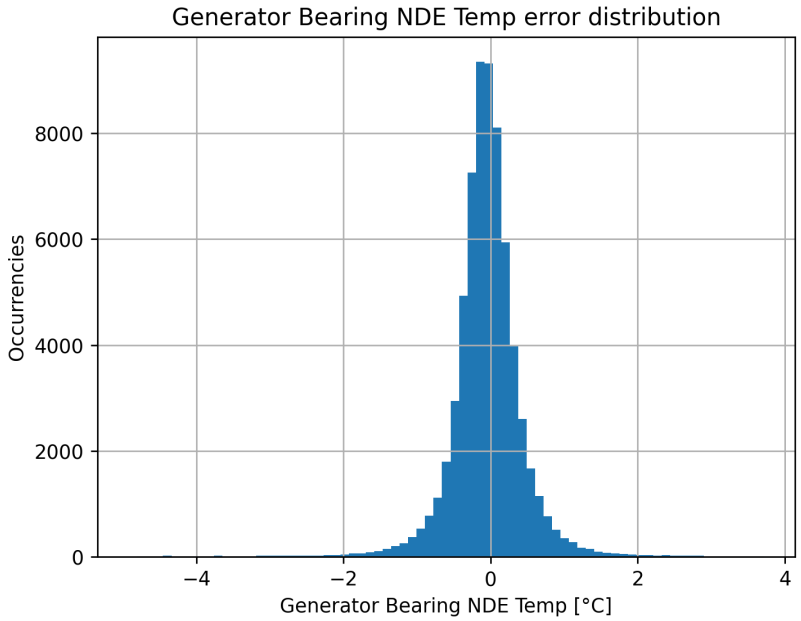

Figure 5.2.8. Generator bearing temperature interpolation and shift model error distribution over test set.

Interpolation

The second model analyzed is the one without the temperature shift, so as to see if the interpolation is sufficient to build a single model that can predict the temperatures of several different WTGs. The results that the model obtains on the test dataset are shown in Table 5.7

Generator Bearing Temp
0.791

Table 5.7. Results of the GRU model using interpolation on the test set in terms of R^2.

In this case, the difference between the model with and without shift is even larger than the gearbox bearing temperature. This is because the shape of the wind-temperature distribution of the generator bearing temperature is very broad, not forming a real curve but simply going over an area of varying shape.
It is already clear from R^2 that this model is less accurate than the previous one, as is confirmed in Figure 5.2.9, which shows a comparison between the predictions made by the digital twin and the actual data produced by the turbine over the course of any given day.

As explained in the previous subsection, predictions from this model are less accurate but are better able to predict the temperature trend, although this can be shifted up or down without straying too far from the interpolation value. In this way, non-ideal values are hardly predicted.
However, unlike gearbox bearing temperature where this shift is small, in this case the shift is often very large and it also happens more commonly that the trend is not correctly predicted.

Figure 5.2.10, shows a histogram showing the error distribution, obtained by calculating the difference between the actual and predicted value for each sample. Most of the errors are between -5 °C and +5 °C from zero, which is not exactly a small error, Only a small part reaches higher magnitudes, thus committing more significant errors.

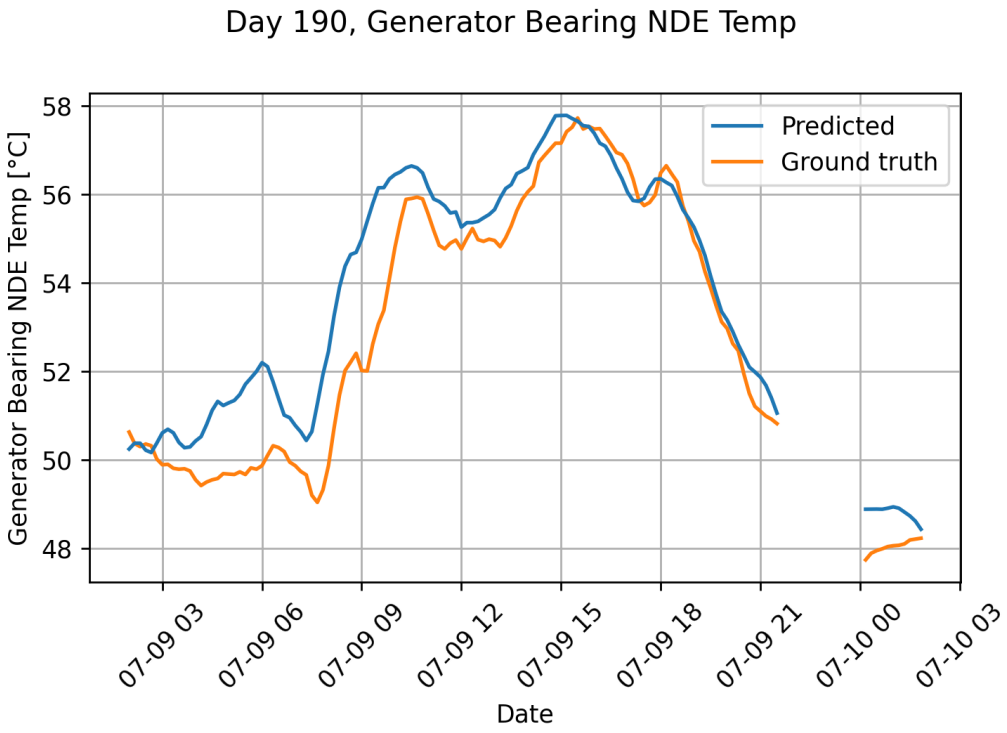

Figure 5.2.9. Generator bearing temperature interpolation model prediction over test set.

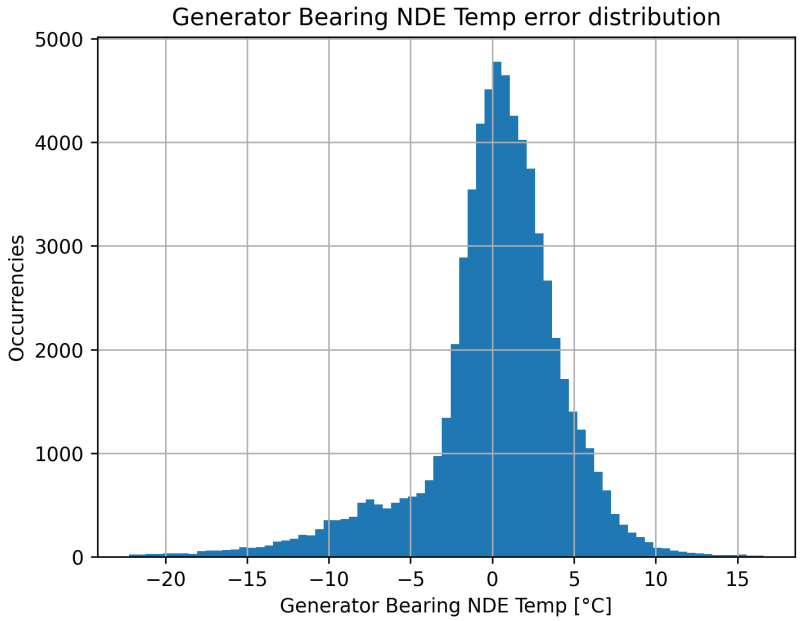

Figure 5.2.10. Generator bearing temp interpolation model error distribution over test set.

Single devices

The last model analyzed is the one trained individually on each turbine, designed to learn to understand and predict the temperature trend of the individual turbine since it is different for each one. The results that the model obtains on the test dataset are shown in Table 5.8.

Turbine	Generator Bearing Temp
WTG 1	0.862
WTG 2	0.554
WTG 3	0.615
WTG 4	0.71
WTG 5	0.452
WTG 6	0.643
WTG 7	0.775
WTG 8	0.852
WTG 9	0.824
WTG 10	0.766
WTG 11	0.662
WTG 12	0.479
WTG 13	0.834
WTG 14	0.878

Table 5.8. Results of the GRU model using the single device model on the test set in terms of R^2. The results are divided by turbine.

The first thing that can be noticed is that the results are not constant among all turbines but rather vary quite a bit. The results are on average worse than the gearbox bearing temperature for the same reason related to the wind-temperature curve explained for the previous model. As with the gearbox bearing temperatures, it is immediately apparent that the results vary greatly from turbine to turbine, with a slightly higher percentage of poor results, however.
Figure 5.2.11 shows a comparison between the predictions made by the digital twin and the actual data produced by one of the turbines with good results over the course of any given day.

As for gearbox bearing temperature, the predictions are not as accurate as in the shift model, and as with interpolation the trend is generally predicted correctly but it may occur that the predicted temperature is not accurate, so the predictions have the correct trend but are shifted up or down from the observed values. The magnitude of this shift depends very much from turbine to turbine, it is generally contained but sometimes it is very high. However, compared with the gearbox bearing temperature, it happens more often that the model also misses the trend or the shift is very high.

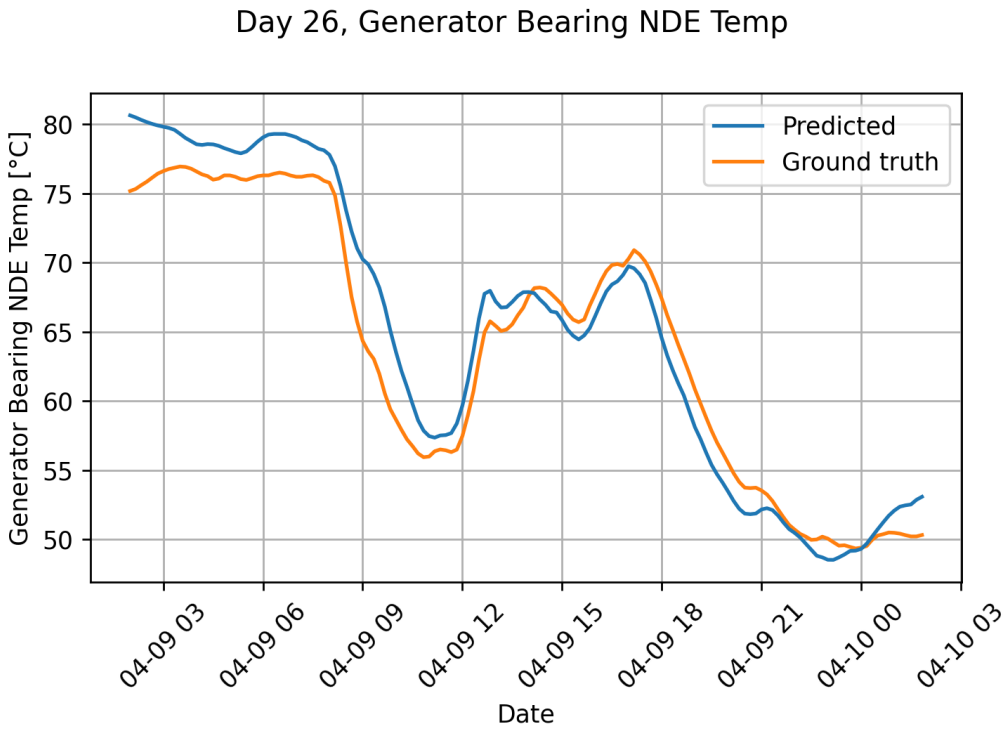

Figure 5.2.11. Generator bearing temperature single device model prediction over test set.

Non-ideal values are not predicted because in training the model has never seen any and has no input temperatures that would cause it to deviate from the standard.

Figure 5.2.12, shows a histogram showing the error distribution in one of the turbines with good results, obtained by calculating the difference between the actual and predicted value for each sample. In most of the turbines, most of the errors are between -5 °C and +5 °C or so from zero, which is not exactly a small error, and in the case of malfunctioning turbines it may even be higher.

In conclusion, this is a good model assuming, however, that the turbine to be analyzed is among the turbines that are predicted well (which are most turbines) and also assuming that the necessary data to train the model for each turbine is available. If these assumptions are met, the model is also better than the model with interpolation. Since the training time is very rapid, having to do one per turbine does not create any particular slowdown.

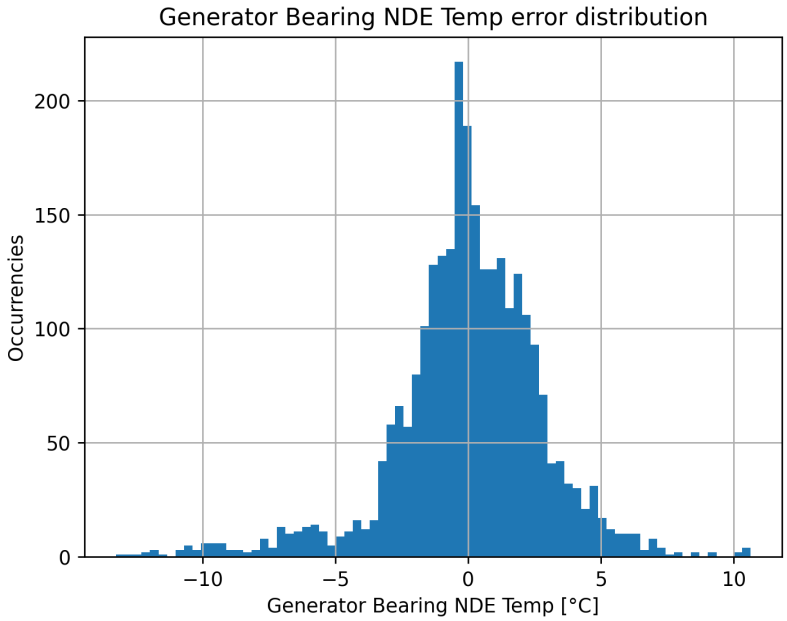

Figure 5.2.12. Generator bearing temperature single device model error distribution over test set.

5.2.3 Gearbox Oil Temperature

As mentioned earlier, gearbox oil temperature has less relevance because it is cooled externally, so it is particularly difficult to predict it. For this reason, it is not analyzed like other temperatures.

Chapter 6

Conclusions

In this thesis, it is developed a digital twin which exploits the knowledge contained in the data to learn how to predict the operation of a wind turbine under ideal conditions, predicting its main variables of interest based on the data derived from the surrounding environment. Various technologies in the areas of machine learning and deep learning, as well as various data management solutions, are proposed and tested for its development.

To reach this goal, it is necessary to perform an important data cleaning to extract only the set of data considered ideal so that the digital twin can be trained only on them to ensure that its predictions always fall within the ideal situation. To do this, several filters are adopted to remove all data related to wind speeds that are too low for power production to begin, or related to zero power produced, typical of when the turbine is stationary. Then, the theoretical production curve is used to remove samples that do not have ideal performance. Finally, DBSCAN is used for outlier detection to remove non-ideal data on internal component temperatures, since there is no theoretical curve for these values.

Three different models were trained multiple times and tested with different combinations of hyperparameters to obtain the best possible results. This process of hyperparameters tuning is repeated for the different groups of measurements to be predicted so that models are optimized specifically to make predictions about them.
Specifically, the three models tested are:

- **SVR**: It belongs to the world of machine learning, it has been tested so that not only neural network-based models are used, but the results obtained are worse than those obtained from the other models. It also proved to be particularly slow to train, which is why the training is done using only a portion of the dataset.
- **FNN**: A deep learning model, it gets the best results where knowledge of previous sample data is not required, with short training times.
- **GRU**: A deep learning model derived from RNNs, which can process temporal sequences of data and create an internal state that represents a memory. This allows it to take into account what has happened previously, which

is particularly useful for studying temperatures, where data from previous samples must also be considered. This model achieves very good results for all outputs, but it is in the prediction of internal component temperatures that it achieves significantly better results than other models.

In addition, because temperatures are particularly difficult to predict, alternative solutions had to be sought in order to achieve better results.
In particular, three different strategies were developed to aid learning:

- **Temperatures shift**: This solution involves including among the inputs the temperature taken from the previous sample, that is, the temperature ten minutes earlier. The idea is to provide a starting point for calculating the current temperature.
  This solution is the one that performed best in terms of R^2, but it has a major flaw that it may predict values that are not ideal.

- **Temperatures interpolation**: This solution involves calculating the interpolation of the wind-temperature curve of each wind turbine so that its standard trend is known. For each sample, the interpolation value calculated with the wind speed of that sample is included as an input. This provides the model with information about the standard temperature values for that specific turbine based on wind speed.
  This solution did not achieve results as accurate as the previous one but it proved to be able to predict trends better.

- **Single device models**: The last proposed solution is different from the previous ones because it does not involve adding any data among the inputs, but instead a single model for each turbine is trained. In this way, it is not necessary to provide information about the standard temperatures of each turbine since there is a model for each one and it can learn them on its own. This model, like the previous one, has slightly worse accuracy than the former but can predict trends better than either.

Finally, once the best models and their hyperparameters have been chosen, a detailed analysis is performed on the test dataset to verify the results obtained by each model, observing examples of predictions and the distribution of errors.

6.1 Future works

The solutions proposed in this thesis can predict active power and rotor RPM with very high accuracy. For temperatures, more specialized solutions are used which are able to achieve excellent results as well. In any case, the latter are the ones that have the most room for improvement and therefore those on which any future work could be focused.

For example, one could try to train a model to predict the difference between the temperature of the previous sample and the current one. In this way, since the model does not have to predict a temperature but only its variation over the last

ten minutes, it might be possible to solve the problem of different temperatures on different turbines since it is no longer necessary to predict the final temperature value. To obtain the final value, it is necessary to add the temperature value of the previous sample to the predicted variation. Probably, in different turbines the temperature also varies differently, however, these variations might be more similar to each other than the temperatures themselves.

Further work could also focus on the models trained on individual devices, trying to understand more accurately why some turbines are predicted better than others. In addition, it would be appropriate to do the training with a larger dataset, as the dataset used in this work has many different turbines but for a relatively narrow period. By being able to use data on a single turbine for each training, it would be appropriate to collect data over a larger period so that the training can be performed on a dataset of appropriate breadth and so that larger validation and test datasets can also be created and thus so that larger tests can be performed.

Finally, another strategy might be to try to use models other than those used in this thesis, so as to see whether or not these can make a substantial difference. Certainly, models that can analyze time sequences to predict temperatures should be chosen.

Bibliography

[1] Iea. *World total energy supply by source, 1971-2019*. URL: https://www.iea.org/reports/key-world-energy-statistics-2021/supply.

[2] NWS Southern Region. *Origin of wind*. URL: https://web.archive.org/web/20090324043730/http://www.srh.noaa.gov/jetstream/synoptic/wind.htm.

[3] Chung-hoi Yung. *Why is the equator very hot and the Poles very cold?* URL: https://www.hko.gov.hk/en/education/edu06nature/ele_srad.htm.

[4] Steve Ackerman. *Sea and land breezes*. URL: http://cimss.ssec.wisc.edu/wxwise/seabrz.html.

[5] Hrvoje Čočić. *How do wind turbines work*. Feb. 2019. URL: https://chrvojeengineering.com/2019/02/03/how-does-wind-turbine-work/.

[6] Fouad Amri et al. «Toward an evolutionary multi-criteria model for the analysis and estimation of wind potential». In: *Journal of Power and Energy Engineering* 03.11 (2015), pp. 14–28. DOI: 10.4236/jpee.2015.311002.

[7] christoph.schilter@meteotest.ch METEOTEST. *The Swiss Wind Power Data Website*. URL: https://wind-data.ch/tools/weibull.php?lng=en.

[8] Ugo Romano and Jacques Ruer. «6.2 - Generazione elettrica dal vento». In: *Enciclopedia degli idrocarburi*. Vol. 3. Istituto della enciclopedia italiana, 2007, pp. 561–574.

[9] *Wind energy basics*. URL: https://web.archive.org/web/20100923194211/http://www.awea.org/faq/wwt_basics.html.

[10] *wind energy — everything you need to know*. URL: https://justenergy.com/blog/everything-you-need-to-know-about-wind-energy/.

[11] *Vertical axis wind turbine — part 1 - ansys innovation courses*. URL: https://courses.ansys.com/index.php/courses/vertical-axis-wind-turbine-part-1/.

[12] Peter J. Schubel and Richard J. Crossley. «Wind Turbine Blade Design». In: *Energies* 5.9 (2012), pp. 3425–3449. DOI: 10.3390/en5093425.

[13] Ammar A. Alshannaq et al. «Structural analysis of a wind turbine blade repurposed as an electrical transmission pole». In: *Journal of Composites for Construction* 25.4 (2021). DOI: 10.1061/(asce)cc.1943-5614.0001136.

[14] *Wind Turbine Components*. Feb. 2023. URL: https://windmillstech.com/wind-turbine-components/.

[15] Ya-Lun Chou. «Statistical analysis: With business and economic applications». In: Holt, Rinehart and Winston, 1975.

[16] Erwin Kreyszig. *Advanced engineering mathematics*. Wiley, 1979.

[17] *2.6 - (Pearson) Correlation Coefficient R*. URL: https://online.stat.psu.edu/stat462/node/96/.

[18] Joseph Lee Rodgers and W. Alan Nicewander. «Thirteen ways to look at the correlation coefficient». In: *The American Statistician* 42.1 (1988), pp. 59–66. DOI: 10.1080/00031305.1988.10475524.

[19] *Correlation matrix in R (3 examples)*. Mar. 2022. URL: https://statisticsglobe.com/correlation-matrix-in-r.

[20] Frank E. Grubbs. «Procedures for detecting outlying observations in samples». In: *Technometrics* 11.1 (1969), pp. 1–21. DOI: 10.1080/00401706.1969.10490657.

[21] Lewis Barnett Vic; Lewis. *Outliers in statistical data*. 1978.

[22] Shubhtripathi. *All about outliers in machine learning*. Oct. 2020. URL: https://shubh-tripathi.medium.com/all-about-outliers-in-machine-learning-2e26be9708f1.

[23] *Cos'è un gemello digitale?* URL: https://www.ibm.com/it-it/topics/what-is-a-digital-twin.

[24] Simone Formentin, Klaske van Heusden, and Alireza Karimi. «Model-based and data-driven model-reference control: A comparative analysis». In: *2013 European Control Conference (ECC)* (2013). DOI: 10.23919/ecc.2013.6669388.

[25] *Scipy.interpolate.univariatespline*. URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.UnivariateSpline.html.

[26] John R. Koza et al. «Automated design of both the topology and sizing of analog electrical circuits using genetic programming». In: *Artificial Intelligence in Design '96* (1996), pp. 151–170. DOI: 10.1007/978-94-009-0279-4_9.

[27] Christopher M. Bishop. *Pattern recognition and machine learning*. Springer, 2016.

[28] Honglak Lee et al. «Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations». In: *Proceedings of the 26th Annual International Conference on Machine Learning* (2009). DOI: 10.1145/1553374.1553453.

[29] Vladimir Estivill-Castro. «Why so many clustering algorithms». In: *ACM SIGKDD Explorations Newsletter* 4.1 (2002), pp. 65–75. DOI: 10.1145/568574.568575.

[30] Martin Ester et al. «A density-based algorithm for discovering clusters in large spatial databases with noise». In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining* (1996), pp. 226–231.

[31] *Demo of DBSCAN clustering algorithm*. URL: https://scikit-learn.org/stable/auto_examples/cluster/plot_dbscan.html.

[32] Hans-Peter Kriegel et al. «Density-based clustering». In: *WIREs Data Mining and Knowledge Discovery* 1.3 (2011), pp. 231–240. DOI: 10.1002/widm.30.

[33] Michael I. Jordan et al. «Support Vector Regression Machines». In: *Advances in neural information processing systems 9*. MIT Press, 1997, pp. 155–161.

[34] Corinna Cortes and Vladimir Vapnik. «Support-Vector Networks». In: *Machine Learning* 20.3 (1995), pp. 273–297. DOI: 10.1007/bf00994018.

[35] Tom Sharp. *An introduction to support vector regression (SVR)*. May 2020. URL: https://towardsdatascience.com/an-introduction-to-support-vector-regression-svr-a3ebc1672c2.

[36] Thomas Hofmann, Bernhard Schölkopf, and Alexander J. Smola. «Kernel methods in machine learning». In: *The Annals of Statistics* 36.3 (2008). DOI: 10.1214/009053607000000677.

[37] Maysam F. Abbod et al. «Application of artificial intelligence to the management of urological cancer». In: *Journal of Urology* 178.4 (2007), pp. 1150–1156. DOI: 10.1016/j.juro.2007.05.122.

[38] Saul Dobilas. *Feed forward neural networks - how to successfully build them in Python*. Feb. 2022. URL: https://towardsdatascience.com/feed-forward-neural-networks-how-to-successfully-build-them-in-python-74503409d99a.

[39] CHRISTIAN W. DAWSON and ROBERT WILBY. «An artificial neural network approach to rainfall-runoff modelling». In: *Hydrological Sciences Journal* 43.1 (1998), pp. 47–66. DOI: 10.1080/02626669809492102.

[40] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. «General Back-Propagation». In: *Deep learning*. The MIT Press, 2017, pp. 211–214.

[41] Ahmed Tealab. «Time series forecasting using Artificial Neural Networks Methodologies: A systematic review». In: *Future Computing and Informatics Journal* 3.2 (2018), pp. 334–340. DOI: 10.1016/j.fcij.2018.10.003.

[42] *Recurrent neural networks cheatsheet star*. URL: https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks.

[43] John F. Kolen, Stefan C. Kremer, and Sepp Hochreiter. «Gradient flow in recurrent nets: the difficulty of learning long-term dependencies». In: *A field guide to dynamical recurrent networks*. IEEE Press, 2001.

[44] Joel C. Heck and Fathi M. Salem. «Simplified minimal gated unit variations for recurrent neural networks». In: *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)* (2017). DOI: 10.1109/mwscas.2017.8053242.

[45] Jonte Dancker. *A brief introduction to recurrent neural networks*. Dec. 2022. URL: https://towardsdatascience.com/a-brief-introduction-to-recurrent-neural-networks-638f64a61ff4.

[46] Javaid Nabi. *Recurrent neural networks (rnns)*. July 2019. URL: https://towardsdatascience.com/recurrent-neural-networks-rnns-3f06d7653a85.

[47] Simeon Kostadinov. *Understanding GRU networks*. Nov. 2019. URL: https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be.

[48] *Mean squared error (MSE)*. URL: https://www.probabilitycourse.com/chapter9/9_1_5_mean_squared_error_MSE.php.

[49] Georges Casella and Roger L. Berger. In: *Statistical inference*. Duxbury/Thomson Learning, 2002, pp. 556–556.

[50] Thomas Bruce. «Analysis of the premature failure of wind turbine gearbox bearings». PhD thesis. 2016.

[51] Lee Brannan. *Omphalos, Uber's parallel and language-extensible time series backtesting tool*. Aug. 2022. URL: https://www.uber.com/blog/omphalos/.

[52] *Python*. URL: https://www.python.org/.

[53] *NumPy*. URL: https://numpy.org/.

[54] *Pandas*. URL: https://pandas.pydata.org/.

[55] *Scikit-Learn*. URL: https://scikit-learn.org/stable/.

[56] *Pytorch*. URL: https://pytorch.org/.

[57] Peter Worcester. *A comparison of grid search and randomized search using Scikit learn*. June 2019. URL: https://medium.com/@peterworcester_29377/a-comparison-of-grid-search-and-randomized-search-using-scikit-learn-29823179bc85.

[58] *Sklearn.svm.SVR*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html.