# POLYTECHNIC OF TURIN

**Master's Degree in Data Science and Engineering**


**Master's Degree Thesis**

# Robustness of Machine Learning algorithms applied to gas turbines

Andres Felipe Cardenas Meza

**Supervisors**
**Prof. Danilo GIORDANO**
**Ing. Jonas DEURELL**
**Prof. Iolanda LEITE**

**06 2024**

# Abstract

This thesis demonstrates the successful development of a software sensor for Siemens Energy's SGT-700 gas turbines using machine learning algorithms. Our goal was to enhance the robustness of measurements and redundancies, enabling early detection of sensor or turbine malfunctions and contributing to predictive maintenance methodologies. The research is based on a real-world case study, implementing the Cross Industry Standard Process for Data Mining (CRISP DM) methodology in an industrial setting. The thesis details the process from dataset preparation and data exploration to algorithm development and evaluation, providing a comprehensive view of the development process. This work is a step towards integrating machine learning into gas turbine systems.

The data preparation process highlights the challenges that arise in the industrial application of data-driven methodologies due to inevitable data quality issues. It provides insight into potential future improvements, such as the constraint programming approach used for dataset construction in this thesis, which remains a valuable tool for future research.

The range of algorithms proposed for the software sensor's development spans from basic to more complex methods, including shallow networks, ensemble methods and recurrent neural networks.

Our findings explore the limitations and potential of the proposed algorithms, providing valuable insights into the practical application of machine learning in gas turbines. This includes assessing the reliability of these solutions, their role in monitoring machine health over time, and the importance of clean, usable data in driving accurate and satisfactory estimates of different variables in gas turbines. The research underscores that, while replacing a physical sensor with a software sensor is not yet feasible, integrating these solutions into gas turbine systems for health monitoring is indeed possible. This work lays the groundwork for future advancements and discoveries in the field.

## Keywords

Gas turbines, machine learning, deep learning, predictive maintenance, software sensor, data quality, Cross Industry Standard Process for Data Mining (CRISP DM), Reliability, Ensemble Methods

# Acknowledgments

I would like to express deepest my appreciation to all the people who provided me with the opportunity and support necessary to complete this work. This journey has been made possible by the guidance and support I have received throughout my academic career from teachers, colleagues, friends and family.

Specially, I want to thank my company supervisor Jonas Deurell for his insightful guidance from the inception of this master thesis to its completion. I am equally thankful with both my academic supervisors from KTH and Polytechnic of Turin: Iolanda Leite and Danilo Giordano, who provided me with the necessary guidance during this process.

Moreover, I want express my deepest gratitude to my family for the constant support I have received from them during my years of study. It has been a journey marked by triumphs and challenges, and I am particularly thankful to my parents and sister. Their unwavering belief in me that has been always a source of encouragement and strength.

Finally, I would like to express my sincere appreciation to my friends from high school in Colombia, from my bachelor's years in Turin, and those I met in Stockholm. Though we may be spread across the globe, these friendships remain close to my heart. Each one has contributed to who I am today and has made my life all the more enjoyable and in different ways helped me to conclude my academic journey with this work. The memories we've created together are deeply cherished.

Finspång, Sweden, June 2024
Andres Felipe Cardenas Meza

# Contents

# List of Figures

# List of Tables

# Listings

# List of acronyms and abbreviations

| | |
|---|---|
| APAC | Agnostic Probably Approximately Correct |
| CFD | Computational Fluid Dynamics |
| CNN | Convolutional Neural Network |
| CRISP-DM | Cross Industry Standard Process for Data Mining |
| DAG | Directed Acyclic Graph |
| FFT | Fast Fourier Transform |
| GRU | Gated Recurrent Unit |
| IGV | Inlet Guide Vane |
| LSTM | Long Short-Term Memory |
| MAE | Mean Absolute Error |
| MAPE | Mean Absolute Percentage Error |
| ML | Machine Learning Algorithm |
| MLE | Maximum Likelihood Estimator |
| MLP | Multi Layer Perceptron |
| NLP | Natural Language Processing |
| OLS | Ordinary Least Squares |
| OOP | Object Oriented Programming |
| PCA | Principal Component Analysis |
| PELT | Pruned Exact Linear Time |
| PGM | Probabilistic Graphical Model |
| RMS | Root Mean Square |
| RMSE | Root Mean Squared Error |
| RNN | Recurrent Neural Network |

| | |
|---|---|
| SVM | Support Vector Machine |
| SVR | Support Vector Regression |
| UML | Unified Modeling Language |

# Chapter 1

# Introduction

Machine Learning techniques are rapidly increasing their relevance in a wide range of practical settings and present a unique opportunity to uncover hidden relationships in complex contexts [1]. This thesis seeks to delve into the capabilities and robustness of these algorithms in the context of gas turbines - complex devices that generate energy and find use in a broad spectrum of practical applications. This thesis aims to lay the foundation for understanding the potential applications of Machine Learning Algorithm in gas turbines by addressing the initial challenge of developing a software sensor within them. The emphasis will be on the comprehensive process of algorithm development, from data collection and cleaning, to the design and validation of such a software sensor. It will further highlight potential areas of future work with the ultimate goal of fully integrating Machine Learning Algorithm (ML) into gas turbine systems. The entirety of this research is done using data from gas turbines at Siemens Energy.

At their core, gas turbines are devices that convert chemical energy into mechanical or electrical energy through the combustion of fuel[2]. They are increasingly relevant in nowadays context for their key role in the pursuit of achieving a sustainable future. They are particularly well suited for complementing renewable energy sources due to their inherent ability to quickly adjust output levels, thus compensating for the typical fluctuations in renewable power generation [3]. Furthermore, they are used in a wide range of applications and through the usage of particular cycles it is possible to achieve remarkable efficiencies [4]. An emerging area of interest is the study of making viable the fuel substitution with hydrogen, at least partially,[5][6][7], which has the advantage of significantly lower emissions. In essence, gas turbines play a key role in our modern society and their relevance is expected

to increase with further advancements.

## 1.1 Background

Gas turbines are complex devices whose main purpose is to convert chemical energy into mechanical or electrical energy [2]. Nowadays, gas turbines are the most versatile turbo-machinery item available in the market [8] due to the variety of applications it can be employed for. As discussed in the book by Soaires Claire [8] its applications encompass propelling vessels, supplying energy to the grid, generating electricity and heating for a district , etc ….

A rudimentary explanation of gas turbines functioning principle is the following [9]:

- **Gas Compression**: A gas is compressed in the first stage. This is in most cases air.
- **Combustion**: The compressed gas is mixed with other type of gas for igniting a combustion process, which releases chemical energy. In most cases the mixture contains natural gas, but it has been also used in combination with hydrogen or ammonia.
- **Expansion**: The compressed high-pressurized gas passes through the blades of the turbine generating work.

A gas turbine has hundreds of sensors that monitor it during its operation. Such sensors are essential for safety and performance reasons. It is also important for predictive maintenance and prevent potential machine damage with long-term usage. Sensors monitor several different quantities such as pressure and temperatures in the different stages of the turbine, flame presence/absence, blade distance to turbine walls (blade tip), amount of gas supplied, generated power, ambient parameters [10]. Finding relationships among all those quantities is non-trivial due to the complex dynamics that govern the system and the thousands of tolerances of all the different components in a gas turbine.

Although gas turbines are renowned for their reliability and infrequent failures, continuous monitoring is essential to identify any deviations from expected performance and operating points, and further minimize potential downtime.

The inception of this master's thesis lies in the potential benefits that could be obtained from systematically discerning these relationships in an automated

manner from the large amount of data amassed from numerous gas turbines. This is further motivated by the advent of machine learning, which has seen a surge in applications across various industries [1].

## 1.2 Problem

Monitoring of the functioning of a gas turbine is a complex challenge primarily for the vast amount of processes and sensors that need to be monitored.

As stated before a gas turbine is an inherently complex system which is governed by specific physical rules and mechanisms. Such a system of equations often has not analytical closed solution and researchers and engineers resort to Computational Fluid Dynamics (CFD) simulations to observe the behaviour during design phases.

Furthermore, numerous efforts have been done, also at Siemens Energy, to develop gas turbine digital twins to characterize a specific-turbine behaviour. This method is purely based on the physical model of the turbine and requires significant expertise and time to develop.[11]

In this context, considering the vast amount of data collected by a company like Siemens Energy from their gas turbines, a data-driven approach poses itself as a solution for obtaining a digital-twin like method for creating redundancies for the gas turbine. The problem we will examine in depth during this master thesis is whether Machine Learning Algorithms are viable and robust to be used in gas turbines.

### 1.2.1 Original problem and definition

The research question that this master thesis project aims to solve is whether Machine Learning Algorithm can be used for building a data-driven software sensor, and what the resulting robustness of such a solution would entail.

A software sensor refers to the concept of a system that computes the expected value of a physical variable purely based on others, non trivially redundant, physical variables. In the context of a gas turbine, one can imagine the computation of the exhaust temperature based on the combustion chamber temperature and the air mass-flow.

### 1.2.2 Scientific and engineering issues

The inherent complexity of gas turbines, from a scientific perspective, adds a layer of difficulty to the problem. Gas turbines are exceptionally non-

linear systems, with thousands of movable or replaceable parts, making the development of a universal solution an immense, if not impossible, challenge. The situation is further complicated by the various fuel mixtures used in gas turbines, which significantly alter their behavior. The complexity is further increased by the fact the flexibility of turbines [8] makes them to be used in a large set of operation modes, which differ from use case to use case; even when considering exclusively terrestrial gas turbines.

From an engineering standpoint, monitoring every possible change in a structured manner is challenging, making it extremely difficult to pinpoint the exact differences between two gas turbines of the same type.

Adding to these complexities is the demand to assemble a comprehensive and clean dataset. As will be further discussed in the thesis, numerous factors contribute to this challenge and it is not always possible to obtain such a dataset due to quality constraints or scarcity of data. Meticulous consideration is necessary when assembling the dataset and selecting appropriate techniques for designing and validating our algorithms.

## 1.3 Purpose

The purpose of this project is to leverage the power of Machine Learning Algorithm for creating redundancies in a gas turbine, which will help Siemens Energy to obtain more robust systems and be able to detect potential sensor or component failure at an early stage. This will help to further reduce the downtime of gas turbines which is an extremely important feature for customers [12].

A further improvement on the reliance of gas turbines is beneficial for easier adoption of gas turbines and better incorporation with renewable sources. As well as the development of a precise redundancy system might allow in the future to reduce the number of physical sensors in the gas turbine thus reducing costs for both client and Siemens Energy and increasing the performance of the gas turbine.

Furthermore, this project will examine the limitations of such system. This is of great importance since overconfidence in the algorithm might lead the operator of the gas turbine to neglect changes not detected by the algorithm and lead to possible catastrophic failures.

## 1.4 Research Methodology

The selected methodology for this research is based on the Cross Industry Standard Process for Data Mining[13]. This model's philosophy is to approach the business problem iteratively, enabling continuous refinement of the proposed solution and its alignment with the requirements as both the data and model improve. From a theoretical perspective the methodology is governed by the principles distilled from the statistical framework for ML presented in [14]. This framework will be further expanded on the Section 2.1.

This choice was based on the fact that this master thesis project was addresses a novel usage of Machine Learning Algorithms in gas turbines. Its iterative structure allows for repeated cycles of development, evaluation, and refinement, which are critical in the pursuit of a production-level product; which can be continued after the master thesis is concluded.

This thesis, from a philosophical perspective, emphasizes the need for safe AI utilization, as overconfidence in the developed algorithms could potentially lead to catastrophic outcomes and unclear accountability [15]. It focuses particularly on evaluating the robustness of these solutions and identifying their observed limitations. This approach aims to encourage safe usage and create awareness about the limits of this technology.

## 1.5 Goals

The goal of this master thesis is to perform a data-driven analysis on Siemens Energy gas turbines data to assess the viability and robustness of a software sensor that tracks the expected value of a turbine physical sensor. Furthermore, we will explore how such a solution can be practically used to detect malfunctions or deviations from normal behaviour and its limitations.

The completion of this major goal has been divided into several minor sub-goals in alignment with the CRISP-DM phases:

1. **Business Understanding**: Understanding of potential applications of a software sensor in gas turbines give characteristics of data.

2. **Dataset Preparation and Data Understanding** : This is a crucial step of the algorithm development as *a model is just as good as its data*[16]. The dataset construction and feature selection cover a significant part of this project work. This will cover as well examination of potential data quality issues and potential fixes.

3. **Model Design**: The modeling of the problem and the choice of the algorithms based on the characteristics of the data. This is the choice of the methods used for solving the business requirements.

4. **Model Validation**: Model validation is particularly relevant in the context of this thesis for correct interpretation of the designed algorithms and comprehensive understanding of their limitations.

5. **Deployment**: This phase will not be covered in this master thesis. It is left for potential future work to observe how this solution performs in a production environment.

The project aims to deliver a solid foundation on top of which further research can be conducted at Siemens Energy for developing and deploying Machine Learning Algorithm for gas turbines.

A significant deliverable of this project is the development of a tool designed to efficiently utilize the data collected by Siemens Energy. This tool will enable the creation of comprehensive and clean datasets that can be used for training ML models. This will significantly increase the velocity at which further iterations carried out after the conclusion of this master thesis can be done. By streamlining the process of dataset preparation and construction, the tool will allow for rapid and flexible data handling, consequently accelerating the pace of further research. On the same line an analysis of data quality issues will be obtained, thus enabling to identify problems and , potentially, improve data governance policy.

## 1.6 Delimitations

There are several kinds of gas turbines in the market offered by Siemens Energy. While most of them operate by the same principles they all have subtle but important differences that need to be assessed individually. This master thesis will focus on the research on turbines of the kind *SGT-700*.

Additionally, this thesis does not require that the software sensor solutions operate in real-time in the strictest sense. Hence, we permit the use of algorithms that function with a certain delay. For example, the application of non-causal filters on signals, such as Gaussian Filters, is permissible.

While it is also desirable to have a software sensor that works in all possible scenarios equally good this thesis will focus in the operation points in which a gas turbine is typically used.

## 1.7 Structure of the thesis

Chapter 2 presents the background necessary to understand the thesis in terms of gas turbines and Machine Learning Algorithms that are employed in this master thesis. It also provides an important overview of previous work conducted both at Siemens Energy and in academia, with which a comparison is possible.

Subsequently the Chapter 3 will present the developed data collection system, particularly relevant for using the data available in Siemens Energy. A discussion about data quality issues is shown in this section.

Following, the results from the data exploration are shown along with the proposed methods for developing this software sensor in Chapter 4.

The Chapter 5 shows the obtained results in different settings. This is arguably one of the most important chapters of the thesis as it discusses the findings and the potential usages of this software sensor as well as the found limitations and considerations to be taken for its usage.

Finally Chapter 6 presents a discussion of the obtained results and potential avenues for future work based on this project.

# Chapter 2

# Background

This chapter aims to give the reader a broad perspective of the theoretical foundation of Machine Learning Algorithms and the various assumptions and considerations fundamental for developing and evaluating them. An introduction to traditional methods such as linear regressors will be presented, along with a link to a statistical framework for ML.

Subsequently, the concepts underlying deep learning methods and the typical architectures used for different tasks will be presented. Together with the concept of AutoEncoders, which will be fundamental for discussing related work in the development of Machine Learning Algorithms for gas turbines as well as for the development of this master thesis.

To ensure a holistic understanding of the subject matter, a brief but complete overview of gas turbines, their diverse components, and applications will be presented. This will also include an exploration of the some measurement systems present in such devices, which form the primary subject of investigation in this master thesis. Moreover, gas turbines will be discussed in the context of their potential role in a sustainable future.

In the end of the chapter, similar works to this master thesis will be explored with a summary of each work's specific contribution and a critical examination of their limitations. Works include previous studies conducted at Siemens Energy, which entail particular relevance as those are exactly the same gas turbines that will occupy this work. Then, works coming from different industries will be presented.

## 2.1 Statistical framework of ML

The entirety of this work can be encompassed in a statistical formal model for ML shown in the book *Understanding Machine Learning* by Shalev [14]; fruit of extensive works on the matter. This statistical model shows the implicit assumptions done when developing a Machine Learning Algorithm. In this section we will provide an overview of that framework and how it can be elegantly connected with the turbine regression task that occupies this work.

Although the principles are more general, for the sake of simplicity, we will limit our discussion to regression. In regression, we aim to predict the value of a certain variable $y$ (or a vector $\boldsymbol{y}$) from a set of predictors $\boldsymbol{x}$. The following basic notions are necessary for describing the model in detail:

- **Domain Set** $\mathcal{X}$: This is the set of possible values that the features can assume. In the context of gas turbines one can think of the physical range in which the sensor variables can exist.
- **Target Set** $\mathcal{Y}$: This is the set of possible values that the target can assume. Similarly to $\mathcal{X}$, this can be associated to the range of values in which the target sensor variable can exist.
- **Prediction Rule** $h : \mathcal{X} \rightarrow \mathcal{Y}$: This is the output of the Machine Learning Algorithm. It is a prediction rule that solves the task we are aiming for (i.e. given a sample $x$ it predicts the target variable $y$)

We then assume that it exists a probability distribution that describes the entire population (i.e. $\exists \mathcal{D} : (\mathcal{X} \times \mathcal{Y}) \rightarrow [0, 1]$ ) and the role of the algorithm $\mathcal{A}$ that we develop is to learn over samples drawn from that distribution to output a *good* prediction rule. Formally the algorithm is a mapping $\mathcal{A}$ that given a collection of samples drawn from $\mathcal{D}$ (i.e. the training set $S = (\boldsymbol{x}_i, y_i) \sim \mathcal{D}_{i=1}^{m}$) outputs $h$.

A *good* algorithm is one that generates a rule $h$ so that ,*in most cases*, the prediction of $h(\boldsymbol{x})$ is close to the true value $y$. Formally, this is expressed as the **generalization error** $\mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{D}}[(h(\boldsymbol{x})–y)^2]$, read as the expected value of the squared difference between the predicted and actual value of the target. The **generalization error** is used to quantify how good an algorithm is. However, in practical scenarios, it is infeasible to obtain the *true* value of the generalization error as this would necessitate infinite sampling from $\mathcal{D}$, which is never the case. Therefore, the best estimate of the generalization error that we can achieve is the test error; this is the error made by the predicting rule $h$ on a dataset $S_t$, where samples are also drawn from $\mathcal{D}$.

The test error provides the best estimate that we have of how well a model will perform *in general*. This underlies the widely accepted practice in the Machine Learning Community to evaluate the performance of a prediction rule $h$ given by a certain algorithm $\mathcal{A}$ a *test dataset*.

The key point here is the assumption that, irrespective of the learning algorithm used, the predicted rule that we learn is meant to be applied on data that comes from the same distribution $\mathcal{D}$. This will be particularly relevant when discussing the results and understanding the success and failure of Machine Learning Algorithms in different turbines. This is a crucial observation in understanding why and when a method will work or will stop working and when we are doing correct assumptions.

### 2.1.1 Overfitting and Learnability

In this section we will discuss the intrinsic limitations of machine learning models and the theoretical origin of them as well as practical considerations on how those are solved or handled both in practice and in theory. Our aim is to answer the question: *How good can a prediction rule $h$ given by an algorithm $\mathcal{A}$ be?* Notice that when asking this question we are not mentioning any information about the distribution $\mathcal{D}$, besides the fact that it must exist to be able to generate a training sample, so the results are general.

The first issue arises when considering the algorithm $\mathcal{A}$ and how to create the prediction rule. A good algorithm candidate is the algorithm that selects the prediction rule $h$ that minimizes the error in the training dataset; often defined as *empirical risk minimization*. However, this algorithm without further consideration suffers from a major problem: **overfitting**. Overfitting is a common challenge in machine learning applications that stems from the constraints presented by data sets of limited size and various other factors [17]. In layman's terms, overfitting occurs when a Machine Learning Algorithm learns patterns that are unique to the training dataset and not generally applicable. Consequently, the algorithm performs exceptionally well on the training dataset but poorly on the test dataset. It means that simply minimizing the error done in the training dataset is not a guarantee for a minimization of the generalization error.

To illustrate the problem, consider the regressor described in Equation (2.1). This is a regressor that predicts $y_i$ if the sample is present in the dataset and $0$ otherwise. It is clear that this regressor will have an exceptional performance in the training dataset, with an empirical error $L(S,h) = \frac{\sum_{i=1}^{n}(h(\boldsymbol{x_i})-y_i)^2}{n} = 0$, but will have poor generalization performance

as it will fail for every point that was not present in the training dataset.

$$y = h(\boldsymbol{x}) = \begin{cases} y_i & \text{if } (\boldsymbol{x}, y_i) \in S \\ 0 & \textit{otherwise} \end{cases} \tag{2.1}$$

The theoretical and practical solution to the overfitting problem lies in learning from a hypothesis class $\mathcal{H}$. This means that the algorithm $\mathcal{A}$ must produce a prediction rule $h$ that belongs to the hypothesis class $\mathcal{H}$. This approach introduces an inductive bias into the learning process, thereby incorporating *a priori* belief into the model.

In practical terms, this explains why introducing structure into machine learning models has proven successful in various fields. A well-known and established example of this are Convolutional Neural Networks. CNNs leverage the expected spatial invariances in images to develop networks that process images efficiently [18]. To date, such networks remain strong in the field of computer vision and have been refined having the same basic principle in mind [19].

Similarly, an inductive bias can be introduced to handle sequential data, where time invariances are expected. Sequential types of networks like Long Short-Term Memory networks, and more recently, transformers [20], have proven successful in this regard. These architectures have proven to be effective in NLP tasks and are particularly relevant in the context of any sequential data modeling such as time series, which is the main topic of this master thesis.

However, the introduction of that inductive bias comes at a cost. It is that the best possible outcome of our learning algorithm $\mathcal{A}$ is a prediction rule that has generalization error $L_D = \min_{h' \in \mathcal{H}} L_D(h')$. The more strong our inductive bias is (i.e the more hypothesis we introduce about the underlying distribution of data) this error $\epsilon_a$, called approximation error, greater will be. This is, we are accepting that in our hypothesis class $\mathcal{H}$ there does not necesarily exist a predictor that is able to achieve perfect generalization error.

Furthermore, there is a hard limit on the performance of **any** prediction rule coming from **any** hypothesis class given by **any** algorithm due to the intrinsic probabilistic nature of the relationship between features and targets, introduced by $\mathcal{D}$. Often, this error is present in real world scenarios due to our intrinsic ignorance about full description of the system and the subsequent non deterministic nature of our problems. This is, that two instances with the same feature values $\boldsymbol{x}$ could result in different targets. In gas turbine terms, it means that even if we know the value of all sensors in the gas turbine except

one it might not be enough to deterministically compute the missing one due to missing information that we do not have access to. This error is called the error of the Bayes predictor and it is always a part of the *approximation error*. In formal terms, the best possible prediction, given a set of features $\boldsymbol{x}$, is the $y$ that satisfies $y = arg\max_y f_{\mathcal{D}}(y'|\boldsymbol{X})$ (i.e. the $y$ that maximizes the conditional probability density of the target given the features). The predictor with this behaviour is called the *Bayes Predictor* and it sets a limit over which we cannot hope to obtain better results; even with the best algorithm and hypothesis class.

### 2.1.2 No free lunch theorem

In this section we will analyze a theoretical result that explains there is no universal hypothesis class. It implies that the inductive bias introduction is necessary and a subsequent trade-off arises to not *underfit* or *overfit* the data. In practical terms, the hypothesis class must be complex enough to represent the problem but not to severely overfit the training dataset.

We first must introduce the concept of APAC learnability; which is central in this discussion. APAC learnability is a property of a hypothesis class $\mathcal{H}$ and in layman's terms it means that there exists an algorithm $\mathcal{A}$ so that for any distribution $\mathcal{D}$ the generalization error is within an $\epsilon$ range of $\min_{h' \in \mathcal{H}} L_D(h')$ with probability $1-\delta$. Basically, it means that a hypothesis class is learnable if it is guaranteed that increasing the number of training examples we minimize the generalization error , with a certain probability and that if we want to further minimize it it is enough to obtain more training samples.

It can be proven the non existence of a *universal learner*. This means that if we consider the hypothesis class $\mathcal{H}$ to be the set of all prediction rules that have a mapping from $\mathcal{X}$ to $\mathcal{Y}$ it is not APAC learnable. In practical terms, it means that there is no learner that performs well in every task and that introducing inductive bias is a necessity. Unfortunately, as discussed in the previous section, introducing a too restrictive hypothesis class $\mathcal{H}$ leads to a large *approximation error* $\epsilon_a$, so it is necessary to strike a balance in the model between flexibility to not *underfit* data and restrictiveness to not incur into *overfitting*. This trade-off is frequently done when deciding on the model to use for solving a particular task and it is extremely relevant in the Machine Learning field.

## 2.2 Linear Models

In this section we will discuss linear models, as they are a cornerstone of Machine Learning Algorithms and exemplify the concepts that were discussed in Section 2.1. Besides that they also offer profound insights on how a problem is modeled when selecting a model. Furthermore, they offer a unique perspective into concepts of Bayesian probability [21] when viewed as a Maximum Likelihood Estimator for some latent variables in a system.

In a regression problem modeled with a linear model, the assumption is that the target $y \in \mathbb{R}$ can be expressed as a linear combination of the features $\boldsymbol{x} \in \mathbb{R}^n$ through some weights $\boldsymbol{w} \in \mathbb{R}^n$, to be determined as shown in equation Equation (2.2) (we can assume that a constant feature is introduced to account for the bias term). Here, one can find how this statement is simply the imposition of a hypothesis class $\mathcal{H}_l$ of linear models. We introduce an inductive bias by saying that the relationship between the targets and the variables is assumed to be linear. Sometimes, this assumption might be too restrictive, which might lead to underfitting; thus meaning that approximation error for linear models in that particular task is high.

$$\hat{y} = \boldsymbol{w}^T \boldsymbol{x} + \epsilon \tag{2.2}$$

Notice we are assuming that the weights $\boldsymbol{w}$ exist and they are considered a latent property of the system as we do not observe them directly. Our aim is to infer them from the data that we collect from the underlying probability distribution $\mathcal{D}$. This training dataset is assumed to have some disturbance on the target $y$ , modeled as a random variable $\epsilon$, which is the reason why all samples with identical features do not necessarily have the same target response. Once again, we can find here that this accounts for the error of the Bayes Predictor when viewed through the lenses of the statistical framework in Section 2.1. It means that our lack of complete knowledge of this system is modelled through this variable $\epsilon$ which encompasses that missing information. In a practical setting, considering measurement systems, that is the value of the disturbances on the measurement.

Typically one has more instances that variables, therefore it is necessary to solve an over-determined linear system. The standard solution for this problem is to use the method of Ordinary Least Squares, although other methods are also possible. This method attempts to find the weights $\boldsymbol{w}$ such that the sum of the residuals $e = \sum_{i=1}^{N} (y - \hat{y})^2$ is minimized (i.e. the empirical error is minimized). A visual representation of this process in a one dimensional

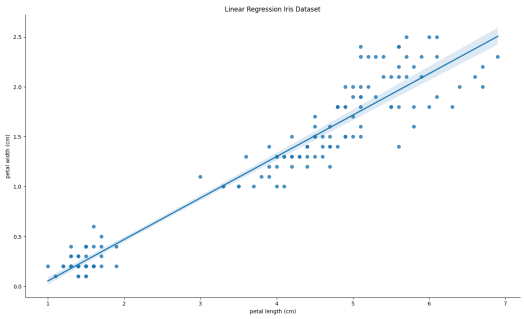



Figure 2.1: **Iris Linear Regression**: Very simple example of OLS in a classic ML dataset.

problem is shown in Figure 2.1. Visually, it appears we are selecting the line that *better fits* the data points. In this case, we are modeling the petal length and width in the classical iris dataset[22]. One can see that , altough the linear regression is good, it is not perfect for modeling the full relationship between them

### 2.2.1 Linear Regression as MLE

In this section we will discuss how a linear regression model can be viewed as a Maximum Likelihood Estimator of the weights $\boldsymbol{w}$. One can think of the weights as an unobserved characteristic under examination of which we aim to have an estimate after observing the dataset.

We must assume that the disturbing noise $\epsilon$ is a normal random variable (i.e. $\epsilon \sim \mathcal{N}(0, \sigma^2)$; independent and identically distributed across samplings. Under such assumption also the target variable $y$ is distributed as a normal random variable $y \sim \mathcal{N}(\boldsymbol{w}^T \boldsymbol{x}, \sigma^2)$. It is straightforward to prove under such conditions that the Maximum Likelihood Estimator for $\boldsymbol{w}$ is equivalent to the least squares estimator.

The proof comes from observing the likelihood of the training dataset as in Equation (2.3)

$$\mathcal{L}(\boldsymbol{w}, \sigma^2) = \prod_{i=1}^{n} \mathcal{P}(y_i | \boldsymbol{w}, x) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y_i - \boldsymbol{w}^T \boldsymbol{x})}{2\sigma^2}} \tag{2.3}$$

The proof is completed by taking the log-likelihood of Equation (2.3) and maximizing it.

### 2.2.2 Regularization

As discussed in Section 2.1 the trade-off between flexibility and adequate generalization of Machine Learning Algorithms is a recurrent topic. In the context of linear models different techniques have been proposed to improve the generalization ability of models. The intuition behind the techniques shown in the following its that , following Occam razor, the simplest model is the better. The techniques used with linear models are Ridge and Lasso regularization[23]; which penalize the unnecessary usage of features thus penalizing large weights

Ridge regularization, formally $L2$ regularization, penalizes coefficients in the regression that are different than $0$, thus forcing the model to *allocate* adequate weights and remove them from variables that provide no information gain. The optimization problem that is then solved is shown in Equation (2.4), instead of the minimization of the single squared residuals we also minimize the norm of the weights of the linear model.

$$Loss_{Ridge} = \sum_{i=1}^{n} ||y_i - \hat{y}_i||_2^2 + \alpha||\boldsymbol{w}||_2^2 \tag{2.4}$$

A more *aggressive* kind of regularization is *Lasso regularization*. Lasso regularization, also known as $L1$ regularization can also reduce some coefficients to zero. This effectively eliminates the least important features, performing feature selection. The optimization problem for Lasso is shown in the equation Equation (2.5)

$$Loss_{Lasso} = \sum_{i=1}^{n} ||y_i - \hat{y}_i||_2^2 + \alpha||\boldsymbol{w}||_1 \tag{2.5}$$

Notice that both in Equation (2.5) and Equation (2.4) the optimization problem is controlled by a hyper-parameter $\alpha$. This is yet another key concept that often emerges in developing Machine Learning Algorithm. The model's performance is often significantly influenced by the value of such hyper-parameters. Consequently, considering them is very important for ensuring optimal model selection.

## 2.3 Probabilistic Graphical Models

In this section we will discuss Probabilistic Graphical Models, which are particularly relevant for modeling the relationships in a system. Relationships

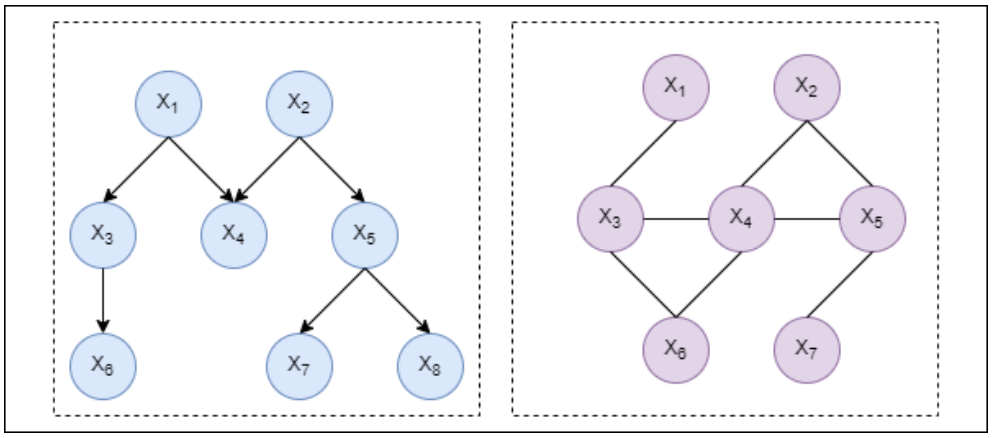



Figure 2.2: **Probabilistic Graphical Models**: This diagram shows two representative kinds of PGMs. To the right a Markov Random Field is represented , to the left a Bayes Network.

for several variables $x_i$ are modeled through the use of a graph structure $\mathcal{G}$[24].

PGMs are instrumental in expressing the structure of a probability distribution $\mathcal{D}$, and subsequently making inferences from it. One type of PGM are Bayesian Networks, which are represented thorough DAGs that model the conditional relationships between variables. The implied structure of the probability distribution in a Bayesian Network is the product of the conditional probabilities of each node given its parents in the graph; defined as $Pa(x_i)$ for node $x_i$. When a node does not have any parents, it simply represented by its marginal distribution, as shown in equation 2.6.

$$\mathcal{P}(x_1, \ldots, x_n) = \prod_{i=1}^{n} \mathcal{P}(x_i | Pa(x_i)) \tag{2.6}$$

Another type of PGM are Markov Random Fields (MRF), which uses an undirected graph to represent relationships. This structure implies that the probability distribution can be expressed as the product of factors $\phi_i$, each of which is associated with a unique clique in the graph. A clique is a subset of nodes in the graph such that all of them have connections between each other. The number of factors equals the number of cliques in the graph, as indicated in equation 2.7.

$$\mathcal{P}(x_1, \ldots, x_n) = \frac{1}{Z} \prod_{i=1}^{M} \phi(\{x_j | x_j \in clique_i\}) \tag{2.7}$$

An example for each type of network is shown in Figure 2.2. The

factorization of the Bayes Network can be written as shown in Equation (2.8) where we made use of the graph for a factorization. On the other hand a factorization for the Markov random field is shown in Equation (2.9).

$$\mathcal{P}(x_1, x_2, x_3, x_4, x_5, x_6) = \mathcal{P}(x_1)\mathcal{P}(x_2)\mathcal{P}(x_3|x_1)\mathcal{P}(x_6|x_3) \\ \mathcal{P}(x_4|x_1, x_2)\mathcal{P}(x_5|x_2)\mathcal{P}(x_7|x_5)\mathcal{P}(x_8|x_5) \quad (2.8)$$

$$\mathcal{P}(x_1, x_2, x_3, x_4, x_5, x_6) = \frac{1}{Z}\phi(x_1, x_3)\phi(x_3, x_4, x_6)\phi(x_2, x_4, x_5) \\ \phi(x_5, x_7) \quad (2.9)$$

## 2.4 Deep Learning and AutoEncoders

This section is devoted to give the reader context on the principles behind deep learning, which is the technology behind most of recent years advanced in the field of Machine Learning. Subsequently, it provides an explanation of the AutoEncoders

### Neural Networks

Neural networks are the building block of any deep learning technology. The reason of their great success is the fact that they have been proven to be universal function learners [25]. This, in practice, means that deep networks can be used for representing arbitrary functions of which the explicit form is not known but for which data is available.

As their name indicates, neural networks are an inter-connection of some neurons. The computation performed in each of those neurons is rather simple can be expressed as $o = f(\boldsymbol{w}\boldsymbol{x} + b)$. As it is possible to see it simply is performing a linear combination of the inputs $x \in \mathbb{R}^D$ through the weights $\boldsymbol{w} \in \mathbb{R}^D$ and the bias term $b \in \mathbb{R}$ with a subsequent application of a non-linear function$f : \mathbb{R} \to \mathbb{R}$. Several alternatives for a linear function have been proposed [26], each with its own advantages and disadvantages. A non-exhaustive list is presented in Table 2.1, being the most common in recent works the Rectified Linear activation function.

The structure described above represents a single neuron. However, several units can be stacked together producing a multi-dimensional output

| Name | Expression |
|---|---|
| Sigmoid | $f(x) = \rho(x) = \frac{e^x}{1+e^x}$ |
| Hyperbolic Tangent | $f(x) = tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$ |
| Rectified Linear | $f(x) = ReLu(x) = x\mathbb{1}_{[0,\infty]}$ |
| Leaky ReLu | $f(x) = x\mathbb{1}_{[0,\infty]} + ax\mathbb{1}_{[}-\infty, 0]$ |

Table 2.1: **Activation Functions**: This table presents a list of the different activation function that can be used in neural networks. This list is non-exhaustive.

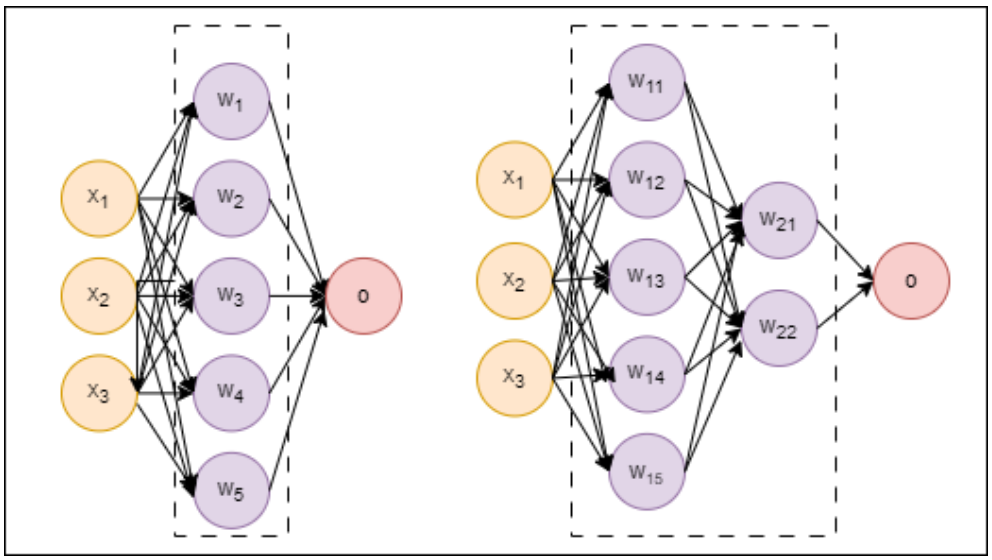



Figure 2.3: **Neural networks**: This figures show two simple examples of neural networks. To the right a neural network with 1 layer and 5 neurons, to the left a neural network with 2 layers of 5 and 2 neurons respectively.

$\boldsymbol{O} \in \mathbb{R}^O$. This can be elegantly expressed with matrix notation as $\boldsymbol{O} = f(\boldsymbol{W}\boldsymbol{x}+\boldsymbol{b})$, where the single weights of each neuron were stacked together in the weight matrix $\boldsymbol{W} \in \mathbb{R}^{O\times D}$, as well as the bias terms in $\boldsymbol{b} \in \mathbb{R}^O$. Here the function $f$ is a shortcut for stating the element-wise application of the original operation. This structure is a single layer neural network. In the same way multiple layers can be stacked for producing the final output by processing on the output of the previous layer, thus creating the so called *deep networks*. A diagram representing two simple neural networks is shown in Figure 2.3.

An interesting observation is that the internal representation of the input in between layers can be thought as a representation of some *latent* representation of the input that the network learns.

In essence a neural network is fully determined by its architecture and the value of the weights $\boldsymbol{W}$ present in different layers. Those weigths can be tuned to produce the expected output from the existing data through a process called *backpropagation*[27]. This is to iteratively process data, compute a loss function $L$ that represents the distance from the desired output and the actual output given by the network and then adjust the weights in such a way to produce results closer to the desired output by using derivatives. Although it is a rather complex process the *automatic differentiation* process has been effectively defined in libraries like PyTorch or TensorFlow [28][29]. Those libraries allow for efficient neural network definition and training.

Lastly, neural networks, as an extremely general *hypothesis class*, might suffer from the problems explained in Section 2.1, therefore various architectures have been devised for introducing the inductive bias required to effectively *learn* on different kinds of tasks. This includes, but is not limited to, architectures for processing images or sequential data [18][30][20]. Thus, leading to the constant expansion of application of deep learning across different domains.

### AutoEncoders

In the following we will discuss of a particular type of architecture that is relevant for learning patterns from extensive data collections. The architecture in question is the AutoEncoder. This is a type of network whose only goal is , in essence, to learn to encode and reconstruct the input from that encoding.

In broad terms the goal of an AutoEncoder is to be able to learn a *latent representation* of the data that is useful for conducting other tasks, or useful by itself for other unsupervised tasks. Having this goal in mind the AutoEncoders solve a related task as explained in the following. The architecture consists of an encoder part, which processes the input, thus generating a latent representation of the original data and an encoder that makes use if this latent representation for reconstructing the input. The intuition behind it is that if the model's latent representation for the input is enough to reconstruct the input itself, then it must contain valuable information about it to solve other downstream tasks. A diagram showing this basic architecture is shown in Figure 2.4.

Several variants of the AutoEncoder have been proposed [31] both in terms of the components used for encoder and decoder, training strategy as well as latent representation. Some of the different tasks typically solved by AutoEncoders are listed below:

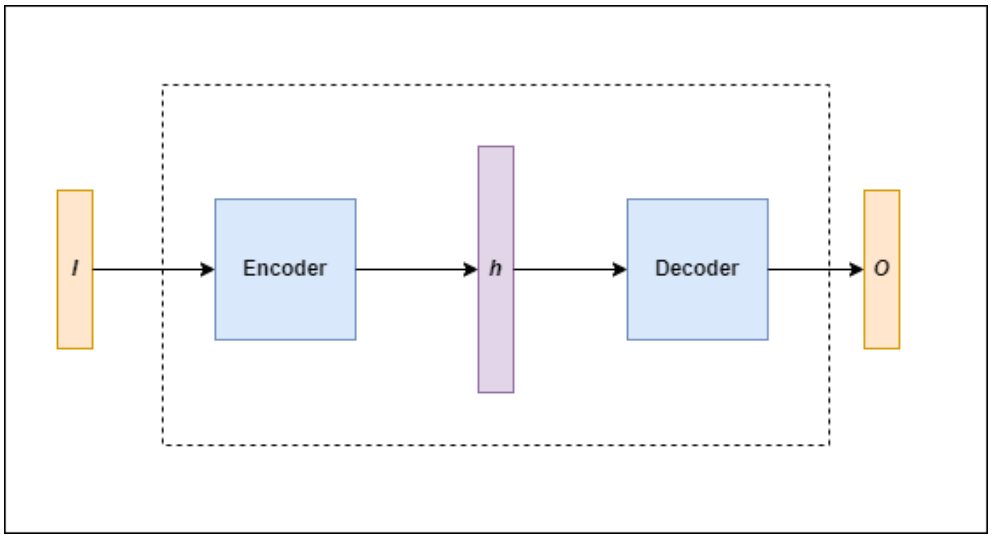



Figure 2.4: **Base AutoEncoder Architecture**: This diagram shows the basic AutoEncoder architecture. The AutoEncoder processes the input $I$ making use of the Encoder and producing the latent representation of the input $h$. Subsequently, the decoder attempts to reconstruct the input, given the latent representation $h$. The goal is to obtain an encoder that extracts a meaningful representation $h$ from the input.

- **Identity AutoEncoder**: This is the base task in which the goal of the AutoEncoder is to reconstruct the input from a latent representation of it. Notice that in this case, if the dimension of the latent representation is larger than the original input dimension the AutoEncoder might not be learning any useful pattern but just an identity mapping between latent representation and the expected output. Whenever the dimension of the hidden representation is lower than the dimension of the input the process can be interpreted as a compression and subsequent decompression of the data.

- **Denoising AutoEncoder**: In this task the original input is corrupted with some noise (which can be of various kinds) and the task of the AutoEncoder is to recover the original data. This prevents the AutoEncoder from learning a trivial representation of the data.

- **Sparse AutoEncoder**: Another variant to overcome the trivial case of an identity learning is to force the latent representations of data to be sparse. This is that it must contain zeros. This is effectively achieved by penalizing non-zero values in the latent representation during training.

Regarding the components used to implement the *Encoder* and the *Decoder* it is heavily influenced by the type of data under examination. When dealing with computer vision typically the implementation is done with CNNs while for Natural Language Processing and time series data usually implementations make use of RNNs of different kinds.

## 2.5 Ensemble Methods

In this section we will briefly explore the theory behind ensemble methods and the justification to make use of them along with their advantages. The ensemble models provide a strategy to *play* with the trade off between a very accurate model and a model whose predictions are robust (i.e. have low variance).

An ensemble method is simply a set of predictors , that are in some sense different, whose predictions are combined to obtain the final prediction of the entire model. Calling $h_j : \mathbb{R}^D \to \mathbb{R}$ the prediction rule established by the predictor $j$ from the features to the target, in a set of $M$ predictors, we have that the final prediction $h : \mathbb{R}^D \to \mathbb{R}$ is simply the average of every single prediction: $h(\boldsymbol{x}) = \sum_{j=1}^{M} h_j(\boldsymbol{x})$.

The expected error of $h$ can be expressed as a bias covariance decomposition [32] as in Equation (2.10). Here it is shown the expected error of the ensemble model is the composition of several characteristics of the base models.

$$\begin{aligned}
\mathbb{E}[h(\boldsymbol{x}) - y]^2 &= bias^2 + \frac{1}{M} var^2 + (1 - \frac{1}{M}) covar \\
bias &= \frac{1}{M} \sum_{j=1}^{M} (\mathbb{E}[h_j] - y) \\
var &= \frac{1}{M} \sum_{j=1}^{M} \mathbb{E}[h_j - \mathbb{E}[h_j]]^2 \\
covar &= \frac{1}{M(M-1)} \sum_{i=1}^{M} \sum_{j=1 \wedge j \neq i}^{M} E[h_i - \mathbb{E}[h_i]](h_j - \mathbb{E}[h_j])
\end{aligned} \tag{2.10}$$

The two key points that can be drawn from this decomposition are that as the number of models increases the variance of the single model's influence decreases but the covariance (i.e. correlation between model predictions)

starts to have a vital role. This observation sets a theoretical motivation to the intuition that adding several models whose predictions are the same is not useful. The key point is that if we design the system such that individual model are not much correlated then the performance of the overall model is expected to increase [32].

Different approaches have been proposed to achieve the goal of having independent predictors. Some particularly relevant methods, shown in [32] are listed in the following:

- Data Diversity: Train a different model in a different dataset. This can be obtained by effectively segmenting the original dataset into several ones, often referred as *bagging*.
- Model Architecture: Use models that have different architectures to generate the predictions.
- Divide and Conquer: It divides the original task into several sub-tasks.

Each of those methods finds diverse implementations both in regression and classification tasks, following the very natural intuition of the *wisdom of the crowd*.

## 2.6 Gas Turbines

In this section, we will present an overview of the operating principles of gas turbines, their various components, and the measurements that will be crucial for this work. Additionally, we will explore the applications of gas turbines and delve into considerations regarding their role in a sustainable future.

A gas turbine is sophisticated industrial apparatus used for converting chemical energy into mechanical or electrical power [2]. A diagram representing a gas turbine is displayed in Figure 2.7, highlighted within the box with a discontinuous contour. The three fundamental components of a gas turbine are: the compressor, the combustion chamber, and the turbine itself.

As a highly complex device gas turbines contain a wide variety of measurement systems that aim to monitor different physical quantities. Over the years, these measurements have proven invaluable for monitoring the health of the machine at different stages [33]. From both an operational and predictive maintenance perspective, these measurements are essential. Critical types of sensors used in gas turbines are pressure, vibration, flow and temperature sensors as well as control signals used such as the IGV [10].

## Compressor

It is the first stage of the gas turbine whose purpose is to supply the necessary gas with the required pressure and temperature for the subsequent stages of the turbine. This supply is crucial to sustain the reaction within the combustion chamber and to drive the turbine to generate the expected power [34].

Due to the inherently extreme conditions under which the compressor operates, constant monitoring is required. As shown in [34], failures at this stage can be both physically and economically devastating. As concluded in that investigation , it is essential to implement diagnostic systems that enable the operator to observe such anomalies and prevent such events. Available sensors at this stage of the turbine include , but are not limited to, pressure and temperature monitoring at different stages of the compressor itself.

## Combustion Chamber

In the combustion chamber, highly pressurized air is combined with fuel to sustain a reaction that effectively releases the fuel's chemical energy. Modern gas turbines are carefully designed to make this process as stable as possible; thereby reducing the need for stabilizing fuel and reducing $NO_x$ emissions[35]. However, due to the dynamic nature of this process and the highly volatile reaction under course, monitoring is crucial to prevent the emergence of *thermo-acoustic* instabilities[35]. These instabilities can excite certain resonance frequencies of the gas turbine, leading to faster deterioration of mechanical parts, higher $NO_x$ emissions and in the worse case scenario a catastrophic failure.

### Pulsations

In this master thesis, a significant portion of the study will be devoted to *pulsations* measurement system. This measurement was chosen due to its critical role in monitoring the health of a gas turbine and its connection with the monitoring of the above mentioned resonance frequencies. The so-called *pulsations* in the context of a gas turbine are a derived measurement from the direct measure of the pressure in the combustion chamber. A pulsation is defined as the power allocated within a specific frequency band of such signal $x$, as shown in Equation (2.11).

$$P(f_l, f_h) = \int_{f_l}^{f_h} |\mathcal{F}(x(t))|^2 dt \tag{2.11}$$

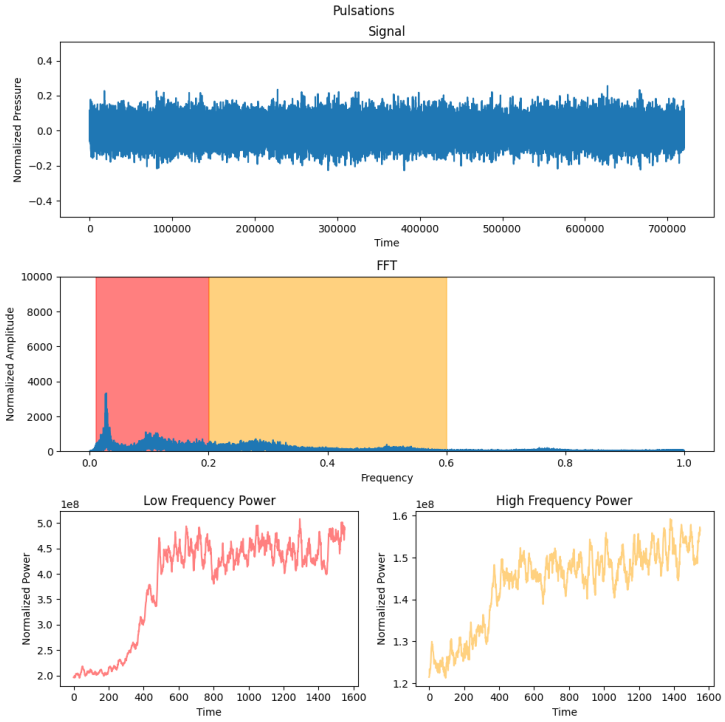



Figure 2.5: **Pulsation Sensor**: This figure shows the computation of *low frequency pulsation* and *high frequency pulsation* sensors from a signal sampled at a high frequency $f_s$. The FFT of the signal is computed in a time window $T$ and the power allocated in interested frequency bands is computed.

In practical applications, one can imagine to calculate the FFT over a window of $T$ seconds. Then, the power allocated within the desired frequency bands of interest is then summed. In Figure 2.5, an example of such a process implementing a sensor is depicted. From the provided signal sampled at a high-frequency, the FFT is computed and the RMS value of two separate frequency bands are monitored over time. The resulting derived measurements are of much lower frequency (in the order of magnitude of Hz). It is widely recognized that maintaining control over these frequencies is crucial, as they have the potential to trigger resonances within the gas turbine that could damage mechanical parts. The Figure 2.6 shows how a different behaviour of the original signal is encoded into the *pulsation sensors*; in that instance the power allocated in low frequencies increases while the power allocated in high frequencies decreases.

The practical application of the measurement chain for pulsation

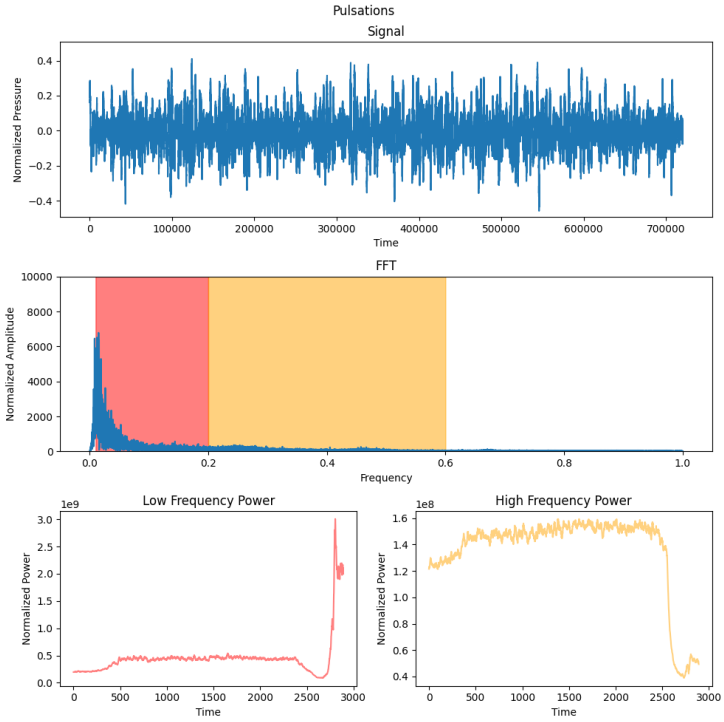



Figure 2.6: **Pulsation Sensor**: This figure shows how the properties of the underlying signal are encoded into the *pulsation sensors*.

measurement is different from the description provided here, but it is conceptually identical. These pulsation measurements, therefore, form an integral part of the overall monitoring system of the combustion chamber, ensuring the optimal functioning and longevity of the gas turbine.

## Turbine

The final stage of the gas turbine is the turbine itself, a critical component comprised of blades that rotate at high speeds, often exceeding 6000 rpm. This rapid rotation is responsible for the generation of mechanical or electrical power, a fundamental output of the turbine's operation.

Designed to withstand extreme conditions, the turbine blades are exposed to high temperatures and pressures from the combustion chamber. This harsh environment necessitates the use of specialized materials and precise engineering designs to ensure the blades' durability and efficiency.

## Operation of gas turbine

The operation mode of a gas turbine largely depends on the application for which it is used and on this it also changes how a normal operation looks like from the sensors point of view. As discussed in [8] gas turbine find extensive applications in very different scenarions. A turbine that is used as power generator will have a very different behaviour from a gas turbine that is used as a mechanical driver.

Another critical factor to consider when analyzing different behaviours is the type of fuel used for the combustion reaction [35].

## Sustainability Considerations

The present times necessitate thoughtful considerations regarding the sustainability of the technologies we develop. Gas turbines are posed as a cornerstone for aiding in the transition towards more sustainable energy sources. A key consideration is the high efficiencies that can be achieved when generating energy with them, compared with traditional sources such as coal . Additionally, they are desirable both for the low emissions achieved when combusting natural gas ($CH_4$) as well as for the current studies on how the combustion can be done with ammonia $NH_3$ or hydrogen $H_2$ thereby leading to minimal emission of greenhouse gases. Moreover, their synergy with clean energy sources makes them suitable candidates to complement and enhance these sources during periods of unavailability. Those aspects will be discussed further in the following.

An extensive set of cycles have been proposed to incorporate gas turbines and boost the efficiency with which energy is produced [4]. Simply recovering the heat lost from the combustion through the exhaust is enough to dramatically increase the efficiency of a gas turbine to nearly double the efficiency a turbine would have by itself. This allows such systems to reach efficiencies up to $58\%$.

A simple cycle in which the exhaust heat is used is shown in Figure 2.7. The heat is used in a secondary steam turbine, to further produce energy. The efficiency achieved through this relatively simple construction surpasses that which could be attained by traditional oil and coal-fired power plants [4]. Incredibly, the construction of such systems is relatively easy and costly thus can be easily implemented in practice. Notice that this is only one of many possible constructions.

Looking ahead, the use of *cleaner* fuels like hydrogen or ammonia in gas turbines has been anticipated for a long time[5]. Utilizing hydrogen as

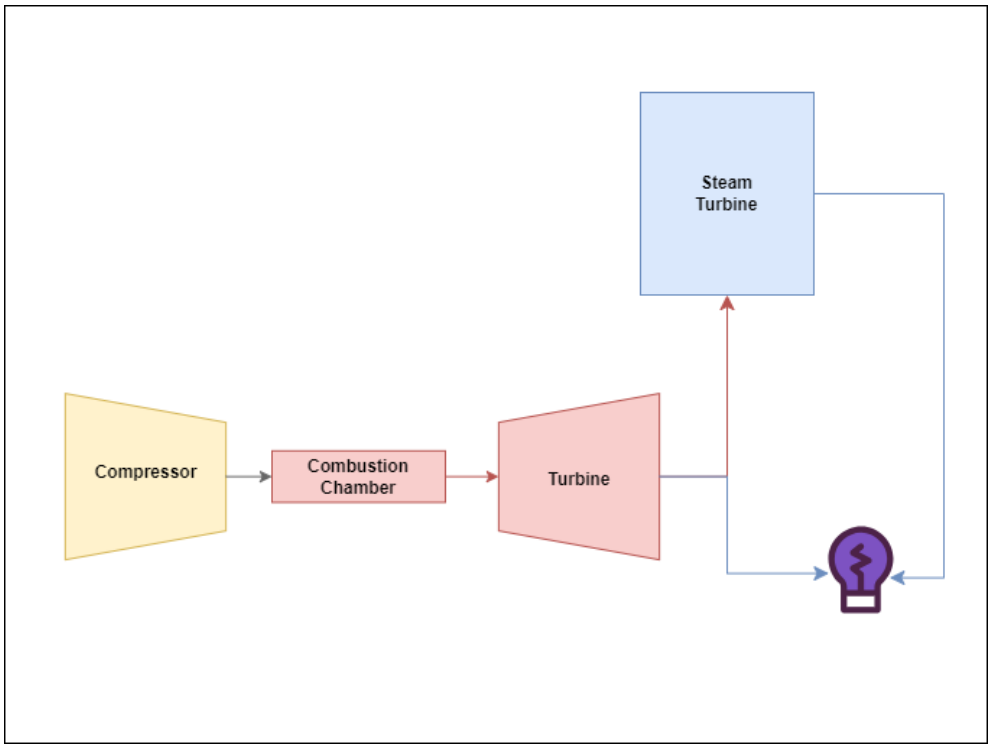



Figure 2.7: **Simple Cycle**: This image depicts a simple cycle of a gas turbine combined with a steam turbine. The design cleverly makes use of the heat lost through the exhaust, augmenting the efficiency of system.

a fuel could significantly reduce greenhouse gas emissions [6]. However, the full-scale implementation of hydrogen as a fuel comes with its own set of challenges.

Practically, hydrogen presents certain difficulties due to its propensity to leak and its high reactivity, which makes it hard to transport and store [36]. This necessitates meticulous handling and stringent safety measures to prevent accidents. Furthermore the combustion stability, essential for a *healthy* machine [35] is also affected. One of the current alternatives it to use a mixture of hydrogen with other fuels [6] or alternative fuels such as ammonia [36], although every choice comes with their respective challenges and trade-off.

Furthermore, there remains an open question regarding the consistent and economically viable production of hydrogen, which requires energy. A potential solution, mentioned in [6], is to integrate gas turbines into a system that also includes a solar energy plant. This combined system could generate the necessary hydrogen for energy production during periods when solar power

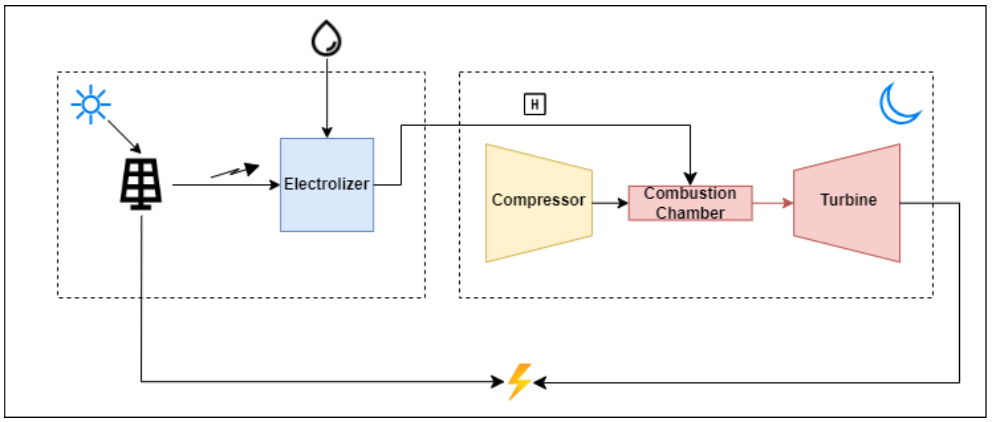



Figure 2.8: **Hydrogen and Solar Energy with Gas Turbines**: This diagram shows a system in which a gas turbine is integrated with a solar energy system to generate sustainable and reliable energy, as proposed in [6]. The solar energy is used to power the electrolizer as well as to generate power during daytime. Subsequently, the stored hydrogen is utilized by the gas turbine to generate energy during periods when solar energy is unavailable, such as during nighttime.

is not available [37].

This approach exemplifies the concept of a hybrid energy system, where different types of energy sources and technologies are combined to compensate for each other's limitations. In this case, the gas turbine can provide a reliable power source when solar energy is insufficient, while the solar plant can produce hydrogen during peak sunlight hours, creating a sustainable fuel source for the gas turbine. A simplified schema of this operating system is shown in Figure 2.8.

## 2.7 Related work

In this section we will examine several works done in industry and academy that tackle closely related tasks to create a software sensor in gas turbines. First, we will provide a review of all previous work done at Siemens Energy followed by examples coming across different industries.

### 2.7.1 Previous Work at Siemens Energy

This subsection focuses on the application of Machine Learning Algorithms in gas turbine studies at Siemens Energy, a large enterprise with numerous

efforts contributing to the body of work over several years. This includes numerous master's theses and research papers done in collaboration with academic institutions.

The concept of a *software sensor* has been in existence for over a decade, as discussed in [12]. In this paper the authors use of Siemens' simulation software to emulate the behaviour of a gas turbine under standard operating conditions and they introduce artificial sensor reading errors for later detection and correction. Despite being a study dated over a decade ago the ideas presented in this paper continue to be innovative, offering an alternative to the popular AutoEncoder solution, as discussed in various other works[38].

The authors propose the use of supervised learning to identify sensor faults, a method possible due to the data generation method used. The suggested classification network is a MLP with outputs designed to indicate either an upper or lower error in sensor $A$ or correct functioning of all sensors. This arrangement results in $2n + 1$ outputs, representing two possible errors for each sensor and an additional output for the scenario where all sensors are functioning correctly. It's important to note that this approach is based on the implicit assumption that sensor faults are infrequent, hence the likelihood of simultaneous failures is relatively low, since the system does not model such situation.

Once a sensor fault has been individuated, a regression task is employed to estimate the correct reading for the failing sensor. For this task the authors also employ an MLP. The results for this regression task are exceptionally high, with all software measurements having predictions with less than $0.5\%$ mean error. Sensors used in the study encompass several different physical quantities such as pressures, temperatures and certain control signals like Inlet Guide Vane. However, it is important to bear in mind that these exceptional results are based on data derived from a turbine simulation. Nonetheless it sets an interesting benchmark for comparison in the current work.

This master's thesis can be viewed as an extension of the above-mentioned study, incorporating and building upon the ideas presented in the original research. However, this work introduces an additional level of complexity by utilizing data from actual turbines. As the authors of the paper suggest one key aspect to consider in future research is the impact of component degradation on this system. This aspect is explored in this master's thesis and is also object of study, in a slightly different way, in the study [2].

The work conducted at [2] can also be viewed as a continuation of the previous work. It employs, opposite to [12], a simple AutoEncoder architecture for aiding in the monitoring of the machine. The result of this

study is that the AutoEncoder reconstruction error increases over time as the machine ages, as expected due to the change in the components. The authors of this study concentrate on the turbine's compressor and propose to use the reconstruction error as a measurement to quantify the level of machine degradation. A significant challenge to effectively use this measurement is to understand *when* the reconstruction error has become excessively large. Seen through the lenses of the statistical framework for ML presented in 2.1 the observed drift in performance is attributed to an underlying change in the distribution $\mathcal{D}$ that generates samples, likely due to the physical change of turbine components. This paper verifies that MLP can be effectively be leveraged in gas turbines to model a real nonlinear system.

Another study that utilizes a similar AutoEncoder architecture for gas turbines was conducted during a master's thesis [39]. This work employed a more complex architecture, specifically an LSTM, with a larger number of sensors, totaling 69. However, one limitation of this master's thesis is that the input was neither masked nor compressed, which could potentially lead to data leakage and impact the validity of the results. therefore *leaking* data and probably invalidating results. Despite it, this master's thesis shows the engineering of a data pipeline for model training, showcasing its vital role in this studies.

Yet another significant work conducted at Siemens Energy is the master's thesis [40], which focused on detecting the presence or absence of a flame within a gas turbine. The author of this study demonstrated the effectiveness of relatively straightforward methods in accomplishing this task, such as a Naive Bayes Classifiers, Support Vector Machine, etc. A detailed correlation analysis of various signals was also carried out, mirroring the approach in [2]. The analysis revealed that some relationships between signals exhibit a high degree of linearity. As we will see throughout this work this is a very important observation. However, despite the promising results achieved by all the tested methods, the critical nature of flame detection in a combustor makes the replacement of optical sensors unlikely. Indeed, the high accuracies obtained in this study are attributed to the use of data from the gas turbine's full operation, not just the critical parts where a flame can effectively extinguish. Instances where a flame disappears when a turbine is operating at full load are so rare that training a model to accurately predict such events is an extremely challenging task.

The final related work conducted within Siemens Energy showcases a comprehensive analysis of the different kind of anomalies that emerge in different measurement systems of a gas turbine [10], also conducted within

the same department that this master thesis was done. This work shows a comprehensive analysis on the usage of more traditional techniques such as statistical charts to detect anomalies of several kinds: sequential, contextual , behavioural.

### 2.7.2 Digital Twin for Predictive Maintenance

As a completely different approach, yet closely related, there have been successful attempts to build a model-based digital twin of a gas turbine. This is a mathematical model that closely reflects the state of the gas turbine and is tuned for a specific turbine. The principal goal of such a system is to be able to predict values on the turbine, given the control signals [41]. This avenue is another potential development in gas turbine predictive maintenance. Opposed to the previous shown data-driven methods a digital twin relies on the physical knowledge of the machine. The main advantage of them over data driven models is that opposed to black box models they are fully explainable.

This approach is so widely established that several academic publications have been done in this regard. In the works of [42][11] a digital twin solution is explored to model a gas turbine and provide sensor faults readings. In this work it is possible to see that, despite the challenging task of modeling a gas turbine, their model is able to do it consistently and it is proven to be an effective strategy for fault detection. However, as the authors acknowledge, it requires a full knowledge of the components of the gas turbine, which is often extremely hard to achieve.

Furthermore, the digital twin is the main strategy employed by General Electric, one of the main Siemens Energy competitors, in their Predix system [41].

This system can be further elaborated by making use of a time aware model. For instance, the usage of Kalman Filters [41] is popular for modeling the time evolution of the several signals. There exist several variants, conceptually similar, using different kinds of filtering.

### 2.7.3 AutoEncoders in Predictive Maintenance

Previous work for detecting anomalies in industrial settings largely focus on the use of AutoEncoders. Due to their nature, by which they try to model the *normal* operation of a particular system they are particularly well suited for detecting anomalies. The main principle that groups all of them is the idea of using an autoencoder to learn the distribution $\mathcal{D}$ of a system and then apply

it on new data. If the data is still coming from the same distribution (i.e. the normal distribution) then the system is operating under normal conditions. If the AutoEncoder suddenly makes errors then it means that , since the AutoEncoder itself did not change, the system must have changed and therefore an abnormal point is detected.

The wide variety of AutoEncoder architectures make the literature around them extensive. An overview of several studies that employ them for fault detection systems is given in [38][31]. In the context of time series anomaly detection, which is closely related with the scope of this master thesis, it is often encountered the usage of Recurrent Neural Networks as encoder and decoder architecture [43][44], often being one among an LSTMs or GRUs to improve *memory* of the network.

One interest strategy presented in [43] is the fact that an different AutoEncoders are trained to learn the patterns present in different types of fault. Thereby, allowing for fault identification by selecting the fault corresponding to the model with the lowest error. Unfortunately, it is not always possible to follow this strategy as data for failure cases is not available. For instance, in the case of gas turbines instances of failures are of so many kinds and infrequent that employing this strategy is not useful and virtually impossible.

As shown by Davari et al. [44] the AutoEncoder paradigm can be used to detect anomalies in complex systems with extremely large datasets. In this work the authors make use of a LSTM based AutoEncoder to detect abnormal points. The simple rule for doing this is to examine the reconstruction error (RMSE) done in the training dataset through a simple boxplot use it is a baseline to detect anomalies during inference. However, this research is also aided by a dataset with recorded failures that, as mentioned earlier, is not always possible to obtain.

Other strategies focus on the latent variables that are generated by the encoder [45].

### 2.7.4 Performance based Predictive Maintenance

Another approach for adopting a prediction maintenance scheme in a gas turbine is to monitor its performance and , based on it, isolate the source of the fault [41]. The concept is to perform a so-called *Gas Path Analysis*. It is expected that deterioration of parts of the gas turbine will lead to a reduced value of several health parameters such as efficiency of the compressor or flow capacity. Based on the specific case it might be possible to further isolate the

fault. As noted by [41] this is not a bullet-proof approach and some faults, for instance in the combustion chamber, might be undetected. Furthermore, the real sensor data is not free from noise and deviation, thus leading to potential wrong conlusions about the turbine state.

### 2.7.5 Optimization-based predictive maintenance

Extensive studies have been done for improved ability to detect and isolate faults. On a completely different approach is the usage of genetic algorithms in combination with a model of the gas turbine and optimization procedures [41]. In this framework the goal is to minimize the residuals obtained between sensor readings obtained from the real turbine and a model of the turbine for which specific faults are inserted.

In a genetic algorithm the main idea is that a population of a certain number of individuals is generated, in this case gas turbines with specific faults, and only the most fitted survive and produce the next population generation. In our case by fitted individual we mean the model of the gas turbine that more closely reflects the condition of the gas turbine. Therefore, the *spring* of turbines with faults that resemble the one in the current machine will aid in the fault isolation. While this method might seem convoluted it has been proven to have success in previous academic research. One important remark on it is that it is also a model-based approach and it will rely on the accuracy of the given models of the gas turbine.

# Chapter 3

# Data Collection

This chapter provides an overview of the data collection process and the different datasets used for experimentation in this master's thesis. It also offers a discussion on the source of the data and quality issues encountered. All of the data collected during this master thesis focused on turbines of the kind SGT-700.

First a discussion about the two origins of data is given in Section 3.2 and Section 3.3. Following this, the chapter outlines the steps taken to ensure high-quality data for training the models and to create a system that can quickly scale and iterate. This is particularly crucial in an industrial environment, especially during experimentation, as it enables the rapid collection and cleaning of new datasets, potentially with a different set of features. Section 3.4 and Section 3.5 show, respectively, the data cleaning procedure and an insight into the software architecture for getting samples for models for different tasks.

Throughout this chapter, we will frequently refer to a concept termed as **turbine run**, which denotes the operation of a gas turbine from a state of no electrical power production (i.e. 0MW electric output) to its operation and eventual shutdown (i.e. return to 0MW electric output). The duration of this run can vary from a few hours to several days in some instances.

## 3.1 Relevance of Automatizing

Significant efforts were invested in this master thesis to automate the construction of the dataset as much as possible. It is widely understood that the development of Machine Learning Algorithms is not a linear process, but rather a cyclical one, particularly in an industrial context. As outlined in the Cross Industry Standard Process for Data Mining, several iterations of data

preparation and subsequent model evaluation are often necessary [13]. The efficient execution of these steps is only possible by eliminating the bottleneck that dataset construction often represents.

The outcomes from the methodologies outlined in this chapter are not only instrumental to this master's thesis, but they are also expected to be beneficial for future projects within the department on the same subject. This significance arises from their contribution to rapid prototype implementation and fast dataset creation. Specifically, the software developed for data fetching and cleaning holds significant value. This analysis also highlights potential areas for improvement, especially in relation to data quality issues.

## 3.2 Turbine Test Data

The first data source is derived from the testing facility at Siemens Energy. In this place, turbines undergo rigorous testing post-construction and prior to customer delivery. This testing aims to provide guarantees concerning the turbine's performance and emission rate, particularly regarding NOx emissions. The nature of this data is such that each turbine is tested briefly, typically for a total duration of one day, under varying load conditions tailored to customer needs. For a specific turbine, there are generally two runs: an initial verification run before the customer's arrival, and a subsequent run conducted with the customer. More runs could be necessary if any extra test is required. During this stage the gas turbine is *tuned* to meet customer requirements and fix potential issues that arise, which often entails changes between the two or more runs. These changes, which may range from modifications in the turbine's instrumentation to specific operational adjustments for enhanced performance, can be challenging or even impossible to quantify due to their diversity.

As it will be noted in Chapter 5, the data from this source presents significant limitations due to its restricted size and frequent violation of the assumption outlined in the statistical framework for ML (Section 2.1), which stipulates that data must come from the same underlying distribution when training models in one run and validating them in the next. Indeed, alterations in turbine components or its operation effectively constitute changes in some latent variable that remains unobserved and, as mentioned, challenging to quantify. Moreover, the scarcity of extensive data for a single turbine often leads to models overfitting and learning patterns exclusive to that particular run.

### 3.2.1 Data Issues

One of the significant challenges encountered when constructing a dataset for training and testing a model, using this specific data source, was the inconsistency in the set of valid sensors across different turbines; after the cleaning procedure. Valid sensors refer to those that recorded high-quality data during the turbine run (i.e. not constant values or NaN). This practically means that datasets for different turbines will have a different set of valid sensors.

Consider that the datasets that we are dealing with contain several turbine runs for a single turbine. One can think of each run as a matrix $\boldsymbol{X_i} \in \mathbb{R}^{T_i \times D_i}$, where $T_i$ is the length of the run and $D_i$ the number of sensors available. Subsequently, we define the set of sensors available in a given run $S_i = \{s_i | s_i \in \text{run}_i\}$. The issue at hand is that , in general, the equality $S_i = S_j,\ i \neq j$ is not satisfied.

A visual representation of this challenge is shown in Figure 3.1. From a *large* set of approximately $300$ sensors and over $100$ turbine runs, one can see the distribution of sensor missingness across the different runs. It can be seen that some sensors are missing in all runs while most of them are only missing in certain runs. A practical solution to this issue will be discussed in the following.

### 3.2.2 Parallel Fetching

The data collection system developed for this dataset is particularly efficient, thanks to the availability of an API for data fetching. This existing API, utilized by Siemens Energy's visualization tool within the Testing department, integrates seamlessly into the developed data pipeline, facilitating a fully automated process for dataset construction.

The current API permits querying a specific sensor in a particular turbine run, a design likely intended to ensure a smooth experience with a GUI. However, this setup is not necessary for data collection and can become a bottleneck if queried sequentially, resulting in hours of dataset querying times. Figure 3.2 illustrates a histogram detailing the API response times. While the responses are quick, sequential querying is impractical.

Fortunately, the modern C# concurrent programming paradigm can be employed to fetch API data in the most efficient manner possible [46]. This concurrent paradigm proves particularly effective when interacting with a network API, as multiple requests can be launched and managed concurrently, resulting in a dramatic improvement in application performance [47]. The pseudo-code for the simple querying algorithm is presented in Algorithm 1.

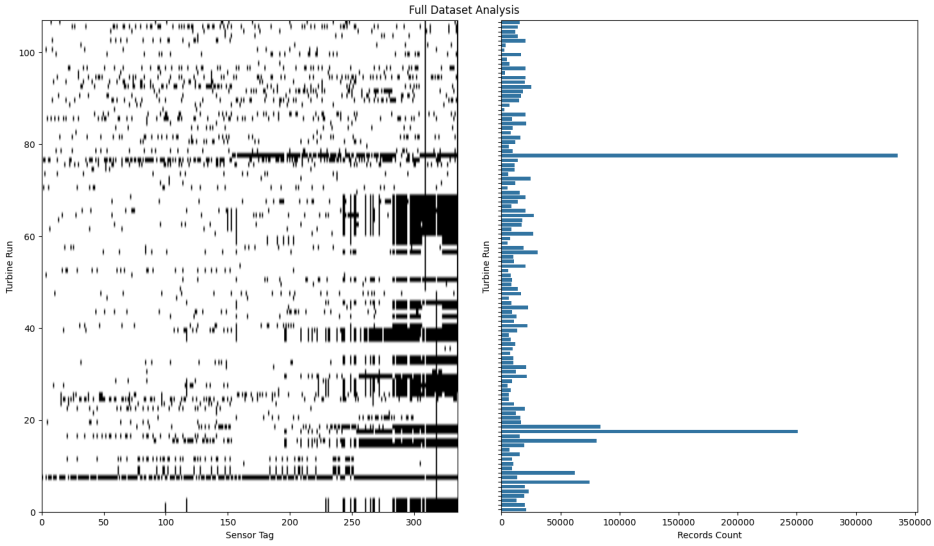



Figure 3.1: **Dataset Overview**: This figure shows the cleaned dataset obtained; composed of several turbine runs for different turbines of the same type. This dataset is heterogeneous due to differences in the testing equipment used in each run as well as potential failures of the system, etc ... The figure in the left shows (in black) missing sensors across different turbine runs and the figure in the right shows the record count for each turbine row.

The results obtained by the developed software during this master's thesis when querying a single turbine run are shown in Figure 3.3.

## 3.3 Fleet Dataset Analysis

In this section data collected from the fleet of gas turbines of Siemens Energy is described in depth. First, a discussion about the most prominent data quality issues with data coming from those sources is given along with the chosen strategy for imputation of missing data.

Then, a description of the datasets acquired from this data source is given. The discussion involves the motive behind the choice of such datasets and their characteristics.

### 3.3.1 Data Quality Issues

The amount of data collected from a gas turbine over time is simply massive. If we make an assumption of having 1000 sensors in a gas turbine , measured

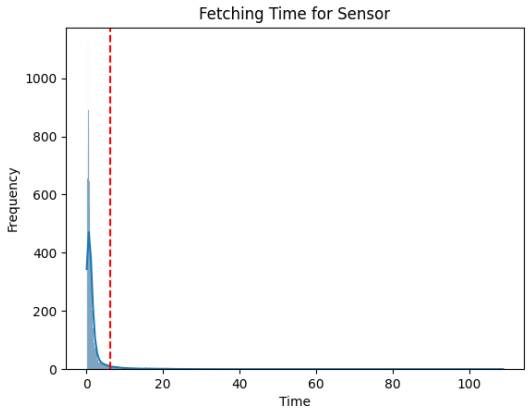



Figure 3.2: **Api Response**: This figure shows the distribution of response time by the API when querying for data for a single sensor. Notice that every query might contain an intrinsically different number of recordings. The 95th percentile is at $6.8$ seconds, while the mean querying time is at $1.6$ seconds.

with floating point numbers (i.e $4$ bytes) and sampled at $1Hz$ one obtains a quantity of around $345$ MB per day; if all the data points were collected during all the day . Although this is probably an overestimation, it underscores the issue that the amount of data generated by a gas turbine adds up quickly and forced engineers at Siemens Energy to use a more clever storage system that does not store unnecessary information.

One of the *compression* solutions is to store values only if the variation from the previously stored value surpasses a certain threshold. As a result the obtained datasets suffer from missing values and this must be addressed before developing Machine Learning Algorithms on top of them. Data coming from

**Algorithm 1** Task Scheduling Procedure

```
1:  procedure ScheduleTasks(turbine_list, api, outputFile)
2:      tasks ← list()
3:      for turbine in turbine_list do
4:          runs ← await api.getRuns(turbine)
5:          for run in runs do
6:              tags ← await api.getTags(run)
7:              for tag in tags do
8:                  task ← launchTaskFetching(turbine, run, tag, outputFile)
9:                  tasks.append(task)
10:     wait all tasks to complete
```

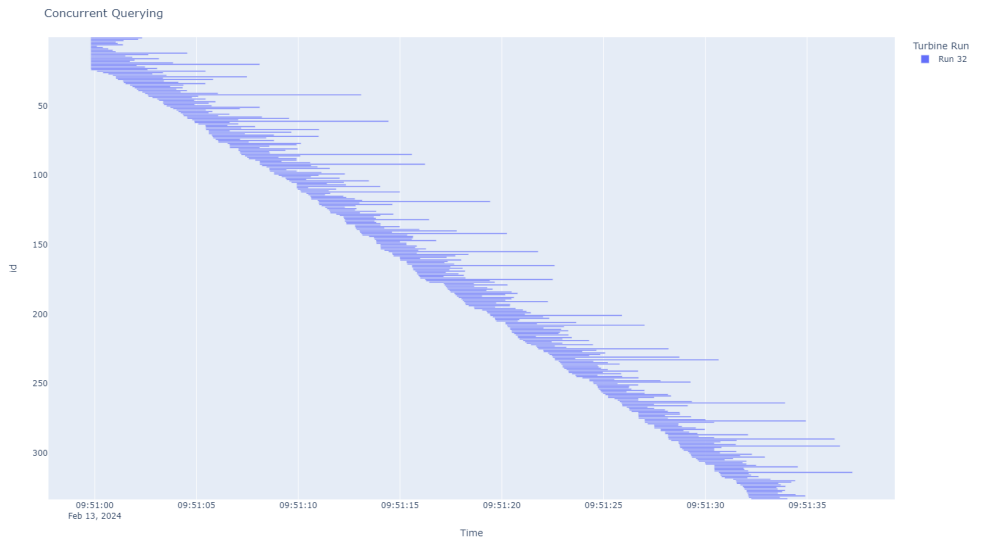



Figure 3.3: **Concurrent Querying**: This figure shows the result of concurrently querying the API for a single turbine run. Every row represents a different sensor which was queried and the length of the bar is the time it took for the request to be completed.

the turbine fleet is characterized by *asynchronous* sampling across signals and not unique sampling rate considering a single signal.

A visual representation of this phenomenon in a macro scale is shown in Figure 3.4. In this image the *missingness* of a particular sensor is represented as a black tile. The active load of the turbine (i.e., MW produced) is displayed alongside for reference. Most sensors appear to be missing during periods when the gas turbine is offline, but function during the operation of the gas turbine. However, it's worth noting that even during turbine operation, some sensors may occasionally fail to record data. This could be attributed to technical malfunctions in the electronic equipment or the data recording system.

Upon closer inspection, it is noticeable that even during operation time, the availability graph presents a grey tone. This is an optical effect resulting from the underlying structure. A zoomed-in image is displayed in Figure 3.5, where it can be observed that during a period of 500 seconds, sensors are sampled asynchronously and exhibit a non-constant missing rate over time.

During operation, assuming that sensors only omit readings when the underlying physical quantity (or rather, our sensor-based estimate of it) has not varied beyond a system-defined tolerance, it is reasonable to use linear

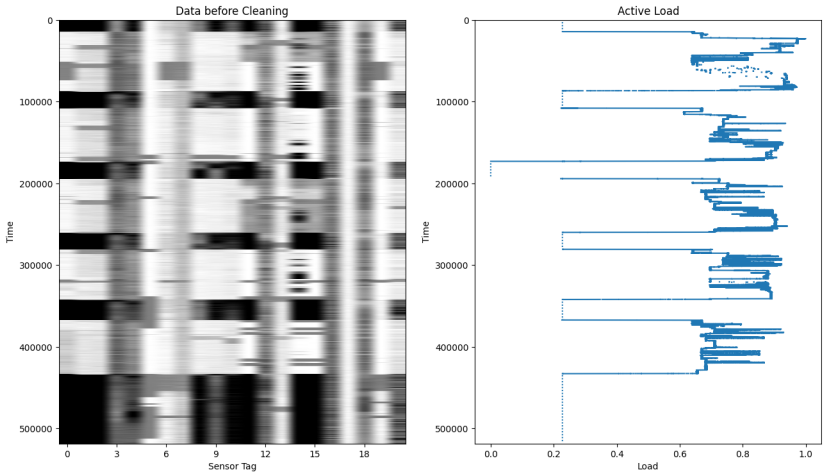



Figure 3.4: **Missing Values in Macro Scale**: In this figure it is possible to observe the pattern for missing values in a fleet gas turbine over an approximate span of one week. Interestingly, some sensors also become unresponsive, or *go blind*, even when the turbine is operational.

interpolation to impute missing data. However, in situations when the turbine is offline, or when there is a clear electrical malfunction during operation (as evidenced in Figure 3.4 where the load sensor itself is missing), it's advisable to preserve data integrity for the underlying algorithm by eliminating those time periods.

### 3.3.2 Datasets

Choice of the datasets was done on the base of the quality of the underlying data and the amount of operation time. The main requirement when choosing a dataset was to have enough data to simulate a practical scenario where an algorithm is trained during some weeks/months and subsequently tested on the following operation time of the gas turbine. Another important factor taken into consideration was the operation usage of the gas turbine. As discussed in Section 1.1 turbines find a wide range of usages which in turn have an impact on the characteristics that they exhibit. A requirement for training a Machine Learning Algorithms is to have a wide range of working points of the gas turbine from which the model can *learn*, therefore the choice was to pick gas turbines whose operation was not constantly steady.

Acquisition of this datasets was time consuming and not efficient as no

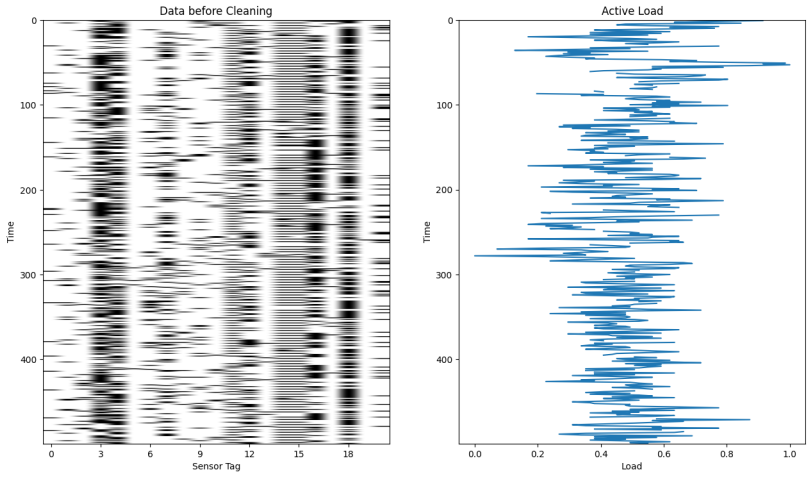



Figure 3.5: **Missing Values in Details**: This figure illustrates missing values in a fleet's gas turbine at a granular level, where the sampling from sensors is neither uniform nor synchronized. Black tiles represent the missing values of a particular sensor.

access to API was obtained. Therefore, only 3 datasets were considered. In Table 3.1 a concise summary of the obtained datasets is shown. The size of the dataset is listed after a data cleaning procedure takes place, this means after the issues presented in Section 3.3.1 are solved with the procedure that will be shown in the next chapter.

### Small - Very Dynamic Dataset

This dataset originates from a turbine utilized as a mechanical drive, resulting in extremely diverse runs characterized by sharp increases and decreases over time. However, as indicated in Table 3.1, this dataset is not extensive. It encompasses training data from a three-week period and validation data from the subsequent two weeks. In the final week (for testing), only one run is available.

This diversity is evident when comparing the load distribution in the datasets between this dataset, as seen in Figure 3.7, and Figure 3.9. The load distribution in the former dataset is extensively distributed, while in the latter, it is concentrated on a specific section of the possible operation points.

As will be discussed in Chapter 5, this dataset will play a crucial role in demonstrating the substantial benefits of employing a deep learning model

| Dataset Name | Characteristics | Size after cleaning | Turbine Model | Turbine Id |
|---|---|---|---|---|
| Small | Turbine with a very dynamic load change over time, used for mechanical drive. | 70.8MB | SGT-700 | 0 |
| Medium | Turbine with dynamic load change over time, used for grid stabilization. | 721MB | SGT-700 | 1 |
| Large | Turbine with dynamic load change over time, used for gird stabilization. | 13.5GB | SGT-700 | 1 |

Table 3.1: **Turbine Fleet Datasets**: This table summarizes the general characteristics of the datasets obtained from the turbine fleet of Siemens Energy for this master thesis. The medium and large datasets correspond to the same gas turbine in disjoint periods of time.

over simpler methods for this kind of application.

### Medium - Dynamic Dataset

The next-in size dataset comprises data coming from a turbine whose objective is to stabilize an electrical grid. As such, it contains dynamic data but not containing as sharp transitions as in the previous dataset. Due to its purpose it is operated intensively with runs that are typically of days, as can be seen in Figure 3.8.

Data in this dataset is collected for a month and a half and then tested in the following weeks. As noted in Table 3.1 this dataset can be considered of medium size. During this period, a total of approximately 1.5 million seconds of data is gathered, following the cleaning of the dataset. The distribution of data into training, validation, and testing can be observed in Figure 3.9.

### Large - Dynamic Dataset

The final dataset gathered from Siemens Energy's turbine fleet focuses on the same turbine as the previous dataset. However, it covers a more extensive, non overlapping time frame, collected between December 2016 and January 2021. The primary objective of this dataset is to provide an estimate of a software sensor's reliability over time. This gas turbine was selected for this purpose because it provided data on the usage of a unique gas turbine at a specific location over several years.

To avoid excessive memory usage, data was collected densely (i.e., data for every week) only during the first year and a half. For subsequent periods, data was gathered for approximately one week per month on average; when available.

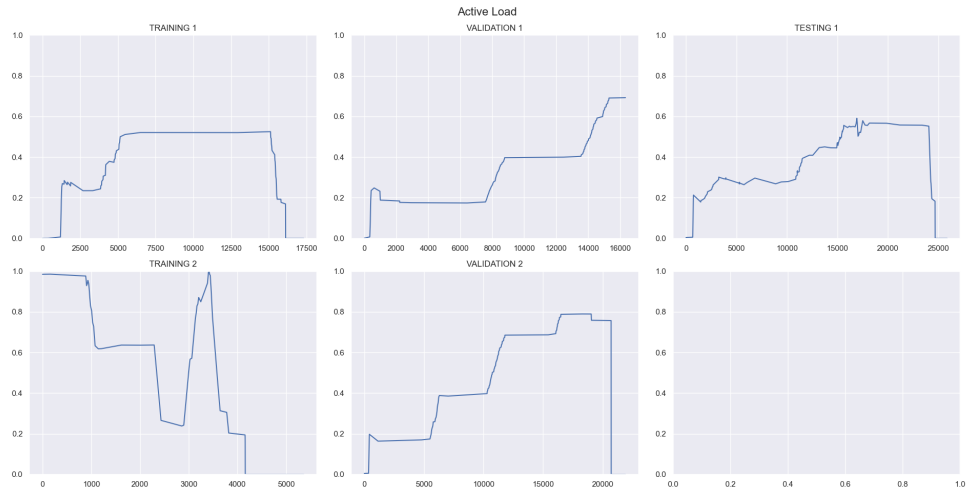



Figure 3.6: **Dynamic Dataset Overview**: This figure shows an overview of the load (MW) of turbine runs in the *dynamic* dataset. It shows some turbine runs used for training and validation and the unique run used for testing. The operation of this gas turbine is very dynamic with very frequent load variations.

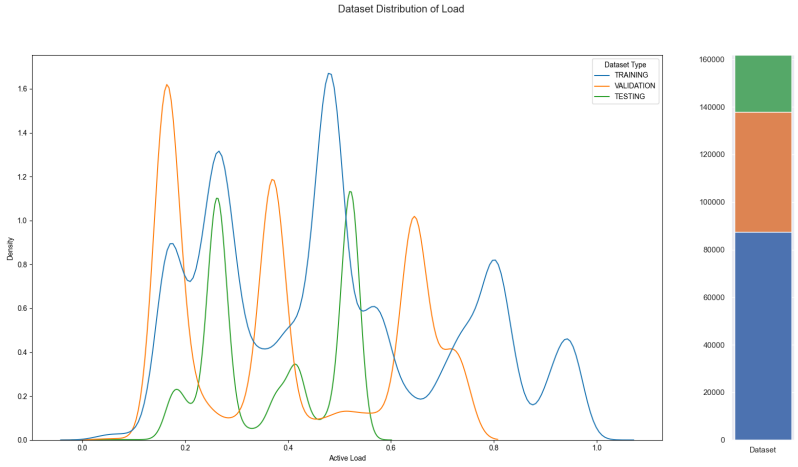



Figure 3.7: **Dynamic Dataset Load Distribution**: This figure characterized the load (MW) distribution of the *dynamic* dataset. To the right, it is shown the number of data samples and the proportion used for training, validation and testing.

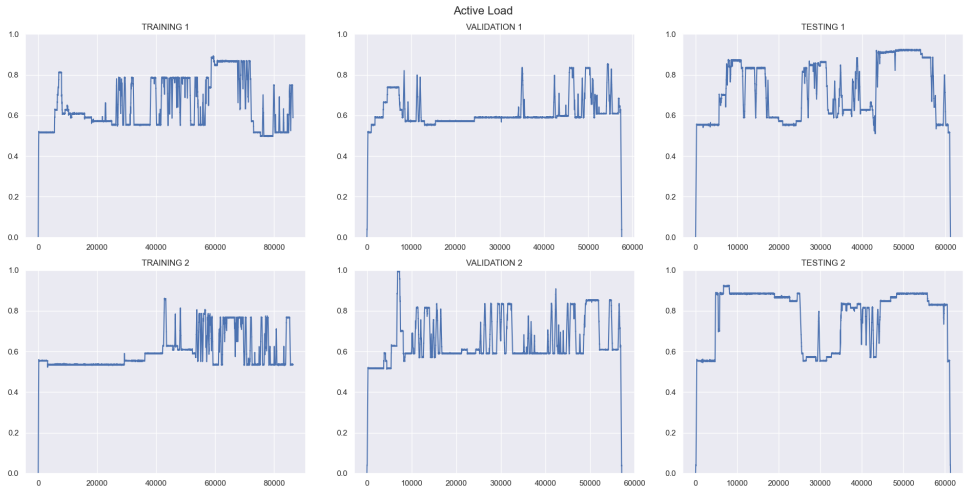



Figure 3.8: **Medium Dataset Overview**: The figure illustrates the load (MW) of several runs derived from the medium dataset, representing data from a turbine used for grid stabilization. The data reflects frequent changes inherent to the operations of the turbine.

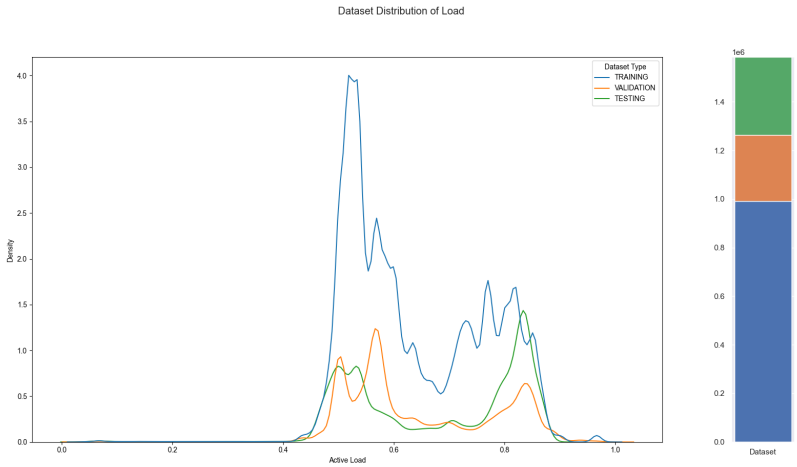



Figure 3.9: **Medium Dataset Load Distribution**: This figure presents a detailed analysis of the load (MW) distribution within the medium dataset. On the right, the total number of data samples is divided into segments showing the proportion allocated for training, validation, and testing.

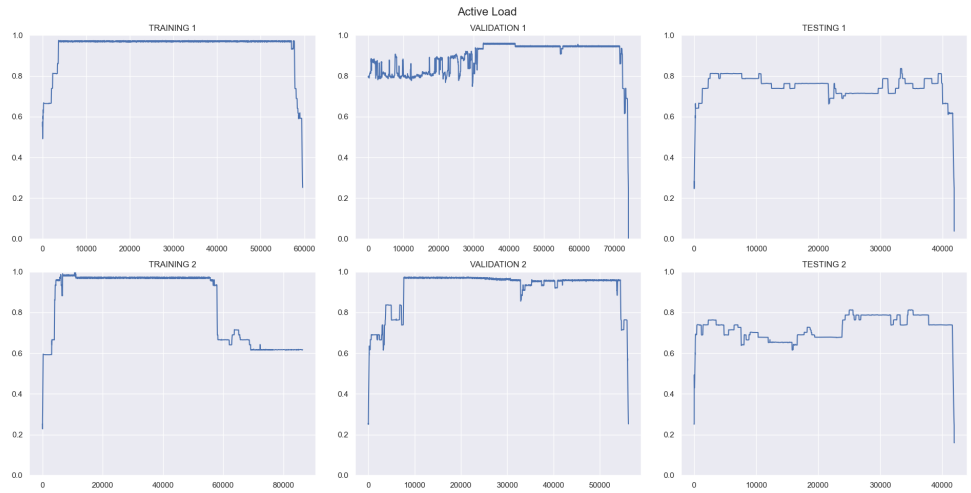



Figure 3.10: **Large Dataset Overview**: The figure illustrates the load (MW) produced in several runs derived from the medium dataset, representing data from a turbine used for grid stabilization. The data reflects frequent changes inherent to the operations of the turbine.

Due to those characteristics the proportion of testing and training data for this dataset is inverted, as shown in Figure 3.9. The testing data is over 50% of the total dataset.

## 3.4 Data Cleaning Procedure

The cleaning procedure is common for both datasets as the issue described in Section 3.3.1 is shared by both data sources, but being much less prominent to the test facility's data, for which missing values during a turbine run are rather rare.

The steps taken for cleaning a specific run $\boldsymbol{X} \in \mathbb{R}^{T \times D}$ are shown in Algorithm 2. Essentially, the procedure consists of removing those dimensions that contain invalid values (Not a Number) over a certain threshold, repeated values, active load under threshold and then segment it into segments $S_i \in \mathbb{R}^{T_i \times D'}$ for which all the values are valid. This means that we discard those temporal intervals that, after the first parts of the procedure, still contain invalid values. The significance of this procedure can be seen when analyzing the result on a turbine run like the one shown in Figure 3.4. Here, this procedure will discard only those sensors that are truly not usable and then will remove those temporal spaces with failures (at around sample 70000 and

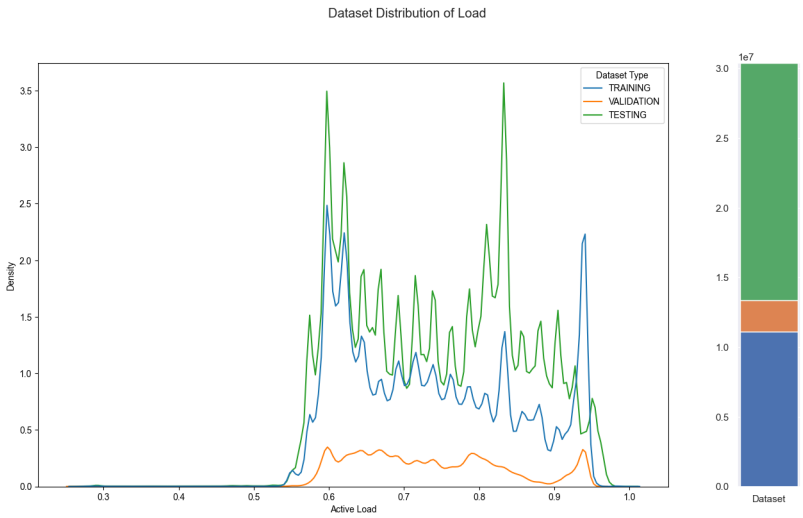



Figure 3.11: **Large Dataset Load Distribution**: This figure presents a detailed analysis of the load (MW) distribution within the medium dataset. On the right, the total number of data samples is divided into segments showing the proportion allocated for training, validation, and testing.

320000). Thus, it preserves the maximum possible amount of data from the turbine run while also satisfying the minimum load constraint.

As a result of this algorithm, each turbine run produces a set of segments of completely valid data that meet all given constraints, and therefore can be utilized by the subsequent stages of the data pipeline.

**Algorithm 2** Turbine Run Cleaning Procedure

1: **procedure** CLEANRUN($\boldsymbol{X}$, $desiredSamplingRate$, $nanThreshold$, $minLoad$,$minSegmentLength$)
2: $\boldsymbol{X} \leftarrow$ resample($\boldsymbol{X}, desiredSamplingRate$)
3: $\boldsymbol{X} \leftarrow$ removeNaNOverThreshold($\boldsymbol{X}, nanThreshold$)
4: $\boldsymbol{X} \leftarrow$ invalidateRowsWithLoadBelow($\boldsymbol{X}, minLoad$)
5: $\boldsymbol{X} \leftarrow$ removeRepeated($\boldsymbol{X}$)
6: $S \leftarrow$ segment($\boldsymbol{X}, minSegmentLength$)

A problem specific to turbine test data is the issue described in Section 3.2.1. Here, the main problem is that, after cleaning the dataset, the obtained segments might have (and usually do) a different set of sensors. This issue does not arise on data coming from the fleet as the set of valid sensors is rather stable. On the fleet's dataset usually temporal intervals are discarded,

rather than the full sensor data.

As shown in Figure 3.1, the prevalence of this issue is high in the testing facilty's dataset. Manually selecting a set of runs with the desired valid columns is not feasible, especially during development when the set of features used is frequently changing following discussions with domain experts. An elegant solution can be achieved by making use of *combinatorial optimization*. There are efficient tools available for solving such problems, which can be expressed through constraint programming [48]. In this master's thesis the CP-SAT solver of Google Operations Research Tools was the software user to address this issue [49].

The problem can be formally expressed by defining the boolean variables $\{r_i\}_{i=1}^{N}$, with $N$ being the number of runs, available and $\{d_i\}_{j=1}^{M}$, with $M$ being the total number of sensors under examination. Those variables represent whether a certain run $r_i$ and a certain sensor $d_i$ is selected or not. The optimization problem can be stated by the following objective and constraints:

- **Objective**: Maximize the number of data samples selected. Alternatively, it is possible to maximize the number of turbine runs selected. Constraint programming is versatile in allowing the user to choose what exactly is to be maximized.

- **Constraint** 1: Selecting a file $f_i$ for which the sensor $d_j$ has no entries (i.e. it is invalid) implies not selecting the sensor $d_j$ for any run. In other words, if a sensor has no valid entries in a specific run the algorithm must select either the sensor or the file (i.e. in this case $d_i = True$ implies $f_j = False$ and vice-versa.

- **Constraint** 2: The user can, optionally, specify a minimum number of runs and sensors to select.

- **Constraint** 3: The user can, optionally, specify a set of sensors that must be selected. This is equivalent to setting their respective boolean variables $d_i = True$. It is also possible for the user to specify a set of turbine runs that must be selected, equivalent to setting $r_i = True$.

- **Constraint** 3: The user can, optionally, specify that if a specific run is selected then at least another run from the same machine must be selected (this is useful for testing algorithms in the same machine).

The advantage of constraint programming lies in its simplicity - we only need to define the constraints that need to be met and the optimizer will deliver

a solution. However, as with any combinatorial optimization problem, finding a solution can be time-consuming. In our usage of this algorithm all solutions were always found in a reasonable amount of time (under 5 minutes). However, the user can also specify a time limit within which the solver must return a result. The possible outcomes of the solution procedure are [50]:

- Optimal Solution Found: This result is attained when the solver is able to prove that, given the specified constraints, the found solution is optimal.
- Feasible Solution Found: This result is attained when the solver did not explore all possibilities but found a solution that satisfied the given constraints. If multiple solutions are found the solution that minimizes/maximizes the objective is returned
- Infeasible: This result is given if the problem, given the constraints, is proven to be impossible.
- Unknown: No feasible solution has been found neither it could be proved no solution exists.

As a result, the creation of a dataset with data from the testing facility and a specified set of sensors is fully automated.

## 3.5 Modular Sample Generation

In this section the solution for generating training, validation and testing samples from the different datasets will be discussed. A modular solution is required due to:

- Complexity of data sources: Due to the followed cleaning procedure and the nature of the data sources are multiple are do not represent an uninterrupted sequence of time. This requires careful consideration when generating samples that use multiple timestamps
- Unique source of truth for different models: To decrease the probability of a human-introduced error and augment the velocity at which it is possible to iterate, as in the CRISP-DM.

Those necessities led to the implementation of a system able to combine multiple data sources and extract from them samples that adapt to the requirements of the model under training/validation.

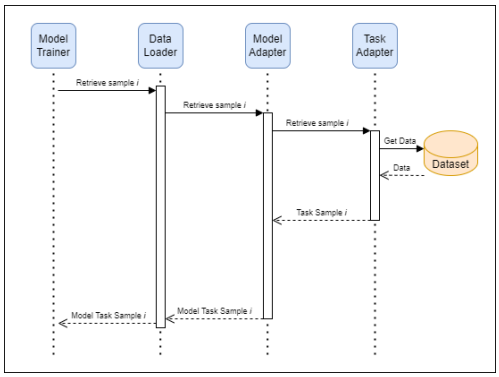



Figure 3.12: **Data Loader Architecture**: For ensuring consistent testing for models

First, the different files containing time sequences are divided into *Data Splits*. This is a basic unit, inspired by Hugging Face's datasets architecture [51], that contains a sequence of data points temporally adjacent with a specific purpose (training, validation, testing).

By grouping all the Data Splits with a specific purpose one obtains the training dataset. To ensure consistent sample generation for all different methods and deliver a clean software product to Siemens Energy the Adapter pattern was used [52]. This pattern was also chosen because it allows for seamless integration with one of the most popular deep learning frameworks: PyTorch [28]. The main idea behind it is to not generate all the samples at once, as this can be impractical due to the large memory requirement it would imply but compute them *on the fly* as the model requires them. This architecture ensures that the system stays scalable even with very large datasets,that would potentially need to handle, and divides the concerns in the software to ensure relatively easy extensibility of the code. A sequential UML diagram is displayed in Figure 3.12 showing a schema of the process for retrieving a training sample. In this system the task samples will be exactly the same for two different models, ensuring that they will receive exactly the same information and the same target.

# Chapter 4

# Software Sensor

This chapter explains the methodology followed to develop the software sensor. It initiates with a presentation of the data exploration methodologies and results conducted on the datasets. This information is critical in the subsequent development of algorithms and selection of feature variables, and is therefore included in this chapter. The final part of this section highlights the algorithms for a pertinent feature extraction mechanism associated with the turbine's operation zone.

The primary target variable used in this study is the low frequency pulsation, which is elaborated in detail in Chapter 2. This variable was selected due to its significant role in the operation of the gas turbine and the complexity involved in its prediction. As described in [35], thermoacoustic instabilities, which are relevant for turbine's health, are measurable in these sensors.

The latter part of this chapter offers an in-depth explanation of the various methods employed to construct this artificial measurement system along with the rationale behind their relative inception. Subsequently, the procedure followed to tune the hyper-parameters of the different models and to configure experiments is presented. Finally, this chapter concludes with a discussion over the evaluation metrics used, highlighting how the best models are chosen and how the results will be presented in the following Chapter 5.

## 4.1 Data Exploration

In this section we will summarize the results of the data exploration. The rationale for including this in the present chapter, rather than in the subsequent Chapter 5, lies in the fact that the insights obtained during this stage significantly steered the subsequent research and selection of both features

and algorithms. Hence, it seems more logical to present this section before delving into the algorithms section.

The key takeaways coming from this exploration can be summarized in the two following points:

- Turbines are unique, or at the very least very different from one another.
- Sensor data is surprisingly correlated, both between sensors and across time.

### 4.1.1 Correlation Across Sensors

The correlation comes at less of a surprise as similar results were attained in previous research [2][40] , showing that sensor data coming from gas turbines exhibit very high values of linear correlation. This result can be visualized for $n$ sensors in the Figure 4.1 . One can notice that there are big blocks of highly correlated features along the diagonal of this matrix, those are less interesting correlations as they represent sensors that are measuring physical quantities that must be the same (either they are measuring the same quantity or a quantities that must be very close to another). More interesting are the clusters that appear far from the diagonal and show correlations across different sets of sensors (for instance, temperature in the compressor and temperature in the outlet). Those are correlations that can be potentially exploited for building anomaly detection systems, thus are very interesting.

The data exploration phase was essential, combined with consultations with domain experts, to carry out the feature and model selection. This process guided us in avoiding predictors that not only had a clear correlation with the target variable, but also shared a high degree of *collinearity*, such as two sensors that are both measuring the same physical quantity. The primary insight from this analysis, which is consistent across all the datasets we collected, is that the measurements taken in a gas turbine are more correlated than one might initially think. In retrospect, it's logical that an increase in certain sensor measurements could lead to a corresponding increase in other sensors. For instance, a rise in temperature in the early stages of the turbine would likely cause higher temperatures in the later stages.

### 4.1.2 AutoCorrelation

Autocorrelation analysis was crucial , taking into account the temporal nature of the system. This analysis revealed that a significant number of sensors exhibit high autocorrelations, an intriguing, though not unexpected, finding.

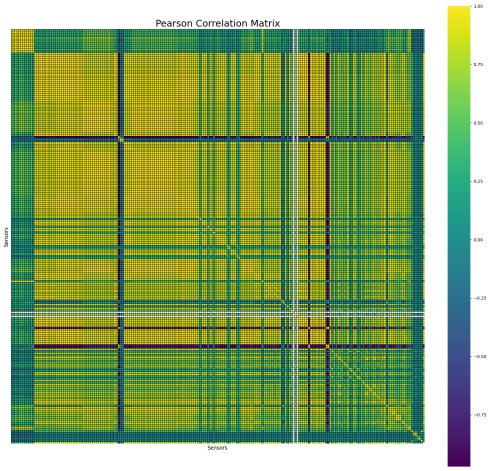



Figure 4.1: **Pearson Correlation of Sensors**: This illustration presents the result of a Pearson correlation analysis conducted on data gathered from 217 distinct sensors in a gas turbine across multiple turbine operations. The analysis reveals significant correlations among various sensors.

These autocorrelations manifest up to a time lag of 500 seconds, which is nearly 8 minutes. This behaviour with high auto-correlation for high lag values emerges from the nature of the data under examination. We attribute this behavior to the tendency of a gas turbine, in conventional applications, to remain in the same operational condition for a considerable length of time. The sensor readings during these periods are nearly constant, with minor variations due to factors such as turbulence, electrical noise, and minor flow changes.

In some sensors, we observed an extreme autocorrelation (over $0.99$ with the values up to $3 - 10$ seconds). This can be attributed to the physical principle that governs their functioning. For instance, temperature sensors are governed by the Seebeck effect[53][10]. This principle explains how a temperature gradient between two points of a metal or semiconductor can generate a voltage difference. As the thermocouple, which operates on this principle, undergoes a temperature transition, it might not capture the system's dynamics instantaneously. The physics of the system could potentially apply a kind of low-pass filter to the actual or underlying system temperature and lead to this high auto-correlation for the first time lags.

The varied behaviors of sensors become evident in Figure 4.2, which depicts the autocorrelations of three different sensor types. Here it is possible to observe that the previous discussion is not valid for all types of sensors. For instance, vibration sensors exhibit a low degree of autocorrelation.

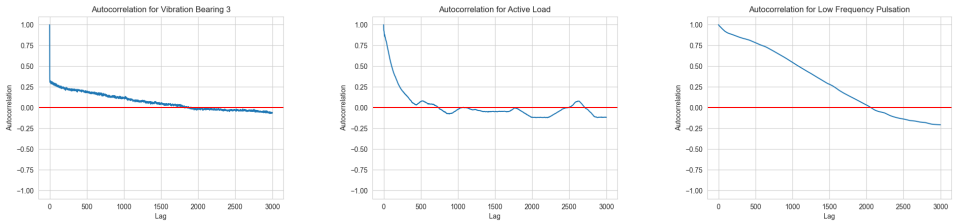



Figure 4.2: **Auto-Correlations**: The three figures above show temporal auto-correlation of three different kinds of sensors. One can observe a high degree of autocorrelation for sensors probably due to the intrinsic slow nature of the data (i.e. gas turbines operate in the same condition for relatively long periods). This pattern is not present for all sensors as it can be seen in the vibration autocorrelation.

### 4.1.3 Turbine Uniqueness

Another finding from the data exploration study is that gas turbines are very different from one another; one might say *unique*. During this study, as mentioned in Chapter 3 we only focused on turbines of the kind SGT-700. Despite this, variations between them are substantial enough to distinguish them by the characteristics they have.

This variation can be attributed to several factors. Unique tuning based on customer requirements and physical differences among thousands of components within the turbine, both large and small, play a significant role. Furthermore, the unique operation type of each specific gas turbine contributes to this diversity; as shown in Figure 3.10 and Figure 3.6. Operational variations could include different levels of active load, specific applications of the gas turbine, and unique tuning requirements such as NOx emission levels. The type of components used, such as the technology utilized in the combustion chamber, can also greatly influence the turbine's behavior.

A straightforward analysis, shown in Figure 4.3, highlights the variations among gas turbines. This image shows the results of Principal Component Analysis applied to data points collected from various turbines operating at full load. By maintaining a constant operation point across all turbines, we can ensure a valid comparative analysis. These differences, which manifest as distinct clusters of data points representing each gas turbine in the figure, will become a focal point in our discussion in Chapter 5.

This key observation is also relevant in deciding what kind of experiment was necessary to observe if the *knowledge* from a specific gas turbine is easily transferable to another or if special considerations must be done in this regard.

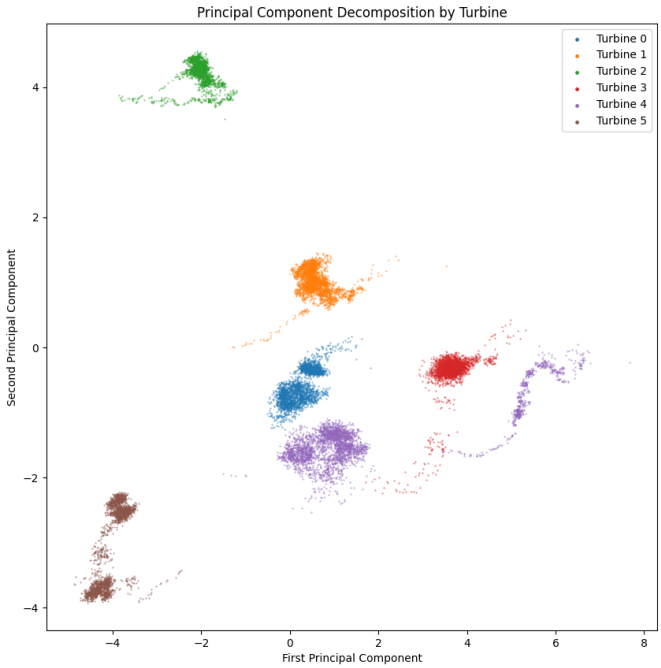



Figure 4.3: **PCA on different turbines**: This image shows the Principal Component Analysis of data points from several testing facility's turbines operating constantly at full load. It is possible to clearly distinguish clusters by turbine. The PCA was done from 19 dimensions.

## 4.2 Operation Points of Gas Turbines

A highly intuitive strategy for enhancing models prediction capabilities, is to incorporate the operation point of the gas turbine. This approach is rooted in the simple assumption that the behavior of a gas turbine will vary significantly between periods of transition and periods of stability. Moreover, the concept of determining whether a turbine is in a transitional or stable state appears viable by merely examining the active power output of the turbine (i.e. the megawatts produced at a specific moment) if one for instance observes the active load represented in Figure 4.4.

However, it's worth noting that gas turbines can, under certain circumstances, operate without generating any energy output, particularly during the shutdown phase of the turbine. This is when the hot air must be ventilated to prevent damage to the components. For the sake of simplicity, we have

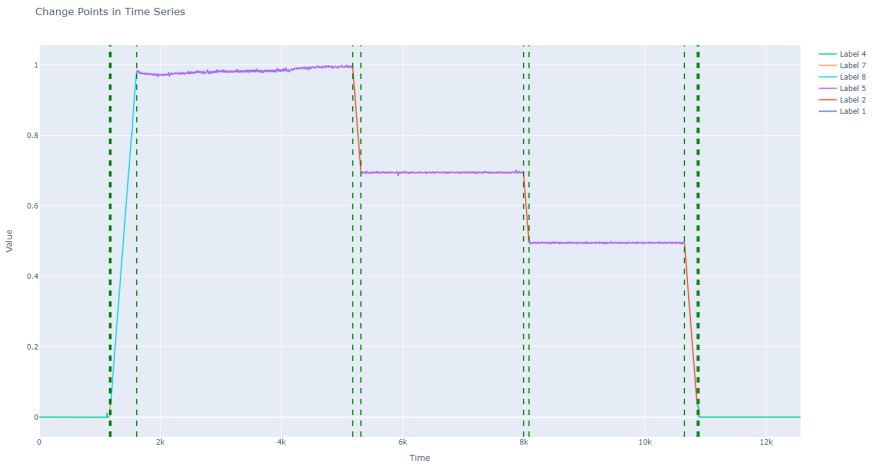



Figure 4.4: **Active Load Segmentation**: This figures shows the result of the active load segmentation procedure in a turbine run of the turbine testing facility. Segment division is shown by the dashed green lines. The color of the line represents the classification of the time segment.

not considered any of these situations. We have pruned out parts of the dataset where the load is zero from our analysis, as elaborated in Section 3.4. Therefore, our primary focus is to differentiate the various operational points of a turbine when it is producing some output load.

The chosen strategy is straightforward: we define as different states when the active load is increasing, decreasing or stable. Furthermore, we segment the *stable* parts by the nominal value of the active load and whether it surpasses a certain threshold $B$.

By employing this method we are effectively *compressing* a *lot* of information into a single point and giving algorithms useful information for making predictions. The steps followed to obtain the result shown in Figure 4.4 are described below.

First, for obtaining better and more stable results the active load signal was smoothed using a Gaussian filter. The chosen window length for this filter was of 10 seconds. As it can be seen in Figure 4.5 , this choice provided a more clear distinction of the different states when looking at the result of the derivative.

The derivative calculation was executed using the FinDiff package in Python [54]. This package works on the principle of numerically approximating the first derivative of the signal in a stable manner. Instead of utilizing the *instantaneous* rate of change, which can be volatile, the method

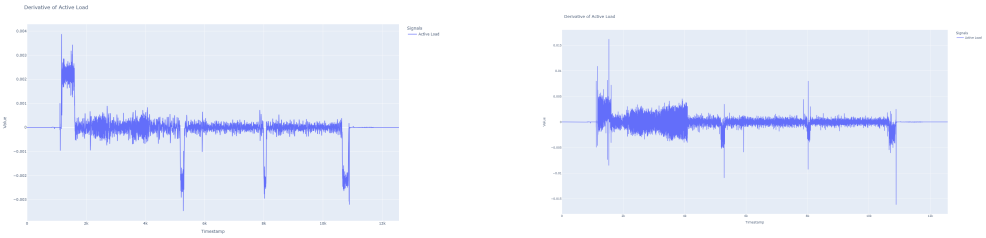


Figure 4.5: **Active Load Derivative**: This figure illustrates the result of the active load derivative, demonstrating the outcomes both without the application of a Gaussian filter (right) and with it (left).

employs a *stencil* for more stable and reliable calculation.

Then, segments are found by making use of a *changing point algorithm* in the active load's derivative. The chosen algorithm was Pruned Exact Linear Time, and the implementation of the *ruptures* library was used [55][56]. As explained by the authors of the library a changing point algorithm requires to specify a search method, a cost function and some constraints.

The changing point algorithm aims to minimize the cost over a the time series by finding an optimal division $\boldsymbol{t} = \{t_1, ...t_N\}$ of it. The cost function is expressed as the sum of a cost for each of the segments defined by the given division $\boldsymbol{t}$.

In our case the search algorithm employed was PELT, which has a time complexity of O(n)[56], consistent with our requirements having datasets with large amount of timestamps. Other algorithms such as Binary Segmentation can be used to replace PELT for a faster execution. For the cost function we made use of the $L2$ cost function, which aims to divide two segments based on its mean value; this means that we expect all points belonging to the same segment to be close to the mean of the segment. The expression of this cost function is shown in Equation (4.1), and its choice come logically observing Figure 4.5.

$$C_{L2}(y_{t_1,...,t_N}) = \sum_{t=t_1}^{t_N} ||y_t - \bar{y}_{t_1,...y_N}||_2^2 \tag{4.1}$$

Finally, each of the segments is classified depending on the mean value of the derivative and the active load. This leads to 9 possible results, but not all of them appear in real data. A summary is given in Table 4.1

Notice that the procedure just described, from the Gaussian filter, to the changing point algorithm can be only applied offline after a turbine run has

| Active Load >B | $\lvert\frac{d}{dt}$ Active Load$\rvert$ >D | Sign($\frac{d}{dt}$ Active Load) | Classification |
|---|---|---|---|
| No | No | Positive/Negative | Off (4) |
| No | Yes | Positive | Starting (7) |
| No | Yes | Negative | Finishing (1) |
| Yes | No | Positive/Negative | Stable (5) |
| Yes | Yes | Positive | Increasing (8) |
| Yes | Yes | Negative | Decreasing (2) |

Table 4.1: **Operative States Definition**: In this table we have the definition of the operative zones of the gas turbine based on the active load value and derivative's value and sign.

finished (actually the Gaussian filter only requires some seconds of delay, being a non-causal kind of filter). This is consistent with the aim of this thesis as we are not developing an online solution. However, as it was done for the larger datasets where this procedure is prohibitively expensive, it can be easily extended to online applications by simply applying a threshold over the value of the active load. This leads to a less informative context but it is yet effective in encapsulating some useful information that the model can leverage.

## 4.3 A simplified model of the problem

In this section we will introduce a simplified model of the turbine for modeling the relationships between the different parts of the gas turbine and a model of the scope of the desired software sensor. The necessity of this model arises from the need to obtain some insight of the relationships that exist in the gas turbine and how the measurements that we obtain are related to them.

The model, shown in Figure 4.6, uses the plate notation [24] to denote the collection of the random variables up to a certain time $T$. The physical variables, at a given instant in time, are modeled using a set of nodes in the graph. However, the network's structure remains unknown to us and is concealed within the turbine dependency box. This is shown by the thick connections; representing multiple possible relations of each variable with other variables in the gas turbine. Then , we model the sensor with other set of nodes $S_i$ and we *assume* that each sensor is solely linked to the physical quantity they are designed to estimate. This is a reasonable assumption as we expect the measuring device to provide some probabilistic guarantee about the real value of the physical quantity, given the sensor reading. Moreover, we model the turbine's operational state with another random variable, which can

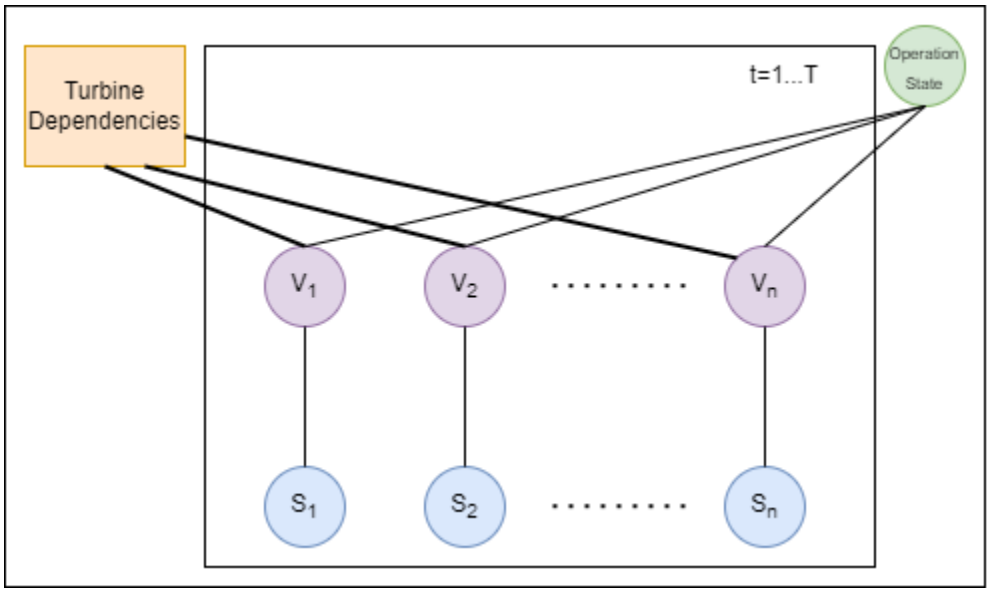



Figure 4.6: **Simplified PGM of gas turbine**: This diagram depicts a simplified model of the gas turbine and the sensors.

give rise to different physical variable behaviors in the gas turbine.

This is, as stated before, an oversimplified model and we must consider that the real system contains temporal dependencies both between a variable and its previous value as well as a variable and the previous value of other physical values. In this sense, we would be modeling the system as a Hidden Markov Model (assuming dependencies are not extending beyond the previous timestamp). Despite being an intriguing approach, this path was not further explored as we lack a probability distribution model for the physical random variables' time evolution. This exploration is earmarked for future work.

## 4.4 Algorithms

In this section, we will describe the various algorithms utilized in the development of the software sensor for the gas turbine. Initially, we will present the baseline, followed by an exploration of methods that leverage both a single time instant and a time window to generate predictions.

The selection of these methods was not predetermined, but rather evolved in a cyclical manner, adhering to the Cross Industry Standard Process for Data Mining methodology. This approach involves a systematic evaluation of the limitations of a developed method, followed by the selection and development

of new methods based on insights gained from the training dataset. In this way, our method selection process is both responsive and adaptive, allowing for continual refinement and improvement. This process can be easily continued at Siemens Energy after the conclusion of this master thesis by making use of the developed data collection system presented in Chapter 3 and the experiment software described described in Section 4.5 and Section 3.5.

### 4.4.1 Baseline

From a scientific perspective, and to satiate the curiosity of authors, it is often appealing to utilize intricate models. However, it is crucial to establish a baseline for comparison, providing a *quick and dirty* solution. In many instances, the added complexity of advanced models is not beneficial; rather, it becomes burdensome. For instance, sophisticated models like deep neural networks often lack explainability for the obtained regressions. Interestingly, it is sometimes observed in academic research that simpler models can achieve comparable results, or even outperform their complex counterparts [57]. Therefore, while the allure of complexity is understandable, it is essential to balance this with the model's transparency and efficiency.

The baselines adopted in this study involve linear regression models, as shown in Equation (4.2). Our primary objective is to pinpoint the circumstances under which a more complex solution becomes necessary, as well as to quantify the resulting performance improvement. Additionally, we will employ a particularly simplistic model that solely relies on the power output (i.e., MW produced at a given moment) of the gas turbine to generate its predictions.

To allow for a fair comparison with other methods we will train *specialized* linear models. This means training a different linear model for different operation zones. This decision was made after observing the huge imbalance in datasets when considering the number of data samples coming from different operative zones. Training a unique model on all areas leads to linear models that only take into account the parts with stable conditions , as they represent the majority of the data.

$$y = \beta_0 + \beta_1 x_1 + ... + \beta_n x_n + \epsilon \tag{4.2}$$

### 4.4.2 Shallow Methods

Variations of linear regression that include regularization terms as shown in Section 2.2 for preventing overfitting . Among them the one that showed better results was Ridge regularization, doing a less aggressive weight shrinking. For the experimentation with those algorithms we made use of the scikit-learn library, which provides an clean and correct implementation of those algorithms [58].

Furthermore, other shallow methods, like SVR and Bayesian Linear Regression, were discarded because it was not scalable to use them in larger dataset. Results for them are reported only on data coming from the testing facility, where the amount of training data samples allows for convergence of algorithms.

### 4.4.3 Deep Neural Networks

The most powerful methods used during this master thesis for the implementation of the software sensor are deep neural networks. As previously discussed in Section 2.1, introducing some level of inductive bias through the architecture is a crucial step towards ensuring the successful performance of the network. The various architectural configurations will be detailed in the following sections.

Deep neural networks are intrinsically highly flexible tools and, as outlined in Chapter 2, they are essentially universal function approximators [25]. This flexibility, however, also makes them susceptible to overfitting. Hence, to mitigate overfitting, common regularization techniques, such as dropout and early stopping, have been employed in this study.

#### Multi-Layer Perceptron

A simple approach, inspired by the work in [2][12], is to perform the prediction by using a deep neural network with a very simple architecture, namely a MLP. The idea is to use sensor data of a given instant of time $t$ to perform the predictions for the target sensor for the same instant $t$.

The expected success of this architecture is also sustained by the fact that , as seen during the data exploration part, our main target sensor (the low frequency pulsation), as well as many other input sensor have high autocorrelations which indicates that including more timestamps might not be necessary for obtaining, as the information might be redundant.

While this network remains simple it is able to capture the expected non-linearities that emerge from a complex system like a gas turbine.

### Ensemble of Multi-Layer Perceptron

As discussed previously, the development of this master thesis was not linear but it was done with a cyclical approach , as per CRISP-DM [13]. This architecture emerged from the idea to make the best use of the available temporal context to reduce the variance of the predictions done by a single MLP [32]. As stated in [32], for an ensemble method to be successful and effectively minimize errors made by a single model, it is mathematically required that the models maintain independence, or achieve as close to independence as possible.

In our application, we adopted a *data diversity* strategy. During the creation of the model, each sub-network is randomly assigned specific timestamps from each sensor within a pre-defined time window for prediction generation, as shown in Figure 4.7. These timestamps remain constant for each network during both training and inference. This method of random assignment creates the necessary independence among the sensors. For simplicity, the architecture of each single MLP is the same . The final result of the model is the mean prediction of every single network.

### Time aware networks

Temporal awareness can be a significant advantage for networks, which allows them to process more data effectively. While this time awareness is somehow present in the ensemble of MLP in this section we explore networks that perform this task in a more structured way.

Firstly, let us explore architectures based on Recurrent Neural Networks, which have been the industry standard for many years in handling time sequences. These types of networks appear to be especially suitable candidates for encoding the temporal information present in turbine data.

Standard Recurrent Neural Networks can often face issues with exploding gradients and memory, which can be overcome by using more advanced implementations of RNNs such as Long Short-Term Memory and Gated Recurrent Unit networks. These networks utilize specific architectures that enable the network to *remember* crucial aspects of the data at hand.

Furthermore, recent studies suggest that incorporating an attention mechanism into these networks can lead to significant improvements. This mechanism allows the network to focus on relevant features and periods.

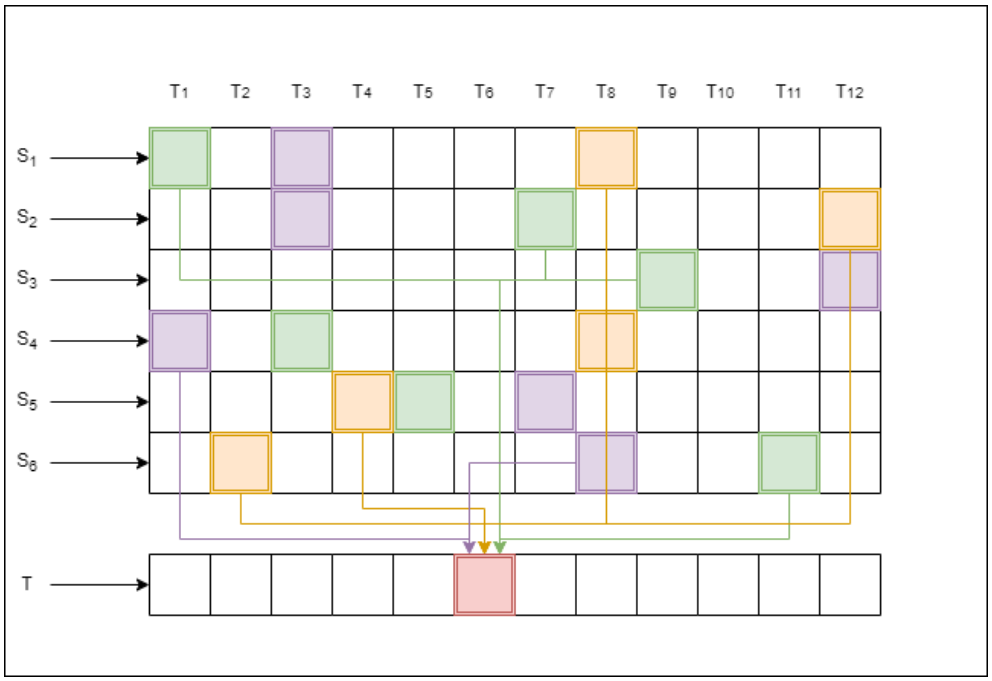



Figure 4.7: **Ensemble of MLP**: This diagram presents the proposed architecture of an ensemble of MLP. During the model's creation, each network is randomly assigned specific timestamps from each sensor within a predetermined time window to generate predictions. These assigned timestamps remain constant during both the training and inference phases, ensuring consistent sensor usage for each network. This random assignment creates the necessary independence between sensors, which is critical for the successful implementation of the ensemble method. Subsequently, these individual network predictions are averaged, culminating in a well-rounded and dependable final prediction.

Therefore, our architecture of choice is strongly inspired by the successful network implemented by Yao Quin et al. [59]. Although this original network was designed for time-series forecasting in economics, its architecture can be easily adapted to our current domain.

We make use of the input attention layer proposed by the authors of the paper to allow the network to focus the attention on the relevant time series; according to the context, for each timestamp. The final encoding after processing all timestamps is then used by our simplified decoder (a single MLP) to make the prediction of the target. Since we are not interested in prediction we allow the target to be within the time frame of the processed data by the network. Notice that there are no data leaks since, differently to the

approach of the original paper, the target variable is not present in the feature variables; given that we are aiming to simulate an artificial measurement of such sensor, thus we assume to have no knowledge about the sensor.

Recurrent Neural Networks have been well known to suffer from a variety of problems from velocity of processing to forgetting past. In the recent years, coming from NLP research, a new kind of network has been proposed with exceptional results both in NLP and also computer vision [20]. The emergence of the transformer architecture has been revolutionary in those fields achieving incredible results. Being somehow an analogous problem to time series processing some research has been done regarding the usage of this architecture in time series modeling. In the research [60] it is shown how the Encoder of the transformer architecture, with a pre-training strategy similar to BERT [61] can be used for developing highly capable time-aware networks employing this architecture for a downstream regression task. However, our preliminary experiments for using it as software sensor showed that this complex architecture did not achieve good performance, in terms of MAPE, in our datasets, probably because of the large amounts of data required by it, complex training mechanism and the characteristics of the data. Hence, we did not include this architecture in the following analysis and did not proceed further with it.

## 4.5 Hyper-parameter tuning & Experiment Setting

The selection of hyper-parameters is central for the successful evaluation of algorithms. Almost all of the algorithms proposed in this work for software sensor implementation necessitate some degree of fine-tuning. This tuning was executed utilizing the validation part of each dataset; which was extracted as specified in Chapter 3.

In this work, we utilized the Optuna framework for efficient hyper-parameter tuning [62] and Hydra for setting up experiments [63]. Also, this project greatly benefited from the usage of PyTorch Lighting [64] for defining the training, validation and inference strategies for deep learning methods.

Optuna is a robust, open-source hyper-parameter optimization framework which allows for automatic selection of hyper-parameters, reducing the time and complexity traditionally associated with this process and allowing to effectively explore the hyper-parameter space.

From the user point of view, the complexity of searching the hyper-

parameter space is reduced to defining the search space through an API. What is remarkable is that Optuna automatizes the traditionally expensive task of exploring promising values for hyper-parameter search (e.g. the researcher discarding values of learning rate that are not successful and continuing to explore the learning rates that lead to convergence of the loss).

The Optuna framework uses a combination of several optimization methods, such as TPE (Tree-structured Parzen Estimator), to find the most efficient parameters. However, a detailed discussion on the methods employed by Optuna is beyond the scope of this thesis. It also provides features for visualizing the optimization process, making it easier to understand and analyze the results. Optuna is particularly powerful in combination with PyTorch lighting [64] as it allows for early stop of non-promising trials, by monitoring the validation loss during training.

The combination of Optuna with Hydra [63] is particularly powerful as it is possible to dynamically define everything about the experiment. During this master thesis we made use of hydra for defining the three main components of each experiment.

- General Configurations: Configurations regarding the task that must be solved by the model. This includes for example the length of the time window of features given to the model (useful for RNN based models)
- Dataset Configuration: Regards which sensors to use as features and target as well as where to retrieve the dataset.
- Model Configuration: This configuration specifies the model to be used, with its hyper-parameters, training hyper-parameters and model adapter (see Section 3.5)

For instance, consider the configuration of a feed forward neural network shown in Listing 4.1. Hydra allows to specify every single parameter of the network like in a YAML configuration file. While is is not the case for the chosen examples, in models that make use of a time window it also allows to explore different window lengths by modifying parts of the *dataset* configuration. Practically, Hydra facilitates the development by defining implicit factories, in OOP sense, [52] for our modules. The power comes when combining such a configuration with Optuna, as shown in Listing 4.2. The effort dedicated into the creation of this framework was well paid off since the grid search of every model defined during this master thesis can be generated by simply defining this YAML file. For more details on how this was implemented the reader can see a commented code in the Appendix A.

```yaml
# Configuration of the feed forward model itself
model:
  # We create the model from the dataset (to dynamically have the
  # number of features)
  _target_: "package.deep_architectures.FeedForwardModule.from_dataset"
  output_size: 1
  dropout: 0.1
  hidden_sizes:
    - 100
    - 50
  # We can specify both optimizer and scheduler from the YAML
  optimizer_config:
    _target_: "package.deep_architectures.DynamicOptimizerConfigurator"
    optimizer_class: "torch.optim.Adam"
    optimizer_params:
      lr: 0.001
    scheduler_class: "torch.optim.lr_scheduler.StepLR"
    scheduler_params:
      step_size: 100
      gamma: 0.5

model_adapter:
  _target_: "package.data.model_adapters.PyTorchAdapter"
  task: ${general.task}
  batch_size: 1024

# Configuration of the training itself
training:
  trainer:
    _target_: "package.deep_architectures.PyTorchModelTrainer.from_config"
  max_epochs: 200
  callbacks:
    - _target_: "lightning.pytorch.callbacks.early_stopping.EarlyStopping"
      patience: 20
      monitor: "val_loss"
```

Listing 4.1: Configuration Example of a Feed Forward Module

```yaml
execution_mode_args:
  n_trials: 200
  # This is just a placeholder for the title of the experiment.
  model: "feed_forward"
  metric: "mae"
  params:
    # Optuna does not support lists. This is possible due to
    # a wrapper class created around Optuna.
    - param_path: "model_framework.model.hidden_sizes"
      param_type: "list"
      param_args:
        name: "hidden_dimensions"
        length:
          low: 1
          high: 3
        elements:
          param_type: "integer"
          param_args:
            low: 50
            high: 200
            step: 50
    - param_path: "model_framework.model.optimizer_config.optimizer_params.lr"
      param_type: "float"
      param_args:
        name: "learning_rate"
        low: 0.00001
        high: 0.001
        step: 0.00005
    - param_path: "model_framework.model.dropout"
      param_type: "float"
      param_args:
        name: "dropout"
        low: 0.05
        high: 0.3
        step: 0.05
```

Listing 4.2: Grid search Configuration Example of a Feed Forward Module

## 4.6 Evaluation Metrics

The task of developing a software sensor is in the root a regression task. The main metric used for evaluating the software sensor is the Mean Absolute Error. This metric is the preferred one as it gives a clear measure of *how far* the predicted measures are from the target value and allows for an easy interpretation of the model's performance.

Although this metric was used internally at Siemens Energy for evaluating the models the results can not be disclosed in this master thesis for the sensibility of this data for the company. Therefore, the main metrics used in this master thesis are the MAPE and the MAE on the normalized sensor values. Under this context, the MAE can be seen as the normalized absolute error with the maximum value seen in a particular dataset ; so the mean absolute error normalized with the maximum value of the dataset.

It is important to be aware of the limitations of the MAPE [65] as an evaluation metric. First, it can give misleading high values for target values close to $0$. Fortunately, this is not a big issue when we handle sensors for which the value is not $0$ (or at least not very often, so we can ignore those values). Secondly, the MAPE is asymmetric, this is that a method trained by minimizing it will *prefer* to predict lower values [65]. In our case this is mitigated by the fact that during the work we minimized the MAE, which is an absolute metric.

While other options are proposed in literature like the symmetric MAPE it is also true that each of them suffers from their respective problems [66] and that is the rationale behind us sticking with this choice.

# Chapter 5

# Results and Analysis

This chapter presents the results obtained by this research at Siemens Energy in both collected datasets coming from the testing facility and from the fleet of gas turbines. The initial portion of this chapter is dedicated to a discussion on the hyper-parameter search for the different models employed.

Following this, the primary insights extracted from these results are systematically presented in the subsequent sections; with each section dedicated to a single insight. To ensure a seamless understanding, a discursive approach is adopted, integrating the discussion with the results themselves. For readers who wish to bypass the detailed discussion, a concise summary of all results can be extracted from the tables within this chapter. Results show the performances of the different algorithms in both the fleet dataset and the testing facility dataset.

The critical experiments from which we draw all of our conclusions are based on the evaluation of different models in the turbines at the testing facility and in the fleet turbines. For the testing facility's dataset, we compared two different training strategies, the results of which are reported in Table 5.4 and Table 5.3. On the other hand, the results from the fleet of gas turbines are presented in Table 5.5. In all of our experiments, we aimed to estimate the low-frequency pulsation in the gas turbine in the gas turbine, which is a sensor with a central role as explained in Chapter 2. Furthermore, in the fleet dataset, we attempted to estimate a control variable, providing insights into the use of this technology for system monitoring. The results of this are provided in Table 5.6.

As a last comment before delving into the results it must be noted that only results with the final set of features are shown and this feature set was developed from the correlation analysis and after careful consideration with

| Hyper-Parameter | Range | Step |
|---|---|---|
| Learning Rate | 0.00001-0.001 | 0.00005 |
| Window Length | 10-60 | 5 |

Table 5.1: **AT-RNN hyper-parameters' space**

a domain expert. Additionally, all results have been normalized in order to maintain the data privacy of Siemens Energy.

## 5.1 Hyper-Parameters Search

A key observation when interpreting the following results is how the hyper-parameter selection was done during this master thesis.

In the data coming from the testing facility, due to the scarcity of data coming from one turbine, and due to the relevance of all segments of those turbine runs it was decided to use the same turbine run for training and validation. The testing data was always the subsequent turbine run in the following days.

On the other hand, in data coming from the fleet of turbines, having more available data, the validation dataset was always different from the training dataset. Proportions are shown in Figure 3.7, Figure 3.9 and Figure 3.10.

For simplicity, in all of our experimentation we made use of Optuna [62] with pre-defined hyper-parameter spaces that were set up according to Section 4.5. The different hyper-parameter spaces for deep networks are shown in Table 5.1 and Table 5.2. In all of our experimentation with ensemble methods we made use of time windows of 1 minute of data. This was decided since adding more parameters to tune to optuna translates into exponential complexity [62]. For the SVR and the Ridge linear models only the regularization parameter was tuned. The number of trials executed by Optuna was dynamic and simply blocked after an experiment had been running for more than 8 hours, or a maximum of 200 trials.

## 5.2 Turbines are very different

One of the major findings of this master's thesis is that the differences between two different gas turbines are substantial. Although this is expected—since each gas turbine is practically tailor-made to meet the specific needs of the customer—the differences are indeed significant.

| Hyper-Parameter | Range | Step |
|---|---|---|
| Learning Rate | 0.00001-0.001 | 0.00005 |
| Hidden Layers Length | 1-3 | 1 |
| Hidden Layer Size | 50-150 | 50 |
| Dropout | 0.05-0.3 | 0.05 |
| Number Models (for ensemble) | 2-20 | 2 |

Table 5.2: **MLP and Ensemble of MLP hyper-parameters' space**

As demonstrated in the preliminary analysis using PCA in Section 4.1, shown in Figure Figure 4.3, the data from different gas turbines *cluster* together by turbine. This suggests that models trained on a specific turbine, even being of the same type SGT-700, cannot be directly transferred and used for inference on another turbine without further consideration. Our results showed indeed that training in one turbine specific turbine and then using that model for inference on another turbine represents a dramatic failure.

Based on this preliminary result we devised an experiment using data from gas turbines of the testing facility. For this experiment we selected gas turbines for which at least 2 turbine runs were available. To ensure validity of the results we excluded those turbines that, after the first (or former) runs, had changes in their components. Then training of each model was conducted in two different ways. First, the model was trained using data of exclusively the same gas turbine for which inference was done and later training was executed on all turbines' training datasets and then inference done in their respective testing datasets. In this way, we evaluated whether a model benefited from a more extensive dataset comprised of data coming from different turbines or from a unique dataset. A comparison of the results is shown in Figure 5.1; which are drawn from Table 5.4 and Table 5.3.

At first glance one can observe that, in general, models benefited from the more extensive dataset. However, when examining closely each of the cases where major improvements were obtained (e.g. Ensemble MLP on T1 and T2) we realized this was due to the fact in the specific training dataset of those gas turbines the model did not have access to the specific operation configuration (i.e. mix of fuel) that was then used in the validation dataset. Therefore, the initial high error was due to a lack of data diversity in their respective datasets. The improvement, for turbine T1 and T2, can be seen in Figure 5.2. Here, it is possible to observe how in one dataset the improvement is due to a more precise estimate of the low frequency pulsation. Perhaps a more interesting result is later where we see how the network, despite still

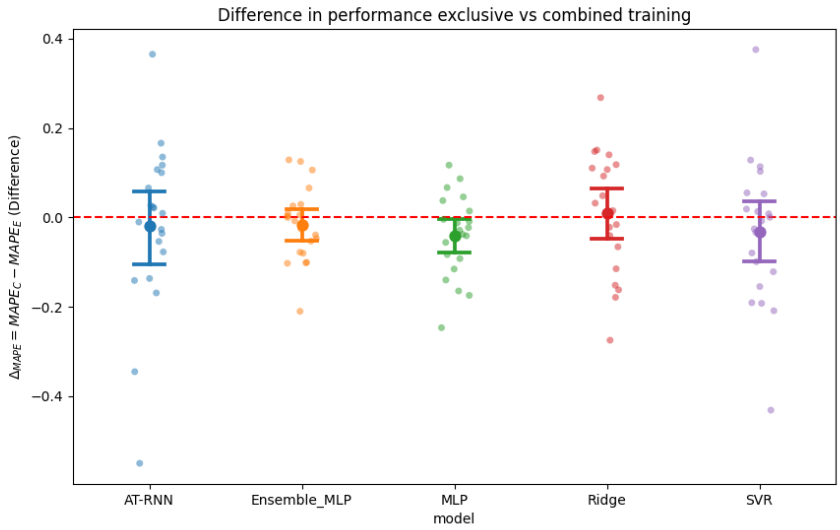



Figure 5.1: **Difference in performance between exclusive and combined training**: In this figure it is possible to see a comparison between training models solely with training data coming from an exclusive gas turbine or from multiple gas turbines. Positive values of the difference imply the model achieved better performance in that dataset by using exclusively data from that gas turbine while negative results imply it benefited from the more extensive dataset.

doing a non-negligible error on the estimation process, learns to capture a transition that was not observed at all before. Furthermore, for this dataset, we experimented by attempting to fine-tune the pre-trained network on the specific datasets, but the results were not optimal, likely due to the data issue mentioned above (i.e., the fine-tuning effectively erased the useful information the network had learned from other turbines).

On the other hand, for some turbines it is detrimental to use data from other turbines in the dataset. Carefully examining the source data one observes the opposite pattern of the turbines for which there is a huge improvement. The training datasets of such turbines were *complete* in the sense that had all the operating points for which the same gas turbine was then tested. This can be examined from the point of view of the statistical distribution that originates the data points of the different gas turbines. By adding more data from other gas turbines we are essentially *contaminating* our original dataset and as a consequence *confusing* our model.

From this discussion, we conclude that , as general guideline, it is only

useful to add data from multiple gas turbines when the data for the turbine under examination does not have all the operating points for which it will be used. Therefore, the apparent improvement suggested by Figure 5.1 must be seen through the lenses of the data that underlies that result. Thus, it was decided for larger datasets coming from the fleet of gas turbines to use data exclusively from each specific dataset, which corresponds to a unique gas turbine.

The idea of transfer learning was no further studied during this master thesis due to the data problems mentioned earlier. However, it remains an open question whether it is possible to train a larger model from a big corpus of gas turbines which is then fine-tuned in a complete dataset (i.e. containing the operating points for which it will be used) and whether this signifies an increased performance.

Another idea that remains unstudied is whether a domain adaptation scheme would be viable to *port* measurements present only in a specific gas turbine to another. This would mean whether it is possible to train in a gas turbine that is used in the same operating points of another and , via domain adaptation, train an algorithm that emulates the missing sensor in the later gas turbine.

## 5.3 Ensemble model success

As shown in the results the performance of the ensemble of feed forward networks is remarkable. This network is often the best performer both in the data coming from the testing facility's gas turbines and the fleet's gas turbines. This can be seen when analyzing the results obtained in Table 5.5 ,Table 5.3 and Table 5.4.

The most interesting result from a theoretical point of view is that, aligning with the theoretical background of ensemble methods [32], the variance of the predictions is substantially reduced yielding better predictions. This result is clear when observing a comparison between the predictions done for the low frequency pulsation both by a single MLP and an ensemble of them in Figure 5.3.

We attribute this increased performance to the structure of ensemble that we used. Indeed, the issue that a single MLP might be suffering is that the output might rely too much on input sensors that might be altered to some random noise not related with the actual state of the machine, for instance a flow turbulence. By employing our ensemble method, which implies sampling from the different sensors across a larger time window, thus having networks

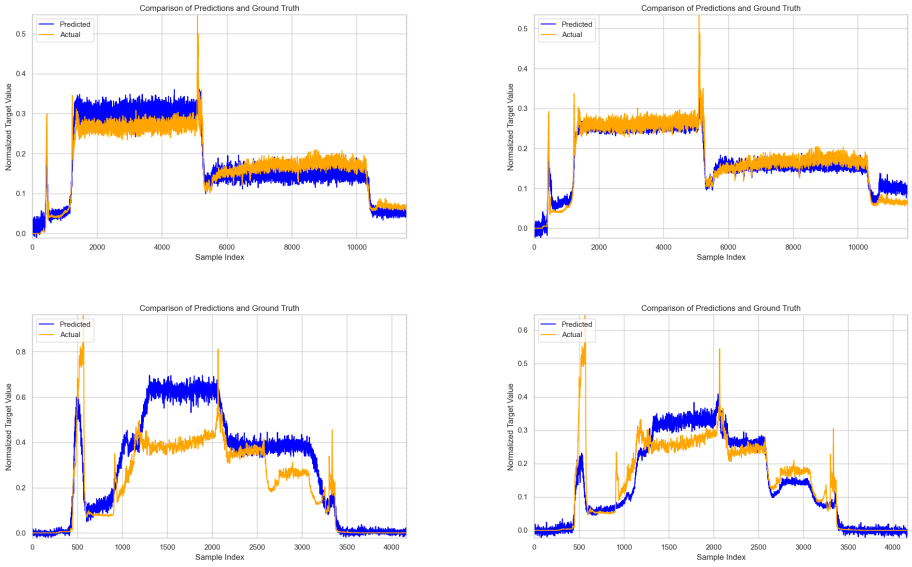



Figure 5.2: **Exclusive Vs Combined Training**: In this figure predictions done by an ensemble of MLP on the turbines $T1$ and $T2$ of the testing facility dataset is shown. Here, one can see how the network benefited from the more extensive dataset because it had the chance to observe turbine operation points not present in the $T1$ and $T2$ training dataset.

that do not rely on the exact same instants we obtain that fluctuations in single sensors are compensated and the accuracy of the prediction is increased.

Furthermore, in the datasets for which we were able to perform grid searches (fleet's gas turbines) , we obtained the theoretically expected result for which this increase is not infinite but it reaches a plateau since a requirement for performance increase is to have model independance. For those datasets the optimal number of predictors found were between 2-5. On the other hand, for gas turbines in the testing facility it was decided to use a total of 5 networks.

## 5.4 The baselines are very good

One of the most surprising results, evidenced throughout all the work, is that simple linear regression models often provide reasonable estimates of the variable under examination. As evidenced in the results coming from the testing facility , shown in Table 5.3, and also results from the fleet of gas turbines in Table 5.5.

The good results from linear regression models came as a surprise, as a gas turbine system is expected to present high non-linearities. However, after careful examination of results one can find a physical meaning behind such results. Indeed, a gas turbine is a physical system , highly nonlinear, but employed in a specific operation point. From a physical point of view one can imagine that , close to that specific point, the system can be *linearized*. Thus, explaining the success of those baselines.

However, this success relies on the assumption that the gas turbine is operated close to a specific operation point. This assumption is not valid, for instance, in the gas turbines in the fleet. The discussion that follows addresses this observation.

### 5.4.1 Where deep learning takes huge advantage

By observing the results obtained in the data coming from the testing facility's gas turbines one might rapidly conclude that, despite complex architectures obtaining better results, the difference is not big enough to justify the added complexity of such methods.

To better understand this phenomenon we will examine the result obtained in a dataset coming from gas turbines coming from more dynamic conditions than the ones in the testing facility. As one can observe in Figure 4.4, in the testing facility the active load is kept constant for long periods of time (i.e. the turbine is operated in a very stable condition). In contrast, in the dataset

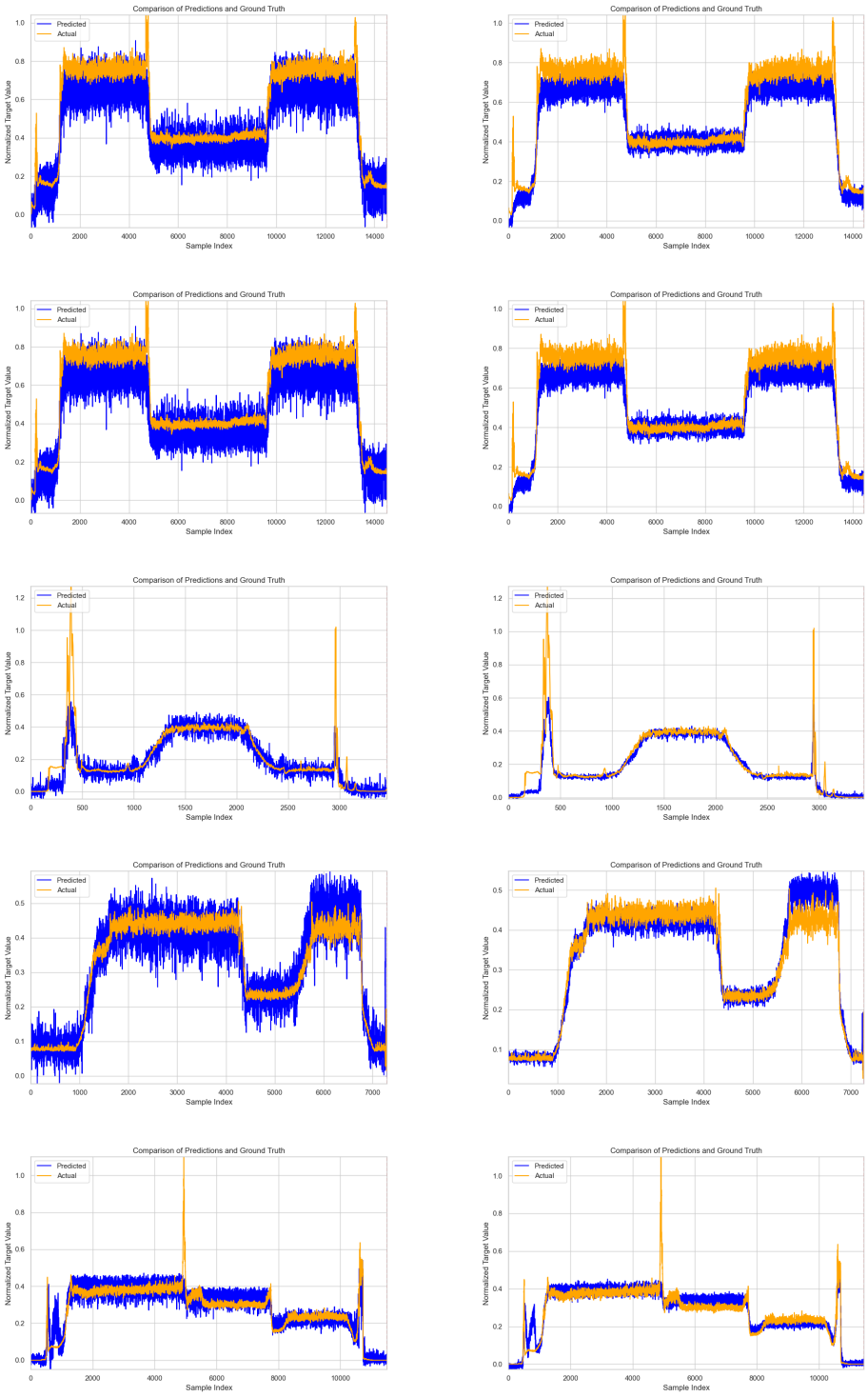



Figure 5.3: **Ensemble Stability**: The provided figure illustrates a comparative analysis between the application of a single MLP (left) and an ensemble of 5 MLP (right) used for making predictions in runs of testing facility's gas turbines. As can be observed from the images, the ensemble approach tends to yield predictions with reduced variance, aligning with our expectations.

| Turbine | Linear | SVR | Ridge | MLP | Ensemble MLP | AT-RNN |
|---|---|---|---|---|---|---|
| T1 | 0.2547 | 0.5441 | 0.2723 | 0.1764 | **0.1428** | 0.1680 |
| T2 | 0.4883 | 0.3055 | 0.3877 | 0.3344 | **0.2994** | 0.4444 |
| T3 | 0.1912 | 0.1730 | 0.1331 | 0.1686 | 0.1140 | **0.1031** |
| T4 | 0.4448 | 0.2247 | 0.2918 | 0.1915 | 0.1383 | **0.0890** |
| T5 | 0.1837 | **0.0635** | 0.1627 | 0.1130 | 0.1089 | 0.1707 |
| T6 | 0.5449 | 0.2118 | 0.2119 | 0.3556 | 0.2108 | **0.1637** |
| T7 | 0.5980 | **0.2149** | 0.2246 | 0.2786 | 0.2167 | 0.2731 |
| T8 | 0.2250 | 0.1259 | 0.0901 | 0.1127 | 0.0859 | **0.0737** |
| T9 | 0.8904 | 0.4558 | 0.3817 | 0.2282 | **0.1867** | 0.2884 |
| T10 | 0.0828 | 0.1471 | **0.0635** | 0.1234 | 0.0881 | 0.0754 |
| T11 | 0.1890 | 0.1236 | 0.1409 | 0.1900 | **0.1219** | 0.7289 |
| T12 | 0.6351 | 0.2897 | 0.3531 | 0.2579 | **0.2064** | 0.3061 |
| T13 | 0.4908 | 0.1661 | 0.1752 | 0.1557 | **0.1204** | 0.1819 |
| T14 | 0.3301 | 0.4380 | 0.2900 | 0.2917 | **0.2444** | 0.3273 |
| T15 | 0.1355 | 0.2421 | 0.1018 | 0.1255 | 0.1222 | **0.1078** |
| T16 | 0.3951 | 0.2767 | 0.1778 | 0.1767 | 0.1013 | **0.0841** |
| T17 | 5.6530 | **0.0953** | 0.1151 | 0.2525 | 0.2219 | 0.2592 |
| T18 | 0.1032 | 0.1078 | **0.0624** | 0.1312 | 0.1028 | 0.1470 |
| T19 | 0.1481 | 0.2160 | 0.1145 | 0.1376 | **0.1048** | 0.1475 |
| T20 | **0.0602** | 0.1252 | 0.0684 | 0.1141 | 0.0752 | 0.0958 |
| T21 | 1.4400 | 0.2876 | **0.1145** | 0.2439 | 0.1903 | 0.1478 |
| Total Best | 1 | 3 | 3 | 0 | **8** | 6 |

Table 5.3: **Performances on testing facility data (LFP) sequential**: This table shows the Mean Absolute Percentage Error obtained by different models in the datasets. The training is done from data coming **exclusively** from the specified gas turbine $T_i$.

| | Linear | SVR | Ridge | MLP | Ensemble MLP | AT-RNN |
|---|---|---|---|---|---|---|
| T1 | 0.2143 | 0.1127 | 0.1102 | 0.1352 | **0.0897** | 0.2751 |
| T2 | 0.1397 | 0.1132 | 0.1128 | **0.0875** | 0.0892 | 0.0991 |
| T3 | 0.3397 | 0.2760 | 0.2839 | 0.1642 | **0.1433** | 0.2693 |
| T4 | 0.4253 | 0.1451 | 0.1399 | 0.0993 | 0.0908 | **0.0622** |
| T5 | 0.2574 | 0.4391 | 0.4308 | 0.1996 | **0.1747** | 0.5359 |
| T6 | 0.4057 | 0.2121 | 0.2149 | 0.1910 | **0.1329** | 0.1849 |
| T7 | 0.2980 | 0.2336 | 0.2397 | **0.1041** | 0.1151 | 0.1364 |
| T8 | 0.2106 | 0.2395 | 0.2376 | 0.1002 | **0.0915** | 0.2089 |
| T9 | 0.1670 | 0.2649 | 0.2667 | 0.1723 | **0.1065** | 0.2346 |
| T10 | 0.3437 | **0.1552** | 0.1561 | 0.1902 | 0.1942 | 0.1755 |
| T11 | 0.2361 | 0.0979 | 0.1002 | 0.1068 | **0.0964** | 0.1785 |
| T12 | 0.5794 | 0.1906 | **0.1742** | 0.2489 | 0.1985 | 0.3720 |
| T13 | 0.4615 | 0.2188 | 0.2237 | 0.2016 | 0.1461 | **0.1047** |
| T14 | 0.2781 | 0.2291 | 0.2242 | 0.1761 | **0.1437** | 0.1583 |
| T15 | 0.2569 | 0.0873 | **0.0858** | 0.1398 | 0.1305 | 0.1170 |
| T16 | 0.3908 | 0.2898 | 0.2959 | 0.2143 | 0.2301 | **0.2010** |
| T17 | 0.1474 | 0.2236 | 0.2254 | 0.2146 | 0.1827 | **0.1180** |
| T18 | 0.3027 | 0.1000 | **0.0943** | 0.2482 | 0.2280 | 0.1366 |
| T19 | 0.1301 | 0.0944 | **0.0925** | 0.1088 | 0.1047 | 0.1117 |
| T20 | 0.2276 | 0.1802 | 0.1755 | 0.0913 | **0.0784** | 0.1189 |
| T21 | 0.2496 | 0.2542 | 0.2546 | 0.1038 | **0.0876** | 0.1741 |
| Total Best | 0 | 1 | 4 | 2 | **10** | 4 |

Table 5.4: **Performances on testing facility data (LFP) combined**: This table shows the Mean Absolute Percentage Error obtained by different models in the datasets. The training is done from data coming from **all** the training datasets of gas turbines $T_i$.

| Dataset/Sensor | Baseline Active Load | Linear | Ridge | MLP | Ensemble MLP | AT-RNN |
|---|---|---|---|---|---|---|
| Medium (LFP) | 0.0895 | 0.0772 | 0.0766 | 0.0650 | 0.0525 | **0.0510** |
| Large (LFP) | 0.0857 | 0.0646 | 0.0645 | 0.0522 | **0.0482** | 0.0566 |
| Dynamic (LFP) | 0.4573 | 0.8885 | 0.4844 | 0.1580 | **0.0886** | 0.1723 |

Table 5.5: **Performances on fleet data (LFP)**: This table shows the Mean Absolute Percentage Error obtained by different models in the different datasets, computing the low frequency pulsation from other 16 sensors spread across the gas turbine.

coming from fleet's turbines we observe a much more dynamic active load pattern , as seen in Figure 3.6, Figure 3.8 and Figure 3.10.

This dynamicity is probably *turning on* non-linearities of the system that do not appear when the turbine is operated in a specific point. Perhaps, the limit example of this behaviour comes from the small and dynamic dataset in which the gas turbine is used as a mechanical drive. For this gas turbine results show a dramatic failure of linear models (and all other baselines) to capture the dynamics of the systems and evidence how , on the contrary, deep learning methods are able to capture it and provide with reasonable estimates for the low-frequency pulsation. The result is shown in Figure 5.4.

In situations where the behaviour of the gas turbine is highly dynamic the expressive power of a linear model is not enough to generalize well or to capture the dynamics of the system. In such cases deep networks perform vastly superior to linear model.

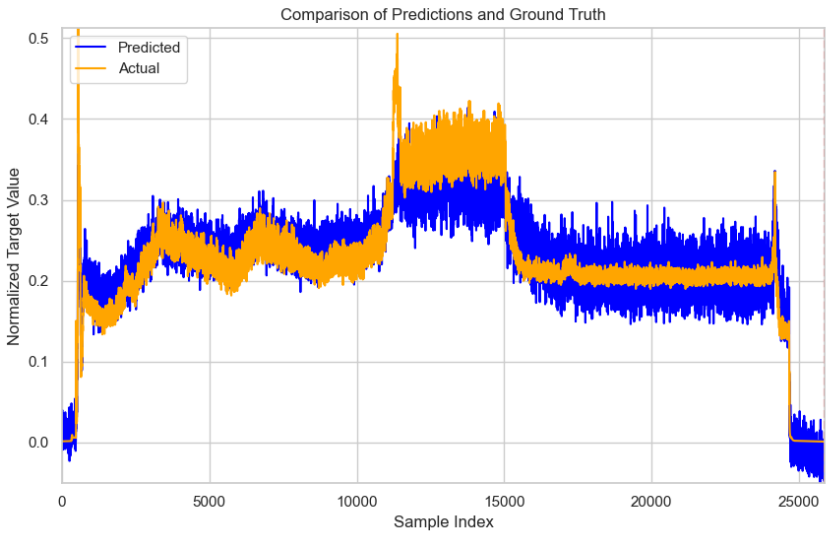
Another key observation is the improvement of deep learning methods in situations with larger amount of samples. Indeed, results in Table 5.3 are lower than results in Table 5.5. However, this can also be attributed to the less-pronounced changes in active load in fleet's gas turbines.

## 5.5 Robustness of model over time

Another key analysis of the models developed in this master thesis is to observe the performance behaviour of the software sensor over a long period of time.

By the usage of gas turbines it is expected that they wear and tear, besides potential bigger changes that might occur in the gas turbine. Ultimately, such small or big changes will end up by changing the latent probability distribution from which the data samples of a specific gas turbine are generated. Therefore, we expect a drift in the performance of the gas turbine over time. This result was also obtained by [2], in Siemens Energy's gas turbines.

We designed an experiment , by taking inspiration on [2], for assessing this performance drift. Following the training of the software sensor over one year of collected data (2017) we proceeded to perform testing in the following years. An analysis shown in Figure 3.10 and in Figure 3.11 was conducted to ensure that the operation mode of the gas turbine remained equivalent during testing and training, so that changes in the software sensor performance could not be attributed to a change in the operative behaviour of the gas turbine. For performing grid search in this dataset we made use of a part of the data from 2017 as validation dataset.




(a) Ensemble Feed Forward Model


(b) Ridge Linear Model

Figure 5.4: **Feed forward ensemble vs Ridge Linear Model** This figure offers a comparative analysis between an ensemble MLP and a linear model equipped with ridge regularization, both applied to the highly dynamic dataset derived from a fleet of turbines. The focus of this comparison is the low-frequency pulsation, which serves as the target variable. The results clearly illustrate that the baseline model was not successful in capturing the dynamics of the system.

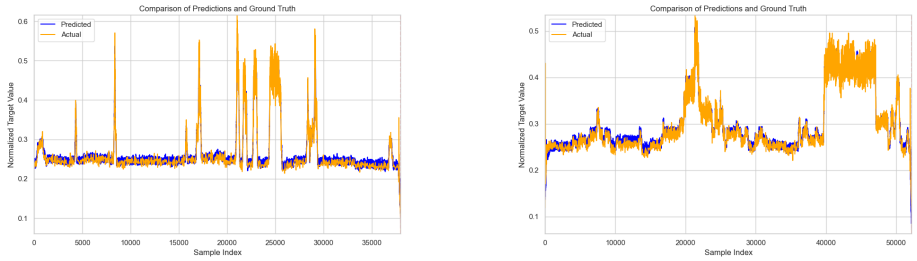



Figure 5.5: **Large Fleet Dataset Results**: These figures show the predictions done for the low-frequency pulsation by an ensemble of MLP in the large dataset from the fleet's turbine. To the left a result from the validation dataset in 2017 and to the right a result during the first 6 months of model validity

The first side result from this experiment is that the increase in training data available for the models boosts their performance with respect to the medium dataset, coming from the same gas turbine, but with less training data available. Remarkable results obtained by this model can be seen in Figure 5.5 where predictions for the low-frequency pulsation are shown for validation dataset and the first months of model validity after training.

The evolution of the Mean Absolute Percentage Error over time is shown in Figure 5.6. In this figure each point represents a turbine run, which can be longer or shorter, which is encoded in the size of the corresponding point. It can be observed that the performance of the software sensor remains equivalent during the first 6 months post-training. However, in July 2018 a substantial degradation of the performance of the software sensor is observed. After this result was obtained further exploration was conducted to better understand *what* changed and what might be the reason of this degradation. We found that the decrease in performance appears after an intensive usage of the gas turbine followed by a full stop of a couple of weeks. After this, predictions degradate indicating some change on the gas turbine. However, we were not able to pinpoint from available information what is the exact reason for this deterioration.

It is worth noticing that the deterioration is not substantial and estimates given by the software sensor are still within a reasonable range.

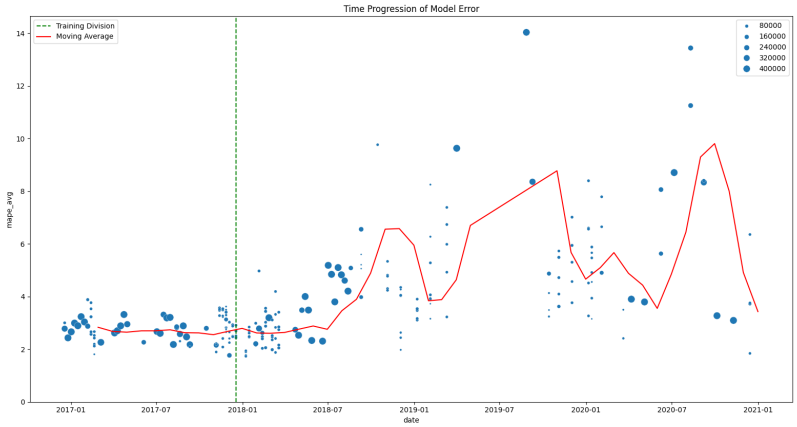



Figure 5.6: **Evolution of Software Sensor Performance**: This figure depicts the progression of a software sensor's efficiency over several years. The sensor maintains its performance for six months after being trained on a year's worth of data, but a significant decline in performance is noted thereafter.

## 5.6 Software Sensors to Replace Physical Sensors?

One of the potential usages of a software sensor in gas turbines posed at the beginning of this master thesis was to make use of them to potentially replace a physical sensor present in the gas turbine. This usage is similar to the one studied in [40] for replacing flame detectors and was also addressed by [12] with a gas turbine simulation. This application would be of substantial value for both company and customers as it would allow to , potentially, save costs in sensors that are expensive due to the extreme conditions under which they operate.

Results from this master thesis show that, with the current development, this application would not be possible or safe. This conclusion is given after analyzing the residuals obtained by the best performing algorithms. As it can be seen from Figure 5.7 the residuals exhibit that the models perform particularly bad for large pulsation values , which are the exact peaks that the sensor is aiming to monitor. In other words, a pulsation's sensor aim is to capture dynamic pressure patterns that imply the existence of some undesired pressure wave excited by the resonance frequencies of the gas turbine. Those pressure waves are undesirable as they might damage mechanical parts of

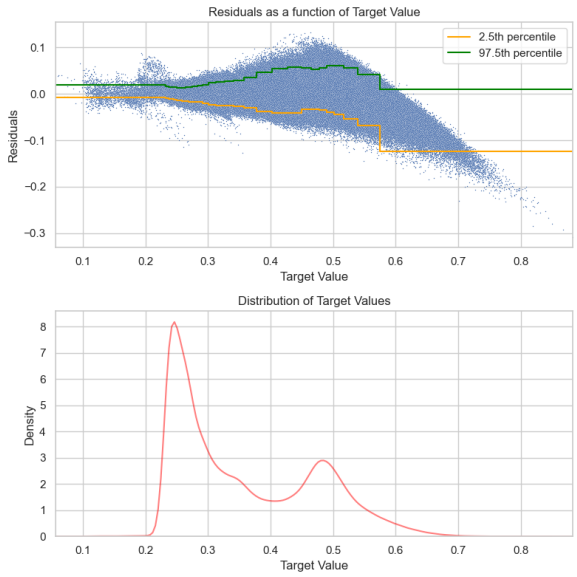



Figure 5.7: **Residual Analysis on fleet's medium dataset**: This figure shows the residuals as a function of the target value for the medium sized dataset coming from the gas turbines fleet. The algorithm under examination is the ensemble of MLP

the gas turbine and our models perform particularly bad on detecting such peaks. This is probably due to the rarity of them in the data combined with the instability of the underlying physical system during those transitions.

This limitation is profound and not simply attributable to the single models and their inability to capture those peaks; it is not a model error. The limitation, as can be seen also from Figure 5.7, comes from a data limitation perspective. This is, in the available data, we have gas turbines operating most of the time adequately. This , in practice, means that occurrences of the thermoacoustic instabilities in data [35] are rare; as they should be since gas turbines are constructed, by design, to avoid such pressure waves. But, from a ML point of view this is a big problem since our models did not see, or saw rarely, this behaviour during training, so it is impossible or hard for them to learn this behaviour. From the data engineer perspective it might seem possible to find data with such peaks but this requires both domain expertise and extensive knowledge of the operation of gas turbines. Furthermore, peaks present in the low frequency pulsation in our datasets are often attributable to the operation itself of the gas turbine (i.e. change in the ratios of fuel given, turn on/off of gas turbine, etc. ) and not of the kind that the sensor is explicitly placed for (i.e. unintended high low-frequency pulsations coming

from a thermoacoustic resonance).

## 5.7 Fault Isolation?

The results show that it is possible to create a *software sensor* that , under certain circumstances, provides accurate estimates of variables in the gas turbine. However, as discussed in the previous section, the potential of software sensors for replacing a physical sensors remains limited at the moment due to the safety issues this would create. Nonetheless, their potential still remains high as they can very likely be employed for predictive maintenance in the gas turbines. In this section, we present , based on our results the potential usages of this technology in this field; supported by our results.

### 5.7.1 Software Sensor for Overall Machine Status

The previous results were focused on the low frequency pulsation prediction in the gas turbines. This prediction was done using 19 sensors from all sections of the gas turbine: compressor, combustion chamber and exhaust. The usage of those sensors was carefully done after discussion with domain experts and the minimal set of requires sensor was aimed to be chosen.

However, this practically implies that , when using the model for inference, in the case of a difference between the model prediction and the physical sensor it is not possible to pinpoint a failure on a specific part of the gas turbine. Indeed, given a software sensor failure, the physical failure can come from *anywhere*. It is possible for it to be a real sensor failure, but it can also be due to a change in the overall health of the machine which changes the relationship between parameters, or a failure of one of the input sensors that is used to generate the prediction. In this sense, the software sensor solution is very similar to having an autoencoder as in [2]. Indeed, given a model deterioration without further information it is only possible to conclude that something changed in the machine, but not exactly isolate what. Therefore, as in [2], it acts only as a health parameter of the gas turbine. This is indeed shown in Section 5.5, where a deterioration in the model performance is linked with a change in the machine. Therefore, a use-case is to use this algorithm to detect unexpected changes.

As an important note for this application is that also changes in the operation mode of the gas turbine will lead to varied performance of the model. Therefore, before drawing conclusions, one must validate that the underlying

data is drawn from the same distribution. This can be done in a simplified way by observing that the active load pattern of the machine remains unchanged.

### 5.7.2 Software Sensor for Control Loop Monitoring

Perhaps one of the most exciting usages of a software sensor is to introduce it into a control loop. Here, we will discuss this idea. The concept is to use the software sensor to predict the value of a control variable from other variables that depend on the control variable itself. This allows for detection of faults in the control device itself or sensor used by it.

To exemplify this usage, we focused on the IGV measurement. We attempted its prediction from a small set of variables within the same turbine section (compressor) from other variables that are directly determined by this control variable (i.e. pressure after compressor, inlet differential pressure , etc. ). The intrinsic relationship between those variables makes predictions done by our software sensor to be *extremely* accurate , as shown in Table 5.6 and Figure 5.9. This is expected as we are predicting a value that is expected to have a direct effect on the variables that we are considering.

Reflecting on the original, simplified, PGM that we used to model the gas turbines we are effectively learning the inter-dependency of those variables by creating a link among them. The kind of fault that we are able to identify, due to the specificity of the software sensor devised, and to its high accuracy is to detect faults in the control system. The schematic of this implementation is shown in Figure 5.8.

Notice that in case of discrepancy between the software measurement and the physical measurement we still are not able to conclude definitely whether it is the software sensor or the physical one that is failing. However, in combination with other predictive maintenance strategy based on the performance of the compressor one can adequately identify a failure in the IGV control system. This is, if we observe that the performance obtained when using the predicted IGV and the predictors' values is consistent with the expected value of performance one can be confident that the *real* value of the IGV is the one predicted by our software sensor and not the measured one. Effectively, this technique would allow to pinpoint the failure to the IGV control system.

This example is highly specific, and it was the product of extensive discussion with a domain expert. However, the concept can be generalized to other systems with a similar *loopy* structure to , in a schematic way, apply redundancies to the different control systems of the gas turbine and in that way

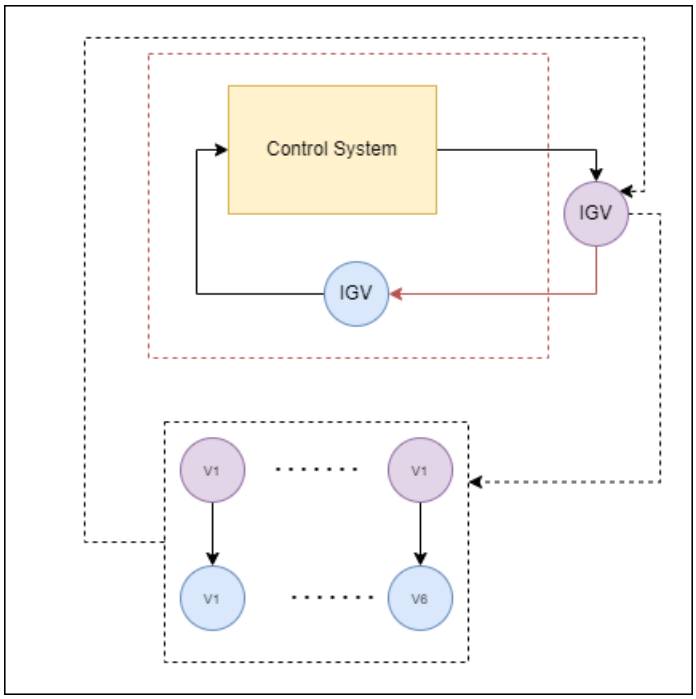



Figure 5.8: **Usage of Software Sensor in Control Loop**: This diagrams shows the usage of a software sensor for detecting failures in the IGV control system of the gas turbine.

implement a mechanism of Failure Detection and Isolation.

## 5.8 Data Pipeline Analysis

As discussed in Chapter 3, the inefficiency and unfeasibility of manual download of testing facility's turbine data necessitated of the development of an automatic procedure for data collection and subsequent processing to create datasets for the conducted experiments.

Results for the developed data collection system are shown in Figure 5.10. It shows that the developed system enables rapid construction of datasets. In this benchmark 250 sensors are queried for a set of around 100 turbine runs. The obtained median turbine run download time is around 10 seconds, with some outliers present due to particularly large turbine tests.

This system proved to be valuable during this master thesis to rapidly iterate and explore feature and dataset modifications. It remains a useful tool for future work conducted in this department to effectively, in combination with the procedures described in Section 3.4, make use of the data available.

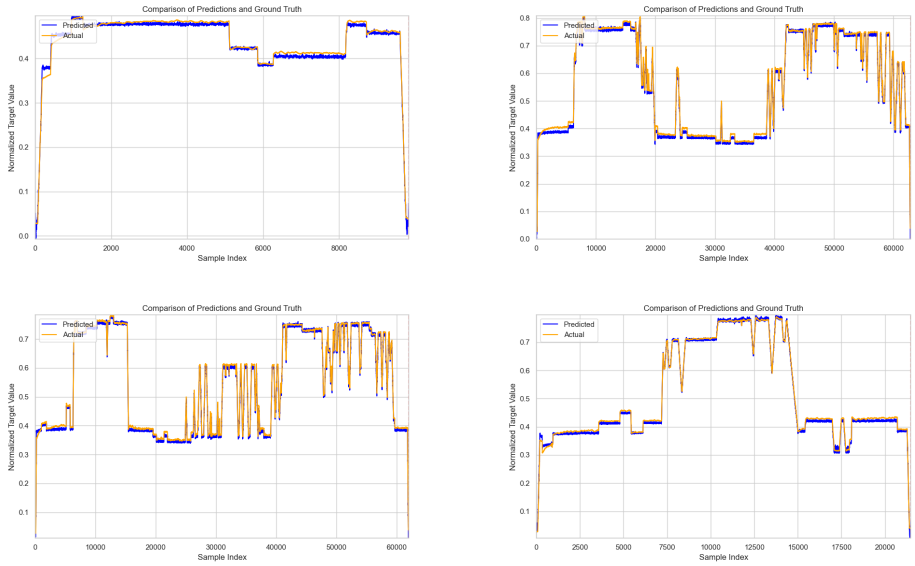



Figure 5.9: **IGV Linear Model Regression**: In this figure it is possible to see predictions done by a simple linear regressor on the control variable IGV from other 6 variables that react to it. We can observe the extreme level of accuracy of the predictions. This is expected as we are using feature variables that directly depend on the IGV value.

| Dataset/Sensor | Linear | Ridge | MLP | Ensemble MLP | AT-RNN |
|---|---|---|---|---|---|
| Medium (LFP) | **0.0122** | 0.0128 | 0.02056 | 0.01753 | 0.01236 |
| Large (LFP) | 0.0224 | 0.0224 | 0.03039 | 0.02206 | **0.0183** |
| Dynamic (LFP) | 0.1244 | 0.1068 | 0.3800 | **0.0968** | 0.1505 |

Table 5.6: **Performances on fleet data (IGV)**: This table shows the Mean Absolute Percentage Error obtained by different models in the different datasets, computing the IGV from 6 sensors in the compressor. Datasets shown are very different in characteristics

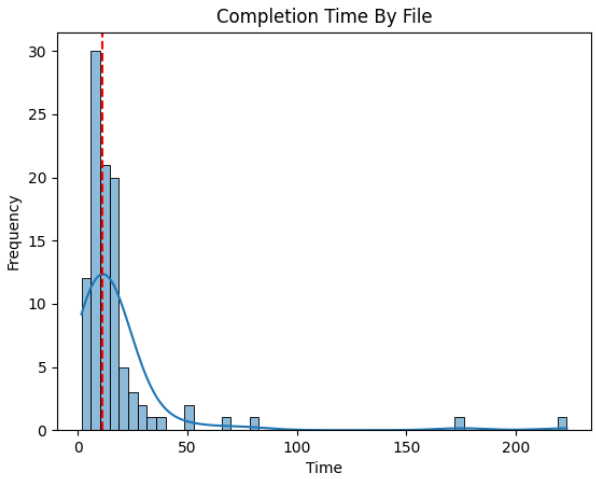



Figure 5.10: **File Download System Benchmark**: This figure shows the distribution of download time of a complete file with over 250 sensors, by using the proposed parallel approach.

On the other hand, the overall system developed for sample extraction from datasets, as described in Section 3.5, proved invaluable. A unique sample generation by task ensured that models were indeed being compared by making use of the same available information. Furthermore, adopting a PyTorch-like solution [28] proved to be beneficial when dealing with larger datasets, as in Section 3.3, for which holding all samples in memory was not efficient.

## 5.9 Data Quality Issues That Emerge in Model Testing

Throughout this master thesis, we have reiterated that the development of Machine Learning Algorithms is not a linear process. This section exemplifies how data quality issues can surface late in the development stage, prompting a re-evaluation of the existing data pipeline and the introduction of new cleaning procedures. For illustrative purposes, we shall discuss a data set that was excluded from all other analyses due to quality reasons identified during this thesis. As we have established in Chapter 2, data quality is a paramount factor in the development of a Machine Learning Algorithm as the efficacy of the models directly correlates to the quality of the data used for training them [16]. We realized the data of a low-frequency pulsation from a specific gas

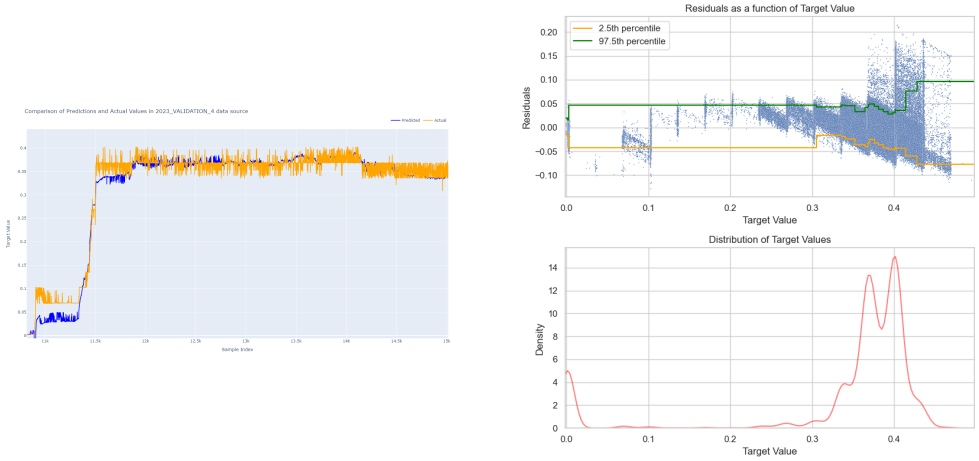



Figure 5.11: **Quantized Signal Effect on Model Outputs**: The quantization pattern emerging from the regression of the low-frequency pulsation based on a ridge linear-model based software sensor. Only after observing this result it was possible to understand the data quality issue in the raw data collected.

turbine was quantized only during the model testing phase.

This quantized data revelation came to light while analyzing the residuals of a linear model attempting to predict the raw data, as depicted in Figure 5.11. The pattern of these residuals mirrors what is typically found in electronics when an analog signal is quantized to convert it into a digital signal. While no electronic device caused the quantization in our dataset, it was apparent upon reviewing the raw signal that ,for reasons unknown to us, the original signal had indeed been quantized, as shown in Figure 5.11.However, this fact was not uncovered during the initial data processing, as its occurrence was highly unexpected.

Due to this unexpected discovery, we decided to discard the dataset, as the results could not be utilized effectively. An open , very interesting, question remains of whether the interpolation done by the linear model in this scenario was accurate. However, lacking access to the pre-quantized signal, it was impossible to evaluate this.

# Chapter 6

# Conclusions and Future work

During this master thesis we conducted a successful study based on the Cross Industry Standard Process for Data Minings on the application of Machine Learning Algorithms for Siemens Energy's gas turbines. The thesis aimed to answer whether it is possible to build a data-driven software sensor for specific gas turbines. It also aimed to answer how this technology can be employed in the gas turbine for either reducing costs or predictive maintenance.

In this chapter, we will present the major results obtained from this master thesis, a discussion of the limitations of the developed approach, future avenues to pursue and a reflection on the ethical issues stemming from this AI usage.

## 6.1 Conclusions

This master thesis successfully demonstrated the efficacy of implementing a cyclical development model in an industrial context, as proposed by the Cross Industry Standard Process for Data Mining [13]. As a result, we have successfully integrated this model within the Siemens Energy R&D department, establishing the necessary software infrastructure for data collection, preparation, and algorithm development.

This project placed specific emphasis on the considerations necessary for constructing a dataset suitable for algorithm development and experimentation. These considerations were essential for addressing the initial project queries and for future project development. A robust, flexible, and efficient data preparation system was proposed and built, which can be used for further experimentation and data-driven solutions.

As a result, we developed a dynamic interface that enables data extraction

from the existing Siemens Energy infrastructure in an efficient and rapid way. The system includes a cleaning procedure for automated data extraction from the raw data. Additionally, a solution based on constraint programming was proposed for data selection. This solution allows for the handling of specific constraints regarding desired features and potential missing sensors in certain turbine runs by making use of constraint programming for solving this combinatorial optimization problem.

Equally relevant, this master thesis highlighted the data challenges tied to gas turbines from both the testing facility and the fleet. The essential considerations for further result analyses and data limitations, particularly regarding data missingness and scarcity, were thoroughly discussed. A crucial consideration is the data limitations concerning the rarity of events intrinsically designed to be captured by a sensor (i.e., unexpected events). This constraint suggests that the algorithms developed may not be sufficiently reliable to replace a physical sensor holding critical value in the machine. This was evidenced through the residual analysis conducted on a software sensor predicting low-frequency pulsation, where it was observed that the model tends to underestimate results for higher target values.

Furthermore, we demonstrated the feasibility of constructing a software sensor that yields accurate and robust results, provided it's trained with diverse data. We utilized a range of methods, from baseline to more complex deep learning architectures, and observed their strengths and limitations. Our observations led us to conclude that, despite the high autocorrelation of data, the time-awareness of algorithms is critical. It helps filter out noise that might be induced by turbulence in gas turbine operation. In particular, we found that ensemble methods are especially pertinent when dealing with potentially noisy data.

We successfully conducted a robustness analysis of such solutions and highlighted their main limitations concluding that while usage for replacing a physical sensor is still not viable it is indeed possible to think on integrating them into a gas turbine system for different kinds of health monitoring. Our analysis shows that a promising avenue is opened for monitoring control systems or general check of unexpected machine changes which might be not detected by operators. This application was further supported by an analysis done of the performance of a model over time and characterizing how changes in the gas turbine are reflected in the performance of a model of this kind.

We conducted a comprehensive robustness analysis of these solutions, underscoring their primary limitations. We concluded that while replacing a physical sensor is not yet feasible, the integration of these solutions into a

gas turbine system for various health monitoring aspects is indeed possible. Our analysis suggests a promising direction for monitoring control systems or general checks for unexpected machine alterations, which may go undetected by operators. This application was further reinforced by an analysis of a model's performance over time, characterizing how changes in the gas turbine impact the performance of a model of this kind.

## 6.2 Limitations

We can observe from the results of this master thesis that a significant limitation is data quality and , ironically, scarcity of data.

A surprising constraint discovered throughout this work is that despite having vast amounts of data, data scarcity remains a real issue. It manifests in various ways, requiring a separate discussion for each dataset used.

In the data derived from the testing facility, we have a considerable quantity of different gas turbines. However, when conditions such as having a similar set of valid sensors during the run are imposed, the dataset is greatly reduced. Indeed, the final dataset contains *only* 21 turbines with the selected features, for which at least 2 runs with the same set of valid sensors are available. Despite this number of turbines being more than sufficient, our algorithms had to learn from, in some cases, a unique turbine run to perform inference on a subsequent run. As highlighted in Section 5.2, this is a significant limitation in datasets that do not contain the required operative points for which inference is performed. Nonetheless, this was crucial as it confirmed an expected fact: the training dataset must contain the operating point for which the algorithm will be used for inference.

Data scarcity is also an issue for the development of a reliable algorithm in the sense that instances of data where an *unexpected* rise in low-frequency pulsation is very rare (fortunately for Siemens Energy business, as it implies that gas turbines are reliable). This limitation applies to both datasets. However, it poses a significant issue for Machine Learning Algorithms as it prevents them from learning under what circumstances those exceptional, but necessary situations to capture by a software sensor, occur. This fundamental problem of the underlying dataset indicates another limitation of our developed solution: it is not entirely reliable to replace a physical sensor and should not be used to substitute a critical measurement of the machine (such as the low-frequency pulsation). Physical sensors remain key in identifying those extremely rare but dangerous conditions.

The final limitation discovered relates to data quality issues in the dataset

from the fleet of gas turbines. This stems from the engineering challenge of storing this massive amount of data. However, the non-uniform strategy by which data is compressed for storage requires careful consideration when analyzing any dataset from this source. Unexpected issues, as shown in Section 5.9, are not uncommon and pose a significant limitation. Furthermore, the underlying data presents a non-uniform sampling rate, and although, as described in Chapter 3, this issue is reasonably resolved by imputing values using linear interpolation, this base assumption should be revisited given access to the original data.

## 6.3 Future work

The intersection of this work with predictive maintenance concepts is substantial. Future work based on this solution could delve into ideas like the integration of this type of software sensor into a control loop for monitoring control systems.

A critical shift in focus for producing meaningful results supported by material evidence involves obtaining data of specific system failures, akin to [10] method. Understanding how to utilize the discrepancy between software and physical sensors to draw conclusions and identify particular system failures is crucial. This shift is necessary as the implementation of methods to detect failures, tested by injecting failures manually into a specific sensor as done in [12], does not accurately represent how a real failure will appear in data. Such a failure would likely affect multiple sensors simultaneously, complicating the task of identifying the problem's source.

By constructing blocks of sensors concentrated in specific areas of the gas turbine, we can enhance our **belief** in the correct functioning of those sensors or turbine areas and assist operators in performing fault isolation.

Another intriguing area of exploration involves the feasibility of a domain adaptation scheme for gas turbine data. It is conceivable to train a model on a gas turbine with a specific available sensor and then attempt to transfer this knowledge to another gas turbine lacking that sensor, assuming similar operating modes for both turbines. It would be insightful to determine whether it is possible to *align* the latent representations to artificially implement a virtual sensor in another gas turbine. A likely limitation of this scheme is the inherent difference between two gas turbines, despite their similarities, as discussed in Section 5.2.

A final unanswered question pertains to the analysis conducted when training on multiple turbines simultaneously. Is it feasible to pre-train a model

on several gas turbines to yield better inferences on a specific gas turbine? A preliminary result was shown in this master's thesis, where we observed that when a specific operation point was not available in the training data, the model benefited from encountering such situations in other gas turbines. However, we did not investigate the extent of this idea. A limitation for this exploration is that not all gas turbines share the same set of valid sensors, thus restricting the size of potential pre-training or necessitating models that can handle such missing values. In future work, it is a possible avenue to handle the full dataset and make the necessary considerations to let the model *interpret* missing values for certain sensors. Another key limitation, as discussed in the results, is that the dataset for fine-tuning on some turbines is *incomplete*, thus the process effectively destroys the previous learnt knowledge.

## 6.4 Ethical, Sustainability, and Final Reflections

A significant reflection on this work pertains to the safe usage of ML. Indeed, the utilization of AI should not be indiscriminate; accountability is essential [67] for the consequences resulting from its usage. The crucial point here is the necessity for awareness of the limitations of this technology to prevent disastrous outcomes stemming from overconfidence in AI use. While the results of this master's thesis demonstrate accurate and satisfactory estimates of different variables in gas turbines, it is important to note the potential danger of using these estimates to completely replace physical sensors, given the observed data limitations.

In terms of sustainability, this project contributes to the refinement of a technology, namely gas turbines, which is effectively aiding humanity's transition to a sustainable future, as highlighted in Chapter 2. The solution presented in this master's thesis, along with its potential future developments, may play a crucial role in enabling the monitoring of gas turbines when used with more unstable fuels such as Hydrogen or Ammonia.

Reflecting on the entirety of this work, it is evident that a significant portion was dedicated to obtaining clean and usable data. This aspect consistently resurfaced during results analysis and conclusion discussions, often taking precedence over model discussions. While model design played a relevant role, this work underscores the equal, if not greater, importance of the data underlying these models.

In conclusion, this master's thesis delved into the complexities and

opportunities associated with Machine Learning Algorithm applications in gas turbines. It illustrated the construction of a methodical solution, with an emphasis on data refinement, and the outcomes derived from our experimentation. The results of this research contribute to the ongoing evolution in the field of gas turbine technology and also pave the way for future advancements and discoveries.

# References

[1] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, vol. 349, no. 6245, pp. 255–260, 2015. [Pages 1 and 3.]

[2] M. de Castro-Cros, S. Rosso, E. Bahilo, M. Velasco, and C. Angulo, "Condition assessment of industrial gas turbine compressor using a drift soft sensor based in autoencoder," *Sensors*, vol. 21, no. 8, p. 2708, 2021. [Pages 1, 2, 23, 30, 31, 52, 61, 79, and 84.]

[3] T.-G. Isaiah, S. Dabbashi, D. Bosak, S. Sampath, G. Di Lorenzo, and P. Pilidis, "Life analysis of industrial gas turbines used as a back-up to renewable energy sources," *Procedia CIRP*, vol. 38, pp. 239–244, 2015. [Page 1.]

[4] A. Poullikkas, "An overview of current and future sustainable gas turbine technologies," *Renewable and Sustainable Energy Reviews*, vol. 9, no. 5, pp. 409–443, 2005. [Pages 1 and 27.]

[5] P. Chiesa, G. Lozza, and L. Mazzocchi, "Using hydrogen as gas turbine fuel," *J. Eng. Gas Turbines Power*, vol. 127, no. 1, pp. 73–80, 2005. [Pages 1 and 27.]

[6] M. Alhuyi Nazari, M. Fahim Alavi, M. Salem, and M. E. H. Assad, "Utilization of hydrogen in gas turbines: A comprehensive review," *International Journal of Low-Carbon Technologies*, vol. 17, pp. 513–519, 2022. [Pages x, 1, 28, and 29.]

[7] S. Öberg, M. Odenberger, and F. Johnsson, "Exploring the competitiveness of hydrogen-fueled gas turbines in future energy systems," *International Journal of Hydrogen Energy*, vol. 47, no. 1, pp. 624–644, 2022. [Page 1.]

[8] C. Soares, *Gas turbines : a handbook of air, land and sea applications*, second edition. ed. Oxford, England ;: Butterworth-Heinemann, 2015 - 2015. ISBN 0-12-410485-1 [Pages 2, 4, and 27.]

[9] L. S. Langston, G. Opdyke, and E. Dykewood, "Introduction to gas turbines for non-engineers," *Global Gas Turbine News*, vol. 37, no. 2, pp. 1–9, 1997. [Page 2.]

[10] J. Lindhof, "Sensor-network anomaly detection system for turbomachinery instrumentation surveillance," Master's thesis, Linköping University, 2022. [Pages 2, 23, 31, 53, and 94.]

[11] S. Cruz-Manzo, V. Panov, C. Bingham *et al.*, "Gas turbine sensor fault diagnostic system in a real-time executable digital-twin," *Journal of the Global Power and Propulsion Society*, vol. 7, pp. 85–94, 2023. [Pages 3 and 32.]

[12] T. Palmé, M. Fast, and M. Thern, "Gas turbine sensor validation through classification with artificial neural networks," *Applied Energy*, vol. 88, no. 11, pp. 3898–3904, 2011. [Pages 4, 30, 61, 82, and 94.]

[13] R. Wirth and J. Hipp, "Crisp-dm: Towards a standard process model for data mining," in *Proceedings of the 4th international conference on the practical applications of knowledge discovery and data mining*, vol. 1. Manchester, 2000, pp. 29–39. [Pages 5, 36, 62, and 91.]

[14] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014. [Pages 5 and 10.]

[15] A. Jobin, M. Ienca, and E. Vayena, "The global landscape of ai ethics guidelines," *Nature machine intelligence*, vol. 1, no. 9, pp. 389–399, 2019. [Page 5.]

[16] "Data-centric ai: Ai models are only as good as their data pipeline," https://hai.stanford.edu/news/data-centric-ai-ai-models-are-only-good-their-data-pipeline, accessed: 2024-02-05. [Pages 5 and 88.]

[17] X. Ying, "An overview of overfitting and its solutions," in *Journal of physics: Conference series*, vol. 1168. IOP Publishing, 2019, p. 022022. [Page 11.]

[18] Y. LeCun, Y. Bengio *et al.*, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995. [Pages 12 and 20.]

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778. [Page 12.]

[20] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017. [Pages 12, 20, and 64.]

[21] L. D. Broemeling, *Bayesian analysis of linear models*. CRC Press, 2017. [Page 14.]

[22] R. A. Fisher, "Iris," UCI Machine Learning Repository, 1988, DOI: https://doi.org/10.24432/C56C76. [Page 15.]

[23] T. Hastie, "Ridge regularization: An essential concept in data science," *Technometrics*, vol. 62, no. 4, pp. 426–433, 2020. [Page 16.]

[24] D. Koller and N. Friedman, *Probabilistic graphical models: principles and techniques*. MIT press, 2009. [Pages 17 and 58.]

[25] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989. [Pages 18 and 61.]

[26] S. Sharma, S. Sharma, and A. Athaiya, "Activation functions in neural networks," *Towards Data Sci*, vol. 6, no. 12, pp. 310–316, 2017. [Page 18.]

[27] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986. [Page 20.]

[28] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019. [Pages 20, 50, and 88.]

[29] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016. [Page 20.]

[30] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014. [Page 20.]

[31] B. Rezaeianjouybari and Y. Shang, "Deep learning for prognostics and health management: State of the art, challenges, and opportunities," *Measurement*, vol. 163, p. 107929, 2020. [Pages 20 and 33.]

[32] Y. Ren, L. Zhang, and P. N. Suganthan, "Ensemble classification and regression-recent developments, applications and future directions," *IEEE Computational intelligence magazine*, vol. 11, no. 1, pp. 41–53, 2016. [Pages 22, 23, 62, and 73.]

[33] B. Kestner, T. Lieuwen, C. Hill, L. Angello, J. Barron, and C. A. Perullo, "Correlation analysis of multiple sensors for industrial gas turbine compressor blade health monitoring," *Journal of Engineering for Gas Turbines and Power*, vol. 137, no. 11, p. 112605, 2015. [Page 23.]

[34] G. Farrahi, M. Tirehdast, E. M. K. Abad, S. Parsa, and M. Motakefpoor, "Failure analysis of a gas turbine compressor," *Engineering Failure Analysis*, vol. 18, no. 1, pp. 474–484, 2011. [Page 24.]

[35] K. Bengtson, "Thermoacoustic instabilities in a gas turbine combustor," Master's thesis, KTH Royal Institute of Technology, 2017. [Pages 24, 27, 28, 51, and 83.]

[36] S. Ito, M. Uchida, T. Suda, and T. Fujimori, "Development of ammonia gas turbine co-generation technology," *IHI Eng. Rev*, vol. 53, no. 1, p. 6, 2020. [Page 28.]

[37] S. Cen, K. Li, Q. Liu, and Y. Jiang, "Solar energy-based hydrogen production and post-firing in a biomass fueled gas turbine for power generation enhancement and carbon dioxide emission reduction," *Energy Conversion and Management*, vol. 233, p. 113941, 2021. [Page 29.]

[38] J. Qian, Z. Song, Y. Yao, Z. Zhu, and X. Zhang, "A review on autoencoder based representation learning for fault detection

and diagnosis in industrial processes," *Chemometrics and Intelligent Laboratory Systems*, p. 104711, 2022. [Pages 30 and 33.]

[39] M. Farahani, "Anomaly detection on gas turbine time-series' data using deep lstm-autoencoder," Master's thesis, Umeå University, 2020. [Page 31.]

[40] S. Sasikumar, "Predictive study of flame status inside a combustor of a gas turbine using binary classification," 2022. [Pages 31, 52, and 82.]

[41] M. Tahan, E. Tsoutsanis, M. Muhammad, and Z. A. Karim, "Performance-based health monitoring, diagnostics and prognostics for condition-based maintenance of gas turbines: A review," *Applied energy*, vol. 198, pp. 122–144, 2017. [Pages 32, 33, and 34.]

[42] H. Minghui, H. Ya, L. Xinzhi, L. Ziyuan, Z. Jiang, and M. Bo, "Digital twin model of gas turbine and its application in warning of performance fault," *Chinese Journal of Aeronautics*, vol. 36, no. 3, pp. 449–470, 2023. [Page 32.]

[43] H. Liu, J. Zhou, Y. Zheng, W. Jiang, and Y. Zhang, "Fault diagnosis of rolling bearings with recurrent neural network-based autoencoders," *ISA transactions*, vol. 77, pp. 167–178, 2018. [Page 33.]

[44] N. Davari, S. Pashami, B. Veloso, S. Nowaczyk, Y. Fan, P. M. Pereira, R. P. Ribeiro, and J. Gama, "A fault detection framework based on lstm autoencoder: A case study for volvo bus data set," in *International Symposium on Intelligent Data Analysis*. Springer, 2022, pp. 39–52. [Page 33.]

[45] J. Ren and D. Ni, "A batch-wise lstm-encoder decoder network for batch process monitoring," *Chemical Engineering Research and Design*, vol. 164, pp. 102–112, 2020. [Page 33.]

[46] N. Benton, L. Cardelli, and C. Fournet, "Modern concurrency abstractions for c#," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 26, no. 5, pp. 769–804, 2004. [Page 37.]

[47] A. D. Birrell, *An introduction to programming with threads*. Digital Systems Research Center, 1989. [Page 37.]

[48] CPAIOR, "Cpaior 2020 master class: Constraint programming," Sept. 2020. [Online]. Available: https://youtu.be/lmy1ddn4cyw [Page 48.]

[49] L. Perron and F. Didier, "Cp-sat," Google. [Online]. Available: https://developers.google.com/optimization/cp/cp_solver/ [Page 48.]

[50] L. Perron and V. Furnon, "Or-tools," Google. [Online]. Available: https://developers.google.com/optimization/ [Page 49.]

[51] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz *et al.*, "Huggingface's transformers: State-of-the-art natural language processing," *arXiv preprint arXiv:1910.03771*, 2019. [Page 50.]

[52] E. Gamma, R. Helm, R. Johnson, J. Vlissides, and D. Patterns, "Elements of reusable object-oriented software," *Design Patterns*, 1995. [Pages 50 and 65.]

[53] S. Kasap, "Thermoelectric effects in metals: thermocouples," *Canada: Department of Electrical Engineering University of Saskatchewan*, 2001. [Page 53.]

[54] M. Baer, "findiff software package," 2018, https://github.com/maroba/findiff. [Online]. Available: https://github.com/maroba/findiff [Page 56.]

[55] C. Truong, L. Oudre, and N. Vayatis, "ruptures: change point detection in python," *arXiv preprint arXiv:1801.00826*, 2018. [Page 57.]

[56] ——, "Selective review of offline change point detection methods," *Signal Processing*, vol. 167, p. 107299, 2020. [Page 57.]

[57] K. Bartol, D. Bojanić, T. Petković, S. Peharec, and T. Pribanić, "Linear regression vs. deep learning: A simple yet effective baseline for human body measurement," *Sensors*, vol. 22, no. 5, p. 1885, 2022. [Page 60.]

[58] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011. [Page 61.]

[59] Y. Qin, D. Song, H. Chen, W. Cheng, G. Jiang, and G. Cottrell, "A dual-stage attention-based recurrent neural network for time series prediction," *arXiv preprint arXiv:1704.02971*, 2017. [Page 63.]

[60] G. Zerveas, S. Jayaraman, D. Patel, A. Bhamidipaty, and C. Eickhoff, "A transformer-based framework for multivariate time series representation learning," in *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*, 2021, pp. 2114–2124. [Page 64.]

[61] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018. [Page 64.]

[62] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019. [Pages 64 and 70.]

[63] O. Yadan, "Hydra - a framework for elegantly configuring complex applications," Github, 2019. [Online]. Available: https://github.com/facebookresearch/hydra [Pages 64, 65, and 106.]

[64] W. Falcon and The PyTorch Lightning team, "PyTorch Lightning," Mar. 2019. [Online]. Available: https://github.com/Lightning-AI/lightning [Pages 64 and 65.]

[65] B. E. Flores, "A pragmatic view of accuracy measurement in forecasting," *Omega*, vol. 14, no. 2, pp. 93–98, 1986. [Page 67.]

[66] P. Goodwin and R. Lawton, "On the asymmetry of the symmetric mape," *International journal of forecasting*, vol. 15, no. 4, pp. 405–408, 1999. [Page 67.]

[67] European Comission, "Ethics guidelines for trustworthy ai," 2019, accessed March 18, 2024. https://digital-strategy.ec.europa.eu/en/library/ethics-guidelines-trustworthy-ai. [Page 95.]

# Appendix A

# Optuna and Hydra integration

In this concise appendix chapter, we briefly demonstrate the exceptional flexibility provided by Python and Hydra in conjunction with Optuna, which facilitates an extremely efficient and clean grid search process.

We have included this code segment to allow readers to appreciate the benefits of the modular solution we have proposed and to see how seamlessly the various components of our architecture work together. This integration results in a unique code that can handle every model and dataset, showcasing the effectiveness and adaptability of our system design.

```
def grid_search(
    config: DictConfig, configuration_name: str, configuration_folder: str, all_normalized: bool = False
) -> None:
    """
    Perform grid search on the given configuration file.

    Args:
        config(DictConfig): The configuration of the grid search job.
        configuration_name(str): The name of the configuration used
        configuration_folder(str): The relative path of the configuration folder
        all_normalized(bool): Whether to compute the metrics on normalized variables
    """
    original_working_directory = hydra.utils.get_original_cwd()
    outter_overrides = (
        hydra.core.hydra_config.HydraConfig().get().job.override_dirname
    )  # noqa
    GlobalHydra.instance().clear()
    params_grid = config["general"]["execution_mode_args"]["params"]

    # Define the dataset once (and forever)
    # Changes to parameters of the dataset are not allowed for the same reason (it is just not performant)
    # However, changes to the model adapter/task adapter are allowed, which is very flexible!
    dataset = hydra.utils.instantiate(config.dataset)

    # Define the objective as a closure function. It is not thread safe
    # It should be inter-process safe but has not been tested
    def objective(trial: optuna.Trial):
        # Instantiate all the parameters from the trial
        # Simply loop through all the parameters specified in configuration
        trial_parameters = get_trial_overrides(trial, params_grid)

        # Evaluate configuration
        evaluated_config = set_trial_overrides(
            overrides=outter_overrides.split(",")
            + [
                trial_param_path
                + "="
```

```
                + ParameterSuggestor.to_override_format(trial_param_value)
                for trial_param_path, trial_param_value in trial_parameters.items()
            ],
            configuration_name=configuration_name,
            configuration_folder=configuration_folder,
            original_working_directory=original_working_directory,
        )

        # Instantiate the loaders and preprocess as indicated
        datasets = dataset.get_loaders(
            model_adapter=hydra.utils.instantiate(
                evaluated_config.model_framework.model_adapter
            ),
            task=Task[evaluated_config.general.task.lower()],
            task_args=evaluated_config.general.task_args,
            grouping=Grouping.BY_PURPOSE,
        )

        data_train, data_validation = (
            datasets.get(SplitPurpose.TRAINING),
            datasets.get(SplitPurpose.VALIDATION),
        )

        trainer = hydra.utils.instantiate(
            config=evaluated_config.model_framework.training.trainer,
            cfg=evaluated_config.model_framework,
            dataset=dataset,
            trial=trial,
        )

        trainer.fit(data_train, data_validation)

        model_tester = ModelValidator(model_trainer=trainer, dataset=dataset, all_normalized=all_normalized)
        logger.info("Validating model")
        _, validation_results = model_tester.validate(data_validation)

        return validation_results[config["general"]["execution_mode_args"]["metric"]]

    # Create the study and perform the optimization
    study = optuna.create_study(
        storage="sqlite:///grid_search_result.db", study_name="Grid Search"
    )
    study.optimize(
        objective, n_trials=config["general"]["execution_mode_args"]["n_trials"]
    )
```

The opportunities with hydra [63] signified a really modular experimentation method. As described in Section 4.5 the software implemented could employ different task, model and dataset interexchangably giving place to shell commands like the one in

Listing A.1: bash version

```
# Training an ensemble of mlp for regression in
pyhton main.py general=train_regressor \\
model_framework=ensemble_mlp \\
dataset=fleet_dataset \\
dataset.data_sources.files_path="/example/path/turbine_1"
```