Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering

Amirhossein Vakili

# A Comparative Analysis of Machine Learning Models in Prognostics and Prediction of Remaining Useful Life of Aircraft Turbofan Engines

Master's thesis in Reliability, Availability, Maintainability and Safety (RAMS)
Supervisor: Professor Yiliu Liu
October 2023

NTNU
Norwegian University of
Science and Technology

Amirhossein Vakili

# A Comparative Analysis of Machine Learning Models in Prognostics and Prediction of Remaining Useful Life of Aircraft Turbofan Engines

Master's thesis in Reliability, Availability, Maintainability and Safety (RAMS)
Supervisor: Professor Yiliu Liu
October 2023

Norwegian University of Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering

NTNU
Norwegian University of
Science and Technology

# Preface

This report is written during the fall semester of 2023 in Reliability, Availability, Maintainability, and Safety (RAMS) at the Department of Mechanical and Industrial Engineering (MTP) at NTNU. My main drive in selecting this topic stemmed from a deep-rooted interest in the concept of prognostics and the prediction of the Remaining Useful Life of components and systems. Pairing this with a curiosity in programming and machine learning made this an exciting journey. While the thesis is primarily targeted at those with a foundational grasp of programming and reliability theory, the methods and techniques discussed, especially those related to machine learning, are introduced in a way that should be approachable for readers unfamiliar with the topic.

Trondheim, 2023-10-31

*Amirhossein Vakili*

Amirhossein Vakili

# Acknowledgment

First and foremost, I would like to express my sincere gratitude to Professor Yiliu Liu for his invaluable guidance throughout this thesis journey. His insights were crucial in shaping my research direction and in navigating the complexities of machine learning and prognostic techniques.

I'm also deeply appreciative of my fellow RAMS student, Mahshid Aghamalizadeh. Her consistent support, sharing of knowledge, and enriching discussions played a significant role in the formulation and completion of this report.

Amirhossein Vakili

# Executive Summary

In the pursuit of enhancing the safety and reliability of complex industrial systems, Prognostics Health Management (PHM) technology, especially its components of fault diagnosis and prognosis, has found increasing relevance in various industries. Fundamental to PHM technology is the prediction of the Remaining Useful Life (RUL). An accurate estimation of RUL not only leads to accident prevention but also facilitates optimal equipment maintenance scheduling in complex systems. This strategic foresight can decrease maintenance costs and spearhead a more systematic approach to predictive maintenance. While RUL prediction techniques span from model-based to data-driven approaches, the latter, which establishes relationships between RUL and historical data through a learning-based model, is gaining attention due to its independence from prior knowledge.

Building on this foundation, this master thesis delves into the application of Machine Learning (ML) models and Artificial Neural Networks (ANNs) to predict the remaining useful life of turbofan engines, using the turbofan degradation dataset by National Aeronautics and Space Administration (NASA) as a case study. The research starts with an introduction to the principles of prognostic health management and RUL, establishing the context and relevance of the study. Subsequent chapters explore the domains of Artificial Intelligence (AI), underscoring its intersections with machine learning and Deep Learning (DL). A comprehensive overview of machine learning paradigms, including supervised and unsupervised methodologies, is provided, setting the stage for a deep dive into various regression models. Given the dataset's properties and the continuous, labeled nature of the data, the theoretical foundations of regression models, including linear regression, polynomial regression, support vector regression, decision tree, and random forest regression, alongside artificial neural networks, are elaborated. Each model is thoroughly constructed, ran on the dataset, and a detailed comparative analysis of results is provided. The study further enhances its practical value by addressing data visualization, feature selection, data treatment and hyperparameter tuning, highlighting the significant improvements achieved after treatment, reinforcing the potential of machine learning and deep learning in prognostics and prediction of remaining useful life.

# Contents

# Chapter 1

# Introduction

## 1.1 Background

The prediction of the Remaining Useful Life (RUL) is pivotal in the domain of Prognostics Health Management (PHM) technology. RUL essentially measures the time from the current point until a system is expected to fail. An accurate RUL prediction can serve as a warning, enabling users to initiate timely maintenance actions, reducing subsequent costs, and offering a foundation for insightful maintenance decisions. As a critical element promoting the safety and reliability of complex industrial systems, RUL prediction, especially within PHM, is essential for various industries. Through its accurate implementation, one can accurately schedule equipment maintenance, optimizing the efficiency and cost-effectiveness of complex systems, while ensuring the proactive care of equipment.Zhang et al. (2022)

There are various methodologies employed for RUL prediction like model-based approaches, data-driven approaches, and hybrid approaches. Model-based strategies require a deep understanding and a precise mechanical model of the target system, with robust prior knowledge. However, data-driven approaches are becoming more and more popular due to their capacity to recognize patterns between RUL and historical data without the need for prior system knowledge. These data-driven methods are typically divided into two major categories: conventional machine learning approaches and the burgeoning deep learning approaches. While conventional machine learning techniques like linear regression, Support Vector Machine (SVR), decision tree and Random Forest are used in the field, they sometimes show constraints in both accuracy and computational efficacy, particularly when handling vast datasets. Zhang et al. (2023)

On the contrary, emerging deep learning methods, such as the Artificial Neural Network (ANN), Recurrent Neural Network (RNN), and convolutional neural network (CNN) have demonstrated superior performance in managing extensive data, potentially overshadowing traditional techniques. This thesis tries to explore both traditional machine learning regression techniques

and compare their efficiency and results with artificial neural network model, primarily focusing on the RUL prediction of turbofan engines, using turbofan degradation data set by NASA.

## 1.2 Objectives

This Master thesis strives to offer a foundational guide for individuals keen on utilizing machine learning for forecasting and predicting the remaining useful life. Initially, it introduces key concepts such as prognostics health management, prognostics, remaining useful life, machine learning, and deep learning. Subsequently, the thesis aims to shed light on ML regression techniques, illustrating how to construct and implement machine learning models, with a special focus on deploying artificial neural network.

## 1.3 Approach

This thesis delves into key areas such as prognostics, remaining useful life, machine learning, and artificial neural networks. Given that the big dataset from NASA comes with labels, the study is focused on supervised ML techniques. Comprehensive details, ranging from theoretical insights to practical steps, coding examples, and relevant tools, are provided to guide readers through the process of forecasting the RUL of turbofan engines. Both traditional regression machine learning models and ANN have been employed, with the corresponding results showcased.

## 1.4 Outline

The remainder of this thesis is structured as follows:

- Chapter 2. Prognostic and remaining useful life: This chapter explains the fundamental principles of prognostics health management, emphasizing prognostic techniques and the concept of remaining useful life. It further dives into understanding RUL as functions of different parameters such as Condition Monitoring (CM) and Reliability Factor (RF).

- Chapter 3. Artificial intelligence, machine learning, and deep learning: This chapter clarifies the theoretical foundations of AI, drawing a clear distinction between machine learning and deep learning. It delves into diverse paradigms of ML such as supervised, unsupervised, semi-supervised, and reinforcement learning. The chapter also provides comprehensive insights into several regression algorithms, including but not limited to linear regression and its variants, support vector regression, decision tree, and random forest.

The latter part of the chapter introduces deep learning concepts, focusing on the architecture and components of artificial neural networks, notably activation and cost functions.

- Chapter 4.Building the machine learning models: In this chapter, the choice of programming language and essential libraries for building the models is discussed. An introduction to the turbofan degradation dataset is provided, followed by a systematic walkthrough of data preprocessing techniques ranging from importing the dataset, visualization, feature extraction using a heatmap to data standardization. Subsequently, the procedure for building a series of machine learning models, including linear regression, polynomial regression, support vector regression, decision tree regression, random forest regression, and an ANN model, is illustrated in detail. The primary objective of these models is to determine the remaining useful life of turbofan engines. This chapter concludes with a presentation of the results derived from each model.
- Chapter 5. Conclusions
- Acronym
- Appendix B: Code templates
- Bibliography

# Chapter 2

# Prognostic and Remaining Useful Life

In the quest for enhanced safety and reliability of complex industrial systems, Prognostics Health Management (PHM) technology, which includes fault diagnosis and prognosis, is gaining widespread attention across various industries. Central to PHM technology is the prediction of the Remaining Useful Life (RUL). Devoted to accident prevention, RUL plays a crucial role, and its accurate estimation ensures equipment maintenance in these complex systems is optimally scheduled. This, in turn, can reduce maintenance costs and facilitate systematic predictive maintenance. Zhang et al. (2023)

Traditionally, engineering prognostics, a tool employed by industries to mitigate business risks arising from unforeseen equipment failures, basically relies on the intuition and expertise of personnel familiar with the respective equipment. Yet, challenges emerge as the reliability of assets improves and the engineering workforce ages, making it harder to collect such expertise. Moreover, human decisions, when confronted with equipment characterized by numerous interrelated failure modes, may not always be consistent or accurate. Recognizing these challenges, recent years have witnessed an increase in research efforts to prepare models that reduce the industry's reliance on individuals, ensuring more accurate and reliable decision-making processes. Sikorska et al. (2011)

## 2.1 Prognostics health management

Maintenance of complex systems ensures that machines operate in their correct functioning conditions and reduce unforeseen downtimes. Recently, research has been shifted towards predictive maintenance and condition monitoring, especially prognostic health management. PHM uses machine learning and artificial intelligence to analyze data from sensors on machines, convert it into valuable insights, determining their remaining useful life and enhancing the maintenance planning process. Calabrese et al. (2019)

## 2.2 Prognostics

There are many definitions for prognostic:

1. *An estimation of time to failure and risk for one or more existing and future failure modes.* Tobon-Mejia et al. (2010)

2. *The capability to provide early detecting of the precursor and/or incipient fault condition of a component, and to have the technology and means to manage and predict the progression of this fault condition to component failure.* Engel et al. (2000)

3. *Predictive diagnostics, which includes determining the remaining life or time span of proper operation of a component.* Hess et al. (2005)

4. *Failure prognosis involves forecasting of system degradation based on observed system condition.* Luo et al. (2003)

From all these definitions we can conclude some characteristics for prognostic such as: Sikorska et al. (2011)

- Prognostics ideally focuses on individual parts or components within a system.
- It is about forecasting how a particular malfunction develops over time until the component breaks down.
- Understanding how the component will work in the future is crucial.
- While prognostics is linked to diagnostics, the two are not identical.

Most experts agree that prognostics is closely tied to, and often depends on, diagnostics. Yet, the clear boundary between them is not always well-defined in existing literature. A straightforward way to differentiate these two is: diagnostics focuses on identifying and measuring damage that has already happened—it looks back. On the other hand, prognostics aims to foresee potential future damage. Even though diagnostics can offer valuable insights independently, prognostics relies on diagnostic findings, meaning prognostics cannot operate entirely on its own. Sikorska et al. (2011)

The ISO13381-1 standard describes prognostics as a forecast of how long it will take for a failure to happen, considering both current and upcoming failure modes. This means that prognostics is not just about predicting outcomes based on known failure modes but also understanding how these problems might initiate other failure modes. More specifically, when thinking about prognostics, we should look into: Sikorska et al. (2011)

1. existing failure mode and how fast they're getting worse,

2. initiation criteria for future failure modes,
3. how different failure modes might relate to each other and their progression rates,
4. how good our monitoring tools and analysis techniques are at catching these progression rates,
5. how maintenance efforts can impact these deteriorations, and
6. the condition and assumption behind the prognoses.

Based on this description, (1) to (3) can be seen as diagnostic questions, while the remaining three pertain to prognostics. Continual diagnostic assessments are vital to identify new events, ensuring that prognostic predictions remain correct and precise.

## 2.3 Remaining useful life

The concept of remaining useful life which is occasionally called remaining service life or residual life denotes the time remaining before a potential failure occurs, considering the current age and state of a machine, as well as its past and future operational profile. RUL for any asset or mechanism is described as the time span from the present moment to the end of the useful life. It is defined as remaining lifetime of a system at any given time t, considering all information of the system at time t. There are two primary mathematical definitions of RUL in the literatures, depending on the estimation methodology and the data at hand: one that views RUL as a function of Condition Monitoring (CM) and another that considers it as a function of the Reliability Function (RF). Moamar and Springerlink (2020)

### 2.3.1 RUL as a function of CM

Here, RUL is considered as an outcome of a system's CM, represented as $Z(t)$. This representation encloses the historical operating state of the system and co-variables outlining its present state. The formulation is: Moamar and Springerlink (2020)

$$RUL(t|Z_t) = T - t \,|\, T > t, Z(t) \tag{2.1}$$

Where:

- $T$: time to failure,
- $t$: current time or age,
- $Z(t)$: condition trajectory up until now.

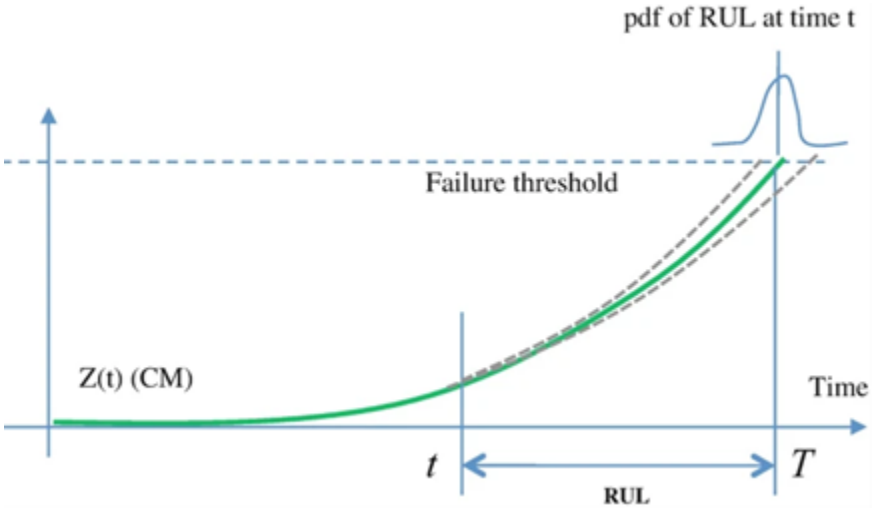



Figure 2.1: RUL as a function of CM from (Moamar and Springerlink (2020)).

This formula is visually represented in Figure 2.1. Depending on the insights from $Z(t)$, RUL might be calculated as either deterministic, statistical (expectation), or a probabilistic variable (probability density function). The dashed gray boundary indicates the uncertainty concerning forthcoming operational condition and the environment of the system. Moamar and Springerlink (2020)

### 2.3.2 RUL as a function of RF

Data derived from condition monitoring can be included into a reliability assessment by assuming the hazard rate function as a probabilistic function. Various techniques exist for calculating both conditional and unconditional reliability functions and for computing the RUL as a function of current conditions. In classical reliability analysis, RF is primarily determined in two scenarios: one that is unconditional, presuming the component has not started operation ($P(T > t)$), and the second that is conditional, based on the assumption that the component has not yet failed until a specific moment x ($P(T > t|T > x)$). Moamar and Springerlink (2020)

When we consider that a system is operating at time $t$, the RUL is denoted as a time $v$. Here, the probability that the system's state $Z$ at $t + v$, denoted $P[Z(t + v) \geq L|Z(t)]$, approximates the known failure probability $q$. The formulation for RUL is:

$$RUL(t, q) = \sup\{v : P[Z(t + v) \geq L|Z(t)] \leq q\} \tag{2.2}$$

In this equation, $L$ is the threshold for failure, and $P[Z(t + v) \geq L|Z(t)]$ is defined as the system's reliability.

RUL is a key measure to understand how a system is performing while it is in use. It serves as a predictor of the remaining operational period a system or component has before it is likely

to fail. In fact, RUL bridges the gap between current health status and the point of operational failure.

Moreover, it gives insights into when the system might need maintenance. This information is critical for scheduling timely maintenance, aiming to minimize any interruptions or downtimes. There are many reasons why knowing RUL is important: Berghout and Benbouzid (2022)

- **Enhancing maintenance strategies**: By transitioning from traditional reactive or calendar-based maintenance to predictive maintenance, industries can ensure the equipment is serviced just in time before failure, ensuring minimal disruptions.
- **Cost efficiency**: An accurate RUL prediction avoids premature replacements and unnecessary inspections, which can lead to substantial cost savings in maintenance operations.
- **Ensuring reliability**: A timely maintenance or replacement, as dictated by the RUL, ensures systems operate at peak performance, thereby guaranteeing operational reliability.
- **Risk mitigation**: Knowing when a system is likely to fail helps industries prepare or make alternate arrangements, thereby minimizing potential risks.

RUL prediction techniques generally fall into a few categories: model-based, data-driven, and hybrid models that combine both approaches. Berghout and Benbouzid (2022)

- **Model-based**: These utilize mathematical models built upon the fundamental principles governing the system's operations. For instance, fatigue analysis for metal components can give insights into their RUL. However, this approach can be intricate and may not consider all real-world variabilities.
- **Data-driven**: Rather than relying on theoretical models, these predictions analyze historical operational data, sensor readings, and past failures to forecast RUL. This method is especially useful for complex systems where traditional modeling can be restrictive.
- **Hybrid models**: By combining the theoretical strengths of model-based predictions and the practical insights from data-driven predictions, hybrid models aim to give a better and more precise RUL estimation. Berghout and Benbouzid (2022); Zhang et al. (2023)

Data-driven approach has drawn attention recently duo to its power to predict complex relationships between sensor data and system degradation or fault. It helps estimation of RUL in many ways such as: Berghout and Benbouzid (2022)

- **Managing complex data sets**: Modern machinery often comes equipped with multiple sensors that generate massive amounts of data. Machine learning algorithms, with their ability to handle high-dimensional data, can analyze these complex data sets to derive meaningful insights.

- **Self-improvement**: ML models can learn from new data. As more data becomes available, these models refine themselves, leading to increasingly accurate RUL predictions over time.
- **Deep learning capabilities**: Deep learning, a subset of ML, can automatically discover the representations needed for feature detection from raw data. This is particularly useful when dealing with unstructured data or when it is challenging to identify what specific features or patterns correlate with system failures.
- **Real-time predictions**: ML models can process data and provide insights in real-time. As systems evolve or external conditions change, ML can quickly adjust its predictions, ensuring industries always have up-to-date information on their machinery's health.

Data-driven RUL predictions are further divided into traditional machine learning models and the newer deep learning techniques. These data-driven approaches are evolving rapidly. They are designed to adapt to various data challenges, addressing attributes such as the sheer amount of data (volume), the speed at which it is generated (velocity), and the diversity of data types (variety). Berghout and Benbouzid (2022)

# Chapter 3

# Artificial Intelligence (AI), Machine Learning (ML) and Deep Learning (DL)

## 3.1 Artificial intelligence

Artificial Intelligence (AI) aims to equip machines with capabilities similar to the human brain. In computer science, AI delves into the study of "intelligent agents", which are devices that can assess their environments and act in ways that maximize their success. Generally, we talk about artificial intelligence when machines can carry out tasks we typically attribute to the human mind, like learning and problem-solving. Given that learning is vital for machines, machine learning emerges as a specialized branch within AI. Shinde and Shah (2018)

## 3.2 Machine learning vs deep learning

Machine learning and deep learning, both popular topics under the artificial intelligence umbrella, have been popular recently. Machine learning is using algorithms to parse data, learn from it, and then decide or predict. It enables computers to learn automatically and improve from experience without being explicitly programmed. Deep learning, which is a subset of machine learning, uses Neural Networks (NN) to tackle complex and difficult problems. Neural networks are inspired by our understanding of the biology of our brains and built with layers, connections, neurons, and directions of data propagation.

The ability of learning without the need for being programmed makes them especially popular in applications where the large datasets are accessible, such as natural language processing, financial decision making and image recognition.

In recent years, due to the growing availability of big datasets as well as increasing computational power of computers, there has been a significant shift in research and advancement in

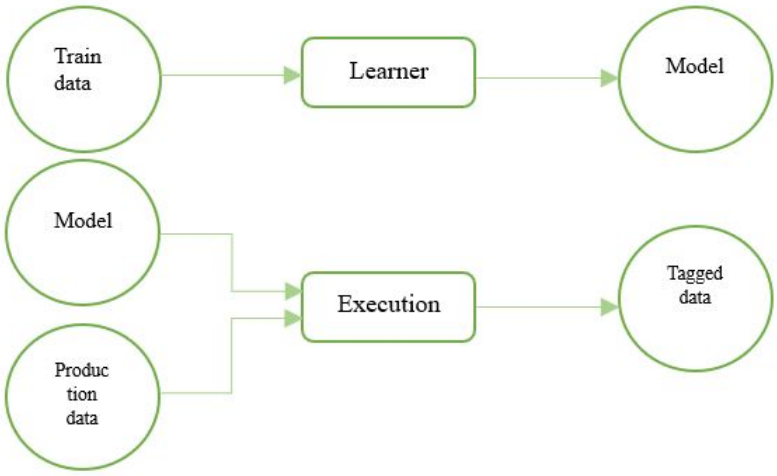



Figure 3.1: Machine learning operational model (Kumar et al. (2020)).

the field of machine learning and deep learning. Sharifani and Amini (2023); Lærum (2018)

## 3.3 Machine learning paradigms

In machine learning, the learning procedure is split into two main stages: training and testing. Initially, the system undergoes several iterations of training. During this phase, it processes training data samples and, using a learning algorithm, picks up features to construct a learning model. Following this, the model is tested. At this stage, the learning model employs an execution engine to predict outcomes for either production or test data. The results from the learning model are referred to as tagged data, which provide either classified outcomes or the final predicted data. ( Figure 3.1) Kumar et al. (2020)

The learning process is divided into different categories:

### 3.3.1 Supervised learning

For this kind of machine learning algorithm, external help is required. The input dataset is split into two groups: training and testing. The training data includes the input as well as the correct output. It is essential to categorize and forecast the output of the training dataset. These algorithms identify patterns from the training data, and then they classify or predict using the test data. There are many algorithms under supervised learning, depending on the nature of the task (classification, regression, etc.) and the type of data. Here are some of the most common supervised learning algorithms: Kumar et al. (2020)

- **Linear regression**: used for regression problems where the outcome is continuous.

- **Logistic regression**: despite its name, it is used for binary classification problems.
- **Decision tree**: can be used for both classification and regression tasks.
- **Random forest**: an ensemble method that creates a 'forest' of decision trees and merges their outputs.
- **Support Vector Machines (SVM)**: mainly used for classification problems but can be adapted for regression.
- **k-Nearest Neighbors (k-NN)**: used for classification and regression. It works by finding the 'k' training examples closest to a point. Bishop (2006)

### 3.3.2 Unsupervised learning

Unsupervised learning algorithms work by discovering patterns and structures in data without explicit labels. They extract features from the data autonomously, and when new data is introduced, the algorithms classify or cluster the data based on the features they have previously identified. Unlike supervised learning, where models are trained using labeled data, unsupervised learning methods operate on datasets without prior knowledge of the outcome. Kumar et al. (2020)

The primary applications of unsupervised learning include:

- **Clustering**: This is the process of grouping similar data points together based on certain features. Common algorithms used for clustering are: Bishop (2006)
  - **K-means clustering**: K-means clustering is a technique used to group similar data points together based on certain features. It works by partitioning the data into 'k' clusters, where 'k' is a user-defined number. The algorithm assigns each data point to the nearest cluster centroid, and it iteratively updates the centroids until convergence. K-means clustering is widely used for tasks like customer segmentation, image compression, and document categorization.
  - **Hierarchical clustering**: Hierarchical clustering is another method for grouping data points. It creates a hierarchy of clusters in the form of a tree-like structure, where each data point starts as its own cluster and then merges or splits based on similarities. Hierarchical clustering can be bottom-up or top-down. It is useful when the number of clusters is not known in advance and provides a visual representation of the clustering process.
- **Dimensionality reduction**: This involves reducing the number of features or dimensions in a dataset while retaining as much information as possible. It is particularly useful for

visualizing high-dimensional data and improving the efficiency of other algorithms. Some of the popular methods are: Bishop (2006)

  - **Principal Component Analysis (PCA)**: PCA is a dimensionality reduction technique that identifies a new set of orthogonal axes (principal components) in the data. These components capture the most significant variance in the data, allowing to represent it in a lower-dimensional space. It is widely used for data compression, feature selection, and visualization.
  - **Independent Component Analysis (ICA)**: ICA is a technique for finding statistically independent sources from observed data. It is commonly used in signal processing and image analysis. It can separate mixed signals into their original source signals, making it useful in applications like blind source separation and noise reduction.

- **Association rule learning**: This is about finding interesting relationships or associations between variables in large databases. A classic example is market basket analysis, where the goal is to discover what products customers tend to purchase together. Bishop (2006)

### 3.3.3 Semi-supervised learning

Semi-supervised learning harnesses the strengths of both supervised and unsupervised learning techniques. This approach is especially useful when there is an abundance of unlabeled data and limited labeled data, as obtaining labels can often be time-consuming, costly, or infeasible. By leveraging both types of data, semi-supervised learning can make better predictions and generalizations compared to using solely labeled or solely unlabeled data. Kumar et al. (2020)

In this method, the labeled data helps guide the model's learning process, while the unlabeled data assists in refining the model's structure and understanding of underlying patterns. An application example of Semi-supervised learning is:

- **Natural Language Processing (NLP)**: In tasks such as sentiment analysis or text classification, obtaining labeled data for every piece of text can be daunting. Semi-supervised learning can utilize the vast amounts of available unlabeled text data alongside smaller sets of labeled data for improved model performance. Bishop (2006)

### 3.3.4 Reinforcement learning

Reinforcement Learning (RL) is a type of machine learning paradigm where an agent learns to behave in an environment by performing certain actions and receiving rewards or penalties in return. The objective is to learn a strategy, often called a policy, that will maximize the cumulative reward for the agent over time. Unlike supervised learning, where the correct answers are

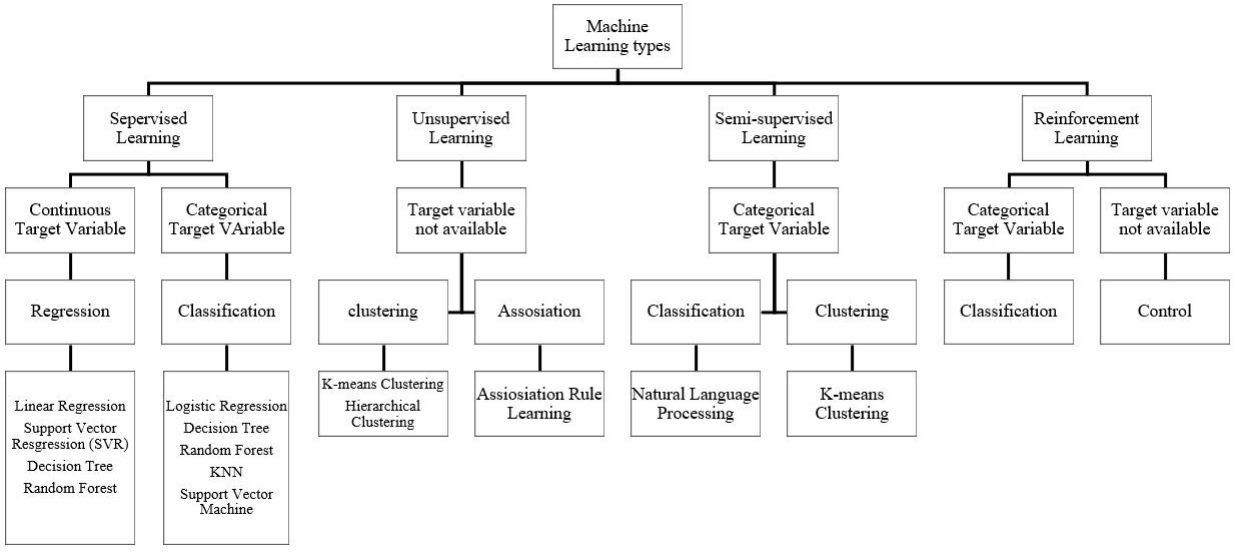



Figure 3.2: Machine learning paradigms, alogorithms, and their applications.

provided, and unsupervised learning, where only input data is given, RL is characterized by not having the right action provided explicitly. Instead, the agent discovers the optimal action by exploring the environment and exploiting its current knowledge. As the agent interacts more with its environment, it uses feedback in the form of rewards or penalties to refine its strategy. This process involves a balance between exploration (trying new actions) and exploitation (relying on known actions that yield good rewards). The mention of "trial and error search" and "delayed outcome" reflects the idea that the agent must try out different actions and might not realize the long-term consequences of an action immediately. Kumar et al. (2020); Bishop (2006)

Figure 3.2 illustrates all machine learning paradigms, related algorithms, and their applications.

## 3.4 Supervised learning algorithms

Supervised learning is a paradigm in machine learning where an algorithm is trained on a labeled dataset. This dataset consists of input examples paired with the correct output, allowing the model to learn the relationship between the inputs and outputs. The main components in supervised learning include: Janiesch et al. (2021)

**Input features (X variables)**: Often known as matrix of features, are the factors or characteristics that influence the prediction. Examples include metrics like the number of sold products or positive user reviews.

**Target variable (Y variable)**: Often known as dependent variable vector, is the output the model is trying to predict based on the input features. It could be a numeric prediction, like the

number of active users on a platform, or a categorical prediction, such as identifying users as "lookers" or "buyers".

Once the algorithm has been trained on this data, it can then be used to make predictions on new, unseen data. The goal is to adjust the model parameters in such a way that it makes accurate predictions.

Two main applications of supervised learning are regression and classification:

- **Regression**: In regression problems, the aim is to predict a continuous value. For instance, forecasting the number of active users on a platform based on various metrics.
- **Classification**: In classification problems, the objective is to predict a discrete label or category. For example, classifying users as "lookers" or "buyers" based on their browsing and purchasing behavior.

Various regression techniques, ranging from linear regression to more complex ones like support vector regression and decision tree as well as random forest regression, offer unique advantages depending on the nature of the data and the problem at hand. In the following these methods and the theory behind them are presented and discussed. Janiesch et al. (2021)

### 3.4.1 Linear regression

Linear regression is a statistical method used to forecast the association between two variables. This approach assumes a linear bond between the independent and dependent variables and tries to identify the optimal line that captures this bond. The chosen line minimizes the sum of the squared differences between the estimated and observed values. Mali (2021)

#### Simple linear regression

Simple linear regression is used for modeling the relationship between a dependent variable and one independent variable. It assumes a linear relationship between the two, aiming to find the slope and intercept of the straight line that best describes this relationship. The slope indicates how much the dependent variable shifts with a unit change in the independent variable. The intercept is the predicted value of the dependent variable when the independent variable is at zero. Eremenko (2023)

Figure 3.3 illustrates the linear connection between the dependent variable (y) and the independent variable (x). The line represents the optimal linear fit. In linear regression the aim is to draw a line that best aligns with the provided data points based on the given information. Mali (2021)

The formula for simple linear regression is:

$$y = \beta_0 + \beta_1 x \tag{3.1}$$

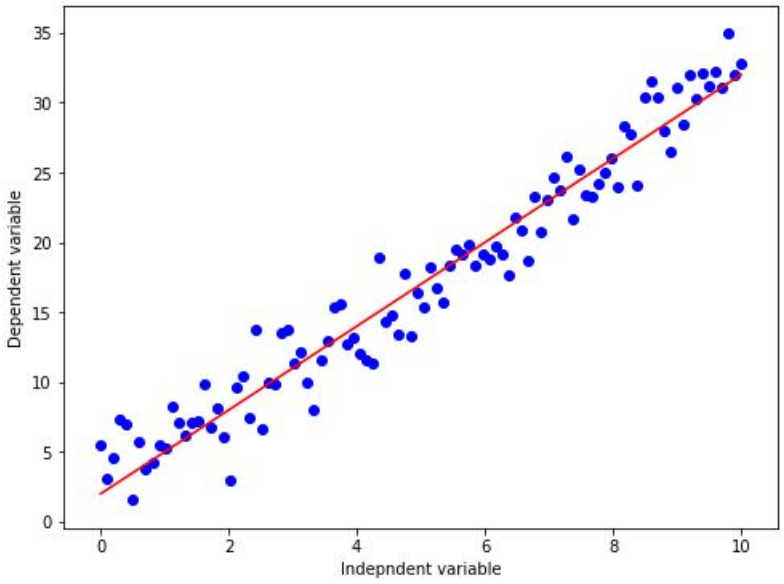



Figure 3.3: Regression line and scatter plot of real values.

- $y$ is the dependent variable.
- $x$ is the independent variable.
- $\beta_0$ is the y-intercept.
- $\beta_1$ is the slope coefficient.

For any given value of $x_i$, the residual $\epsilon_i$, refers to the difference between the actual value of the dependent variable ($y_i$) and its estimated value ($y_{\text{predicted i}}$).

$$\epsilon_i = y_{\text{predicted i}} - y_i \tag{3.2}$$

where:

$$y_{\text{predicted i}} = \beta_0 + \beta_1 x_i \tag{3.3}$$

**Multiple linear regression**

Multiple linear regression is a statistical technique that is used to understand the relationship between a single dependent variable and multiple independent variables. In simple linear regression, there is a one-to-one relationship between the input variable and the output variable. However, in multiple linear regression, there is a many-to-one relationship, between a number of independent variables and one dependent variable. Ray (2019); Alzubi et al. (2018)

The formulation for multiple linear regression is similar to simple linear regression, with the small change that instead of having one beta variable, we will now have betas for all the variables used. The formula is given as:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n \tag{3.4}$$

where:

- $y$ is the dependent variable.
- $\beta_0$ is the intercept.
- $\beta_1, \beta_2, \ldots, \beta_n$ are the coefficients for the independent variables.
- $x_1, x_2, \ldots, x_n$ are the independent variables.

The beta coefficients can be estimated using a variety of methods, such as ordinary least squares. The estimated beta coefficients can then be used to predict the value of the dependent variable for a given set of independent variables. Mali (2021)

**Polynomial regression**

Polynomial regression is a type of linear regression that incorporates polynomial terms because of the non-linear relationship between the predictor and the response variable. This allows the model to capture more complex patterns in the data.

In this approach, the relationship between the output and input variable is expressed as an nth-degree polynomial. For instance, a second-degree polynomial leads to a quadratic model, while a third-degree polynomial results in a cubic model, and the pattern continues for higher degrees.

In mathematical term: Mali (2021)

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \cdots + \beta_n x^n \tag{3.5}$$

During the preprocessing phase, before inputting data into a model, we transform the input variables by adding polynomial terms based on a specified degree. The choice of this degree is a hyperparameter that requires careful selection. Choosing a higher degree of polynomial might lead to overfitting, while a lower degree might result in underfitting. Thus, it is essential to find the best degree value. Typically, polynomial regression models are calibrated using the least squares method, which aims to minimize the variability of the coefficients. Mali (2021)

### Ordinary Least Square method (OLS)

The Ordinary Least Squares (OLS) method is used to find the best-fitting line through the data points such that the sum of the squared vertical distances of the points from the line is minimized.

The technique focuses on reducing the total of squared differences between observed values (actual values of the dependent variable) and the values estimated by the model (predicted values). The difference between an observed and estimated value is called a residual, sometimes referred to as an error. The sum of these squared differences is called the Residual Sum of Squares (RSS). By determining the optimal values for the coefficients to achieve the least RSS, the OLS method defines the most fitting line for the data, named the regression line. Kumar (2022)

Mathematically, this is expressed as minimizing following equation:

$$\sum (y_i - y_{\text{predicted}i})^2 \tag{3.6}$$

where $y_i$ is the actual value, $y_{\text{predicted}i}$ is the predicted value.

### Evaluation metrics

The best fit line is the one that most closely matches the provided scatter plot. In linear regression the aim is to minimize the differences between the observed value and the predicted value. The effectiveness of a linear regression model can be assessed through different evaluation indicators. These metrics typically give insights into how accurately the model predicts observed outputs. Mali (2021)

The most commonly employed metrics include:

- **Coefficient of determination or R-squared**: R-squared quantifies the proportion of variance in the data that is captured by the model. Its value lies between 0 and 1. Generally, a higher R-squared value indicates a better fit of the model to the data.

  In mathematical terms, it is expressed as:

  $$R^2 = 1 - \frac{RSS}{TSS} \tag{3.7}$$

  The residual sum of squares represents the total of the squared differences between the predicted and actual values for every data point in the dataset.

  $$RSS = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{3.8}$$

  Where:

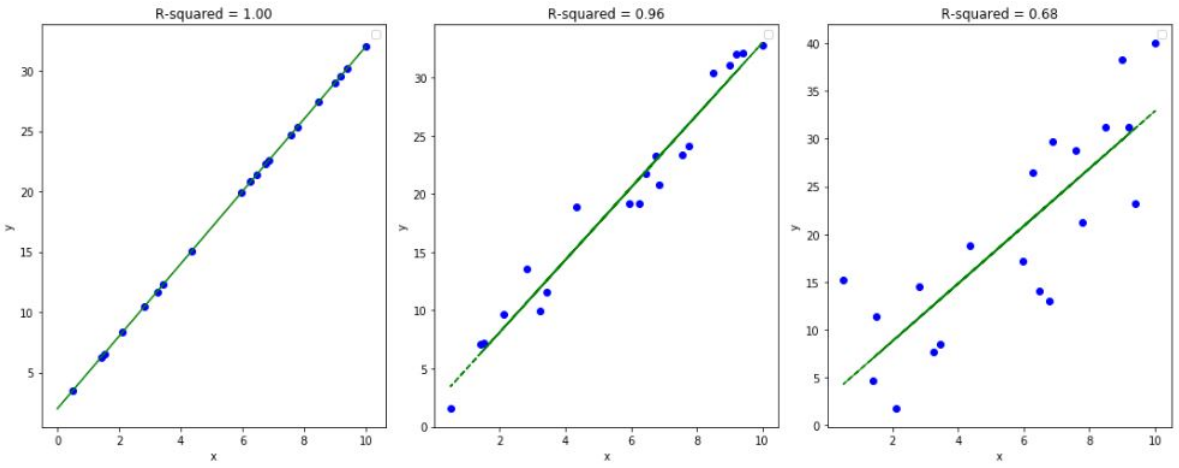



Figure 3.4: Different R-squared values for different regression lines.

- $y_i$ represents the actual observed value.
- $\hat{y}_i$ represents the predicted value from the regression.

The Total Sum of Squares (TSS) represents the cumulative squared deviations of the data points from the average of the response variable. In mathematical notation, TSS is expressed as:

$$TSS = \sum_{i=1}^{n} (y_i - \bar{y})^2 \tag{3.9}$$

Where:

- $\bar{y}$ represents the mean of the observed data.

Figure 3.4 illustrates different values for R-squared.

- **Root Mean Squared Error (RMSE)**: The Root Mean Squared Error (RMSE) is derived from the square root of the residual variances. It measures the model's accuracy in terms of how near the observed data points align with the forecasted values. In mathematical terms, RMSE is given by:

$$RMSE = \sqrt{\frac{RSS}{n}} \tag{3.10}$$

Where:

- $n$ is the number of observations or data points.

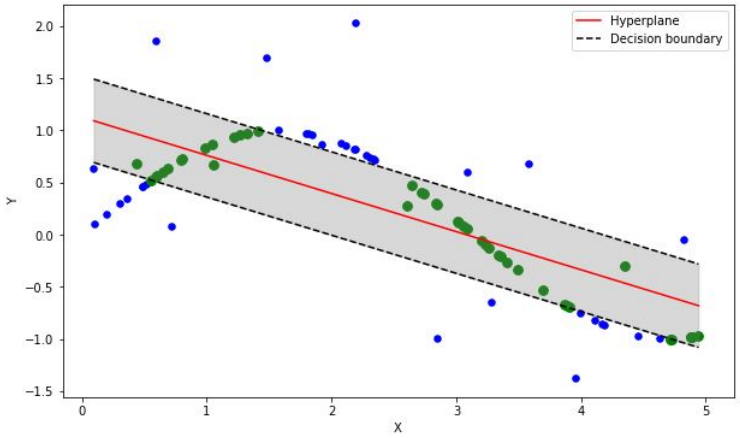



Figure 3.5: Hyperplane and decision boundaries in SVR.

R-squared is a more preferable metric than RMSE. This is because the value of the root mean squared error is unit-dependent (i.e., it is not standardized), so its value can vary if the unit of the variables changes. Mali (2021)

### 3.4.2 Support Vector Regression (SVR)

Support Vector Machines (SVM) aim to determine an optimal hyperplane within an n-dimensional space to effectively separate or predict data points. Critical to this method are the data points, called support vectors, that are close but outside to this hyperplane from both sides. These vectors play a pivotal role in shaping the hyperplane's orientation and position, helping to build the SVM's structure. Awad and Khanna (2015)

Unlike support vector machines which are utilized for classification, SVR is designed to determine a hyperplane that represents data points in a continuous space. This objective is accomplished by projecting the input variables into a higher-dimensional feature space and finding a hyperplane that both maximizes the distance between itself and the nearest data points and minimizes the prediction error.

SVR is adept at navigating non-linear correlations between input variables and the target variable, thanks to kernel functions that transport the data to a higher dimensional space. This capability makes SVR a potent tool for regression challenges where complex relationships may exist between input variables and the target variable. Mali (2021)

Think of the two gray dashed lines as the boundary for making decisions (Figure 3.5), with the red line representing the hyperplane. In SVR, our main focus is on the data points contained within these gray boundary lines. The optimal hyperplane is the one including the highest number of these points.

Assume these boundaries are set at a specific distance, called $\epsilon$, from the hyperplane. These lines are drawn at positions $+\epsilon$ and $-\epsilon$ relative to the hyperplane.

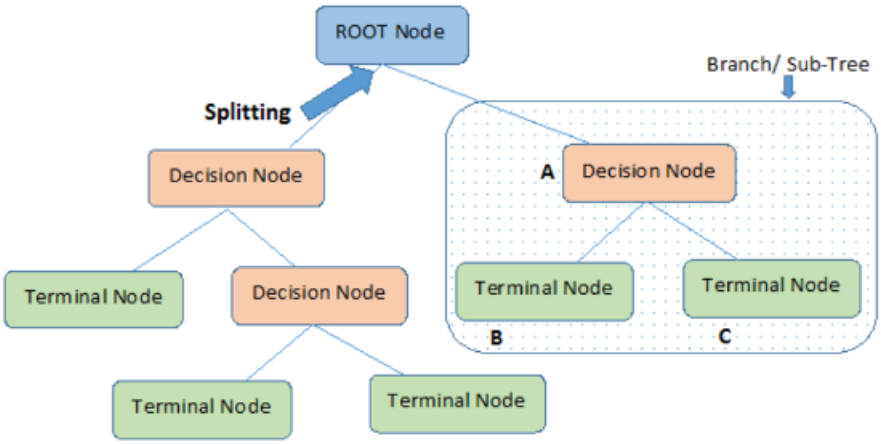



Figure 3.6: Decition tree architecture from (Mali (2021)).

Assume the following is the equation for the hyperplane:

$$f(x) = \langle w, x \rangle + b \tag{3.11}$$

The decision boundaries can then be mathematically expressed as:

$$f(x) + \epsilon \tag{3.12}$$

$$f(x) - \epsilon \tag{3.13}$$

Therefore, for a hyperplane to be valid in our SVR framework, it should fulfill:

$$-\epsilon < f(x) - (\langle w, x \rangle + b) < \epsilon \tag{3.14}$$

### 3.4.3 Decision tree

A decision tree is a predictive model structured similarly to a flowchart; guiding decisions based on input data. It divides the data into subsets through branches, eventually leading to outcomes at its terminal points, or leaf nodes. This model is versatile, suitable for both classification and regression tasks, and stands out for its intuitive, visual representation. Mali (2021)

This hierarchical tool shows various decisions and their potential consequences. It operates using conditional control statements and represents a non-parametric, supervised learning method. The tree's structure originates from a root node according to Figure 3.6, extending through branches and internal nodes, ultimately culminating in leaf nodes, which are terminal points where decisions are made.

Essentially, a decision tree systematically breaks down data based on feature distinctions, starting from a foundational root node, and progressing through a series of splits, resulting in decisions at the leaves.

Key terminologies associated with decision trees include: Mali (2021)

- **Root nodes**: The starting point of a decision tree, from which data begins to branch based on various features.
- **Decision nodes**: Nodes derived from the division of root or other internal nodes.
- **Leaf nodes (terminal nodes)**: Endpoints of the tree where no further splits occur, representing final decisions or outcomes.
- **Sub-tree**: A smaller section of the larger tree, similar to how a sub-graph is a portion of a larger graph.
- **Pruning**: The process of trimming certain branches or nodes to prevent overfitting, thereby optimizing the tree's performance.

### 3.4.4 Random forest

Random Forest (RF) is a versatile machine learning technique employed for both classification and regression tasks. Essentially, it works by combining multiple decision trees to generate a consolidated prediction. The power of RF lies in its ensemble approach, where numerous trees are built from the training data and internally tested to provide predictions for future data points. There are various forms of RF, differentiated by three main aspects: Boulesteix et al. (2012)

- **Tree construction**: This relates to the methodology employed to build each individual tree within the forest.
- **Data generation**: This refers to the approach used to create the altered datasets on which every individual tree is formulated.
- **Prediction aggregation**: This deals with how the outputs from each tree are combined to generate a singular, consensus prediction.

In the general random forest method, every tree is a conventional Classification or Regression Tree (CART). This method uses a specific criterion known as the decrease of Gini impurity (DGI) for splitting nodes. Furthermore, instead of using all predictors for splits, only a random subset of predictors is chosen at each node split, adding a layer of randomness that often

improves model performance. To build each tree, a bootstrap sample (a sample taken with replacement) is drawn from the original dataset. Finally, when it comes to making predictions, the outputs from all trees are aggregated, often through a majority vote, ensuring a robust and less overfitting outcome. Boulesteix et al. (2012)

Random forests are particularly esteemed for their ability to handle large data sets with higher dimensionality. They can also assess the importance of different features in the dataset, which can be instrumental for feature selection. While they tend to be resistant to overfitting due to their ensemble nature, they can still become complex and computationally intensive when the number of trees is significantly large. Boulesteix et al. (2012)

## 3.5 Artificial Neural Network (ANN)

Deep learning, a specialized branch of machine learning, draws its inspiration from the intricacies of the human brain. It employs Artificial Neural Networks (ANN) that attempt to replicate the functionalities of the human brain, enabling machines to learn from experiences and data, much like we do. Deepanshi (2021)

Artificial neural networks, also known as Neural Networks (NNs) are advanced computational structures inspired by the human brain. They have been applied across diverse fields, from computing and medicine to engineering and economics. Rooted in optimization theory, an ANN is constructed using a series of artificial neurons, or processing units, linked by various weights. These weights signify the strength of connections between neurons, dictating how one neuron influences another. The term 'network' in neural network points to the intricate links between these neurons across multiple layers of the system. Zakaria et al. (2014)

Similar to our brain, artificial neural networks consist of multiple layers of neurons (Figure 3.7), often termed as nodes. In the context of ANNs, the dendrites in biological networks symbolize inputs, the cell nuclei stand for nodes, synapses indicate weights, and axons depict the output. Kalita (2022)

The most basic form of a neural network consists of a single unit, referred to as a neuron or node, depicted in Figure 3.8. To construct a more complex NN, multiple such neurons are combined. The procedure of processing data from input to output is consistent across all neurons in the system. A typical NN comprises an input layer, one or several hidden layers, and an output layer. Every layer contains a specified number of nodes. Any layer that is not explicitly an input or output layer is considered a hidden layer. The output layer produces the predicted data or results. Lærum (2018)

In this architecture, each neuron undertakes basic tasks: it multiplies its inputs $(x_1, x_2, \ldots, x_n)$ by specific weights $(w_1, w_2, \ldots, w_n)$, aggregates the results $(Z)$, and then passes this sum through a typically nonlinear activation function $(a)$, producing an output $(y)$, as illustrated in Figure 3.9.

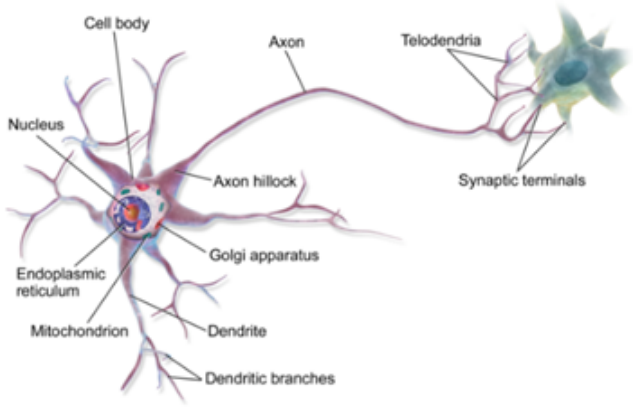



Figure 3.7: Human brain's neuron from (Kalita (2022)).

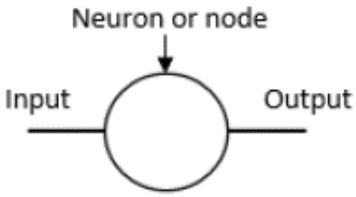



Figure 3.8: ANN with a single neuron from (Lærum (2018)).

Despite the simplicity of individual nodes, when grouped into dense networks, they have the capacity to represent highly intricate functions. These functions are characterized by the neuron weights, which are well-tuned during the training phase. Rosa et al. (2019)

Figure 3.9 depicts key components of a single neuron, and they are as follows: Kalita (2022)

- **Input**: This refers to the specific features provided to the model for training.
- **Weights**: Weights determine the degree of influence an input has on the outcome. Less relevant inputs get reduced emphasis, while more related ones are given more impor-

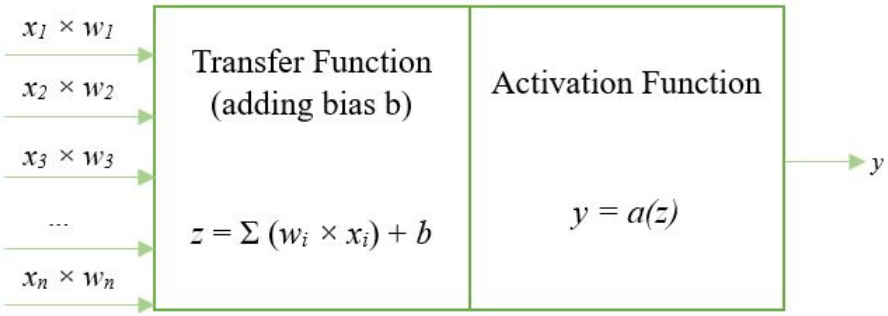



Figure 3.9: Key components of a neuron from (Rosa et al. (2019)).

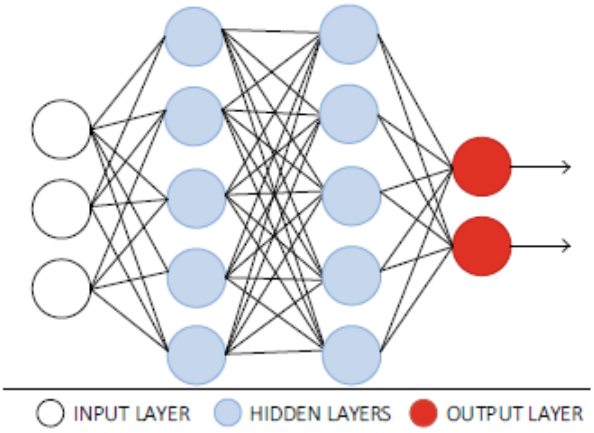



Figure 3.10: ANN with two hidden layers from (Rosa et al. (2019)).

tance. For instance, in sentiment analysis, a negative term would be more influential in the decision-making process compared to a neutral term.

- **Bias**: Bias helps regulate the activation of a neuron. It can either accelerate or delay this activation.
- **Transfer function**: This function's role is to combine the inputs (which have been multiplied by their weights) in preparation for the application of the activation function.
- **Activation function**: This function determines if a neuron should be activated. It employs mathematical operations to decide whether the provided input is significant enough for the neuron to be activated.

When several neurons are grouped together, they form a layer. Likewise, when multiple such layers are combined, it results in a multi-layer neural network as shown in Figure 3.10.

The primary elements of this arrangement include: Kalita (2022)

- **Input layer**: This is the initial layer in a neural network. Its function is to bring the data into the system for subsequent processing by the succeeding layers of artificial neurons.
- **Hidden layers**: These layers follow the input layer. Their role is to transform the incoming inputs in a non-linear manner. Each hidden layer can produce outputs tailored to specific tasks.
- **Output layer**: This is where the data, having been processed by the hidden layers, makes the final prediction. The ultimate result or decision of the network emerges from this layer.

All incoming inputs are summed up, with each input being adjusted based on its specific weight. This scaling, determined by the weights, is optimized during the network's training process to ensure the neuron responds to the right features. In addition to the weights, the neuron's bias is another trained parameter that plays a role in this summation. The bias acts as an offset addition to the weighted sum, effectively shifting the activation function's outcome either positively or negatively. Kalita (2022)

Let's assume we have a neuron that receives two inputs, $x_1$ and $x_2$. These inputs are each multiplied by their respective weights, $w_1$ and $w_2$. These product values are then combined and an additional bias, $b$, can be added. Here, you can see the representation of the input, horizontal vector $\mathbf{X}$, and the weight, vertical vector $\mathbf{W}$.

$$\mathbf{X} = [x_1, x_2] \tag{3.15}$$

$$\mathbf{W} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \tag{3.16}$$

The equation that represents the combined total of inputs multiplied by their weights, along with the bias, is as follows:

$$Z = \mathbf{X} \cdot \mathbf{W} + b \tag{3.17}$$

After having multiplication of two vectors, we have: Kalita (2022)

$$\mathbf{X} \cdot \mathbf{W} = \sum_{i=1}^{n} x_i w_i = x_1 w_1 + x_2 w_2 \tag{3.18}$$

After adjusting the inputs according to their weights, combining them, and adding the bias to get the value $Z$, we need to use the activation function on $Z$ to determine the neuron's output as previously depicted in Figure 3.9.

### 3.5.1 Activation function

Activation functions play an important role in neural networks. Without activation functions, neural networks would be much less powerful and unable to model complex, non-linear relationships in data. They transform the linear combinations of weights and inputs in a way that allows the model to approximate any function, given a suitable architecture and enough data. The primary purpose of the activation function is to introduce non-linearity into the network. Without non-linearity, no matter how many layers the network has, it would behave just like a single layer because summing these layers would give just another linear function. Non-linearity allows the network to capture complex patterns and relationships in the data, making deep learning possible. Moreover, activation functions help in transforming and shaping the

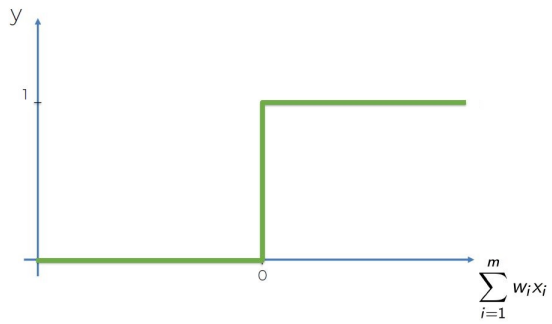



Figure 3.11: Threshold activation function from (Eremenko (2023)).

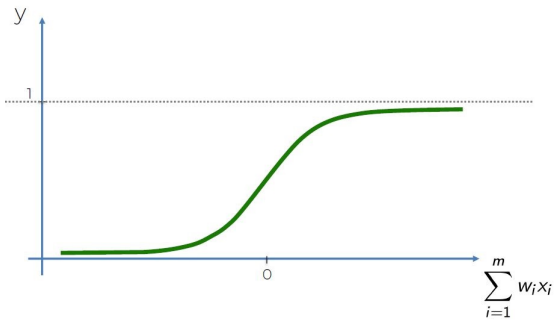



Figure 3.12: Sigmoid activation function from (Eremenko (2023)).

output in a way that can be helpful for the subsequent layer to learn from. This sequential transformation of features allows the network to learn hierarchical representations. Eremenko (2023)

Most common activation functions are as follows: Eremenko (2023)

- **Threshold function**: The threshold function is a basic form of activation function. It operates by choosing a threshold. If the weighted sum input, here denoted as $Z$, surpasses this threshold, which is zero in this case, then the neuron gets activated. (Figure 3.11)

$$f(Z) = \begin{cases} 1 & \text{if } Z \geq 0 \\ 0 & \text{if } Z < 0 \end{cases} \tag{3.19}$$

- **Sigmoid function**: The sigmoid activation function is commonly used in neural networks. It is characterized by its s-shaped curve. Sigmoid function converts the value between the range 0 and 1. (Figure 3.12)

$$f(Z) = \frac{1}{1+e^{-Z}} \tag{3.20}$$

- **Hyperbolic Tangent (Tanh)**: The hyperbolic tangent activation function is like the sig-

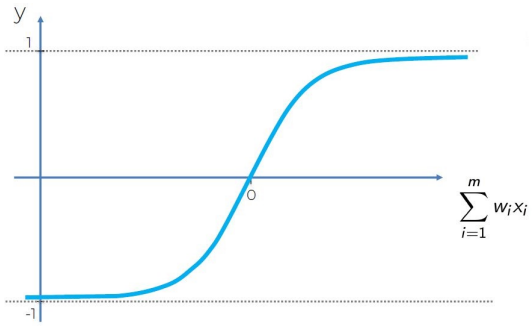



Figure 3.13: Hyperbolic tangent activation function from (Eremenko (2023)).

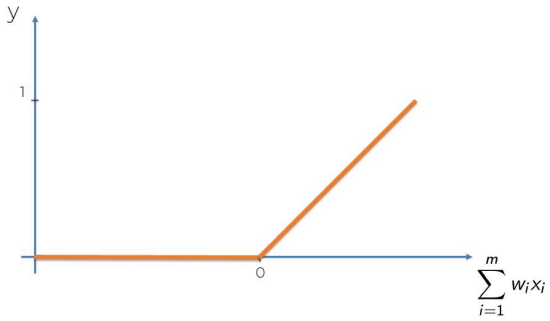



Figure 3.14: Rectified linear unit activation function from (Eremenko (2023)).

moid function, representing an s-like curve. However, its output values range from −1 to 1. (Figure 3.13)

$$f(Z) = \frac{e^{Z} - e^{-Z}}{e^{Z} + e^{-Z}} \tag{3.21}$$

- **The Rectified Linear Unit (ReLU)**: ReLU is a type of activation function that is widely used in deep learning models. Graphically, the function looks like a ramp that starts from 0 and grows linearly with positive values of $Z$. If the input is $Z$, the output will be $Z$ if $Z$ is positive, otherwise, it outputs zero.(Figure 3.14)

$$f(Z) = \max(0, Z) \tag{3.22}$$

### 3.5.2 Cost function

After training the neural network, it is essential to evaluate its performance. This is where the concept of a cost function becomes pivotal. The cost function quantifies the discrepancy or error between the predicted outcomes and the actual results. It essentially offers a numerical representation of the model's inaccuracies or mispredictions. In neural networks, the learning

or training phase revolves around optimizing this cost function. The objective is to adjust the model's parameters in such a way that this error measure, often denoted as $J$ or $C$, is minimized. Commonly used cost functions include the Mean Squared Error (MSE) and the Mean Absolute Error (MAE) as defined in (3.23 & 3.24), where $M$ represents the total number of samples, $x_i$ denotes the input vector, $y_i$ is the expected output, and $w$ stands for the model's weights. By optimizing these functions, neural networks refine their predictions, aiming to achieve better performance on new data. Rosa et al. (2019)

$$MSE(x, y, w) = \frac{1}{M} \sum_{i=1}^{M} [ANN(x_i, w) - y_i]^2 \tag{3.23}$$

$$MAE(x, y, w) = \frac{1}{M} \sum_{i=1}^{M} \left| ANN(x_i, w) - y_i \right| \tag{3.24}$$

To address this optimization problem, gradient descent and backpropagation are used. Gradient descent is a repetitive process that navigates the cost function in the opposite direction of the gradient. When training ANNs, backpropagation is used to determine the gradient of the cost function with respect to the weights. Throughout the training process, the error's derivative with respect to each weight is propagated back through the network (which is why it is termed backpropagation), facilitating the adjustment of the weights according to equation (3.25): Rosa et al. (2019)

$$w^{(n+1)} = w^{(n)} - \eta \frac{\partial J}{\partial w}\Big|_{w^{(n)}} \tag{3.25}$$

In this context, $J$ signifies the cost function, $w$ represents a generic weight, and $w^{(n+1)}$ is the value of that weight when it gets updated. Additionally, $\eta$ is a scalar referred to as the learning rate. Backpropagation uses the chain rule, offering a simplified method to iteratively compute the cost function's gradient based on the derivatives of the activation functions. The rule for backpropagation is denoted as (3.26), where $u_w$ indicates the input of the weight's net in the feed-forward network, and $v_w$ stands for the input of the same weight's net during the backpropagation process. Rosa et al. (2019)

$$\frac{\partial J}{\partial w}\Big|_{w^{(n)}} = u_w v_w \tag{3.26}$$

The learning rate controls how much influence the gradient has on weight adjustments. When set too high, the learning rate might cause the algorithm to surpass the lowest point of the loss function or, in more extreme cases, stay away from it. Conversely, a too small learning rate may result in a slow convergence, as depicted in Figure 3.15. The learning rate is a pivotal hyperparameter as it governs the direction (either towards convergence or divergence) of the optimization algorithm. Rosa et al. (2019)

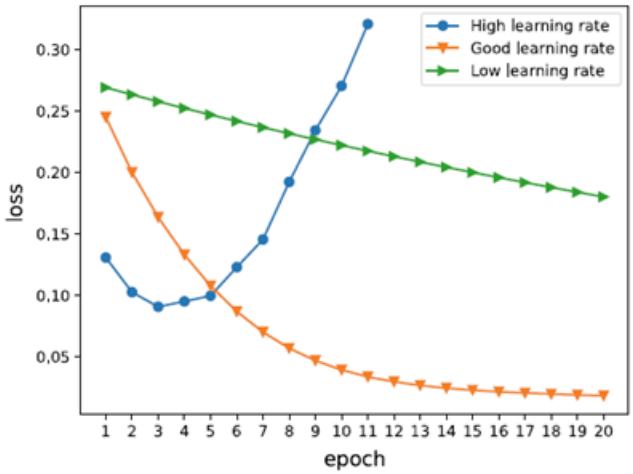



Figure 3.15: Evolution of the error with different learning rates from (Rosa et al. (2019)).

# Chapter 4

# Building The Machine Learning Models

This chapter provides an overview of the programming language Python, tools, libraries, and platforms employed in the development of machine learning models. Then presents the process of constructing and programming these models for preliminary testing. All models, along with the data preprocessing, aim to predict the remaining useful life of the aircraft turbofan engine based on the degradation simulation dataset from NASA. The performance metrics adopted are root mean squared error (RMSE) and R-squared. Libraries such as NumPy, Scikit-learn, TensorFlow, Pandas, and Matplotlib are employed in this work. All coding tasks are executed within the Python Jupyter notebook environment.

## 4.1 Programming language

Python, created by Guido van Rossum and first released in 1991, is a high-level, versatile programming language renowned for its clear syntax and readability, which was inspired by the ABC language. Over the decades, Python has evolved and gained widespread adoption, especially in scientific and academic communities. In the realm of machine learning, Python has become the language of choice due to its vast ecosystem of specialized libraries like TensorFlow, Keras, and Scikit-learn. These libraries provide pre-built functions and tools that simplify the development and implementation of complex machine learning algorithms. Additionally, Python's adaptability and integration capabilities make it easy to work with large datasets, conduct data analysis, and interface with other technologies. Inoxoft (2022)

There are several types of environments that we can develop to run Python code like:

- Text editors
- Full IDEs
- Notebook environments

In this master thesis, the Jupyter notebook as the most popular notebook environment for executing Python code is employed. Jupyter notebook provides an interactive interface that facilitates the integration of live code, visualizations, and narrative text within a single document. This notebook environment is particularly advantageous for Python programming and data analysis as it allows for real-time execution and feedback, enabling iterative development and immediate visualization of results. Eremenko (2023)

## 4.2 Libraries

In programming, a library refers to a set of precompiled codes that can be utilized in subsequent programs for some specific operations. Beyond just precompiled codes, a library can have documentation, configuration settings, message templates, and various elements such as classes and values. In Python, a library represents an assembly of related modules. These modules consist of code segments that can be used across different programs repeatedly, thereby simplifying the programming process, and improving efficiency. By eliminating the need to repeatedly write the same code for different applications, Python libraries become crucial, especially in areas like machine learning, data science, and data visualization. Eremenko (2023)

- **NumPy**: NumPy, which stands for Numerical Python, is one of the foundational packages for numerical computations in Python. It provides support for large multi-dimensional arrays and matrices, along with a collection of high-level mathematical functions to operate on these arrays. Eremenko (2023)

- **Pandas**: Pandas is a powerful and widely used open-source data analysis and manipulation library for the Python programming language. It provides flexible data structures that make it easy to efficiently manipulate structured data. Given its capabilities and versatility, Pandas has become a vital tool for data scientists, analysts, and researchers working with Python, especially for tasks related to data cleaning, transformation, and exploration. Eremenko (2023)

- **Matplotlib**: Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. It can produce a wide variety of plots and figures, including line plots, scatter plots, bar plots, histograms, error charts, pie charts, box plots, and even more complex visualizations like 3D plots. Eremenko (2023)

- **Scikit-learn**: Scikit-learn, often referred to simply as Sklearn, is a machine learning library in Python. It is built based on NumPy, SciPy, and Matplotlib. It is known for its user-friendly and straightforward Application Programming Interface (API), as well as its

Table 4.1: The dataset description of aircraft turbofan engine. (Zhang et al. (2023))

| Dataset | FD001 | FD002 | FD003 | FD004 |
|---|---|---|---|---|
| Fault modes | 1 | 1 | 2 | 2 |
| Operational conditions | 1 | 6 | 1 | 6 |
| Training engines | 100 | 260 | 100 | 249 |
| Testing engines | 100 | 259 | 100 | 248 |

efficiency and utility in standard machine learning tasks. It includes a wide variety of algorithms for supervised and unsupervised learning like classification, regression, clustering, dimensionality reduction, and more. Beyond algorithms, it provides tools for model selection, data preprocessing, and evaluation. This includes functions for feature scaling, encoding categorical variables and imputing missing values among others. Eremenko (2023)

- **TensorFlow**: TensorFlow is one of the most widely used open-source frameworks for deep learning and machine learning tasks. While TensorFlow is known primarily for deep learning, it is versatile enough to be used for a wide range of machine learning tasks. It provides both high-level APIs for quick model development and low-level APIs for more custom, intricate work. TensorFlow's comprehensive features make it a leading choice for researchers and developers aiming to harness the power of machine learning and deep learning. Eremenko (2023)

## 4.3 Introduction of the dataset

The Prediction Center at NASA Ames offers a comprehensive dataset for aircraft turbofan engines, derived from the Commercial Modular Aero-Propulsion System Simulation (CMAPSS). This dataset contains data across four distinct operational conditions. Table 4.1 presents the structure and details of this dataset. Zhang et al. (2023)

Each of these subsets offers simulated run-to-failure paths of turbofan engines under varied operational conditions and fault modes. Each subset is further divided into a training dataset and a testing dataset. The datasets include readings of 21 sensors as presented in Table 4.2, that record the run-to-failure data. Every dataset forms a matrix with $m \times 26$ dimension, where $m$ represents the number of data points for each engine. In these matrices, rows indicate cycle-wise data, while columns represent elements like engine number, operational cycle number, three different operational settings, and the values from the 21 sensors. In total, the CMAPSS dataset consists of four training sets, four testing sets, and four RUL files. A distinguishing factor between the training and testing datasets is the endpoint data. In the training sets, the last data represents the engine's failure time, whereas the testing sets stop capturing sensor data some

Table 4.2: Details of 21 sensors for a turbofan engine. (Esfahani et al. (2021))

| Sensor No. | Symbol | Description |
|---|---|---|
| 1 | T2 | The total temperature at the fan inlet |
| 2 | T24 | The total temperature at the LPC outlet |
| 3 | T30 | The total temperature at the HPC outlet |
| 4 | T50 | The total temperature at the LPT outlet |
| 5 | P2 | Pressure at fan inlet |
| 6 | P15 | The total pressure in bypass-duct |
| 7 | P30 | The total pressure at HPC outlet |
| 8 | Nf | Physical fan speed |
| 9 | Nc | Physical core speed |
| 10 | epr | Engine pressure ratio (P50/P2) |
| 11 | Ps30 | Static pressureat HPC outlet |
| 12 | phi | The ratio of fuel flow to Ps30 |
| 13 | NRf | Corrected fan speed |
| 14 | NRc | Corrected core speed |
| 15 | BPR | Bypass ratio |
| 16 | farB | Burner fuel-air ratio |
| 17 | htBleed | Bleed enthalpy |
| 18 | Nf-dmd | Demanded fan speed |
| 19 | PCNfR-dmd | Demanded corrected fan speed |
| 20 | w31 | HPT coolant bleed |
| 21 | w32 | LPT coolant bleed |

time ahead of the actual failure, and the estimation of remaining useful life of the engine is expected. However, for validation purposes, actual RUL values are provided separately. For the scope of this master thesis, the first dataset, FD001, which operates under a single fault mode and one operational condition is used, to evaluate the efficacy and functionality of previously proposed machine learning and ANN models. In this chapter, the accuracy and performance of proposed models are analyzed based on two evaluation metrics, R-squared and RMSE. Esfahani et al. (2021)

## 4.4 Data preprocessing

The dataset is provided in a zip-compressed text file consisting of 26 columns of numbers, each separated by a space. Each row represents a set of data from a single operational cycle, while each column stands for a unique variable. These columns represent: NASA Repository (2023)

- Column 1: Unit number
- Column 2: Cycle
- Column 3 to 5: Operational setting
- Column 6 to 26: Sensor measurements

Within the training set FD001, each engine's measurements start from cycle one and continue for each cycle up to the point of failure. The data recorded at each cycle or row acts as an example. However, each example lacks a directly associated RUL. This means that there is not a direct corresponding $y$-value for each $x$.

Based on this data structure, several preprocessing steps are necessary. The initial step involves importing three files including `train_FD001`, `test_FD001` and `RUL_FD001` as training set, test set, and RUL for test set, respectively into the Jupyter notebook. Subsequently, finding a corresponding label or $Y$-value for train set or $X$. Finally, it is essential to remove the "Unit Number" and "Cycle" columns, as they are not relevant features for RUL.

The first number in the dataset, labeled "unit number", corresponds to a specific engine. For every new engine, the "cycle" number always starts from one and continues to increase with every operational cycle until the engine breaks down. By inverting the cycle column for every engine, each example or row can be assigned a respective RUL, or the number of cycles left before failure.

### 4.4.1 Importing the dataset

In the following code, Figure 4.1, first the necessary libraries are imported. For enhanced clarity, each column is assigned a name. Subsequently, all three files are loaded as Pandas framework: the training set, test set, and the remaining useful life (RUL). The first 5 rows of the training set are illustrated in Figure 4.1.

The `y_test` vector has the shape of $(100,1)$ and represents the remaining useful life (RUL) for 100 engines. On the other hand, the `x_test` matrix has a shape of $(13096,26)$. In `x_test`, each engine starts from an initial point and ends at a random cycle, with all the cycle records provided. However, for RUL prediction purposes, only the last cycle for each engine is of interest. Since the true RUL values in the `y_test` for the test set are provided solely for the last time cycle

```
# importing libraries
import pandas as pd
import numpy as np
# define column names for easy indexing
index_names = ['unit_number', 'cycles']
setting_names = ['setting_1', 'setting_2', 'setting_3']
sensor_names = ['s_{}'.format(i) for i in range(1,22)]
col_names = index_names + setting_names + sensor_names
# read data
train = pd.read_csv('train_FD001.txt',sep='\s+', header=None, names=col_names)
x_test = pd.read_csv('test_FD001.txt',sep='\s+', header=None, names=col_names)
y_test = pd.read_csv('RUL_FD001.txt', sep='\s+', header=None, names=['RUL'])
train.head()
```

| | unit_number | cycles | setting_1 | setting_2 | setting_3 | s_1 | s_2 | s_3 | s_4 | s_5 | ... | s_12 | s_13 | s_14 | s_15 | s_16 | s_17 | s_18 | s_19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | -0.0007 | -0.0004 | 100.0 | 518.67 | 641.82 | 1589.70 | 1400.60 | 14.62 | ... | 521.66 | 2388.02 | 8138.62 | 8.4195 | 0.03 | 392 | 2388 | 100.0 |
| 1 | 1 | 2 | 0.0019 | -0.0003 | 100.0 | 518.67 | 642.15 | 1591.82 | 1403.14 | 14.62 | ... | 522.28 | 2388.07 | 8131.49 | 8.4318 | 0.03 | 392 | 2388 | 100.0 |
| 2 | 1 | 3 | -0.0043 | 0.0003 | 100.0 | 518.67 | 642.35 | 1587.99 | 1404.20 | 14.62 | ... | 522.42 | 2388.03 | 8133.23 | 8.4178 | 0.03 | 390 | 2388 | 100.0 |
| 3 | 1 | 4 | 0.0007 | 0.0000 | 100.0 | 518.67 | 642.35 | 1582.79 | 1401.87 | 14.62 | ... | 522.86 | 2388.08 | 8133.83 | 8.3682 | 0.03 | 392 | 2388 | 100.0 |
| 4 | 1 | 5 | -0.0019 | -0.0002 | 100.0 | 518.67 | 642.37 | 1582.85 | 1406.22 | 14.62 | ... | 522.19 | 2388.04 | 8133.80 | 8.4294 | 0.03 | 393 | 2388 | 100.0 |

5 rows × 26 columns

Figure 4.1: Importing libraries and dataset files.

```
# removing all extra rows and keep only last cycl of each engine
x_test = x_test.groupby('unit_number').last().reset_index()
x_test.shape
```

```
(100, 26)
```

Figure 4.2: Removing extra rows in `x_test`.

of each engine, the test set is adjusted to represent this as illustrated in Figure 4.2. As a result, all rows except the last one for each engine are removed and now the `x_test` and `y_test` are synchronized.

With both test sets (`x_test` and `y_test`) now prepared, our attention shifts towards the construction of the training sets, `x_train` and `y_train`. Unlike the test set, the training set does not have a direct RUL column for each cycle. To address this, a function called `add_remaining_useful_life`, detailed in Figure 4.3, is introduced.

The function operates by determining the RUL for each row in the following manner: it subtracts the current cycle number from the maximum cycle of the respective engine. This computation yields the RUL value for that specific row. After this RUL computation, the auxiliary column labeled `max_cycle` (which was used temporarily to facilitate the computation) is discarded. Heading of the modified training set can be reviewed in Figure 4.3.

In Figure 4.4, a detailed representation of the training set is presented. One noteworthy observation from this figure is the consistency in the values of the `setting_3` column. Given its static values, it is evident that this column does not show any variability, making its contribution to the predictive modeling process negligible. So, this column is removed from the train set.

```
def add_remaining_useful_life(df):
    # Get the total number of cycles for each unit
    grouped_by_unit = df.groupby(by="unit_number")
    max_cycle = grouped_by_unit["cycles"].max()

    # Merge the max cycle back into the original frame
    result_frame = df.merge(max_cycle.to_frame(name='max_cycle'), left_on='unit_number', right_index=True)

    # Calculate remaining useful life for each row
    remaining_useful_life = result_frame["max_cycle"] - result_frame["cycles"]
    result_frame["RUL"] = remaining_useful_life

    # drop max_cycle as it's no longer needed
    result_frame = result_frame.drop("max_cycle", axis=1)
    return result_frame
train = add_remaining_useful_life(train)
train.head()
```

| ıber | cycles | setting_1 | setting_2 | setting_3 | s_1 | s_2 | s_3 | s_4 | s_5 | ... | s_13 | s_14 | s_15 | s_16 | s_17 | s_18 | s_19 | s_20 | s_21 | RUL |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | -0.0007 | -0.0004 | 100.0 | 518.67 | 641.82 | 1589.70 | 1400.60 | 14.62 | ... | 2388.02 | 8138.62 | 8.4195 | 0.03 | 392 | 2388 | 100.0 | 39.06 | 23.4190 | 191 |
| 1 | 2 | 0.0019 | -0.0003 | 100.0 | 518.67 | 642.15 | 1591.82 | 1403.14 | 14.62 | ... | 2388.07 | 8131.49 | 8.4318 | 0.03 | 392 | 2388 | 100.0 | 39.00 | 23.4236 | 190 |
| 1 | 3 | -0.0043 | 0.0003 | 100.0 | 518.67 | 642.35 | 1587.99 | 1404.20 | 14.62 | ... | 2388.03 | 8133.23 | 8.4178 | 0.03 | 390 | 2388 | 100.0 | 38.95 | 23.3442 | 189 |
| 1 | 4 | 0.0007 | 0.0000 | 100.0 | 518.67 | 642.35 | 1582.79 | 1401.87 | 14.62 | ... | 2388.08 | 8133.83 | 8.3682 | 0.03 | 392 | 2388 | 100.0 | 38.88 | 23.3739 | 188 |
| 1 | 5 | -0.0019 | -0.0002 | 100.0 | 518.67 | 642.37 | 1582.85 | 1406.22 | 14.62 | ... | 2388.04 | 8133.80 | 8.4294 | 0.03 | 393 | 2388 | 100.0 | 38.90 | 23.4044 | 187 |

Figure 4.3: Defining a function to create RUL column in train set.

| | unit_number | cycles | setting_1 | setting_2 | setting_3 | s_1 | s_2 | s_3 | s_4 | s_5 | ... | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| count | 20631.000000 | 20631.000000 | 20631.000000 | 20631.000000 | 20631.0 | 2.063100e+04 | 20631.000000 | 20631.000000 | 20631.000000 | 2.063100e+04 | ... | 20631.00 |
| mean | 51.506568 | 108.807862 | -0.000009 | 0.000002 | 100.0 | 5.186700e+02 | 642.680934 | 1590.523119 | 1408.933782 | 1.462000e+01 | ... | 2388.09 |
| std | 29.227633 | 68.880990 | 0.002187 | 0.000293 | 0.0 | 6.537152e-11 | 0.500053 | 6.131150 | 9.000605 | 3.394700e-12 | ... | 0.07 |
| min | 1.000000 | 1.000000 | -0.008700 | -0.000600 | 100.0 | 5.186700e+02 | 641.210000 | 1571.040000 | 1382.250000 | 1.462000e+01 | ... | 2387.88 |
| 25% | 26.000000 | 52.000000 | -0.001500 | -0.000200 | 100.0 | 5.186700e+02 | 642.325000 | 1586.260000 | 1402.360000 | 1.462000e+01 | ... | 2388.04 |
| 50% | 52.000000 | 104.000000 | 0.000000 | 0.000000 | 100.0 | 5.186700e+02 | 642.640000 | 1590.100000 | 1408.040000 | 1.462000e+01 | ... | 2388.09 |
| 75% | 77.000000 | 156.000000 | 0.001500 | 0.000300 | 100.0 | 5.186700e+02 | 643.000000 | 1594.380000 | 1414.555000 | 1.462000e+01 | ... | 2388.14 |
| max | 100.000000 | 362.000000 | 0.008700 | 0.000600 | 100.0 | 5.186700e+02 | 644.530000 | 1616.910000 | 1441.490000 | 1.462000e+01 | ... | 2388.56 |

8 rows × 27 columns

Figure 4.4: Training set headings.

```
import matplotlib.pyplot as plt
def plot_sensor(sensor_name, ax):
    for i in train['unit_number'].unique():
        if (i % 10 == 0):  # only plot every 10th unit_number
            ax.plot('RUL', sensor_name,
                    data=train[train['unit_number'] == i])
    ax.set_xlim(250, 0)  # reverse the x-axis so RUL counts down to zero
    ax.set_xticks(np.arange(0, 275, 25))
    ax.set_ylabel(sensor_name)
    ax.set_xlabel('Remaining Useful Life')
fig, axes = plt.subplots(7, 3, figsize=(15, 17.5))  # 7 rows, 3 columns
for sensor_name, ax in zip(sensor_names, axes.ravel()):
    plot_sensor(sensor_name, ax)
plt.tight_layout()
plt.show()
```

Figure 4.5: Function for plotting sensor data.

### 4.4.2 Data visualization

Before deploying machine learning models, data visualization plays a pivotal role in shaping the analytics process. It allows for a comprehensive understanding of data distributions, aiding in recognizing inherent patterns and potential anomalies. By illustrating data graphically, outliers (which might distort the model's performance) can be easily identified and addressed. Beyond outliers, visual tools can shed light on how various features interact with one another. This can be instrumental in refining feature selection or engineering.

So, the next step is examining the sensor signals in relation to RUL to differentiate between "useful" and "irrelevant" sensors. Specifically, the aim is to identify sensors rich in information versus those that are not as informative. To achieve this, a function called `plot_sensor` is used and depicted in Figure 4.5, which plots the sensor signals for every tenth turbofan. Illustrations of these plots for all sensors can be viewed in Figures 4.6–4.8.

Upon examining the visualizations, it appears that sensors 1, 5, 10, 16, 18, and 19 offer minimal or no significant information to aid in predicting the remaining useful life (RUL). These columns should be removed before any prediction tasks to enhance the speed and generalization of the algorithms.

Additional analysis, revealed by the provided heatmap in the next step, suggests that sensors resembling the patterns of sensors 6, 9, and 14 should also be excluded from the features.

It is noticeable that the remaining sensors exhibit a strong relationship with the RUL, particularly in the range of 0 to 125. This observation implies that for RUL values greater than 125, there is not any noticeable change in sensor data. So, in order to ensure the model performs better, all higher values of RUL (above 125) should be clipped to 125 to help improve RUL prediction. However, all machine learning models will be applied without clipping the RUL. Then, for comparative analysis, we will subsequently clip the RUL and examine the impact on our results and predictions.

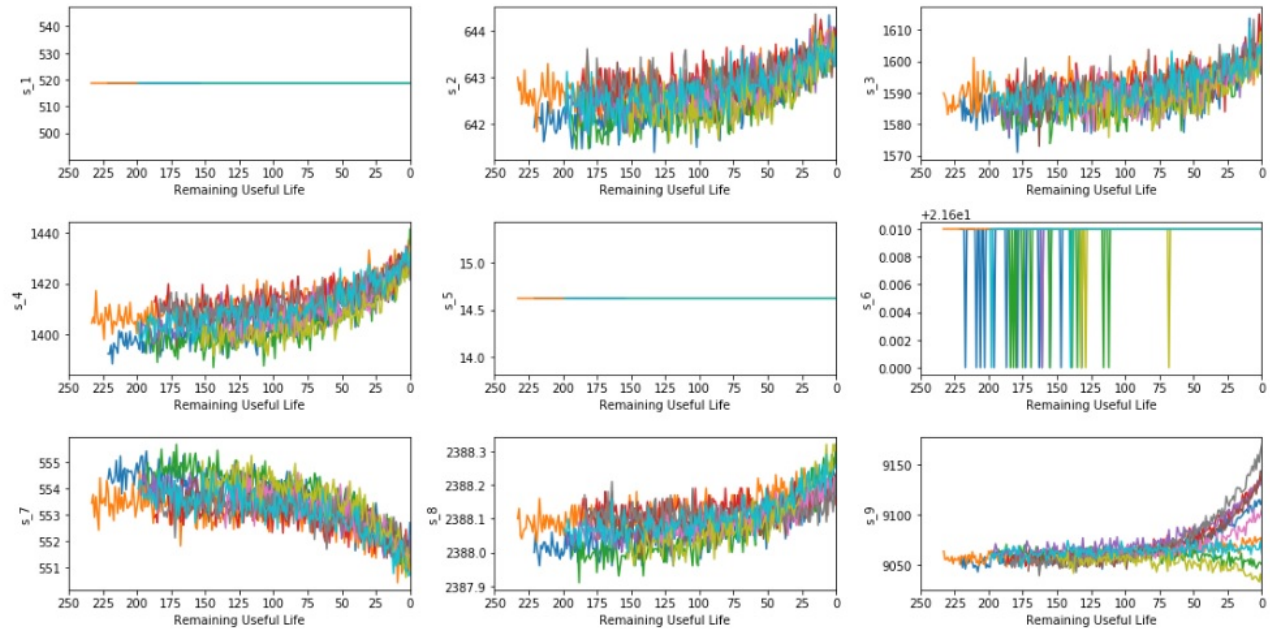



Figure 4.6: Sensor's plots from s_1 to s_9.

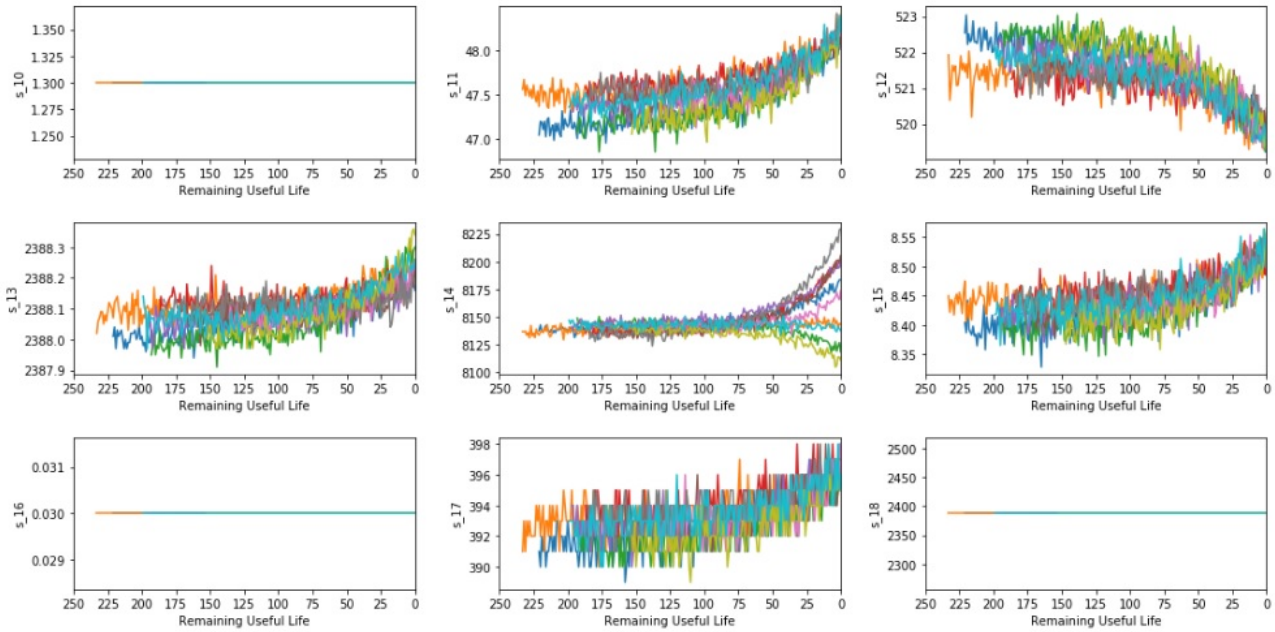



Figure 4.7: Sensor's plots from s_10 to s_18.

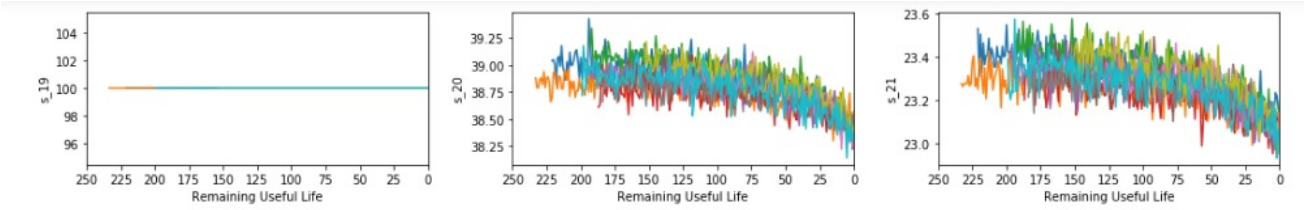



Figure 4.8: Sensor's plots from `s_19` to `s_21`.

### 4.4.3 Feature extraction

Heatmap is a data visualization tool that uses color to represent the correlation between features. A heatmap can provide a visual representation of how two variables (features) are related. Correlation quantifies the degree to which two variables (or features) move in relation to one another. Measure of correlation ranges between -1 and 1.

- A correlation of 1 indicates a perfect positive relationship.
- A correlation of -1 indicates a perfect negative relationship.
- A correlation close to 0 indicates little to no relationship.

Figure 4.9 illustrates the heatmap of features in train set.

Based on the code presented in Figure 4.10, only those features with an absolute correlation value with RUL of 0.5 or greater are selected. This is carried out to select only the essential features for model building, thereby preventing the issue of overfitting. Clearly, sensors that have a strong correlation with RUL, as shown in Figure 4.9, demonstrate the previous findings from visualizing the sensor data in section 4.4.2.

Now, only these important features will be retained in both the train and test sets. Moreover, the train set should be split into `x_train` and `y_train` to start building models. All these steps are shown in Figure 4.11.

Before beginning the model-building process, a function to evaluate machine learning models' performance is created as depicted in Figure 4.12. This function provides both R-squared and RMSE values, helping to eliminate redundancy in writing evaluation code for each model.

### 4.4.4 Standardization

Standardization is a preprocessing method used in machine learning and statistics to scale features (or variables) to have a mean of zero and a standard deviation of one. This is done to ensure that all features have the same scale, which can be important for many machine learning algorithms. When features in a dataset have different scales, algorithms might be influenced

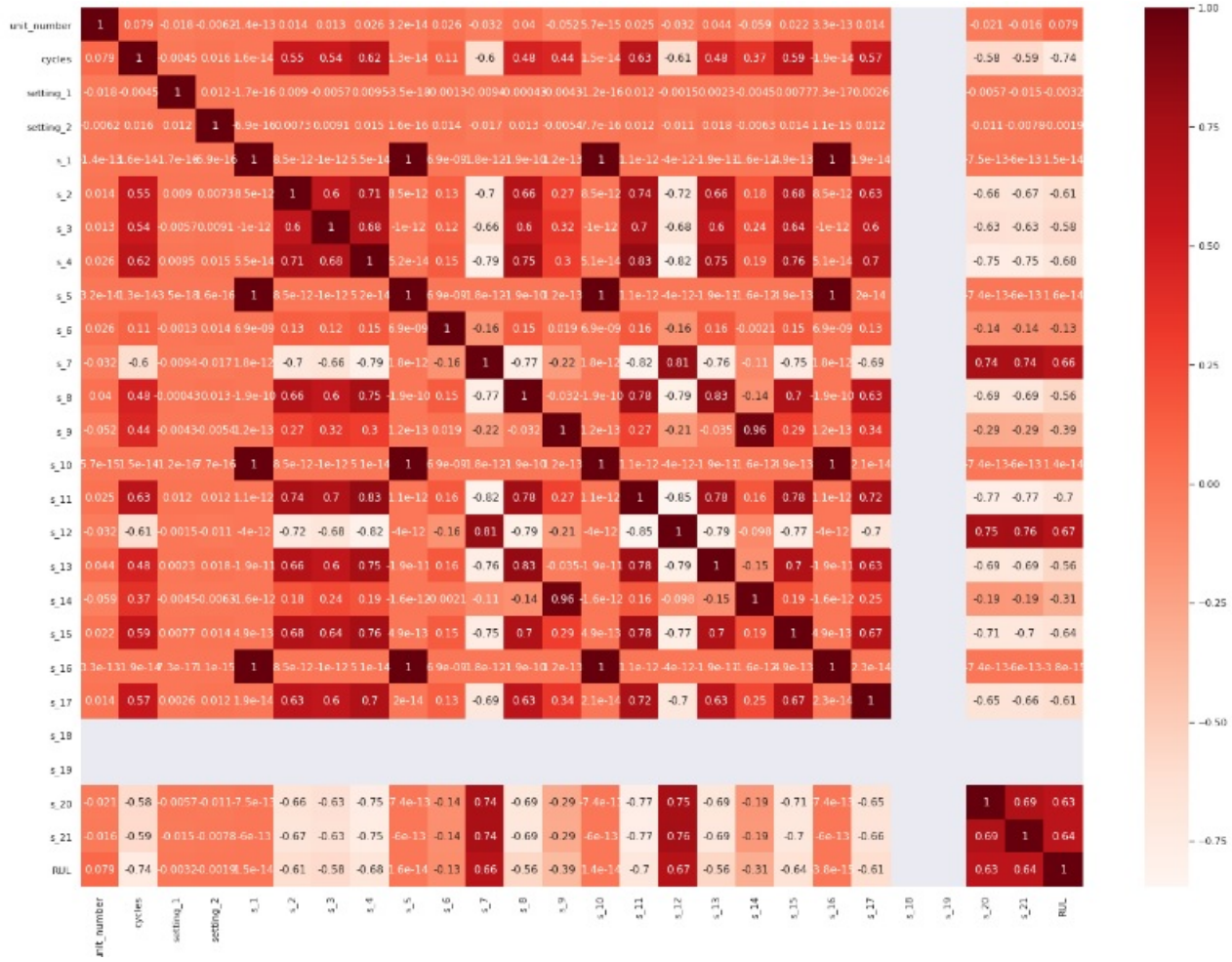



Figure 4.9: Heatmap visualization of sensors' correlation.

```
correlation=train.corr()
#Selecting highly correlated features
train_relevant_features = correlation[abs(correlation['RUL'])>=0.5]
train_relevant_features['RUL']
```

```
cycles   -0.736241
s_2      -0.606484
s_3      -0.584520
s_4      -0.678948
s_7       0.657223
s_8      -0.563968
s_11     -0.696228
s_12      0.671983
s_13     -0.562569
s_15     -0.642667
s_17     -0.606154
s_20      0.629428
s_21      0.635662
RUL       1.000000
```

Figure 4.10: Selecting only relevant features to RUL.

```
# creating a list of only important features.
list_relevant_features=train_relevant_features.index
list_relevant_features=list_relevant_features[1:]

# Now we will keep onlt these imprtant features in train set.
train=train[list_relevant_features]

# splitting train set into x_train and y_train(RUL).
y_train=train['RUL']
x_train=train.drop(['RUL'],axis=1)

# keeping only train columns/features in the test set as well.
x_test=x_test[x_train.columns]

x_train.head()
```

| | s_2 | s_3 | s_4 | s_7 | s_8 | s_11 | s_12 | s_13 | s_15 | s_17 | s_20 | s_21 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 641.82 | 1589.70 | 1400.60 | 554.36 | 2388.06 | 47.47 | 521.66 | 2388.02 | 8.4195 | 392 | 39.06 | 23.4190 |
| 1 | 642.15 | 1591.82 | 1403.14 | 553.75 | 2388.04 | 47.49 | 522.28 | 2388.07 | 8.4318 | 392 | 39.00 | 23.4236 |
| 2 | 642.35 | 1587.99 | 1404.20 | 554.26 | 2388.08 | 47.27 | 522.42 | 2388.03 | 8.4178 | 390 | 38.95 | 23.3442 |
| 3 | 642.35 | 1582.79 | 1401.87 | 554.45 | 2388.11 | 47.13 | 522.86 | 2388.08 | 8.3682 | 392 | 38.88 | 23.3739 |
| 4 | 642.37 | 1582.85 | 1406.22 | 554.00 | 2388.06 | 47.28 | 522.19 | 2388.04 | 8.4294 | 393 | 38.90 | 23.4044 |

Figure 4.11: Keeping only significant sensors and splitting the train set.

```
# create an evaluate function to calculate R-squared and RMSE
from sklearn.metrics import r2_score, mean_squared_error
def evaluate(y_true, y_hat, label='test'):
    mse = mean_squared_error(y_true, y_hat)
    rmse = np.sqrt(mse)
    variance = r2_score(y_true, y_hat)
    print('{} set RMSE:{}, R2:{}'.format(label, rmse, variance))
    return rmse,variance;
```

Figure 4.12: Evaluation function.

```
# Standardization of features
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train_trans = sc.fit_transform(x_train)
x_test_trans = sc.transform(x_test)
```

Figure 4.13: Standardization of features.

by those with larger scales. Standardizing the data helps in making the training process more efficient and can lead to better model performance. (Lærum (2018))

As depicted in Figure 4.4, there is a noticeable variance in values across different sensors. Given this observation, it is important to standardize these values, ensuring all features are rescaled to possess a mean of zero and a variance of one. This standardization process can be conveniently executed using Sklearn's `StandardScaler()` method followed by `fit_transform()`. Consequently, as illustrated in Figure 4.13, both `x_train` and `x_test` have been standardized. In the subsequent stages of model building, the transformed values, `x_train_trans` and `x_test_trans` will be used.

## 4.5 Building models

In this section, various machine learning models, including linear regression, polynomial regression, support vector regression, decision tree, random forest, and the artificial neural network are constructed, to compare their results. The process of building and training these models is made considerably straightforward due to the robust capabilities of the Scikit-learn and TensorFlow libraries. In the following, all codes related to model construction are provided.

### 4.5.1 Linear regression

Figure 4.14 displays the code block for the linear regression model. Both R-squared and RMSE are employed as evaluation metrics for the training and test sets. Observing the training set, we note an R-squared value of approximately 0.56. While this is not too good, it can potentially be related to the data not being strictly linear. However, when applying the model to the test set, the R-squared value decreases to 0.35, which is considerably low. Given this, we will proceed to build further more models to see how they are performing.

### 4.5.2 Polynomial regression

Executing the polynomial regression does not yield superior results. As depicted in Figure 4.15, the R-squared value for the training set is marginally better than that of the linear regression with the value of 0.61. However, its performance on the test set is inferior to the linear regression with R-squared of 0.29.

```
# Linear regression model
from sklearn.linear_model import LinearRegression
lm = LinearRegression()
lm.fit(x_train_trans, y_train)
# predict and evaluate
y_hat_train = lm.predict(x_train_trans)
RMSE_Train,R2_Train=evaluate(y_train, y_hat_train,'train')
y_hat_test = lm.predict(x_test_trans)
RMSE_Test,R2_Test=evaluate(y_test, y_hat_test,'test')
```

```
train set RMSE:45.61466077800371, R2:0.5614378099126991
test set RMSE:33.301161070181, R2:0.3578164045379345
```

Figure 4.14: Linear regression code block.

```
# Polynomial regression
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
poly_reg = PolynomialFeatures(degree = 4)
x_poly = poly_reg.fit_transform(x_train_trans)
lin_reg_2 = LinearRegression()
lin_reg_2.fit(x_poly, y_train)
# predict and evaluate
y_hat_train = lin_reg_2.predict(x_poly)
RMSE_Train,R2_Train=evaluate(y_train, y_hat_train,'train')
y_hat_test = lin_reg_2.predict(poly_reg.fit_transform(x_test_trans))
RMSE_Test,R2_Test=evaluate(y_test, y_hat_test,'test')
```

```
train set RMSE:42.89372356002194, R2:0.6121982317015551
test set RMSE:34.87992039035894, R2:0.295483083124645
```

Figure 4.15: Polynomial regression code block.

### 4.5.3 Support Vector Regression (SVR)

From the results depicted in Figure 4.16, it is evident that support vector regression outperforms both linear and polynomial regression, achieving an R-squared value of 0.54 on the test set. Typically, SVR exhibits superior performance compared to other classical machine learning models, especially when dealing with non-linear data. We continue to try other models.

### 4.5.4 Decision tree regression

The results for the decision tree are presented in Figure 4.17. While it exhibits the best performance on the training data, boasting an R-squared value of 0.71 compared to previous models, its performance significantly declines on the test set with an R-squared value of 0.15. One possible explanation for this discrepancy is model overfitting, where the model performs exception-

```
# SVR
from sklearn.svm import SVR
regressor = SVR(kernel = 'rbf')
regressor.fit(x_train_trans, y_train)
# predict and evaluate
y_hat_train = regressor.predict(x_train_trans)
RMSE_Train,R2_Train=evaluate(y_train, y_hat_train)
y_hat_test = regressor.predict(x_test_trans)
RMSE_Test,R2_Test=evaluate(y_test, y_hat_test)
```

```
test set RMSE:45.28091055080134, R2:0.5678320152955884
test set RMSE:28.04397904661384, R2:0.5445720042978832
```

Figure 4.16: Support vector regression code block.

```
# Decision Tree
from sklearn.tree import DecisionTreeRegressor
dt = DecisionTreeRegressor(random_state=42, max_depth=15, min_samples_leaf=10)
dt.fit(x_train_trans, y_train)
# predict and evaluate
y_hat_train = dt.predict(x_train_trans)
RMSE_Train,R2_Train=evaluate(y_train, y_hat_train, 'train')
y_hat_test = dt.predict(x_test_trans)
RMSE_Test,R2_Test=evaluate(y_test, y_hat_test)
```

```
train set RMSE:36.77513644517892, R2:0.7149435492153497
test set RMSE:38.31216453330952, R2:0.150009965299418
```

Figure 4.17: Decision tree regression code block.

```
# Random Forest
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(random_state=42, n_jobs=-1, max_depth=6, min_samples_leaf=5)
rf.fit(x_train_trans, y_train)
# predict and evaluate
y_hat_train = rf.predict(x_train_trans)
RMSE_Train,R2_Train=evaluate(y_train, y_hat_train, 'train')
y_hat_test = rf.predict(x_test_trans)
RMSE_Test,R2_Test=evaluate(y_test, y_hat_test)
```

```
train set RMSE:44.793331149655884, R2:0.5770889724209527
test set RMSE:32.42845735268905, R2:0.39103401537952087
```

Figure 4.18: Random forest regression code block.

ally well on the training data but fails to generalize effectively to new data.

### 4.5.5 Random forest regression

Based on the results for the random forest regression model presented in Figure 4.18, it did not face the same overfitting issue as the decision tree model. This is evident in its improved performance on the test set, achieving an R-squared value of 0.39. However, this is still not an optimal outcome. In subsequent sections, an artificial neural network (ANN) model will be implemented to evaluate if it can outperform the other models.

### 4.5.6 Artificial neural network

In this section, we attempt to construct an artificial neural network by selecting hyperparameters that optimize performance for our dataset. Hyperparameters dictate the architecture and behavior of the network during its training phase. They encompass various elements including the number of layers, nodes within each layer, activation functions, batch size, choice of optimizer, and number of epochs, among others. The act of hyperparameter tuning in neural networks is essentially a search to find the best combination of these variables. This process involves exploring different hyperparameter values, training and assessing the network performance under each configuration. Importantly, there is not a generic formula for this process. In fact, selecting the right hyperparameters is largely an iterative and intuitive process.

As depicted in Figure 4.19, we first start with an ANN architecture comprising two hidden

```
# ANN
import tensorflow as tf
ann = tf.keras.models.Sequential()
ann.add(tf.keras.layers.Dense(units=10, activation='relu'))
ann.add(tf.keras.layers.Dense(units=10, activation='relu'))
ann.add(tf.keras.layers.Dense(units=1))
ann.compile(optimizer='adam', loss='mean_squared_error')
ann.fit(x_train_trans,y_train, batch_size=32,epochs=75)
# predict and evaluate
y_hat_train = ann.predict(x_train_trans)
RMSE_Train,R2_Train=evaluate(y_train, y_hat_train, 'train')
y_hat_test = ann.predict(x_test_trans)
RMSE_Test,R2_Test=evaluate(y_test, y_hat_test,'test')
```

```
Epoch 1/75
645/645 [==============================] - 2s 2ms/step - loss: 7156.7144
```

Figure 4.19: Artificial neural network code block.

layers, each with 10 nodes. The rectified linear unit (ReLU) was selected as the activation function for both layers, and a batch size of 32 was employed. Considering the performance, ANN's results are somewhat comparable to the linear regression model, achieving an R-squared value of 0.35 on the test set and 0.58 on the training set.

## 4.6 Results

After evaluating all the models, the support vector regression (SVR) demonstrated the highest performance on the test set, achieving an R-squared score of 0.54. To gain a deeper understanding, we visually assessed the result of the SVR model to ensure its accuracy and reliability. It is essential to visualize predicted values against actual ones because a single metric, such as R-squared, does not always capture the complete picture. While the majority of predictions might be accurate, the presence of a significant outlier could make the model unsuitable for production. Figure 4.20 demonstrates the actual values of RUL versus predicted one in test set for SVR model.

From Figure 4.20, it is evident that the algorithm tends to overestimate its predictions especially in higher range of RUL. To address this issue, we try to clip the RUL, as elaborated in Section 4.4.2. Implementing RUL clipping, make the maximum RUL value in train set 125, which will enhance prediction accuracy, as we will see subsequently. This approach logically makes sense because for instance, sensor readings for the RUL of 175 are similar to those with an RUL of 125, making it challenging for the algorithm to differentiate effectively among this range. Following this strategy, we clipped the RUL at 125 and re-executed all models. A comparative result before and after clipping the data are provided in Table 4.3.

When we compare the results before and after clipping the dataset, there is a remarkable enhancement in prediction accuracy. Specifically, the R-squared value nearly doubled for all models after clipping. The RSME saw a decline of approximately 13 for the ANN model, with almost similar reductions observed for other models, a truly noteworthy improvement. From

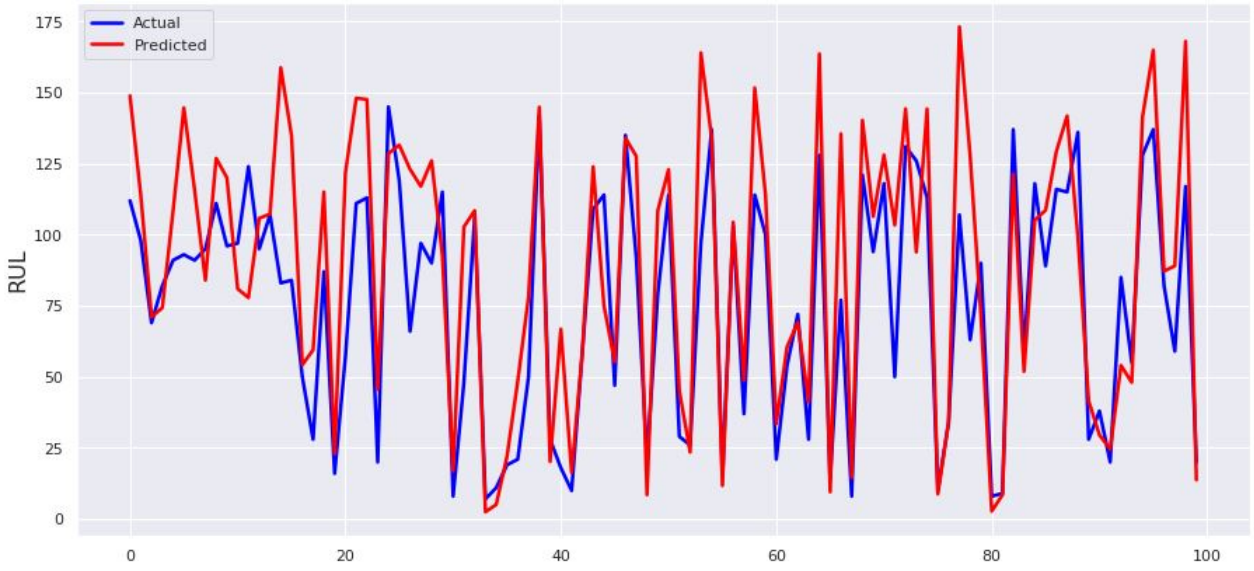



Figure 4.20: Actual RUL vs predicted RUL of the test data for SVR model.

Table 4.3: Models' results on train and test set.

| | Before clipping | | | | After clipping | | | |
|---|---|---|---|---|---|---|---|---|
| Model | RSME-Train | R2-Train | RSME-Test | R2-Test | RSME-Train | R2-Train | RSME-Test | R2-Test |
| LR | 45.61 | 0.56 | 33.30 | 0.35 | 22.73 | 0.70 | 22.91 | 0.69 |
| PR | 42.89 | 0.61 | 34.87 | 0.29 | 19.99 | 0.77 | 21.26 | 0.74 |
| SVR | 45.28 | 0.57 | 28.04 | 0.54 | 21.42 | 0.73 | 21.82 | 0.72 |
| DT | 36.77 | 0.71 | 38.31 | 0.15 | 17.51 | 0.82 | 21.18 | 0.74 |
| RF | 44.79 | 0.57 | 32.42 | 0.39 | 21.11 | 0.74 | 20.81 | 0.75 |
| ANN | 44.23 | 0.58 | 33.46 | 0.35 | 20.35 | 0.76 | 20.34 | 0.76 |

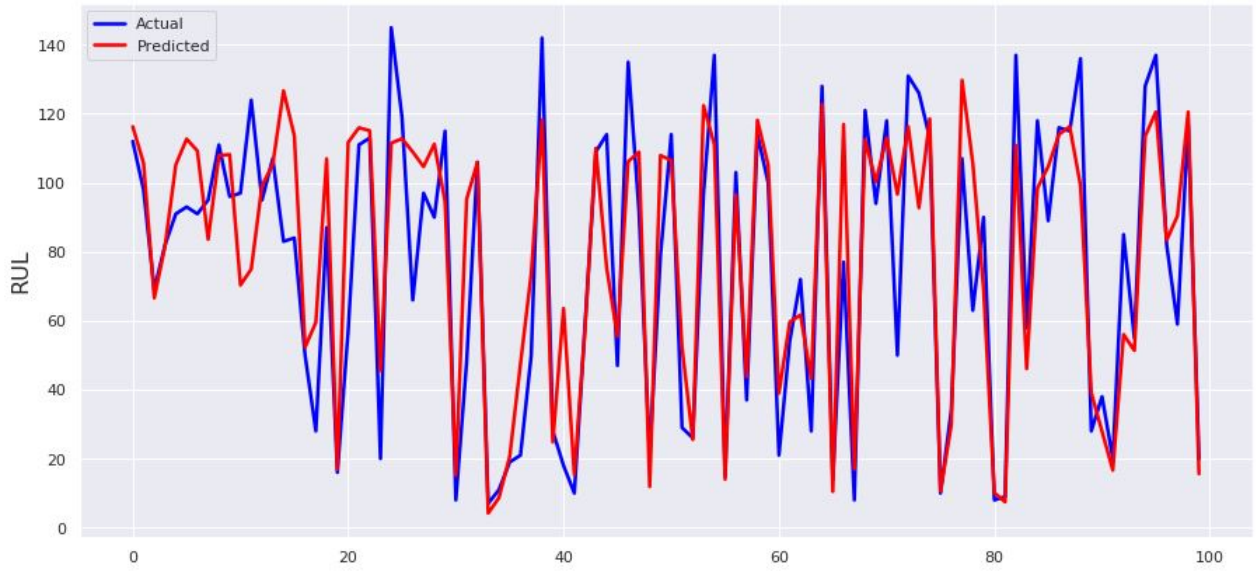



Figure 4.21: Actual RUL vs predicted RUL of the test data for ANN model.

these outcomes, it is evident that the ANN model has the best performance with an R-squared of 0.76. It is closely followed by the random forest and decision tree models with R-squared of 0.75 and 0.74, respectively. Polynomial regression recorded 0.74, SVR 0.72, while linear regression, being the least effective, registered an R-squared of 0.69. Figure 4.21 illustrates a comparison between the actual and predicted RUL values for the ANN model on the test set. Our preference for the ANN model stems from its standout performance, same reason in the previous section for showcasing the predictive outcomes of the SVR model. Within these illustrations, it is apparent that model accuracy in predicting RUL has improved, particularly in the upper RUL range.

### Hyperparameter tuning

The last step for improving the result is applying the hyperparameter tuning. Hyperparameter tuning is the process of systematically searching for the best combination of hyperparameters that will produce the optimal performance of the ANN model. Hyperparameter tuning with grid search is applied in this part to find the best set of hyperparameters for the ANN model. Typical hyperparameters include the batch size, optimizer, number of neurons in each layer, number of epochs and activation functions.

Figure 4.22 shows the definition of the function called `create_model` which creates and returns our ANN model. This allows us to modify hyperparameters within the function.

According to Figure 4.23, we define a dictionary of hyperparameters we would like to search over. The dictionary keys are the parameter names, and the values are lists of settings to try.

Applying grid search, shows that setting with this combination of hyperparameters (batch size = 20, epochs = 50, optimizer = `adam`, number of nodes = 15, activation function = `relu`)

```
# import library
from tensorflow.keras.wrappers.scikit_learn import KerasRegressor
# defining function which returns ANN model with ability to get different hyperparameters
def create_model(optimizer='adam', units=10, activation='relu'):
    ann = tf.keras.models.Sequential()
    ann.add(tf.keras.layers.Dense(units=units, activation=activation))
    ann.add(tf.keras.layers.Dense(units=units, activation=activation))
    ann.add(tf.keras.layers.Dense(units=1))
    ann.compile(optimizer=optimizer, loss='mean_squared_error')
    return ann
model = KerasRegressor(build_fn=create_model)
```

Figure 4.22: Defining function which returns ANN model.

```
# Grid search parameters
param_grid = {
    'batch_size': [10, 20, 32, 40],
    'epochs': [10, 50, 75],
    'optimizer': ['adam', 'rmsprop'],
    'units': [5, 10, 15],
    'activation': ['relu', 'tanh']
}
```

Figure 4.23: Grid search hyperparameters to search through.

improves the R-squared of the ANN model to 0.77.

# Chapter 5

# Conclusions

## 5.1 Summary and conclusions

In the pursuit of enhancing the safety and reliability of industrial systems this thesis explored the domain of Prognostics Health Management (PHM) technology with a keen focus on predicting the Remaining Useful Life (RUL) of aircraft turbofan engines. The core objective of this research was to critically assess the capabilities of data-driven models in predicting the remaining useful life. As industrial landscapes evolve, higher importance is placed on safety, reliability, and predictability. Accurately predicting when a component might degrade or fail has profound implications for operational safety, maintenance costs, and system efficiency.

Key insights from this investigation include:

- **Data-driven over model-based approaches**: Traditionally, predictive models in industries often relied on well-established deterministic models or domain-specific expertise. However, this study highlighted the increasing shift towards data-driven methods, particularly leveraging machine learning models and artificial neural networks. The data-driven approach's main strength lies in its capability to establish complex relationships from historical data without relying on prior domain knowledge.

- **Significance of data preprocessing**: Data, in its raw form, often requires rigorous refinement to be suitable for modeling. The initial dataset, though rich in information, was laden with redundant variables and lacked direct RUL values for each cycle. The preprocessing steps were pivotal in transforming raw data into structured and meaningful inputs for models. This step was not just technical but foundational, ensuring that subsequent models operated on optimized and relevant datasets that led to enhancement of overall model performance.

- **Power of data visualization**: Empirical analysis was complemented by data visualization

to obtain deeper insights from the datasets. Beyond graphical representation, visualization played a pivotal role in understanding data distributions, inherent patterns, and potential anomalies. By thoroughly filtering out less informative sensors, the study ensured that the predictive models interfaced with the most significant and relevant variables.

- **Algorithmic evaluation**: A comparative analysis was conducted across multiple algorithms, ranging from linear and polynomial regression to sophisticated artificial neural networks, to determine the most effective predictive model. While each algorithm had its strengths and challenges, the decision to clip the RUL in train set to address overestimation issues in the upper range, a strategy derived from the data visualization, was instrumental in achieving enhanced prediction accuracies.

- **Superiority of neural networks**: Although, amongst the evaluated models, traditional machine learning models like SVR showcased worthy performance, the artificial neural network emerged as a frontrunner. Its inherent capability to process complex, non-linear relationships positioned it as the most proficient model for this specific prognostic task. Its performance, with an impressive R-squared value of 0.77 after hyperparameter tuning, was indicative of its potency in handling intricate prediction tasks.

- **Essentiality of hyperparameter tuning**: The significance of hyperparameter tuning cannot be understated. It transforms a well-performing model into an outstanding one. Through iterative and time-consuming grid search trials, the study was able to refine the ANN model to its optimal performance.

## 5.2 Recommendation for further work

Future avenues for exploration are, indeed, manifold. Despite the focus on classic machine learning models and a foundational artificial neural network in this thesis, the results, as summarized in Table 5.1, are already promising. It is worth noting that we conducted a comprehensive evaluation of various traditional machine learning models, including linear regression, support vector regression, polynomial regression, decision tree, and random forest, in addition to the ANN. These models were assessed based on the Root Mean Square Error (RMSE) as a performance metric and their results as well as some other recent works are provided in Table 5.1.

These findings, when compared to other recent works in the field, indicate that although our models may not have achieved the lowest RMSE values, they still demonstrate strong predictive capabilities. In particular, the data preprocessing techniques employed in this thesis like visualizing the data and feature extraction using heatmap, as well as the utilization of RUL clipping in the dataset, have contributed to the effectiveness of our classic machine learning models. While

Table 5.1: RUL prediction performance comparision with other recent works.

| Approach | RSME |
|---|---|
| Linear regression | 22.91 |
| Support vector regression (SVR) | 21.82 |
| Polynomial regression | 21.26 |
| Decision tree | 21.18 |
| Random forest | 20.81 |
| KNN (Zhang et al. (2017)) | 20.46 |
| ANN | 20.14 |
| CNN (Babu et al. (2016)) | 18.45 |
| CNN + RNN (Zhang et al. (2019)) | 16.89 |
| CNN + LSTM (Kong et al. (2019)) | 16.13 |
| LSTM (Li et al. (2018)) | 13.52 |

there are models in recent research that outperform our results, these findings underscore the significance of our methodology, especially in contexts where the implementation of advanced neural architectures might not be feasible due to resource constraints or other practical considerations.

Recent advancements in the field, particularly in neural network architectures, offer promising directions for continued research. For instance, delving into more advanced neural architectures such as Recurrent Neural Networks (RNN), Long Short-Term Memory (LSTM), and Convolutional Neural Networks (CNN) could substantially enhance the RUL prediction capabilities as illustrated in Table 5.1, especially given their demonstrated proficiency in handling sequential and complex data structures. Moreover, the emerging trend of Transfer Learning (TL) has drawn significant attention due to its powerful capacity to leverage pre-trained models on sizable and intricate datasets.

Furthermore, integrating domain-specific expertise with these advanced data-driven techniques may offer another promising avenue. This hybrid approach could converge the strengths of traditional domain knowledge with current machine learning paradigms, thereby combining the benefits of both methodologies.

In conclusion, the capability to precisely predict the remaining useful life of machinery represents a profound intersection of domain expertise, classic modeling, and state-of-the-art machine learning methodologies. While our current findings are promising, they also highlight the expansive horizon of opportunities for further refinement and innovation. The landscape of prognostics health management is dynamically evolving, and its future holds immense poten-

tial for enhanced predictive accuracy, operational safety, and system efficiency.

# Appendix A

# Acronyms

**AI** Artificial intelligence

**ANN** Artificial neural network

**API** Application programming interface

**CART** Classification and regression tree

**CM** Condition monitoring

**CMAPSS** Commercial modular aero-propulsion system simulation

**CNN** Convolutional neural network

**DGI** Decrease of gini impurity

**DL** Deep learning

**ICA** Independent component analysis

**KNN** k-nearest neighbors

**LSTM** Long short-term memory

**MAE** Mean absolute error

**ML** Machine learning

**MSE** Mean squared error

**NASA** National aeronautics and space administration

**NLP** Natural language processing

**NN** Neural network

**OLS** Ordinary least square

**PCA** Principal component analysis

**PHM** Prognostic health management

**ReLU** Rectified linear unit

**RF** Reliability function

**RL** Random forest

**RNN** Recurrent neural network

**RSME** Root mean squared error

**RSS** Residual sum of squares

**RUL** Remaining useful life

**SVM** Support vector machine

**SVR** Support vector regression

**TL** Transfer learning

**TSS** Total sum of squares

# Appendix B

# Code Templates

This chapter presents a collection of code templates used throughout this master's thesis. These code examples have played a role in carrying out experiments and analyses. While some of the code templates were inspired by existing works and the wider research community, I would also like to acknowledge the contribution made by Eremenko (2023)'s "Machine Learning A-Z™: AI, Python & R [2023]" course on Udemy. This course has provided me with knowledge and practical skills enabling me to create many of these code templates.

The purpose of including these code templates in this chapter is to enhance transparency regarding this research methodology and provide a resource for researchers and practitioners in the field.

```python
# importing libraries
import pandas as pd
import numpy as np

# define column names for easy indexing
index_names = ['unit_number', 'cycles']
setting_names = ['setting_1', 'setting_2', 'setting_3']
sensor_names = ['s_{}'.format(i) for i in range(1,22)]
col_names = index_names + setting_names + sensor_names

# reading data
train = pd.read_csv('train_FD001.txt',sep='\s+', header=None, names=col_names)
x_test = pd.read_csv('test_FD001.txt',sep='\s+', header=None, names=col_names)
y_test = pd.read_csv('RUL_FD001.txt', sep='\s+', header=None, names=['RUL'])

# removing all extra rows and keep only last cycl of each engine
x_test = x_test.groupby('unit_number').last().reset_index()
x_test.shape

# removing 'setting_3' from train set
train=train.drop('setting_3',axis=1)
```

Figure B.1: Importing data set and defining label for columns.

```python
# defining function to ad RUL column in train set
def add_remaining_useful_life(df):
    # Get the total number of cycles for each unit
    grouped_by_unit = df.groupby(by="unit_number")
    max_cycle = grouped_by_unit["cycles"].max()

    # Merge the max cycle back into the original frame
    result_frame = df.merge(max_cycle.to_frame(name='max_cycle'), left_on='unit_number', right_index=True)

    # Calculate remaining useful life for each row
    remaining_useful_life = result_frame["max_cycle"] - result_frame["cycles"]
    result_frame["RUL"] = remaining_useful_life

    # drop max_cycle as it is no longer needed
    result_frame = result_frame.drop("max_cycle", axis=1)
    return result_frame
train = add_remaining_useful_life(train)
```

Figure B.2: Defining a function to add RUL in train set.

```python
# ploting sensor data
import matplotlib.pyplot as plt
def plot_sensor(sensor_name, ax):
    for i in train['unit_number'].unique():
        if (i % 10 == 0):  # only plot every 10th unit_number
            ax.plot('RUL', sensor_name,
                    data=train[train['unit_number'] == i])
    ax.set_xlim(250, 0)  # reverse the x-axis so RUL counts down to zero
    ax.set_xticks(np.arange(0, 275, 25))
    ax.set_ylabel(sensor_name)
    ax.set_xlabel('Remaining Useful Life')
fig, axes = plt.subplots(7, 3, figsize=(15, 17.5))  # 7 rows, 3 columns
for sensor_name, ax in zip(sensor_names, axes.ravel()):
    plot_sensor(sensor_name, ax)
plt.tight_layout()
plt.show()
```

Figure B.3: PLoting the sensors' data.

```python
# visualizing heatmap to find correlations
import seaborn as sns; sns.set()
plt.figure(figsize=(25,18))
sns.heatmap(train.corr(),annot=True ,cmap='Reds')
plt.show()

# Selecting highly correlated features
correlation=train.corr()
train_relevant_features = correlation[abs(correlation['RUL'])>=0.5]

# creating a list of only important features.
list_relevant_features=train_relevant_features.index
list_relevant_features=list_relevant_features[1:]

# Now we will keep onlt these imprtant features in train set.
train=train[list_relevant_features]

# splitting train set into x_train and y_train(RUL).
y_train=train['RUL']
x_train=train.drop(['RUL'],axis=1)

# keeping only train columns/features in the test set as well.
x_test=x_test[x_train.columns]
```

Figure B.4: Feature selection and splitting the dataset.

```
# create an evaluate function to calculate R-squared and RMSE
from sklearn.metrics import r2_score, mean_squared_error
def evaluate(y_true, y_hat, label='test'):
    mse = mean_squared_error(y_true, y_hat)
    rmse = np.sqrt(mse)
    variance = r2_score(y_true, y_hat)
    print('{} set RMSE:{}, R2:{}'.format(label, rmse, variance))
    return rmse,variance;

# Standardization of features
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train_trans = sc.fit_transform(x_train)
x_test_trans = sc.transform(x_test)

# clipping the RUL(y_train) at 125
y_train= y_train.clip(upper=125)
```

Figure B.5: Creating evaluation function, data standardization and clipping RUL.

```
# Linear regression model
from sklearn.linear_model import LinearRegression
lm = LinearRegression()
lm.fit(x_train_trans, y_train)
# predict and evaluate
y_hat_train = lm.predict(x_train_trans)
RMSE_Train,R2_Train=evaluate(y_train, y_hat_train,'train')
y_hat_test = lm.predict(x_test_trans)
RMSE_Test,R2_Test=evaluate(y_test, y_hat_test,'test')
```

Figure B.6: Linear regression model.

```
# Polynomial regression
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
poly_reg = PolynomialFeatures(degree = 4)
x_poly = poly_reg.fit_transform(x_train_trans)
lin_reg_2 = LinearRegression()
lin_reg_2.fit(x_poly, y_train)
# predict and evaluate
y_hat_train = lin_reg_2.predict(x_poly)
RMSE_Train,R2_Train=evaluate(y_train, y_hat_train,'train')
y_hat_test = lin_reg_2.predict(poly_reg.fit_transform(x_test_trans))
RMSE_Test,R2_Test=evaluate(y_test, y_hat_test,'test')
```

Figure B.7: Polynomial regression model.

```
# SVR
from sklearn.svm import SVR
regressor = SVR(kernel = 'rbf')
regressor.fit(x_train_trans, y_train)
# predict and evaluate
y_hat_train = regressor.predict(x_train_trans)
RMSE_Train,R2_Train=evaluate(y_train, y_hat_train)
y_hat_test = regressor.predict(x_test_trans)
RMSE_Test,R2_Test=evaluate(y_test, y_hat_test)
```

Figure B.8: Support vector regression model.

```
# Decision Tree
from sklearn.tree import DecisionTreeRegressor
dt = DecisionTreeRegressor(random_state=42, max_depth=15, min_samples_leaf=10)
dt.fit(x_train_trans, y_train)
# predict and evaluate
y_hat_train = dt.predict(x_train_trans)
RMSE_Train,R2_Train=evaluate(y_train, y_hat_train, 'train')
y_hat_test = dt.predict(x_test_trans)
RMSE_Test,R2_Test=evaluate(y_test, y_hat_test)
```

Figure B.9: Decision tree regression model.

```
# Random Forest
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(random_state=42, n_jobs=-1, max_depth=6, min_samples_leaf=5)
rf.fit(x_train_trans, y_train)
# predict and evaluate
y_hat_train = rf.predict(x_train_trans)
RMSE_Train,R2_Train=evaluate(y_train, y_hat_train, 'train')
y_hat_test = rf.predict(x_test_trans)
RMSE_Test,R2_Test=evaluate(y_test, y_hat_test)
```

Figure B.10: Random forest regression model.

```
# ANN
import tensorflow as tf
ann = tf.keras.models.Sequential()
ann.add(tf.keras.layers.Dense(units=10, activation='relu'))
ann.add(tf.keras.layers.Dense(units=10, activation='relu'))
ann.add(tf.keras.layers.Dense(units=1))
ann.compile(optimizer='adam', loss='mean_squared_error')
ann.fit(x_train_trans,y_train, batch_size=32,epochs=75)
# predict and evaluate
y_hat_train = ann.predict(x_train_trans)
RMSE_Train,R2_Train=evaluate(y_train, y_hat_train, 'train')
y_hat_test = ann.predict(x_test_trans)
RMSE_Test,R2_Test=evaluate(y_test, y_hat_test,'test')
```

Figure B.11: Artificial neural network model.

```
# Plot Actual Vs Predicted RUL for Test Data
fig = plt.figure();
plt.figure(figsize=[15,7])
plt.plot(y_test,color="blue", linewidth=2.5, linestyle="-",label="Actual")
plt.plot(y_hat_test,color="red",  linewidth=2.5, linestyle="-",label="Predicted")
fig.suptitle('Actual and Predicted', fontsize=20)           # Plot heading
plt.xlabel('Index', fontsize=18)                            # X-label
plt.ylabel('RUL', fontsize=16)                               # Y-label
plt.legend()
plt.title("Actual RUL Vs Predicted RUL for Test Data")
```

Figure B.12: Plotting the prediction results vs actual values.

```
# import library
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.wrappers.scikit_learn import KerasRegressor
from sklearn.model_selection import GridSearchCV

# defining function which returns ANN model with ability to get different hyperparameters
def create_model(optimizer='adam', units=10, activation='relu'):
    ann = tf.keras.models.Sequential()
    ann.add(tf.keras.layers.Dense(units=units, activation=activation))
    ann.add(tf.keras.layers.Dense(units=units, activation=activation))
    ann.add(tf.keras.layers.Dense(units=1))
    ann.compile(optimizer=optimizer, loss='mean_squared_error')
    return ann
model = KerasRegressor(build_fn=create_model)

# Grid search parameters
param_grid = {
    'batch_size': [10, 20, 32, 40],
    'epochs': [10, 50, 75],
    'optimizer': ['adam', 'rmsprop'],
    'units': [5, 10, 15],
    'activation': ['relu', 'tanh']
}

# running grid search and setting cross validation on 3
grid = GridSearchCV(estimator=model, param_grid=param_grid, scoring='neg_mean_squared_error', cv=3)
grid_result = grid.fit(x_train_trans, y_train)
```

Figure B.13: Applying grid search and hyperparameter tuning.

# Bibliography

Zhang, J., Li, X., Tian, J., Luo, H., and Yin, S. (2023). An integrated multi-head dual sparse self-attention network for remaining useful life prediction. Reliability Engineering and System Safety, 233, 109096–109096. https://doi.org/10.1016/j.ress.2023.109096

Zhang, J., Jiang, Y., Wu, S., Li, X., Luo, H., and Yin, S. (2022). Prediction of remaining useful life based on bidirectional gated recurrent unit with temporal self-attention mechanism. Reliability Engineering and System Safety, 221, 108297. https://doi.org/10.1016/j.ress.2021.108297

Saxena, A., Goebel, K., Simon, D., and Eklund, N. (2008). Damage propagation modeling for aircraft engine run-to-failure simulation. 2008 International Conference on Prognostics and Health Management. https://doi.org/10.1109/phm.2008.4711414

Sikorska, J. Z., Hodkiewicz, M., and Ma, L. (2011). Prognostic modelling options for remaining useful life estimation by industry. Mechanical Systems and Signal Processing, 25(5), 1803–1836. https://doi.org/10.1016/j.ymssp.2010.11.018

Tobon-Mejia, D., Medjaher, K., and Zerhouni, N. (2010). The ISO 13381-1 standard's failure prognostics process through an example. 2010 Prognostics and System Health Management Conference, 1-12.

Engel, S. J., Gilmartin, B. J., Bongort, K., and Hess, A. (n.d.). Prognostics, the real issues involved with predicting life remaining. 2000 IEEE Aerospace Conference. Proceedings (Cat. No.00TH8484). https://doi.org/10.1109/aero.2000.877920

Hess, A. P., Calvello, G., and Frith, P. (2005). Challenges, issues, and lessons learned chasing the "Big P". Real predictive prognostics. Part 1. https://doi.org/10.1109/aero.2005.1559666

Luo, J., M. Namburu, Pattipati, K. R., Qiao, L., Kawamoto, M., and Shunsuke Chigusa. (2003). Model-based prognostic techniques [maintenance applications]. AUTOTESTCON. https://doi.org/10.1109/autest.2003.1243596

Berghout, T., and Benbouzid, M. (2022). A Systematic Guide for Predicting Remaining Useful Life with Machine Learning. Electronics, 11(7), 1125. https://doi.org/10.3390/electronics11071125

Calabrese, F., Regattieri, A., Botti, L., and Galizia, F. G. (2019). Prognostic Health Management of Production Systems. New Proposed Approach and Experimental Evidences. Procedia Manufacturing, 39, 260–269. https://doi.org/10.1016/j.promfg.2020.01.333

Shinde, P. P., and Shah, S. (2018, August 1). A Review of Machine Learning and Deep Learning Applications. IEEE Xplore. https://doi.org/10.1109/ICCUBEA.2018.8697857

Sharifani, K., and Amini, M. (2023). Machine Learning and Deep Learning: A Review of Methods and Applications. Social Science Research Network. https://ssrn.com/abstract=4458723

Lærum, K.H. (2018). A study of Machine Learning for Predictive Maintenance - A topic and programming guidance.

Copeland, M. (2016, July 29). The Difference between AI, Machine Learning, and Deep Learning? NVIDIA Blog. The Official NVIDIA Blog. https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/

Kumar, Y., Kaur, K., and Singh, G. (2020, January 1). Machine Learning Aspects and its Applications Towards Different Research Areas. IEEE Xplore. https://doi.org/10.1109/ICCAKM46823.2020.9051502

Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer.

Janiesch, C., Zschech, P., and Heinrich, K. (2021). Machine learning and deep learning. Electronic Markets, 31, 685–695. Springer. https://doi.org/10.1007/s12525-021-00475-2

Ray, S. (2019, February 1). A Quick Review of Machine Learning Algorithms. IEEE Xplore. https://doi.org/10.1109/COMITCon.2019.8862451

Eremenko, K. (2023). Machine Learning A-Z™: AI, Python and R + ChatGPT Bonus. Udemy: https://www.udemy.com/

Mali, K. (2021, October 4). Linear Regression | Everything you need to Know about Linear Regression. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/10/everything-you-need-to-know-about-linear-regression/

Alzubi, J., Nayyar, A., and Kumar, A. (2018). Machine Learning from Theory to Algorithms: An Overview. Journal of Physics: Conference Series, 1142, 012012. https://doi.org/10.1088/1742-6596/1142/1/012012

Kumar, A. (2022, February 27). Ordinary Least Squares Method: Concepts and Examples. Data Analytics. https://vitalflux.com/ordinary-least-squares-method-concepts-examples

Awad, M., and Khanna, R. (2015). Support Vector Regression. Efficient Learning Machines, 67–80. https://doi.org/10.1007/978-1-4302-5990-9-4

Boulesteix, A.-L., Janitza, S., Kruppa, J., and König, I. R. (2012). Overview of random forest methodology and practical guidance with emphasis on computational biology and bioinformatics. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 2(6), 493–507. https://doi.org/10.1002/widm.1072

Deepanshi. (2021, May 25). Artificial Neural Network | Beginners Guide to ANN. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/05/beginners-guide-to-artificial-neural-network/

Zakaria, M., Mabrouka, A. S., and Sarhan, S. (2014). Artificial neural network: a brief overview. neural networks, 1, 2.

Kalita, D. (2022, March 30). An Overview and Applications of Artificial Neural Networks. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2022/03/an-overview-and-applications-of-artificial-neural-networks-ann/

Rosa, Daniel, Nuno, Ricardo, and Nuno. (2019). Using Artificial Neural Networks for Analog Integrated Circuit Design Automation. Springer Nature.

4 Reasons Why is Python Used for Machine Learning | Inoxoft. (2022, April 1). Inoxoft.com. https://inoxoft.com/blog/why-use-python-for-machine-learning/

Esfahani, Z., Karim Salahshoor, Behnam Farsi, and Eicker, U. (2021). A New Hybrid Model for RUL Prediction through Machine Learning. Journal of Failure Analysis and Prevention, 21(5), 1596–1604. https://doi.org/10.1007/s11668-021-01205-8

Prognostics Center of Excellence Data Set Repository - NASA. (n.d.). Retrieved October 25, 2023, from https://www.nasa.gov/intelligent-systems-division/discovery-and-systems-health/pcoe/pcoe-data-set-repository/

Moamar Sayed-Mouchaweh, and Springerlink (Online Service. (2020). Artificial Intelligence Techniques for a Scalable Energy Transition : Advanced Methods, Digital Technologies, Decision Support Tools, and Applications. Springer International Publishing, Imprint Springer.

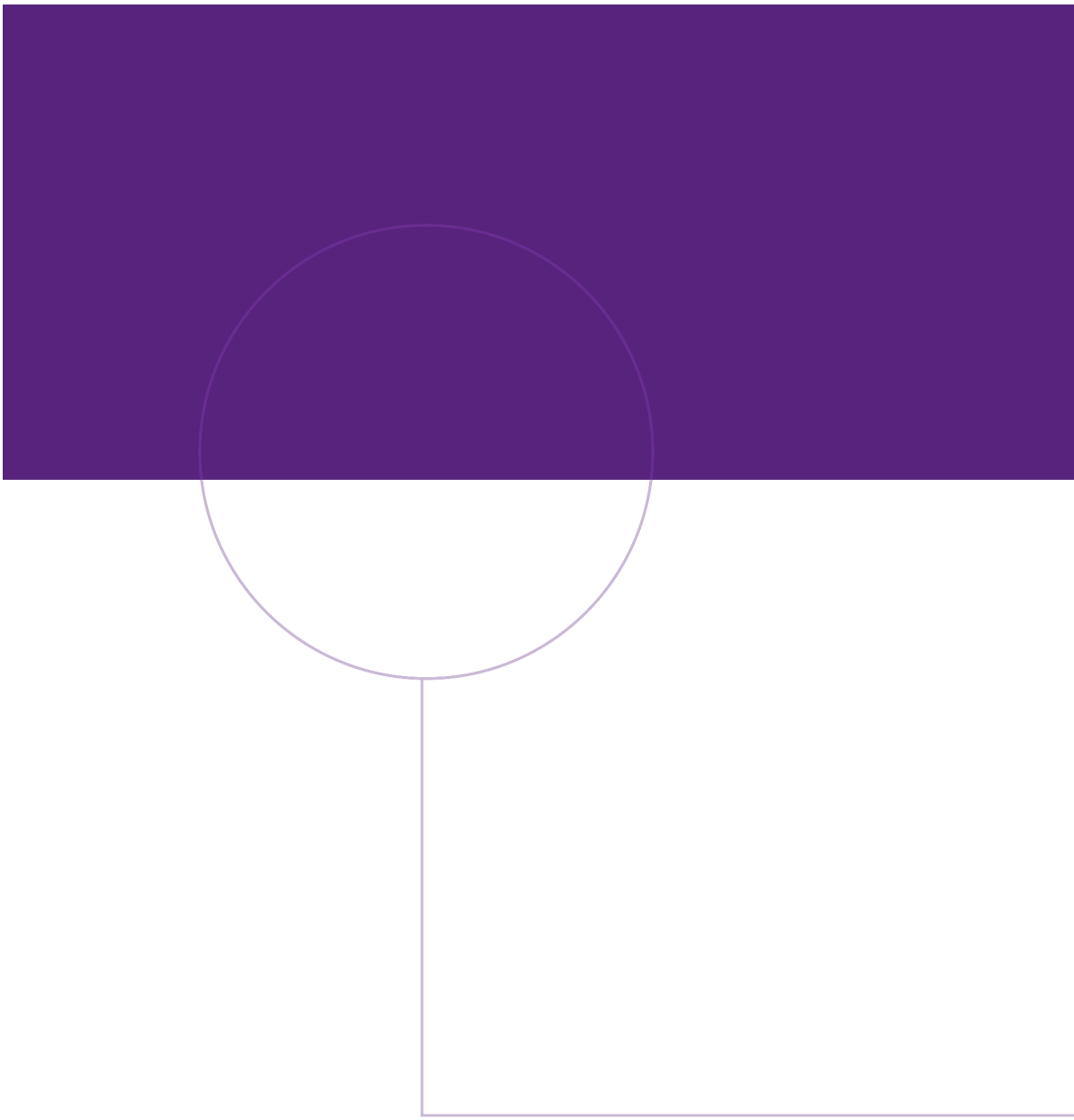
Zhang, C., Lim, P., Qin, A. K., and Tan, K. C. (2017). Multiobjective Deep Belief Networks Ensemble for Remaining Useful Life Estimation in Prognostics. IEEE Transactions on Neural Networks and Learning Systems, 28(10), 2306–2318. https://doi.org/10.1109/tnnls.2016.2582798

Babu, G.S., Zhao, P., and Li, X. (2016). Deep Convolutional Neural Network Based Regression Approach for Estimation of Remaining Useful Life. International Conference on Database Systems for Advanced Applications.

Zhang, X., Dong, Y., Wen, L., Fang, L., and Li, W. (2019). Remaining Useful Life Estimation Based on a New Convolutional and Recurrent Neural Network. https://doi.org/10.1109/coase.2019.8843078

Kong, Z., Cui, Y., Xia, Z., and Lv, H. (2019). Convolution and Long Short-Term Memory Hybrid Deep Neural Networks for Remaining Useful Life Prognostics. Applied Sciences, 9(19), 4156. https://doi.org/10.3390/app9194156

Li, X., Ding, Q., and Sun, J.-Q. (2018). Remaining useful life estimation in prognostics using deep convolution neural networks. Reliability Engineering and System Safety, 172, 1–11. https://doi.org/10.1016/j.ress.2017.11.021



NTNU

Norwegian University of
Science and Technology