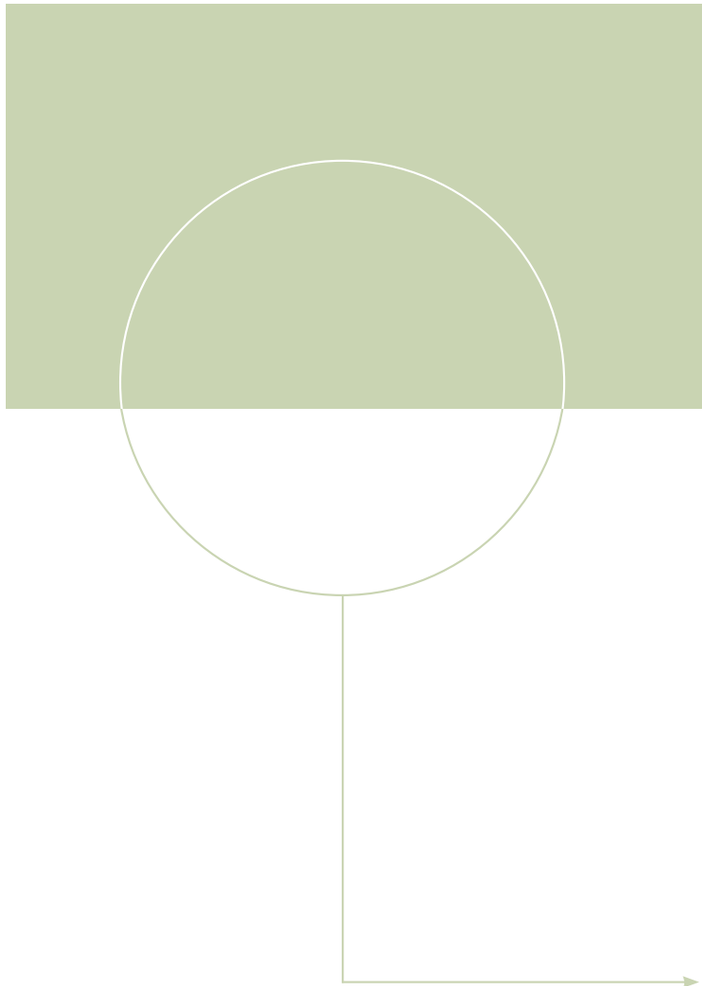


Master's thesis

2019

Simone Heggelund

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

Simone Heggelund

Unsupervised Anomaly Detection based on Machine Learning

An Application for a Centrifugal Air Compressor System

June 2019

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

Unsupervised Anomaly Detection based on Machine Learning

An Application for a Centrifugal Air Compressor System

Simone Heggelund

Reliability, Availability, Maintainability and Safety (RAMS)
Submission date: June 2019
Supervisor: Jørn Vatn
Co-supervisor: Torleif Berg

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

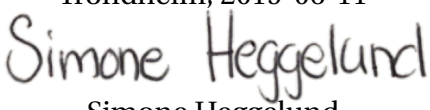
Preface

This master thesis is conducted within the specialization in Reliability, Availability, Maintainability and Safety (RAMS), at the department of Mechanical and Industrial Engineering (MTP). The thesis is a part of the five-years program in Mechanical Engineering at NTNU, and is written during the spring semester 2019.

The project is carried out in cooperation with Norsk Hydro ASA and supervisor Jørn Vatn from the RAMS department at NTNU. Based on the project thesis written during the fall 2018, where a comprehensive literature review on Remaining Useful Life was carried out, the topic for this master thesis was founded. This thesis aims to bridge the gap between the current situation in the industry today in relation to data availability, and research promoted within the fields of RAMS, Health Management and Artificial Intelligence.

This report assumes that the reader has some background knowledge within Health Management, Lifetime Analysis and Maintenance Optimization, preferably fifth-years RAMS-students who have taken the courses; TPK4140 - Maintenance Optimization and Safety, TPK4120 - Industrial Safety and Reliability and TPK4450 - Data Driven Prognostics and Predictive Maintenance. In addition, the reader is assumed to have basic programming knowledge.

Trondheim, 2019-06-11



Simone Heggelund

Acknowledgment

I would like to express my gratitude to those who have contributed to the work related to this master thesis.

First of all, I would like to thank Norsk Hydro ASA, represented by Arnt Johnsen and Torleif Berg, for providing valuable discussions and insight on Health Management, Prognostics and Anomaly Detection from an industry point of view. This has contributed to highlight the challenges faced by the industry today, and the need for sufficient modelling tools. In addition, all the relevant technical documentation and sensor data for the Compressor System are provided by Norsk Hydro ASA, which has enabled to carry out analysis on a real-world case.

Furthermore, I would like to give a sincere thank to my supervisor Jørn Vatn, for weekly discussions and follow up meetings during the whole period. The prior knowledge of Jørn Vatn has contributed to understanding the topic at stake, and provided helpful guidance related to state-of-the-art literature.

These key contributors have motivated me to continue working within the fields of Health Management, Prognostics and Anomaly Detection in the future, and provided me valuable insight that has contributed strongly to this thesis.

S.H.

Executive Summary

In the context of Industry 4.0, new emerging technologies have enabled a shift within the manufacturing sector, where data extracted from all relevant sources is the key driver to create value. Deriving value from these streams of continuous time series data is the main focus of this master thesis, with the objective of building an *unsupervised anomaly detector* based on machine learning. The system under study is a three-stage Centrifugal Air Compressor system, equipped with in total 21 sensors monitoring the system continuously. The current status in the industry today in relation to data availability is examined, concluding that failure history and data associated with labels indicating the health of the equipment is hard to obtain for complex systems. Hence, an unsupervised anomaly detector is required, not relying on labeling or historical failures. Based on machine learning fundamentals, state-of-the-art machine learning models and several anomaly detection surveys, it is concluded that the following approaches will solve the objective; *unsupervised anomaly detection based on residuals* and *unsupervised anomaly detection based on clustering*. A framework for how these approaches can be implemented is presented, both building on the principle of modelling the normal behavior of the system, and flagging samples deviating from this behavior as anomalous.

For the residuals based approach, three state-of-the-art machine learning models are reviewed and implemented, namely the *Decision Tree Model*, the *Random Forest Model* and a *Feedforward Neural Network*. These models are aiming to predict the target variable, which in this case is the pressure, based on a learned relationship with the input features. The magnitude of the residuals between the predicted and actual target variable determines if a sample is classified as normal or abnormal, depending on a chosen confidence interval. For the clustering based approach, the *K-Means clustering algorithm* is reviewed and implemented. This model is aiming to group the data into clusters with similar patterns, forming a reference pattern for the normal behavior of the system. Any new sample falling outside this reference pattern is classified as anomalous, depending on a predefined outlier fraction. Along with this reviews, the basic framework associated with machine learning is presented, involving training, validating and testing the algorithms. The objective of building models that perform well on *never-before-seen* data is emphasized, mitigating the effects of overfitting and underfitting.

For the residuals based approach, a benchmark is performed between the *Decision Tree Model*, the *Random Forest Model* and the *Feedforward Neural Network*. It is concluded that the *Random Forest Model* has to overall best performance both on the validation and testing set, predicting the pressure with an accuracy of 0,98. The results gained from this model is used to calculate the residuals, giving a result of 180 detected abnormal samples out of 7233 in total. For

the clustering based anomaly detector, the same result were obtained using an outlier fraction of 0,02. By comparison of the presented detectors, it becomes evident that approximately the same samples are classified as anomalous, which are big irregular patterns deviating strongly from what is assumed to be normal operating behavior. Due to the lack of failure history, it is concluded that the determination of the decision boundaries for abnormal classification requires expert judgment and system knowledge, such that the risk of false positives and false negatives are minimized. This thesis aims to pride the gap between theory and practice, introducing an interdisciplinary collaboration between research promoted within the fields of RAMS and Health Management and IT and Artificial Intelligence. By such interdisciplinary collaboration, it is believed that the value derived from the continues arriving streams of data can be maximized.

Sammendrag

Satsningen på digitalisering og implementering av konsepter som inngår i Industri 4.0 har aldri vært større, hvor tilstandsovervåkning, implementering av sensorteknologi og prediktive strategier for vedlikehold står i fokus. Denne oppgaven tar sikte på å utvikle en unormalitetsdetektor basert på maskin læring, gjennom analyse av kontinuerlige tidsseriedata. Systemet som studeres er en tre-trinns sentrifugalkompressor for kompresjon av luft, utstyrt med totalt 21 sensorer som overvåker systemet kontinuerlig. Kritisk produksjonsutstyr monitoreres kontinuerlig med intensjon om å detektere unormaliteter før systemsvikt oppstår, noe som gjør at tilgangen på data som beskriver feilhistorikk og degraderingsmekanismer er svært begrenset. For implementering av prediktivt vedlikehold kreves dermed en unormaldetektor som ikke baserer seg på feilhistorikk og indikatorer som sier noe om utstyrets degraderingsnivå, såkalt uassistert unormalitetsdeteksjon. Med utgangspunkt i fundamentale maskin lærings konsepter, en rekke maskinlæringsmodeller og litteratur som dekker ulike tilnærminger til uassistert unormalitetsdeteksjon, presenteres et rammeverk for implementering av følgende tilnærminger; *unormalitetsdeteksjon basert på residualer* og *unormalitetsdeteksjon basert på gruppering*. Begge de presenterte tilnærmingene baserer seg på å modellere systemets normale atferd ved hjelp av maskin læring, for så å detektere atferd som avviker fra denne som unormal.

For den residualbaserte tilnærmingen gjennomgås og implementeres tre state-of-the-art maskinlæringsmodeller, *Decision Tree Model*, *Random Forest Model* og *Feedforward Neural Networks*. Disse modellene tar sikte på å predikere lufttrykket basert på et lært forhold med de andre systemvariablene. Størrelsen på residualene mellom det predikerte og faktiske trykket avgjør om atferden klassifiserer som normal eller unormal, med utgangspunkt i et forhåndsbestemt konfidensintervall. For den klyngebaserte tilnærmingen gjennomgås og implementeres *K-means clustering algoritmen*, som tar sikte på å gruppere data i klynger med lignende atferd. De grupperte klyngene danner et referansemønster for systemets normale oppførsel, hvor data som ikke overlapper med referansemønsteret klassifiseres som unormal, avhengig av en forhåndsbestemt fraksjonsprosent. I tillegg presenteres et grunnleggende rammeverk knyttet til maskinlæring, hvor trening, validering og testing av maskinlæringsmodeller gjennomgås, samt grunnleggende konsepter knyttet til overtilpasning, undertilpasning og optimalisering av modellenes hyperparametre.

Videre gjennomføres en sammenligning av den den oppnådde prestasjonen til både *Decision Tree Model*, *Random Forest Model* og *Feedforward Neural Networks*. Det konkluderes med at *Random Forest Model* presterer best, med evne til å predikere trykket med en nøyaktighet på 0,98. Resultatene som er oppnådd fra denne modellen brukes videre til å beregne residualene,

og for en testperiode på 6 dager detekteres totalt 180 unormaliteter, ut av totalt 7233 mulige. For den klyngebaserte unormalitetsdetektoren oppnås samme resultat med en fraksjonsprosent på 0,02. Ved sammenligning av de to presenterte detektorene konkluderes det med at tilnærmet de samme partiene klassifiseres som unormale, som i hovedsak er store uregelmessige partier som avviker sterkt fra det som antas å være normal driftsadferd. For å optimalisere beslutningsreglene for klassifisering av unormaliteter kreves det domenekunnskap og kunnskap om mulige feilmoder og unormaliteter, da svikthistorikk ikke er tilgjengelig. Beslutningsgrensene må optimaliseres med sikte på å redusere tilfeller av falske positive og falske negative deteksjoner, som både reduserer reliabiliteten til detektoren og øker risikoen for systemsvikt. Denne oppgaven bidrar med en tverrfaglig tilnærming til analyse av kontinuerlig tidsseriedata, hvor tradisjonelle fremgangsmåter fra et RAMS perseptiv kombineres med ny forskning og litteratur fra et IT og Kunstig Intelligens perspektiv.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, the background for the problem at stake is presented, with a further description of the problem formulation and the objectives to be solved. In addition, the scope and limitations associated with the project are presented, and the approaches used to solve the objectives. Finally, a structural overview of the report is given.

1.1 Background

Manufacturing companies are consistently facing high levels of margin pressure, and are aiming to reduce costs by pushing for remaining untapped potential that has not yet been optimized (Wee et al., 2015). During the last decade, new emerging technological solutions have enabled rabid changes in the industry and in the society, which can be regarded as a paradigm shift. This paradigm shift is forecasted to become the fourth industrial revolution, commonly referred to as *Industry 4.0*. The term Industry 4.0 first appeared in an economic policy for Germany in 2011 (Vasja et al., 2016), and has ever since been used to describe the rabid changes new emerging technologies have enabled. Industry 4.0 extends the digitization and automation seen in the third industrial revolution in the late 1900s (Prisecaru, 2017), by further digitization of the manufacturing sector (Wee et al., 2015).

The vision behind Industry 4.0 is to integrate data extracting tools in every subsystem, from start to end in the supply chain, including all external and internal operations (Blanchet et al., 2014). In order for the data extracted from the manufacture and value chain to provide valuable information, it must be aggregated to a higher value context, such that all processes can be simulated and analyzed in real-time (Wee et al., 2015). Industry 4.0 focuses on digital op-

timization instead of physical optimization, with an end-to-end information flow through the entire life cycle. By implementing the concepts of Industry 4.0, the products will have a digital representation in the *cyber physical world*, connected through the *Internet of Things* (IoT) (Wee et al., 2015). Dr. Jan Stefan Michels, Head of Standardization and Technology Development at Weidmüller states that; "Industry 4.0 requires the convergence of business IT and manufacturing IT systems. Applications and different engineering disciplines have to grow together and collaborate in an interdisciplinary way, in order to create additional value through better usage of data." (Wee et al., 2015).

Machinery and assets accumulate significant costs for manufacturing companies, and improved asset utilization drives significant value in terms of reduced downtime costs and higher asset performance while operating. Newly emerging technologies, data availability and the consistently requirements of improving overall equipment effectiveness, have also shifted the focus within maintenance strategies. The industry aims at reducing and replacing corrective maintenance actions with preventive and predictive maintenance strategies, in order to reduce planned and unplanned downtime and increase equipment performance (Wee et al., 2015).

Extracting relevant data from all products and manufacturing equipment through sensors and monitoring technologies is the core driver for enabling real-time decision support and predictive maintenance actions. Collected data does not have an inherent value itself, and should arguably be collected with the intention of maximizing value. In a comprehensive survey carried out by McKinsey and Company covering industrial readiness for Industry 4.0, it is stated that; "Even though we have all the enablers to make Industry 4.0 feasible such as connectivity technology, affordable IoT hardware, standardized communication protocol, collecting meaningful data and analyzing for implications are still the biggest challenges to driving the impact from Industry 4.0. " (Wee et al., 2015).

From a RAMS perspective, Prognostics and Health management (PHM) is widely discussed in relation to predictive maintenance. Here, the main focus is on the phases involved with diagnosing the current system health, predicting future behavior of the system and estimating the Remaining Useful Life (RUL)(Vachtsevanos et al., 2006). Predicting the the RUL is promoted as the optimal prognosis output, required for sufficiently implementing predictive maintenance strategies. Hence, research on approaches to best estimate the RUL has gained a lot of attention. Several researchers have attempted to classify and summarize the existing prognosis techniques used for RUL-estimation, among them Gao et al. (Gao et al., 2015), Sikorska et al. (Sikorska et al., 2011) and Peng et al. (Peng et al., 2010). Gao et al. (Gao et al., 2015) classifies the prognosis techniques into three main categories; Physics-based, Data-Driven and Model Based approaches. Sikorsha et al. (Sikorska et al., 2011) proposed a classification that distin-

guish between Knowledge-based models, Life expectancy models, Artificial Neural Networks and Physical models, whereas Peng et al. (Peng et al., 2010) proposed a classification between Physical models, Knowledge-based Models, Statistical models and Combination models. These classifications are somewhat similar, in the sens that all three classifications distinguish between Physical models, Statistical models and Data-Driven models, such as machine learning. Despite the available models, there is a huge gap between the theory and practice. In order to predict the RUL, independent of the chosen model, historical data containing failures with associated labels indicating the health associated with the data is required. For complex and critical systems, this information is rarely available, argued by the fact that these systems not are *run-until-failure*. In order to derive value from these streams, modeling in an *unsupervised fashion* is required, referring to models that do not rely on labeling or historical examples illustrating the *run-until-failure* behavior of the system.

Furthermore, as the amount of data grows big, referred to as *Big Data*, traditional modeling tools will quickly be inefficient, which rises the need for an interdisciplinary collaboration between IT and manufacturing disciplines. Machine learning has quickly grown to become the most popular sub-field of Artificial Intelligence, since it first started flourishing in the 1990s (Chollet, 2017) (Goodfellow, 2017). This trend is mainly driven by two factors; data availability and faster and more robust hardware. There exists tight connections between classical mathematical statistics and machine learning, yet there are some fundamental differences. Classical statistical analyses and physical modeling tend to be impractical for large and complex data sets, whereas machine learning models tend to increase their performance as the amount of data grows. Furthermore, machine learning models are more engineering oriented than mathematical models, such that relatively little mathematical theory are available, commonly referred to as the concept of a "black box". Hence, within machine learning, ideas are often proven empirically rather than theoretically, through a trial-and-error procedure (Chollet, 2017).

From a Health Management and RAMS perspective, the system architecture and system components are traditionally analyzed with the intention of assessing potential failure modes and models appropriate for predicting future degradation and the RUL. From an IT and Artificial Intelligence perspective, the data steaming from the system is the main focus. From this perspective, it is believed that the results gained through data analytics will speak for itself, reducing the need to truly understand and assess all potential failure modes. Within both fields, the objective is to assist real-time decision support by giving early warnings if abnormal behavior is observed. Based on the current status in the industry today in relation to data availability and promoted research from both the presented fields, an *unsupervised anomaly detector* is in reach, whereas predicting the RUL for complex systems still is in the fields of research (Lavin

and Ahmad, 2015).

Problem Formulation

The problem to address in this master thesis is how valuable insight can be gained from streaming, continuous time series data, with the objective of detecting anomalies at an early stage. The system under study is a three-stage Centrifugal Air Compressor system located at Hydro Sunndalsøra, which is continuously monitored by in total 21 sensors. The compressor system is considered to be highly critical for the production of Aluminum at Sunndalsøra, argued by the fact that compressed air is used to transport the raw material Alumina. Along with energy, Alumina is the key ingredient used used to produce Aluminum. Hence, detecting abnormal behavior at an early stage is of high interest for Hydro, such that preventive maintenance actions can be taken before failures progress until system shutdown.

The continuously arriving stream of data is not associated with a label indicating the health of the equipment and the data does not include *run-until-failure* measurements. The compressor system is continuously monitored with the intention of preventing failures and system downtime, such that historical data containing representative examples of how degradation progress until failure not is in the near future for this system. Arguably, the problem at hand raise the need for a model capable of detecting abnormal behavior at an early stage, without relying on historical failures, commonly denoted an *unsupervised anomaly detector*.

1.2 Objectives

The main objective of this master thesis is to build an unsupervised anomaly detector based on machine learning, capable of giving early warnings for real-time decision support. In order to achieve this objective, the three-stage Centrifugal Air Compressor system is studied in detail, along with the fundamental work flow and concepts of machine learning and unsupervised anomaly detection on continuous time series data. To reach the main objective, the following sub-objectives are established;

1. Present the architecture and key components of the three-stage Centrifugal Air Compressor system, along with the associated instrumentation and control system. In addition, review the problems that can be detected through the instrumentation system.
2. Present the key concepts of Prognostics and Health Management for RUL-estimation,

along with challenges faced by the industry today in relation to data availability.

3. Perform a literature review on approaches for anomaly detection on continuously time series data, emphasizing the approaches for unsupervised anomaly detection.
4. Perform an in-depth literature review on the fundamental framework of machine learning, along with a review of state-of-the-art machine learning models.
5. Establish a work-flow-procedure for building unsupervised anomaly detectors based machine learning.
6. Build and implement the unsupervised anomaly detectors on the three-stage Centrifugal Air Compressor time series data.

1.3 Approach

The objectives of this master thesis can be broadly classified into two main parts, a literature review part and a practical implementation part. The literature review covers health management and prognostics, anomaly detection approaches, machine learning fundamentals and state-of-the art machine learning models, while the practical implementation part involves performing descriptive statistics, training, validation and testing the machine learning models and building the unsupervised anomaly detectors.

The research platforms used to assess the relevant literature were *ORIA*, *ScienceDirect*, *Engineering Village*, *IEEE transactions* and *Google Scholar*. In order to assess the most relevant literature, the publishing date, the number of citations and publishing place were used as guidelines. In addition, state-of-the-art articles related to unsupervised anomaly detection and machine learning were assessed based on expert knowledge from the supervisor in the RAMS-department, and IT-students more familiar with machine learning and anomaly detection. In order to get a deeper understanding of each machine learning model, research articles presenting both general theory, the mathematical framework associated with the model and a numerical case study demonstrating the obtained results were weighted highly. In addition, the mathematical framework and the key properties related to each model were assess through state-of-the-art books, covering machine learning fundamentals and unsupervised anomaly detection.

The practical implementation is based on relevant literature covering the fundamental work flow of machine learning and unsupervised anomaly detection, focusing on how to build and

implement these models. In addition, open source tutorials demonstrating how to train, validate and test the machine learning models have been studied. Python is used for programming, along with the open source libraries *Pandas*, used for preprocessing the data, *Seaborn*, used to perform descriptive statistics and *Scikit-Learn*, used to train, validate and test the machine learning models. These open source libraries include all the relevant documentation required for implementation, which has been heavily used for successfully developing the anomaly detectors.

1.4 Contributions

The main contribution of this master thesis is a comprehensive review on machine learning fundamentals and unsupervised anomaly detection, along with the development of two successful anomaly detectors, from the perspective of a RAMS student. Traditionally, assessing all potential failure modes and models appropriate for predicting future degradation and the RUL has been the main focus within RAMS. From an IT and Artificial Intelligence perspective, the data steaming from the system is the main focus, with the belief that the results gained through data analytics will speak for itself. This master thesis contributes with a combined perspective, considering both traditional research promoted within the fields of Health Management and Prognostics from a RAMS, with emerging trends and research within IT and Artificial Intelligence, such as machine learning. It is of strong beliefs that such interdisciplinary collaboration will bridge the gap between theory and practice, and within research fields.

In addition, this thesis provides insight on how value can be created from continuous time series data, which not includes labels or historical failures. The current status in the industry today in relation to data availability has been highly weighted, with the objective of creating models capable of detecting anomalies with the current available data. Hence, this thesis provide a framework, grounded in research withing health management, machine learning and anomaly detection, for building efficient unsupervised anomaly detectors, capable of detection anomalies at an early stage. The presented modelling framework for machine learning and anomaly detection is believed to cover all the relevant theory and implementation procedures needed for a RAMS student to implement similar models, with the only requirement of basic programming skills.

1.5 Limitations

Maintenance strategies and maintenance optimization options are not discussed in detail. Optimization criteria such as costs, safety and availability, and existing maintenance strategies are considered to be known for the reader, and is excluded from the main scope of this thesis.

Based on the fact that this thesis considers the whole three-stage Centrifugal Air Compressor System, with all associated sensors, a FMECA covering all failure modes has not been carried out in detail. Instead, detectable problems that possibly can be reveled by deviations in air pressure, air temperature and equipment vibration are discussed.

The data provided by Norsk Hydro ASA only contains measurements from December 2018 to January 2019. Due to this, possible seasonal, cyclic and trend variations are not included in the data set. Ideally, the machine learning models should have been trained on an interval for at least one year, to capture the dynamics over a longer time period. However, the provided data is sufficient for demonstrating how unsupervised anomaly detectors can be built.

The anomalies detected by the presented models have not been proven to be real-life anomalies. This is due to the fact that the data does not include any known historical failures or labeling classifying the data as normal or abnormal. Therefore, irregular patterns observed through visualization of the data are assumed to be anomalies, such that the anomaly detectors are optimized to classify these patterns as anomalous.

As a RAMS student, it is not guaranteed that the code associated with the machine learning models and anomaly detectors are optimized to the full extend. However, effort has been made to optimize the code as much as possible, to reduce unnecessary operations and long running times.

1.6 Outline

The report is structured as follows:

- Chapter 1: Presents the background for this topic, the main objectives to be solved, and the approaches and limitations related to the work.
- Chapter 2: Presents the architecture and main components of the three-stage Centrifugal Air Compressor system, along with the instrumentation installed on the system. In addition, detectable problems related to pressure, temperature and vibration are highlighted.

- Chapter 3: Introduces the general framework within Health Management and Prognostics, and reviews anomaly detection approaches for continuous time series data.
- Chapter 4: Presents the fundamental framework of machine learning, along with a review of state-of-the-art machine learning models.
- Chapter 5: Presents the established work-flow-procedure on how to build unsupervised anomaly detectors based on machine learning.
- Chapter 6: Presents the data preprocessing and descriptive statistics performed on the data.
- Chapter 7: Presents the trained, validated and tested machine learning models, along with a performance benchmark between the models.
- Chapter 8: Presents the implementation and optimization of the unsupervised anomaly detectors.
- Chapter 9: Presents the summary, conclusion and discussion for the master thesis, along with recommendations for future work.
- Appendix A: Presents acronyms relevant for this thesis.
- Appendix B: Presents the programming codes for the work carried out.
- Bibliography

Chapter 2

The Three-Stage Centrifugal Air Compressor System

2.1 System Location

The system under study is a three-stage Centrifugal Air Compressor system, running in parallel with four other compressors. The five compressors are located at Hydro Sunndalsøra, and are responsible of producing compressed air that is used for transportation of *alumina*. Each year, Hydro Sunndalsøra produces four hundred thousand tons of melted aluminum, which further is cast and used in rolled or extruded products. Aluminum production starts with extraction of the raw material bauxite, which contains approximately fifteen percent alumina. The bauxite is then processed in a refinery, where alumina is extracted. Along with energy, alumina is the key ingredient in the electrolysis process, used to produce aluminum, as seen in Figure 2.1. Hence, in order for primary melted aluminum to be produced, alumina needs to be transported to the electrolysis facility. Alumina is transported in pipelines with compressed air, produced by the five compressors running in parallel. The electrolysis cells producing aluminum are continuously running, such that if the supply of alumina is prevented, the production will be heavily affected.

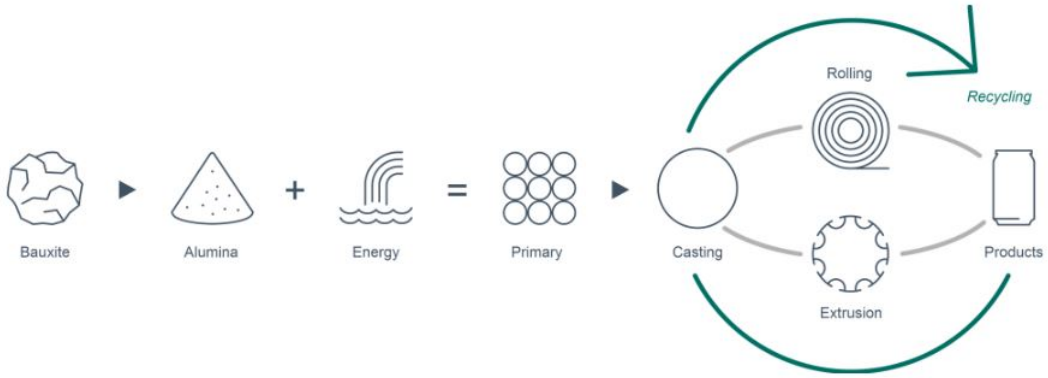


Figure 2.1: Illustration of the aluminum life cycle, where alumina is a key ingredient.

2.2 System Description

In the following, the architecture and the key components of the three-stage Centrifugal Air Compressor system is presented. As seen from the system drawing in Figure 2.2, the air is led in through a *inlet filter*, before entering the *inlet guide vane valve*. The *inlet guide vane valve* leads the air flow into the *first stage compression*, before entering the *second stage* and *third stage compression*. During three stages, the air is compromised from approximately 3,5 bar to 7,2 bar. A *heat exchanger* for cooling and an *air-water separator* are located between each compression. An *electrical motor* drives the *gear*, which drives the *impellers* inside the compressors to rotate. Furthermore, oil is used for lubrication, both in the motor system and in the gear, while water is used as cooling medium in the cooling system. In the following, the function of these sub-systems and components are explained.

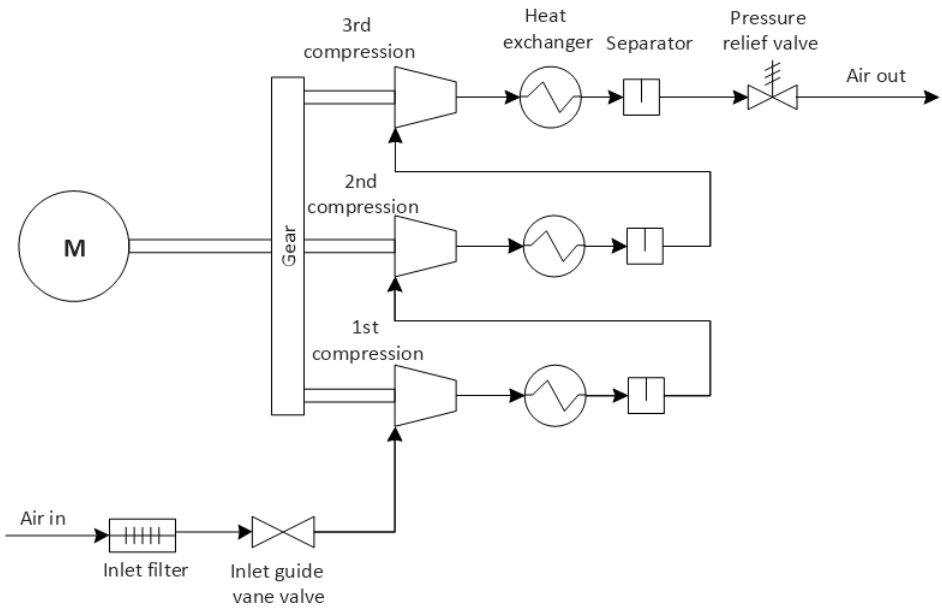


Figure 2.2: The three-stage Centrifugal Air Compressor system architecture.

2.2.1 Electrical Motor

The *electrical motor* converts electrical energy into mechanical energy, through the interaction of a magnetic field and electric current in a wire. The moving part in an electrical motor is the rotor, which is turning the shaft that delivers mechanical power. Bearings are supporting the rotor, allowing the rotor to rotate around its own axis. Furthermore, the stator refers to the stationary part of the electromagnetic circuit associated with the motor (Gottlieb, 1997).

2.2.2 Gear / Mechanical Drive

The *Gears*, also known as *Mechanical Drives*, transfer power from a prime mover to an actuator based on an rotary motion. In this case, an actuator refers to an operational machine or operational member, such as the impeller located inside the compressor. The gears or mechanical drives are located between the prime mover and the actuator, and are connecting these through couplings or clutches. A gearbox contains several gears, each having the function of transmitting and receiving motion from the connected gear-teeth. When gears are meshing with one another, torque moment is transmitted. In addition, gears have the ability to change both the speed, direction and torque generated from the power source (Jelaska, 2012).

2.2.3 Inlet Filter

The *inlet filter* is responsible of filtering out particles, such as dust and other chemicals, from the air before entering the system. Dust and chemicals entering the compressor system may damage the other components, which is why the inlet filter is located at the very beginning of the inlet pipeline (Giampaolo, 2010). In this case, the inlet filter is a three-stage filter, serving different purposes. The first stage is a particle filter, the second stage is a chemical filter responsible for separating out sulfur, while the third stage is a particle filter for even smaller particles.

2.2.4 Inlet Guide Vane Valve and Inlet Butterfly Valve

Inlet guide vane valve and *Inlet butterfly valve* are the two most common valves used to control the inlet flow of a centrifugal air compressor, usually mounted at the inlet before the first stage compression. Both valves are actuated by the compressor control system based on system demand, yet with different operating principles. The butterfly valve directs the airflow straight into the impeller, in an axial direction, whereas the inlet guide vane valve has adjustable vanes that can be positioned such that the inlet air is caused to swirl in the same direction as the rotating impeller. When the inlet air begins rotating before entering the impeller, the first stage impeller is required to do less work, such that the overall efficiency of the compressor is improved. For inlet butterfly valves, the drives are responsible of providing all the energy required to rotate the air with the impellers, without the help of a swirl generated by the valve. It is estimated that inlet guide vane valves can reduce the energy consumption by 9 percent. However, when the compressor is working at 100 percent load, the inlet guide vane valve and the butterfly valve operate based on the same principle, without any swirling of the air. Hence, the inlet guide vane valve only differs from the butterfly valve when the compressor is running in the throttle range of the compressor, meaning that the compressor not is running at full capacity (Joseph, 2001). On the compressor system at hand, an inlet guide vane valve is operating in parallel with an inlet butterfly valve, where the inlet butterfly valve is used as a backup valve.

2.2.5 Compressor Unit

Compressors types can be classified into two distinct categories, namely *dynamic* and *positive displacement* compressors. For dynamic compressors, one distinguish between *axial* and *centrifugal compressors*. Centrifugal compressors achieves compression through rotating *impellers*, which apply inertial forces to the air. Each compression stage consist of a rotating impeller and a

stationary diffuser. There exists three types of impeller design, open, semi-enclosed or enclosed design. Similar for all three is that the impeller consists of radial vanes, which are attached to a backing plate or disc. As the air enters the impeller, most often perpendicular to the axis, a centrifugal force is applied through the rotating impeller. Before exiting the impeller, the air enters a diffuser, also known as a flow decelerator. The flow deceleration generated by the diffuser causes pressure to built up, such that compressed air exits the centrifugal air compressor. The only element adding energy the the air is the impeller, whereas all the other components are stationary (Giampaolo, 2010).

2.2.6 Heat Exchanger and Separator

The *heat exchangers* serve the purpose of cooling the compressed air, in order to prevent equipment from being damaged. As the air pressure increases during the compression, the temperature increases accordingly, described mathematically by for example the *Ideal Gas Equation*, $pV = nRT$ (Giampaolo, 2010). High-temperature air has a detrimental effect on the lubrication and sealing of the equipment material, requiring a heat exchanger to be located as close as possible to the air compression exists. In addition, the high-temperature air exiting the compressor contains high levels of moisture vapor, which eventually will condense as the air is cooled naturally throughout the system. If no actions are taken, the condense will cause rust, scale build-up and possible frost issues during winter operation. Hence, the heat exchangers serve two functions; reduce the air temperature to prevent damage caused by high temperatures itself, and speed up the condense process such that water can by separated from the air using a *separator*. The heat exchangers, also known as after coolers, can either be based on water cooling or air cooling. In both cases, the air is cooled through heat transferring between the compressed air and the cold cooling medium, which in this case is water. For a three-stage compressor, a heat exchanger and a separator are located between each compression stage, to prevent high-temperature air and water developed through condense damaging the equipment of the preceding compression stages (Morel, 2017).

2.3 System Demand and Operating Principals

The five compressors running in parallel are responsible for the supply of compressed air at approximately 7,2 bar, with a total nominal performance equaling 1476 m3/min. Taking into account reduced efficiency caused by operational wear and other external and internal factors, it is assumed that the maximum performance equals 1400 m3/min, given that all compressors

are operating. However, the air demand varies as the production in electrolysis cells varies, from a demand at approximately 420 m3/min (requiring only 2 compressors to run), to a demand at approximately 1200-1400 m3/min (requiring 5 compressors to run). Hence, not all the compressors are operated simultaneously to all times.

2.3.1 Control System

The compressors are controlled by an automatic control system regulating the operation based on a common *load distribution principle*. This means that several compressors are cooperating on the supply of compressed air, such that the total capacity in the inlet valve is utilized and the usage of *blow off valves* are reduced. When running on the common load distribution principle, the inlet valve is throttled, such that the amount of air entering the compressor is reduces, which reduces the motor power consumption, as explained in section 2.2.4. In cases where a further pressure reduction is required, the blow off valve is activated. The blow of valve drops the air out at the roof, which is a direct loss. Hence, it is of interest that the compressors are run as much as possible based on common load regulation and not based on *local pressure regulation*, due to economical losses. In order for the compressors to be operating based on the load distribution principle, at least three of the centrifugal air compressors need to be operating, while two can be in standby mode. In cases where the demand only requires two compressors to be operating, the load distribution control will be switch off, and a local pressure control will be turned on. The common load distribution principle cause the compressors to be automatically loaded, started, and unloaded, switch off, based on the required demand from the production facilities. During loading and unloading, there will be irregular patterns in the time series data, deviating from stable operating behavior. In addition, there will evidently be periods where each of the compressors are in standby mode, implying that the measurement output values, which are explained in more detail in section 2.4, equal zero.

2.4 System Instrumentation

Each of the three-stage Centrifugal Air Compressors are equipped with in total 21 sensors, continuously monitoring the system. The mounted instrumentation automatically sends signals to the control system, which based on the input signals operates the compressors based on the common load regulation principle, as described in section 2.3.1. In addition, data extracted from the instrumentation system is the most powerful tool for detection of abnormal behavior and faults (Giampaolo, 2010). The mounted instrumentation is presented in Table 2.1, along

with the function and measurement unit associated with each sensor. The instrumentation is mounted as close to the source as possible, aiming to capture the true dynamics of the system, as illustrated in the system drawing with instrumentation in Figure 2.3.

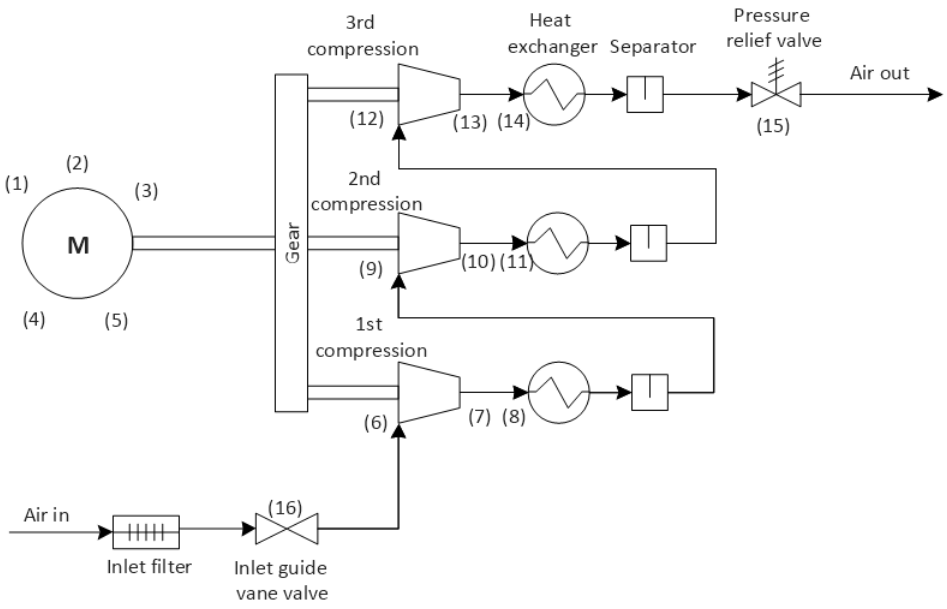


Figure 2.3: The three-stage Centrifugal Air Compressor system architecture with instrumentation.

Table 2.1: Instrumentation mounted on the compressor system

Number	Sensor name	Unit	Description
(1)	Motor-TemperaturePhase1	C	Measures the motor temperature in phase 1.
(2)	Motor-TemperaturePhase2	C	Measures the motor temperature in phase 2.
(3)	Motor-TemperaturePhase3	C	Measures the motor temperature in phase 3.
(4)	Bearing-TemperatureDE	C	Measures the bearing temperature on the drive-end of the motor.
(5)	Bearing-TemperatureNDE	C	Measures the bearing temperature on the non-drive-end of the motor.
(6)	Stage1-Vibration	μm	Measures the vibration on the rotating impeller in stage 1.
(7)	Stage1-AirTemperature	C	Measures the air temperature after stage 1 compression and cooling.
(8)	Stage1-AirPressure	kPa	Measures the air pressure after stage 1 compression.
(9)	Stage2-Vibration	μm	Measures the vibration on the rotating impeller in stage 2.
(10)	Stage2-AirTemperature	C	Measures the air temperature after stage 2 compression and cooling.
(11)	Stage2-AirPressure	kPa	Measures the air pressure after stage 2 compression.
(12)	Stage3-Vibration	μm	Measures the vibration on the rotating impeller in stage 3.
(13)	Stage3-AirTemperature	C	Measures the air temperature after stage 3 compression and cooling.
(14)	Stage3-AirPressure	kPa	Measures the air pressure after stage 3 compression.
(15)	BlowOffValve-Position	%	Measures the position, opened or closed, of the pressure relief valve.
(16)	InletValve-Position	%	Measures the position, openness and angle, of the inlet guide vane valve.
(-)	Oil-Pressure	kPa	Measures the oil pressure in the system.
(-)	VentFanTemperature	C	Measures the temperature in the ventilation fan.
(-)	Motor-Current	A	Measures the motor current in the electrical motor driving the rotating impellers.
(-)	Capacity	m3/min	Measures the overall capacity of the system.
(-)	PowerConsumption	kW	Measures the power consumption used by the electrical motor.

2.5 Detectable Problems

In general, faults or anomalies associated with a complex compressor system driven by an electrical motor can be classified into three categories; *mechanical problems*, *electrical problems* and *performance problems* (Giampaolo, 2010). In the following, the symptoms associated with the problems in each category are presented;

1. **Mechanical problems** are associated with damage on the mechanical equipment, and typical symptoms are vibration, sound or leaks (air, gas, water, oil). The cause may not always be self-evident, while the symptoms of the problem often are.

2. **Electrical problems** are related to damage on the electrical equipment, which often is associated with high voltage to the electrical components, such as motors, motor starter and switcgear. In addition, electrical problems can appear evident through low voltage to the instrumentation and control system. In terms of fault diagnosis, electrical problems are generally harder to diagnose than mechanical problems.

3. **Performance problems** are usually a result of mechanical faults or degradation within the mechanical components in the compressor system, such as problems with the heat exhangers or problems with the rotating impellers. These problems will impact the system performance, and eventually cause system shutdown. Hence, the system performance is a strong indication of the systems health, and can therefore be used as an indirect indicator of equipment condition (Giampaolo, 2010).

The three-stage Centrifugal Air Compressor system is equipped with in total three pressure sensors, three air temperature sensors and three vibration sensors, as presented in Table 2.1. These sensors are highly important for monitoring both mechanical and performance related problems, and are therefore considered the main focus of this thesis. In Table 2.2, possible causes and detection methods for *Low System Air Pressure*, *High Air Temperature* and *High Vibration* are presented. As seen from Table 2.2, the air pressure, air temperature and vibration are in total correlated with nearly all the other sensors mounted on the compressors system, which is revealed trough the possible detection methods for the presented problems.

Table 2.2: Detectable problems associated with a three-stage Centrifugal Air Compressor system (Giampaolo, 2010).

Problem	Possible Causes	Detection
Low System Air Pressure	Dirty inlet filter, incorrect valve calibrations, valve malfunctions, air leakages, degraded or damaged impellers, incorrect control calibration or system demand greater than capacity.	Detected directly by air pressure sensors, detected indirectly through valve positions, high vibration levels, low air temperature or abnormal power consumption.
High Air Temperature	Damaged heat exchangers, inadequate water flow, high water temperature, water flows backwards or plugged water passages.	Detected directly by air temperature sensors, detected indirectly through high vibration and low air pressure, caused by damaged equipment from high air temperatures.
High Vibration	Excessive build-up on impellers, oil not drained properly from bearings, high oil pressure, unbalanced motor rotor, damaged bearings or worn rotor assembly parts or worn couplings.	Detected directly by vibration sensors, detected indirectly through low air pressure and low air temperature, or increased power consumption to maintain system pressure.

2.5.1 Possible Root Causes of Pressure Deviations

The *Stage3-AirPressure* is considered to be the strongest indication of the overall system health, argued by its strong correlation to both temperature, valve positions, vibration levels, inlet filter function and the other pressure sensors. First, the pressure is strongly correlated with temperature, proven by for example considering the *Ideal Gas Equation*, $pV = nRT$, such that deviations in the pressure also imply temperature deviations (Giampaolo, 2010). High temperature air may damage the equipment, and effort should be made to detect failures in the cooling system as soon as possible. Furthermore, low air pressure may be an indication of damaged or degraded impellers, struggling to compress the air to sufficiently high pressure levels. In addition, damaged impellers cause vibration, such that high vibration on any of the three impellers may eventually cause the pressure to drop. Furthermore, dirty inlet filter struggling to filter out dust and particles from the inlet air may damage the impellers, which further leads to reduced air pressure. Malfunctions in the valve positions, both the inlet valve and blow off valve, are strongly

correlated with pressure deviations. For example, if the blow off valve fails to open when necessary, the system pressure, and also the *Stage3-AirPressure* will increase, to levels which can damage the equipment. In addition, the *Stage3-AirPressure* is logically strongly correlated with both the *Stage2-AirPressure* and the *Stage1-AirPressure*, argued by the fact that the air is compressed in three-stages (Giampaolo, 2010).

As seen, the pressure sensors, and most importantly the *Stage3-AirPressure*, are a strong indication of the overall system health, incorporating deviations in many of the key components in the systems. In order to successfully chose an appropriate performance variable describing the system, it is of high importance to be familiar with possible detectable problems and associated diagnosis. In relation to the fields of RAMS and Health Management, being familiar with such root causes is promoted as highly important, enabling efficient diagnosis and scheduling of predictive maintenance (Sikorska et al., 2011).

Chapter 3

Health Management and Anomaly Detection

3.1 Prognostics and Health Management

In order to prevent equipment from failing unexpectedly, prognosis plays an essential role for managing business risks such that future outcomes can be accesses in advance (Sikorska et al., 2011). During the last decade, research on prognostics has been widely promoted by several international societies and conferences, such as Center of Advanced Life cycle Engineering, Phm-society, ESRA and PHM Conference, IEEE Conference on Prognostics and Health Management and Prognostics and System Health Management Conference (Barros, 2018). Within the fields of Prognostics and Health Management, various definitions of prognostics have been proposed. The French normalization ISO 13381 -1 defines prognostics as “Prognostic is the estimation of the life before failure and the estimation of the risk of existence or of the risk of future apparition of one or several failure modes” (Sikorska et al., 2011). Whereas Byington et al. (Byington et al., 2002) states that; “Prognostic is the ability to predict the future condition of a machine based on the current diagnostic state of the machinery and its available operating and failure history data”, and Buruah et al. (Baruah and Chinnam, 2005) argue that: "Prognostics builds upon the diagnostic assessment and are defined as the capability to predict the progression of this fault condition to component failure and estimate the remaining useful life (RUL)". According to these definitions, prognostics is related to both diagnostics of current health states, based upon condition monitoring data collected either online or off-line, and the prediction of future health states and failure modes. In order to provide failure prediction, historical information, age, current health state, future usage and operating environment need to be considered.

Within Prognostics and Health Management, the *RUL* is commonly used as a prediction indicator, denoting the *remaining useful life* of the system at time t, based upon all the available information up to time t (Barros, 2018).

3.1.1 Remaining Useful Life

Estimation of the RUL is essential within Prognostics and Health Management. The RUL is typically a random and unknown variable, implying that it must be estimated from available condition and health monitoring information (Si et al., 2011). Similarly as for prognostics, various definitions of the RUL exists, such as the definitions proposed by Si et al. (Si et al., 2011); "Remaining useful life (RUL) is the useful life left on an asset at a particular time of operation", or "The remaining useful life (RUL) of an asset or system is defined as the length from the current time to the end of the useful life" (Si et al., 2011). A general definition can be formulated by letting the $RUL(t_j)$ denote a random variable that corresponds to the remaining useful life at time t_j, such that;

$$RUL(t_j) = inf\{h : Y(t_j + h) \in S_L | Y(t_j) < L, Y(s)_{0 \leq s \leq t_j}\}, \tag{3.1}$$

where $Y(t_j)$ denotes the condition of the unit at time t_j, which is related to diagnosis. The future health state is denoted by $Y(t_j + h)$, which is the part related to prognosis. Furhteremore, S_L denotes a set of unacceptable states representing failure and L represents a fixed threshold limit defining unit or system failure if exceeded. Hence, the RUL of an asset is a random variable that depends on the current age and condition of the asset, the operational environment and the available condition monitoring (CM) and health information (Si et al., 2011) (Barros, 2018) (Gao et al., 2015).

3.2 Prognostics and Diagnostics

Considering the RUL-estimates as an ideal prognostic output, various steps from diagnostics to prognostics are required. Sikorska et al. (Sikorska et al., 2011) exemplifies this process by investigating the process a component undergoes from a healthy state to final failure, by answering the following questions;

i) Is the component in a degraded state?

ii) Which failure mode has initiated the degradation?

iii) How severe is the degradation?
iv) How quickly is degradation expected to progress from its current state to functional failure?
v) What novel events will change this expected degradation behaviour?
vi) How may other factors, such as the type of model and measurement noise, affect the RUL estimates ? (Sikorska et al., 2011)

The questions concerning component degradation and current health state are related to anomaly detection and diagnostics, while future expected degradation behavior and noise effects that impacts the RUL-estimates are related to prognostics. Furthermore, Sikorska et al. (Sikorska et al., 2011) define the relationship between diagnostics and prognostics as; "diagnostics involves identifying and quantifying the damage that has occurred (and is thus retrospective in nature), while prognostics is concerned with trying to predict the damage that is yet to occur.". This definition implies that prognostics relies upon the diagnostic outputs, such as fault indicators and degradation rates, which is also demonstrated in the general formalism denoting the $RUL(t_j)$ in section 3.1.1 (Sikorska et al., 2011).

The french standard ISO1338-1 suggests a broader approach to describe the steps involved in prognostics (Sikorska et al., 2011). *Data pre-processing* is described as the first step, which includes anomaly detection, diagnostics, failure definitions, identifying potential future failure modes and selecting an appropriate prognostic model. The second step is referred to as *existing failure mode prognosis*, which involves estimation of time to failure (ETTF) of all incipient failures and calculating the system's or component's RUL with the failure mode that has the lowest ETTF. This needs to be done iteratively until the RUL with the desired confidence limit is obtained. This procedure is followed by *future failure mode prognosis*, which involves assessing the most likely future failure modes and assessing the RUL with an appropriate confidence for all future failure modes. Finally, *post-action prognosis*, which refers to the process of identifying potential incidents that could eliminate, retard or halt the progression of critical failure modes and prevent future failure modes. The previous mentioned modelling processes need to be repeated with this information. To assess the impact the predicted failures have on operational and maintenance activities, a *system-oriented* approach to prognostics needs to be assessed. A system oriented approach takes maintenance planning options, logistic concerns, inventory and supply management issues and other non-engineering factors into account. (Sikorska et al., 2011).

Sikorska et al. (Sikorska et al., 2011) have developed a comprehensive flowchart based upon the french standard ISO1338-1, covering the steps involved with RUL prediction. As seen in Figure 3.1, the first steps are related to anomaly detection and diagnostics, while the final steps are related to prognostics of various complexity. Here, the prognostics are categorized by three

levels. **Level 1** refers to *existing failure mode prognostics*, **Level 2** refers to *future failure mode prognostics* and **Level 3** refers to *post-action prognostics*. The different levels indicate the increasing level of modelling complexity which involves in realistic prognosis.

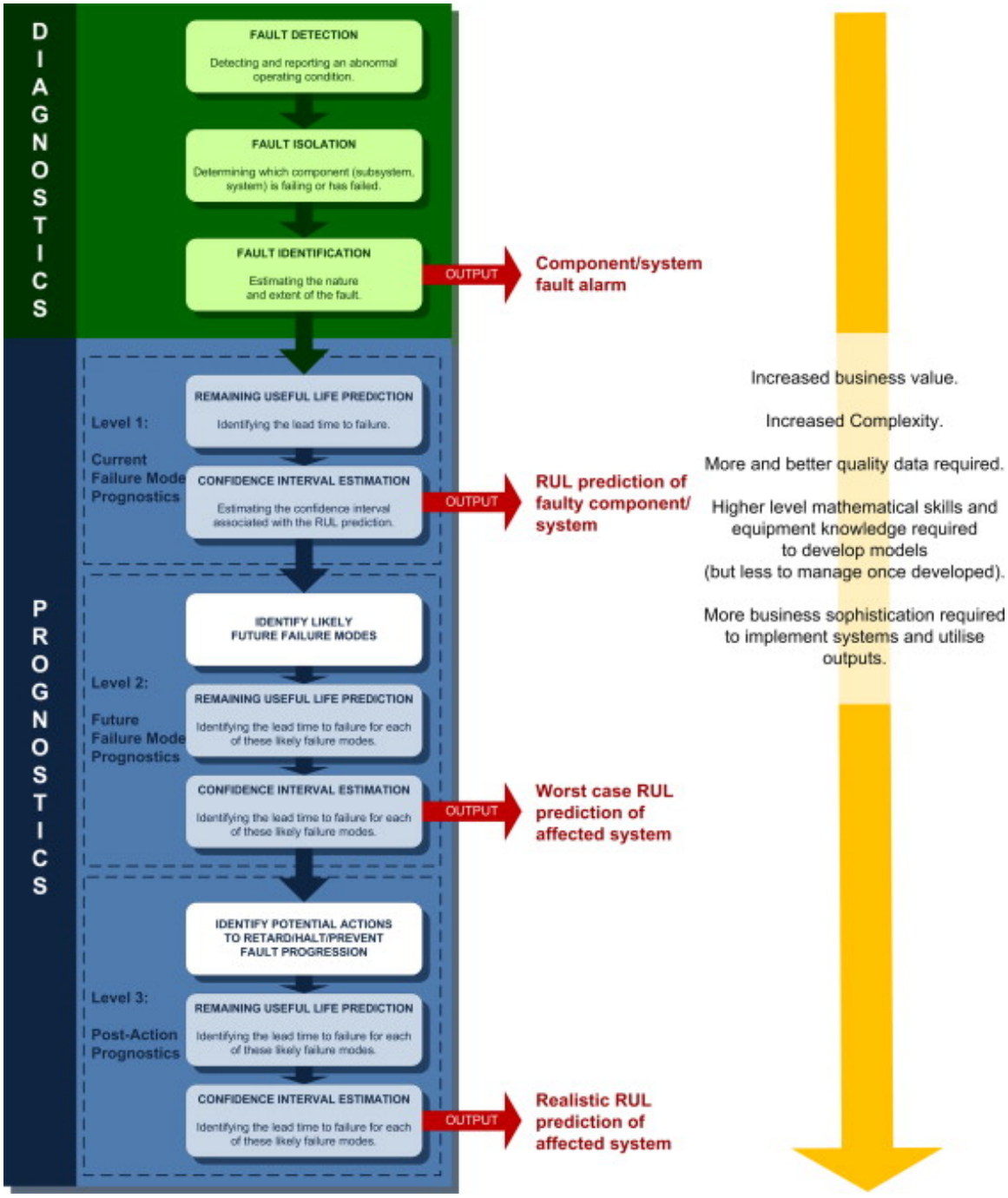


Figure 3.1: Flowchart illustrating the various steps involved in realistic prognostics and RUL-estimation (Sikorska et al., 2011)

3.3 Data Availability - Current Status in the Industry Today

Across every industry, an increase in the amount of streaming, time-series data is observed, produced by connected real-time data sources (Lavin and Ahmad, 2015). Within the fields of *Prognostics and Health Management*, predicting the *residual useful lifetime* is presented as the optimal prognostics output, with the objective of reducing downtime and improving equipment efficiency by scheduling predictive maintenance actions. The current status in the industry today indicates that there is still a huge gap between theory and practice. In order to predict the RUL, the following constraints need to be fulfilled;

1. **Historical data** containing information that is *representative* of both past and potential future system behavior is required, which implies that the data needs to contain relevant examples of *run-until-failure* behavior, for all potential failure modes.

2. **Labeling**: The time series data needs to be associated with a label for each instance of time, indicating the system health. The RUL is predicted based on the current health of the system, and historical examples illustrating how a given health status progress until failure.

Within the industry, critical equipment is continuously monitored with the objective of detecting abnormal behavior before the equipment is damaged, to prevent system downtime. The monitoring is in most cases performed based on economical initiatives and for safety reasons, implying that the systems rarely are *run-until-failure*. Furthermore, real-time sensor data does not contain labels indicating the current health associated with the equipment, logically argued by the fact that this information either needs to be based on *expert judgment* or *historical experience* of similar cases. Hence, the constraints required to predict the RUL for complex systems are rarely fulfilled. However, the continuously monitoring creates huge amount of time series data, which has an inherent value itself, if aggregated to a higher value context. In order to derive value from these streams, modeling in an *unsupervised fashion* is required, referring to models that do not rely on labeling or historical failures. Based on the status in the industry today, an unsupervised, real-time anomaly detector is in reach, whereas predicting the RUL for complex system still is in the fields of research.

3.4 Anomaly Detection

Anomaly detection, also known as *outlier detection*, is used within several fields for detecting critical events, such as within cyber security, credit card fraud, health care and system break-

down (Olson et al., 2018) (Chandola et al., 2009a). According to Chandola et al. (Chandola et al., 2009a), "anomalies are patterns in data that do not conform to a well-defined notion of normal behavior.", while Hawkins (Hawkin, 1980) states that; "an outlier is an observation that deviates so much from other observations as to arouse suspicion that it was generated by a different mechanism.". Hence, *anomalies*, also known as *outliers*, correspond typically to newly observed, unknown and often extreme data (Olson et al., 2018), such that an effective anomaly detector can be built by modeling what is normal, in order to discover what is not (Dunning and Friedman, 2014), (Chandola et al., 2009a). Chandola et al. (Chandola et al., 2009a) carried out a comprehensive survey covering existing approaches to detect anomalies, concluding that the nature of the input data determines the applicability of the anomaly detection technique. The input data associated with the compressor system is *continuous*, *multivariate time series* data without *labeling*, most probably exposed to both *trend*, *seasonal* and *cyclic* variation.

3.5 Time Series Modelling and Forecasting

Time series data is defined as a set of observations x_t, each one being recorded at a specific time t. One distinguish between *discrete time series*, where the observations x_t are made at discrete times in a given interval, and *continuous time series*, denoting observations that are recorded continuously over a given time period (Brockwell and Davis, 2002). An *univariate* time series denotes a time series only consisting of one variable, whereas a time series consisting of several variables is denoted *multivariate* time series (Hyndman and Athanasopoulos, 2018). In addition, the times series may be *stationary* or *non-stationary*. For stationary time series, the statistical properties do not vary with time, such that the mean and variance is constant, whereas for non-stationary time series, which is mostly exhibited in the real world, these statistical properties vary with time. Hyndmand et al. (Brockwell and Davis, 2002) define time series modelling as the procedure of approximating the joint distribution of which the observed set of observations, $[x_t]$, are generated from, which is tightly linked to *time series forecasting*. *Time series forecasting* aims to estimate how the sequence of observations will continue into the future, which in the context of fault detection, anomaly detection and diagnosis, plays an essential role. In order to model and forecast time series, the components associated with time series observations need to be interpreted (Hyndman and Athanasopoulos, 2018) (Chandola et al., 2009b).

3.5.1 Time Series Components

Time series data often includes *trend*, *seasonal* and *cyclic* variations. A *trend* denotes the phenomena of a long-term increase or decrease in the data, which can be caused by external or internal changes, such as changes in operating conditions, environmental changes or equipment wear. *Seasonal variations* occur when the time series is affected by seasonal factors, such as the time of the year or specific months. Seasonal variations are commonly assumed to appear by a fixed and known frequency. *Cyclic variations* denote fluctuations in the time series data through rises and falls that not occur by a fixed frequency. Examples of such cyclic behavior is the business cycle, where economic conditions rise and fall within a cycle of a given time period. In order to properly analyze the time series data, these patterns need to be identified, argued by the fact that the system dynamics is highly affected by these variations. In addition, a model capable of capturing these dynamics are required. Mathematically, these components can be explained by a trend component, T_t, a cyclic component, C_t, a seasonal component, S_t, and a reminder, R_t, denoting the stochastic variations not captured by the other components. These components can be combined by an *additive decomposition* or by an *multiplicative decomposition*. The additive decomposition assumes that the components are independent of one-another, such that $Y_t = T_t + S_t + C_t + R_t$, whereas the multivariate decomposition assumes that the components are dependent, $Y_t = T_t * S_t * C_t * R_t$ (Hyndman and Athanasopoulos, 2018), (Brockwell and Davis, 2002), (Adhikari and Agrawal, 2013).

3.6 Anomaly Detection Approaches

Several researchers have attempted to classify existing approaches for anomaly detection, among them Chandola et al. (Chandola et al., 2009a), Sanz-Bobi (Sanz-Bobi, 2016) and Patcha and Park (Patcha and Park, 2007). Chandola et al. (Chandola et al., 2009a) suggest to distinguish between the following approaches; classification based, nearest neighbor-based, clustering based, statistical based and spectral anomaly detection techniques. Sanz-Bobi (Sanz-Bobi, 2016) suggests to distinguish between physical models, statistical models, neural networks models and clustering based models. Similarly, Patcha and Park (Patcha and Park, 2007) suggest techniques based on statistical approaches, machine learning based approaches and data mining based approach. For the last classification, the machine learning based approach includes regression tasks and the data mining approach includes classification based and clustering based approaches.

These suggested approaches all include a separation between *unsupervised* and *supervised* anomaly detection. The supervised approaches include *classification* based anomaly detection, where

the anomaly detector relies on *labeled data* to classify the data as anomalous or normal. The *unsupervised* approaches include *statistical models*, *physical models* and *machine learning models*, not relying on labels. For the unsupervised approaches, the main idea is to model the normal behavior of the system, and declare any observation in the data that does not belong the the normal region as an anomaly (Chandola et al., 2009a), (Sanz-Bobi, 2016), (Patcha and Park, 2007). Both statistical models, physical models and machine learning models are capable of modeling the normal behavior of the system, yet with different constraints. In the following, these unsupervised approaches to anomaly detection are explained.

3.6.1 Physical Models

Physical models refer to mathematical models that describe the normal operating behavior of the process based on established physical relationships between the variables (Sanz-Bobi, 2016). For a compressor system, the *ideal gas equation*, $pV = nRT$, is one example of such physical model (Giampaolo, 2010). In an abnormal situations, the established relationship between these variables may change or stop yielding, such that the physical model does not longer describe the model behavior. Hence, an anomaly detector can then be build by calculating the *residuals* between the monitored process behavior, and the modeled process behavior expected in a normal situation. If the magnitude of the residuals exceeds a given threshold, there is evidence implying that the established relationship between the variables no longer yields, such that something abnormal has occurred. Physical modelling is also known as modelling based on *the first principal* (Sanz-Bobi, 2016). For complex systems, a physical model capable of capturing the true dynamics of the system is hard to obtain (Gao et al., 2015). In addition, modelling the normal behavior based on the first principal assumes that the input variables are correlated in a specific way, which may not be the case in the real-world.

3.6.2 Statistical Models

Similar as for physical models, a statistical model can be applied to model the normal behavior of the system, for example through linear or non-linear regression model, or by fitting a probability distribution to data associated with normal operating behavior. An anomaly detector can then be created by calculating the residuals between the estimated output from the statistical model and the actual measured output, or by considering the probability distribution to newly available data. For distribution based methods, anomalies are detected when the probability distribution corresponding to normal operating behavior no longer represents the

newly available data (Sanz-Bobi, 2016). The main drawback with this method is that the data characterizing normal behavior is assumed to follow a specific distribution, or the variables are assumed to be correlated in a specific way through a regression model. Similar as for physical models, one may fail to capture the true dynamics of the system. Comprehensive reviews covering anomaly detection based on statistical models are presented by Markou and Singh (Markou and Singh, 2003a), Chandola et al. (Chandola et al., 2009a) and Patcha and Park (Patcha and Park, 2007).

3.6.3 Machine Learning Models

Machine learning can be applied to model the normal behavior of the system, either based on regression or based on clustering. For regression tasks, the relationship between a given *target variable*, such as for example the *Stage3-AirPressure*, and the *input features* is learned and approximated by some function. The established relationship makes it possible to predict the target variable, only by considering newly available samples from the input features. Examples of such models are the *Decision Tree model*, *The Random Forest model* and *Neural Networks*. The anomaly detector can accordingly be established by calculating the residuals between the predicted and actual target variable, using the predicted variable as a reference describing normal behavior. This approach is in the following chapters referred to as *anomaly detection based on residuals*. For clustering based approaches, the main idea is to group the data belonging to normal operating behavior into clusters. Any new sample not belonging to a cluster, or belonging to small clusters, is classified as anomalous, depending on a predefined distance requirement (Sanz-Bobi, 2016). This approach is in the following chapters referred to as *anomaly detection based on clustering*.

Several case studies have been presented on anomaly detection based on clustering, among them Breunig et al. (Breunig et al., 2000), using the *local outlier factor*, Schölkopf et al. (Schölkopf et al., 2000) using *support vector machines* and Bezdek et al. (Bezdek et al., 1984), applying a *K-means* clustering algorithm. These are well-known machine learning clustering algorithms, all grouping the data into distinct patterns, known as clusters. For anomaly detection based on Neural Networks, Ryan et al. (Ryan et al., 1998), Vasconcelos et al. (Vasconcelos et al., 1995), Markou and Singh (Markou and Singh, 2003b) and Augusteijn and Folkert (Augusteijn and Folkert, 2002) illustrate relevant examples. These examples all build on the principal of modeling the relationship between a target variable and the input features for normal operating behavior, and flagging anomalies when these relationships stop yielding. The same anomaly detector procedure can be applied by using other machine learning algorithms capable of predicting the

target variable based on a learned relationship with the input features, such as the *Decision Tree model* and the *Random Forest model* (Chollet, 2017).

3.7 Requirements for Anomaly Detection in Streaming Data

The *Numenta Anomaly Benchmark* is the first temporal benchmark designed for anomaly detection in streaming time series data. The benchmark has been design such that open source algorithms for anomaly detection can be compared and ranked according to an established scoring system. Numenta was founded in 2005, and excels on modeling and predicting patterns in continuously streamed data. In the past, there existed no such common scoring mechanism that properly evaluated algorithms for anomaly detection on time series data (Lavin and Ahmad, 2015). In order to develop the scoring system, the ideal characteristics of an anomaly detection algorithm for time series data have been identified, concluding the following;

1. Anomaly detection must be made online and in real time, implying that the algorithm must fit the data in real time and not based on historical data only.
2. Due to seasonal and cyclic variations, possible drifts, noise and other factors causing the normal behavior of the system to change, the algorithm must have the ability to continuously learn and adapt itself to newly available data.
3. The algorithm needs to continuously learn without needing to store the entire stream, argued by the fact that this would require an infinitely large storage capacity.
4. The anomaly detection model can not rely no labeled or reconstructed data, such that the algorithm needs to run in an unsupervised fashion.
5. Anomalies should be detected as early as possible.
6. The number of false positives and false negatives should be minimized (Lavin and Ahmad, 2015).

Numenta argues that algorithms capable of giving early detections of anomalies and continuous learning new and normal patterns should be given extra credit, such that anomalies are detected in real-time. Within traditional methods, such as for physical and statistical models, these aspects have not been the main focus (Lavin and Ahmad, 2015), and arguably, machine learning based methods for anomaly detection are suggested for the problem at hand.

Chapter 4

Machine Learning Framework and Models

In order to build anomaly detectors based on machine learning, the basic concepts of machine learning are introduced. This includes the difference between machine learning categories, approaches for training, validating and testing the algorithms, handling the challenge of overfitting and underfitting and the performance matrices needed to properly evaluate the model performance. In addition, four machine learning models are presented, namely the *Decision Tree model*, the *Random Forest model*, *Feedforward Neural Networks* and the *K-means clustering algorithm*. These models are chosen based on their capability to perform well on a wide range of problems, proven through various machine learning competitions and highlighted in state-of-the-art machine learning literature (Chollet, 2017) (Goodfellow, 2017) (Ketkar, 2017). Moreover, these four models will later on be used to build the anomaly detector based on residuals and the anomaly detector based on clustering. Three state-of-the-art machine learning books are mainly used to presented the reviewed literature, which are the work presented by Chollet (Chollet, 2017), Goodfellow (Goodfellow, 2017) and Ketkar et al. (Ketkar, 2017).

4.1 Machine Learning Fundamentals

Machine Learning has quickly grown to become the most popular sub-field of Artificial Intelligence, since it first started flourishing in the 1990s (Chollet, 2017) (Goodfellow, 2017). This trend is mainly driven by two factors; data availability and faster and more robust hardware. Machine learning has opened a new programming paradigm, and there are distinct differences

between the fundamental concepts of classical programming and machine learning. In classical programming, established rules describing how to treat the data are the main input, among with the data itself, and the output is the answers. For machine learning, the main input is the data itself and the answers, and the output is the rules linking the answers to the input data. This difference is clearly illustrated in Figure 4.1. These rules can then again be applied to new input data, such that answers close to the original ones can be obtained. As stated by Chollet; "A machine-learning system is trained rather than explicitly programmed. It's presented with many examples relevant to a task, and it finds statistical structure in these examples that eventually allows the system to come up with rules for automating the task." (Chollet, 2017).

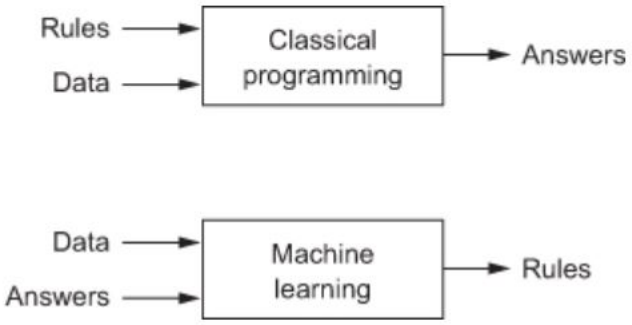


Figure 4.1: The main difference between machine learning and classical programming (Chollet, 2017).

4.2 Machine Learning Categories

Machine Learning can be broadly classified into two main categories, namely *supervised* and *unsupervised* learning (Chollet, 2017) (Goodfellow, 2017) (Ketkar, 2017). In the following, the main difference between these categories is presented, along with the associated application areas for each category (Goodfellow, 2017).

4.2.1 Supervised Learning

Supervised learning builds on the principle of learning the relationship between the input features and known targets variables, by giving the algorithm a set of examples demonstrating the relationship (Chollet, 2017). A prerequisite for performing supervised learning is a data set containing features that are associated with a label or target value. For the problem at hand, the

target variable could potentially be the *Stage3-AirPressure* for each instance of time, and the input features are then all the other sensor measurements. Mathematically, the input features are denoted by a set of random vectors x, with associated target values denoted by the vector y. The main objective is then to learn the relationship between the input features x and the target variable y, such that the future behavior of the target variable y can be predicted by only considering new *never-before-seen* input features. The origin of the term supervised learning is related to the though of an instructor or guide giving the machine learning algorithm examples of how the input features are linked to known targets (Goodfellow, 2017).

4.2.2 Unsupervised Learning

Unsupervised learning builds on the principal of learning interesting transformations and correlations in the data at hand, instead of attempting to learn the relationship between the input features and the target variable directly. Mathematically, only examples of random vectors x are considered, with the objective of learning the properties and characteristics associated with these random vectors (Goodfellow, 2017). For the problem at hand, a random vector x could for example be the $Stage3 - AirPressure$ or any of the other features. In contradiction to supervised learning, there exist no such instructor or guide illustrating the relationships between the input features and target variable, due to the fact that no target values are defined or given (Chollet, 2017) (Goodfellow, 2017).

4.2.3 Application Areas for Supervised and Unsupervised Learning

The two categories can roughly be used to categorize the various tasks that can be solved with machine learning, even though the line between supervised and unsupervised learning is not that distinct. In general, regression, classification and structured output tasks are classified as supervised learning problems, while density estimation commonly is classified as unsupervised learning (Goodfellow, 2017). Density estimation can be performed to better understand the data at hand, and for purposes of data denoising, data compression and data visualization. Clustering, which groups the data into smaller subsets containing similar patterns, and dimesionality reduction are other examples of unsupervised learning (Chollet, 2017). However, real-world data sets do not always correspond directly with the prerequisite of any of these two categories, such that other variants of learning paradigms also exists. Among them, *semi-supervised learning, multi-instance learning* and *reinforcement learning* (Goodfellow, 2017). To limit the scope, these categories will not be discussed further in detail. In addition, it is worth mentioning that

unsupervised anomaly detectors include all approaches not relying on labeled data, associating the feature with a normal or abnormal mark. Hence, the anomaly detector can be unsupervised, despite the fact that the machine learning algorithm itself is supervised, such as for regression tasks.

4.3 Machine Learning Framework

In order to build a machine learning model capable of predicting the target variable based on the input features, known as supervised learning, the algorithm needs to to trained, validated and tested. In the following, the basic machine learning framework associated with such tasks is emphasized, along with the challenge of optimizing the algorithm to perform well on *never-before-seen* data. Similar framework is relevant also for the unsupervised learning tasks, such as clustering based approaches, yet with some key differences presented at the end of this chapter.

4.3.1 Training, Validation and Testing Data Sets

The main objective for any supervised learning task is to perform well on *never-before-seen data*, requiring the machine learning model to be trained, validated and testing on three different data set. The *training set* is used to learn the relationship between the input features and the target variable, implying that the algorithm is allowed to see both the input features and the target variable.

The *validation set* is used to optimize the *hyperparameters* of the machine learning model. All machine learning models have a set of hyperparameters that are decided outside of the learning process. Examples of such hyperparameters are the number of hidden layers in a Neural Network or the number of trees in a Random Forest model, which are explained in detail in section 4.4.3 and 4.9. The validation set is then used to evaluate the model's performance, given a fixed set of hyperparameters. During this optimization process, data characteristics or information stored in the validation set leak into the model, known as information leaks, due to the fact that the model is optimized to best predict the target variable of the validation set. The process of hyperparameter optimization is iterative, and the more the model is adjusted based on the performance on the validation set, the more information from the validation set is leaked into the model. If the hyperparameters are adjusted only to fit the validation data set, the problem of *overfitting* occurs, while if the data set fails to capture phenomenas that are necessary to de-

scribe important relationships between the input features and the target variable, the problem of *underfitting* occurs. Hyperparameter optimization is considered to be a part of the training process, which is why an additional test set is required.

The *testing set* is used to evaluate the performance of the trained and optimized machine learning algorithm. Here, the algorithm is fitted to only *never-before-seen* input features, attempting to predict the target variable based on the learned relationship. The model is evaluated by comparing the predicted output of the target variable and the actual target variable belonging to the test period (Chollet, 2017), (Ketkar, 2017), (Goodfellow, 2017).

4.3.2 Capacity, Overfitting and Underfitting

In machine learning, *generalization* refers to the ability to perform well on *never-before-seen* data, mitigating the phenomenas of *overfitting* and *underfitting* (Goodfellow, 2017) (Chollet, 2017). In order to evaluate the generalization capability of a machine learning model, a generalization error is required, often referred to as the testing error. The generalization error is measured by evaluating the performance of the machine learning algorithm on the testing set, which for regression and classification is achieved by comparing the predicted output with the actual output (Chollet, 2017).

Goodfellow (Goodfellow, 2017) states that there are two factors determining how well a machine learning algorithm performs; i) the algorithms ability to make the training error small, ii) the algorithms ability to make the gap between the testing and training error as small as possible. The challenges of overfitting and underfitting are directly linked to these to factors. Underfitting occurs when the model fails to capture key characteristics and relationships between the input features and the target variable, making the model too general. In this case, the machine learning model does not have the ability to generate a sufficiently low error on the training set. Overfitting refers to the phenomena of learning the characteristic relationship between the input features and the target variable in a too detailed level, which generates a gap between the training and testing error that is too big. The challenge associated with overfitting and underfitting can be controlled by considering the *model capacity*. In this context, capacity refers to the model's ability to fit a wide range of functions. If the model capacity is too high, overfitting might occur, such that properties that only are associated with the training set are weighted too high. On the other side, models with too low capacity will struggle to capture the true characteristics of the training set (Goodfellow, 2017). The phenomenas of overfitting and underfitting are illustrated in Figure 4.2.

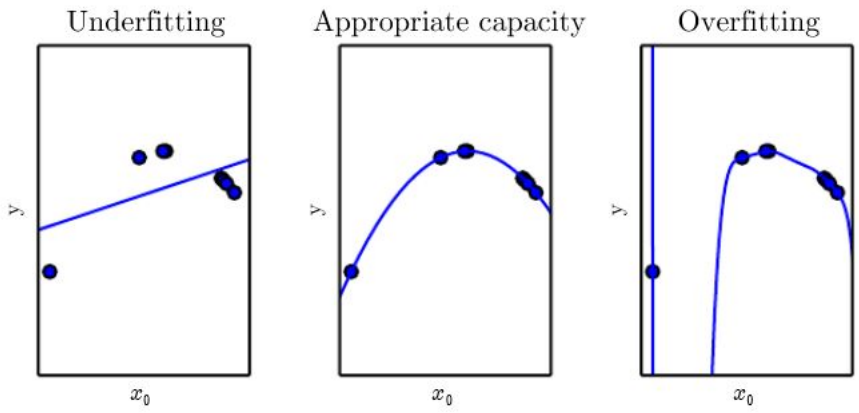


Figure 4.2: Illustrative example of the concept of underfitting and overfitting when the model's capacity is adjusted (Goodfellow, 2017).

Regularization refers to the process of fighting overfitting, and can be defined as "any modication we make to a learning algorithm that is intended to reduce its generalization error but not its training error" (Goodfellow, 2017). There exists several regularization techniques, among the most common; i) Add more training data, to increase the number of examples demonstrating the true relationships between the input features and the target variable, ii) Reduce the model's capacity iii) Add constraints on what information the model is allowed to learn, by adjusting the model's hyperparameters (Goodfellow, 2017).

In relation to capacity, the following yields; "machine learning algorithms will generally perform best when their capacity is appropriate for the true complexity of the task they need to perform and the amount of training data they are provided with" (Goodfellow, 2017). In other words, the capacity of the model needs to be adjusted to the complexity of the task, implying that complex tasks require high capacity, while simple tasks require reduced capacity.

In general, the training error reduces as the capacity increases, such that the training error asymptotically approaches the *minimum training error* value. The generalization error is typically a U-shaped curve, as a function of the model's capacity (Goodfellow, 2017). These phenomenas are illustrated in Figure 4.3, where the optimal capacity is located at the point of the U-curve where the generalization error is minimized.

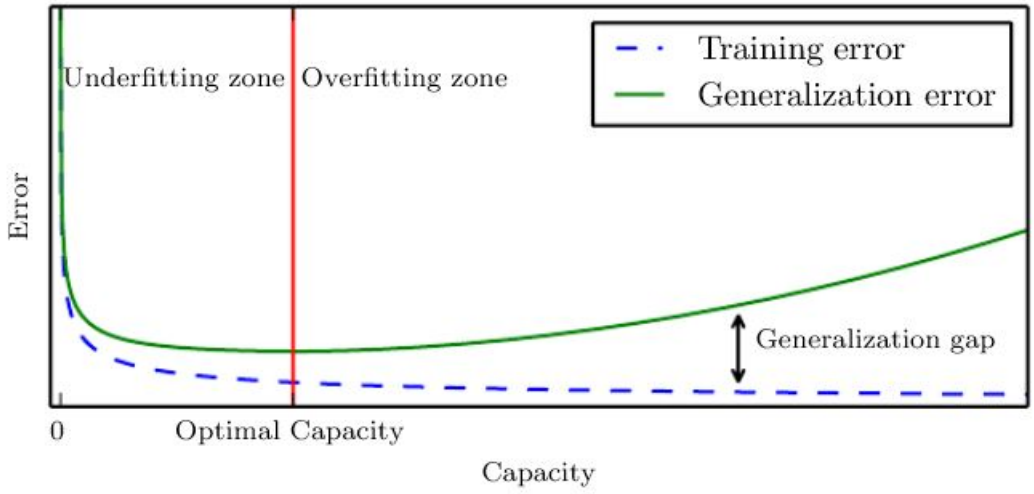


Figure 4.3: Balancing the training and generalization error to obtain the optimal level of capacity (Goodfellow, 2017).

4.3.3 Hold-Out Validation and K-Fold Validation

There exist several ways to split the data set into training, validation and testing sets, such that the model can be evaluated and optimized to obtain a required level of generalization. The most common ways are *Hold-out validation* are *K-fold cross-validation* (Chollet, 2017).

Hold-Out Validation

Hold-out validation builds on the principle of setting apart a fraction of the data for training and validation, while testing on the remaining part. From a broad perspective, the training data then contains a training set to train on and a validation set used to optimize the hyperparameters of the model (Chollet, 2017). As seen in Figure 4.4, the data set is divided between a training and validation set, and in addition comes a *never-before-seen* test set to evaluate the performance of the algorithm. Before testing the algorithm on the test set, the model is commonly trained on the on both the training and validation data set (Chollet, 2017).

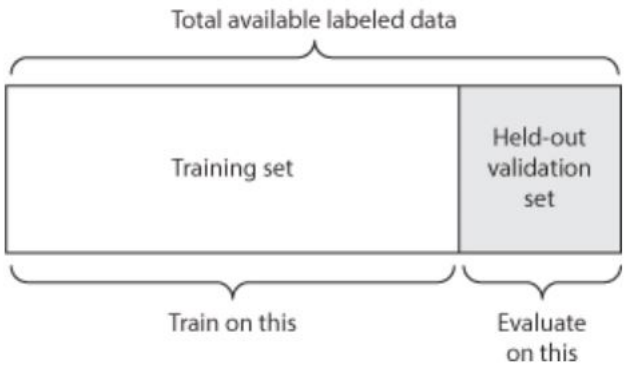


Figure 4.4: Illustrative example of simple hold-out-validation on the training and validation set, where the label refers to the target variable (Chollet, 2017).

The main drawback with this method arises when the amount of data available is small, such that the validation or testing data not are statistically representative of the true data characteristics. In order to address this problem, *K-fold cross-validation* can be applied (Chollet, 2017).

K-Fold Cross-Validation

K-fold cross-validation builds on the principle of splitting the data in K partitions of equal size, using one of these partitions for validation while using the rest for training. The hyperparameters of the model are then optimized on the one partition used for validation. The model performance is obtained by taking the average of the K scores obtained on the validation sets (Chollet, 2017). For three-fold validation, the data is split into three subsets, and the procedure of optimizing and evaluating are applied three times, as illustrated in Figure 4.5.

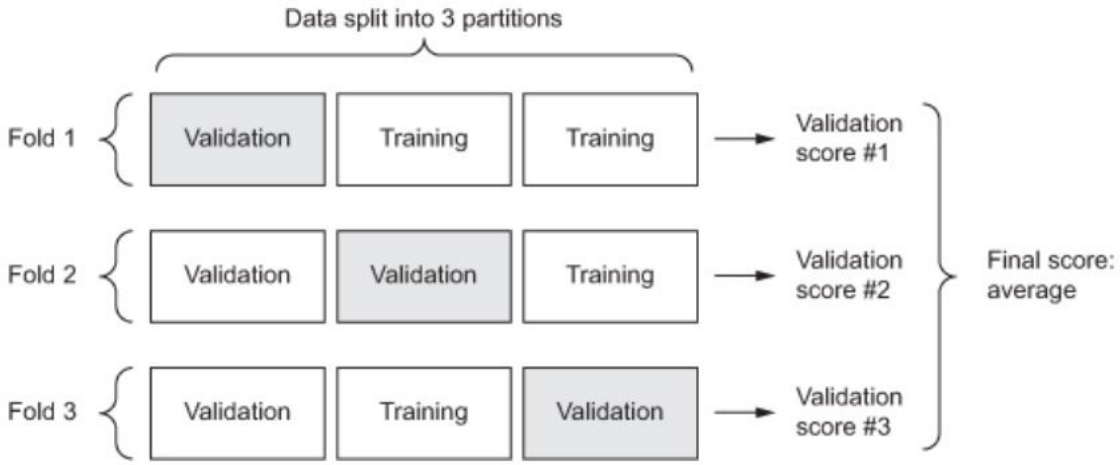


Figure 4.5: Illustrative example of K-fold cross-validation on the training and validation data sets (Chollet, 2017).

In cases where the amount of data available is small, K-fold cross-validation is beneficial, due the the fact that this approach does not require to use a distinct validation set for optimizing the hyperparameters (Chollet, 2017).

4.3.4 Performance Metrics

There exists several performance metrics that are appropriate for evaluating the performance of supervised machine learning models, more specifically, regression tasks. Among the most common are The Mean Absolute Error (MAE), The Mean Absolute Percentage Error (MAPE), The Mean Squared Error (MSE) and The Root Mean Squared Error (RMSE) (Brockwell and Davis, 2002), (Janacek, 2010), (Adhikari and Agrawal, 2013), (Hyndman and Athanasopoulos, 2018). In the following, these five performance metrics are explained, emphasizing their key properties and main drawbacks. For all metrics described, y_t depicts the actual value, f_t depicts the predicted value and $e_t = y_t - f_t$ describes the prediction error.

The Mean Forecast Error (MFE)

The Mean Forecast Error (MFE) measures the average deviation between the predicted values and the actual values, and is also known as the Forecast Bias, defined by Equation 4.1.

$$MFE = \frac{1}{n}\sum_{t=1}^{n} e_t, \tag{4.1}$$

The MFE gives an indication of the direction of the error, positively or negatively, which can be used when optimizing the hyperparameters of the algorithm. Furthermore, the effects of the positive and negative errors cancel out, while extreme errors not are weighted higher such as for the MSE. In addition, MFE is dependent on the data transformation and the scale of the data, which should be taken into account when comparing models using different input data. Having a minimum bias is desirable for high prediction accuracy, which is obtained when the MFE is approaching zero (Brockwell and Davis, 2002), (Janacek, 2010), (Adhikari and Agrawal, 2013).

The Mean Absolute Error (MAE)

The Mean Absolute Error (MAE), also referred to as the Mean Absolute Deviation, measures the average absolute deviation of the predicted values compared to the actual values (Brockwell and Davis, 2002), and is defined by Equation 4.2.

$$MAE = \frac{1}{n}\sum_{t=1}^{n} \mid e_t \mid, \tag{4.2}$$

The MAE describes the magnitude of the overall error between the predicted and true values, without canceling out the effect of positive and negative errors. Furthermore, the MAE does not indicate any direction of the error and is dependent on the data transformation and scale of the input data. In order to verify if the predicted values are close to the true values, the MAE should be minimized (Brockwell and Davis, 2002) (Adhikari and Agrawal, 2013).

The Mean Absolute Percentage Error (MAPE)

The Mean Absolute Percentage Error (MAPE) measures the absolute average percentage error between the predicted value and the actual value, defined by Equation 4.3

$$MAPE = \frac{1}{n}\sum_{t=1}^{n} \mid \frac{e_t}{y_t} \mid * 100, \tag{4.3}$$

In contradiction to the MFE and MAE, the MAPE is independent of the scale of the input data, while dependent on the data transformation. Similar as for the MAE, MAPE does not indicate any direction of the error and positive and negative errors does not offset each other (Brockwell and Davis, 2002), (Janacek, 2010).

The Mean Squared Error (MSE)

The Mean Squared Error (MSE) depicts the average squared error between the predicted and actual values, defined by Equation 4.4.

$$MSE = \frac{1}{n}\sum_{t=1}^{n} e_t^2, \tag{4.4}$$

The MSE is heavily affected by large individual errors, implying that few large errors cause the MSE to grow quickly compared to many small errors. Similar as for MAE and MAPE, MSE does not provide any idea of the direction of the error and is dependent on both the scale and data transformation of the input data (Brockwell and Davis, 2002), (Janacek, 2010).

The Root Mean Squared Error (RMSE)

The Root Mean Squared Error (RMSE) has the same properties as the MSE, except from the fact that it is squared (Brockwell and Davis, 2002), as given by Equation 4.5.

$$\sqrt{MSE} = \sqrt{\frac{1}{n}\sum_{t=1}^{n} e_t^2}, \tag{4.5}$$

4.3.5 Data Representativeness

In order for the model to properly learn and predict the target variable, the data used for training, validation and testing needs to be representative of the true data characteristics. When applying machine learning for classification and regression, the issue of data representativeness needs to be dealt with in different manners. For classification on data containing for example ten different classes/labels, ordered from class one to class 10, random shuffling needs to performed on. Without random shuffling the data, one risk to have a training and validation set containing input data with only the first classes, while the final classes are stored in the testing data set.

For time series, the arrow of time needs to be taken into account, which makes random shuffling inappropriate. For time series data, the goal is often to predict the future given the past. By random shuffling the data before splitting it into training, validation and testing set, a temporal leak will be created, such that the time space is reorganized causing the model to train on data from the "future". Hence, the data in the training and validation set should be collected from

a time interval before the time interval in the testing set. In this way, properly evaluating the model's performance by comparing the predicted values to the true values in the testing set is possible.

In addition, data redundancy needs to be considered for both regression and classification tasks. Data redundancy refers to the phenomena of data points or data intervals appearing twice or several times in the data set, which is common in real-world data sets. In this case, random shuffling will increase the risk of having redundant data in the training and validation set, which in practice cause the model to be validated on the training data. Hence, the training, validation and testing set need to be disjoint, implying that no samples are overlapping (Chollet, 2017).

4.4 Machine Learning Models

In the following, three supervised machine learning models are presented, namely the *Decision Tree model*, the *Random Forest model* and *Feedforward Neural Networks*. The main objective is to highlight how these models are built and optimized, which forms the framework for building the anomaly detector based on residuals. In addition, the unsupervised learning algorithm *K-means clustering* is presented, which forms the framework for building the anomaly detector based on clustering.

4.4.1 Decision Tree Model

Decision Tree models are powerful supervised learning algorithms, which can perform both classification, regression and multioutput tasks. In addition, the Decision Tree model is a fundamental component in the Random Forest model, such that the understanding one single decision tree is required to build a Random Forest model (Géron, 2017).

The Decision Tree model builds on the principle of predicting the target variable based on a decision tree. The decision tree is built based on a learned relationship between the target variable and the input features, and forms a set of decision rules for prediction of the target variable. The input features are represented in the branches of the tree, while the target variable is presented in the leaf nodes of the tree. The root node refers to the top branch of the tree, while the interior nodes refer to the nodes in the tree located in-between the root node and the leaf nodes (Géron, 2017). Furthermore, the depth of the tree is defined by the number of layers containing nodes in the same hierarchical level, while the width of the tree refers to the number of interior nodes located at the lowest hierarchical level. In Figure 4.6, a decision tree for regression is visualized,

having a depth of three, a with of four and eight leaf nodes containing possible target values depending on the decision outcome.

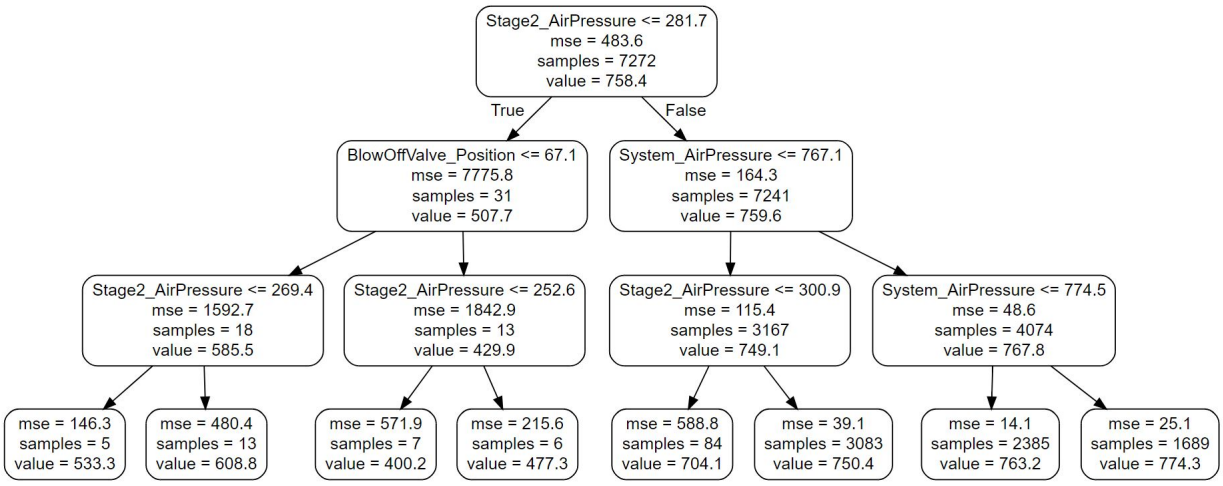


Figure 4.6: Illustrative example of a Decision Tree model for regression tasks.

There are some main differences between Decision Tree models used for classification and regression. For decision tree classification, the decision tree leafs represent the predicted class/label, while the tree branches represent the input feature conjunction leading to each specific class. For decision tree regression, the leafs of the tree represent the predicted target variable, as shown in figure Figure 4.6 (Géron, 2017). However, the learning process for decision tree classification and regression builds on the same procedure.

Building a Decision Tree Model

The training of Decision Tree models, known as "growing" trees, builds on the principal of first splitting the training set in two subsets based on one single feature k, with a given threshold t_k. For example, as illustrated in the root node in Figure 4.6, the training set is split at "Stage2-AirPressure<= 281.7". The pair (k, t_k) that produces the purest subsets is chosen by the algorithm, such that the training set is split in a way that minimizes the Mean Squared Error (MSE). After splitting the training set in two subsets, each subset is again split based on the same logic, which again is performed on the sub-subsets. This is a recursive process that continues until reaching a specified maximum depth of the tree. Hence, the Decision Tree model has a top-down learning approach, where the features in each hierarchical level in the tree are chosen such that a "best split" procedure is applied. Generally, the homogeneity of the predicted target values provided by the tree for different subsets is used to determine the "best split" (Abu-Mostafa

et al., 2019), (Géron, 2017). For regression tasks, the following costs function is minimized during the splitting procedure;

$$J(k, t_k) = \frac{m_{\text{left}}}{m}\text{MSE}_{\text{left}} + \frac{m_{\text{right}}}{m}\text{MSE}_{\text{right}}, \tag{4.6}$$

where

$$\text{MSE}_{\text{node}} = \sum_{i \in \text{node}} (\hat{y}_{node} - y^{(i)})^2, \tag{4.7}$$

and

$$\hat{y}_{node} = \frac{1}{m_{\text{node}}} \sum_{i \in \text{node}} y^{(i)}. \tag{4.8}$$

The predictions suggested in each leaf node are based on the average calculated target value of the number of samples associated with this node. In addition, the MSE is given for each predicted target value, based on the number of samples for this leaf node (Abu-Mostafa et al., 2019),(Géron, 2017).

Hyperparameter Optimization

One of the key characteristics of a Decision Tree model is that few assumptions are made about the input training data, in contradiction to for example a linear statistical regression models, assuming that the features and targets have a linear relationship. Similar yields for the Random Forest model and a Feedforward Neural Network. If few constraints are specified, the decision tree will fit itself closely to the training data, leading to overfitting. In order to prevent overfitting, regularization needs to be performed, by reducing the freedom associated with the decision tree during training. For decision trees, reducing the maximum allowed depth of the tree is a powerful regularization tool that will increase generalization and reduce the risk of overfitting. In addition, the minimum number of sample leafs, the minimum number of sample splits, the maximum number of leaf nodes and maximum number of features that are evaluated for splitting at each node are hyperparameters that can be adjusted to prevent overfitting and underfitting. In general, by reducing the maximum level of any hyperparameter containing a "maximum" limit, or increasing the minimum level of any hyperparameter containing a "minimum" level will regularize the model, such that optimization and generalization are balanced (Géron, 2017).

Decision Tree Performance

The Decision Tree model have several advantages, such as its good ability to handle tabular data containing numerical features or categorical features (Abu-Mostafa et al., 2019). In addition, these models are simple to understand and interpret, versatile and powerful. However, the Decision Tree model suffers from some limitations. The performance of the algorithm is sensitive to rotations in the training set and variations in the training data. The problem of instability is limited by the Random Forest model, as explained in the following (Géron, 2017).

4.4.2 Random Forest Model

The Random Forest model is capable of performing both classification and regression, and builds on an ensemble learning method. Ensemble learning refers to the principle of aggregating the predictions of a group of predictors, with the thought that a group of predictors will gain better results than one individual predictor (Géron, 2017). Random Forests models are based on an ensemble of decision trees, where each decision tree provides specific target predictions. For regression, the final output are obtained by taking the average of all outputs given by the predictors in the ensemble, as illustrated in Figure 4.7. For classification, the final output is based on the class getting the most votes in total by the decision trees in the ensemble (Géron, 2017).

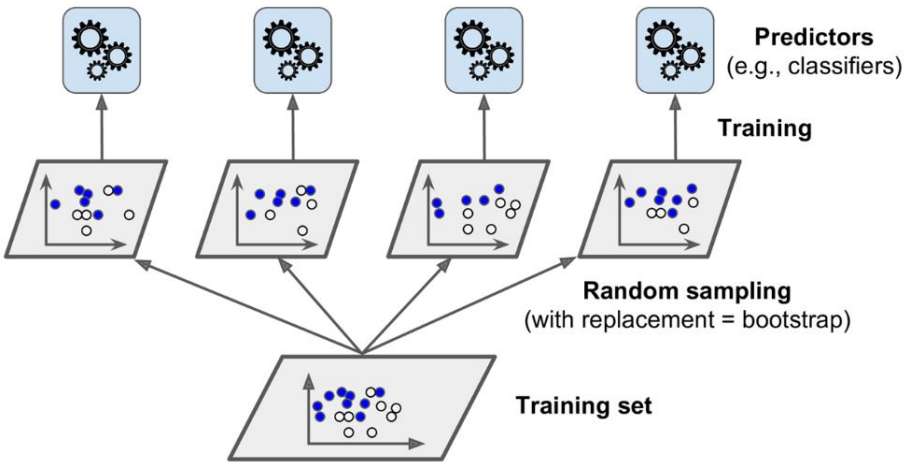


Figure 4.7: Illustrative example of the Random Forest model building on ensemble learning with a group of predictors (Géron, 2017).

Building a Random Forest Model

The Random Forest algorithm is based on bootstrap aggregation, also known as bagging (Géron, 2017). Bootstrap aggregation builds on the principle of random sampling with replacement, meaning that B random samples of the same training set are taken. Decision trees are fitted to each of these samples (James et al., 2013), described by the following procedure;
i) A training set, with input features $\mathbf{x} = x_1, ..., x_n$ and targets $\mathbf{Y} = y_1, ..., y_n$ is sampled with replacement B times, such that $b = 1, ..., B$. In this way, n training subsets are obtained, denoted $\mathbf{X}_b$ and $\mathbf{Y}_b$.
ii) A decision tree, f_b, is then built from each of this subsets $\mathbf{X}_b$, $\mathbf{Y}_b$.

The target variable predictions, denoted x', are then obtained by taking the average value of all the predictions generated from the decision trees (James et al., 2013), as illustrated in Figure 4.8. This is defined mathematically by;

$$\hat{f} = \frac{1}{B} \sum_{b=1}^{B} f_b(x') \tag{4.9}$$

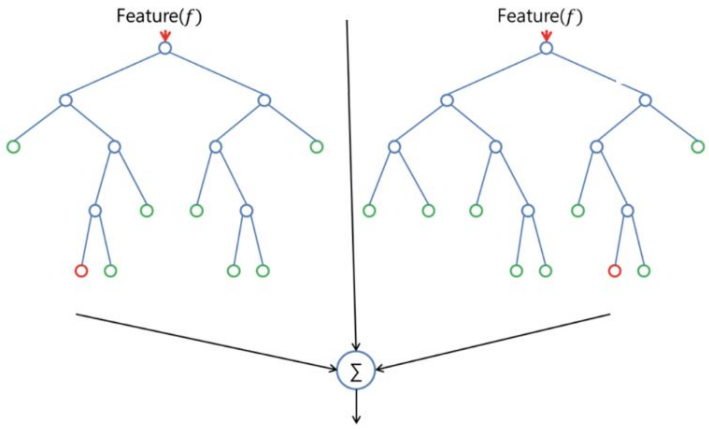


Figure 4.8: The target value is obtained by taking the average value of the predicted output from several decision trees (Donges, 2018).

Hyperparameter Optimization

The hyperparameters to optimize for a Random Forest model are similar to the hyperparameters associated with a Decision Tree model, which was explained in section 4.4.1. In addition, hyperparameters that are specifically intended to control the bootstrap aggregation with the

ensemble of predictors need to be optimized. For example, the hyperparameter controlling the maximum samples in each training set in the ensemble and the maximum number of individual decision trees to grow (Géron, 2017).

Random Forest Performance

Bootstrap aggregation leads to increased prediction performance, in terms of reduced variance and reduced risk of overfitting. As explained in section 4.4.1, one single decision tree is sensitive to variations and rotation in the training set, whereas the average prediction generated by many trees does not suffer from the same limitation. However, there are some factors to be considered when building a Random Forest model that can limit the performance. As the number of trees grow, such that many trees are built from the same training set, the risk of strongly correlated trees grow. The risk of highly correlated trees is reduced through the introduction of extra randomness when growing trees, known as "feature bagging". Feature bagging refers to the concept of considering only a random subset of the input features for each candidate split during training, instead of searching for the very best feature for the split among all features. Features having strong correlations to the target variable will be selected in many cases, such that many of the B trees contain the exact same splits. Hence, by considering only a random subset of features for each split, the correlation between the trees is reduced, and the algorithm is forced to consider all features (James et al., 2013).

Feature Importance

Decision Tree models and Random Forest models make it possible to measure the relative importance of each input feature, which for complex problems with many features is highly valuable information. The feature importance is calculated by considering how much all the nodes in a decision tree using on specific feature contribute to reducing the average error. Hence, the feature importance in based on a weighted average, calculated based on the consideration of the number of training samples associated with each node. The feature importance result obtained after training and optimization is often scaled, such that the sum equals 1 (Géron, 2017).

4.4.3 Deep Learning - Feedforward Neural Networks

Deep learning is a specific subfield of machine learning, where the *deep* in *deep learning* refers to the idea of learning successive *layers* of increasingly meaningful representations. The *depth* of

of the model denotes the number of layers contributing to map the relationship between the input features and the target variable. In deep learning, *Neural Networks* are most commonly applied to learn these layered representations, where the relationship between the input features and target variable is learned via deep sequences of simple data transformations, performed by the layers. From the 2010s, deep learning models, more princely Neural Networks, have shown remarkable results within complex fields, such as speech recognition, image classification, autonomous driving and text-to-speech conversion. However, researchers are still exploring the full extent of what deep learning models can accomplish. *Feedforeward Neural Networks*, also known as *Multilayer Perceptrons* (MLP), are the building blocks of all types of Neural Networks, among them *Recurrent Neural Networks* (RNN), *Long-Short-Term Memory* (LSTM) and *Gated Recurrent Neural Networks* (GRNN). For the problem at hand, a Feedforward Neural Network will be investigated, based on the fact that these models in its simple form have proven to accomplish complex tasks (Ketkar, 2017),(Goodfellow, 2017), (Chollet, 2017).

Building a Feedforward Neural Network

The main objective of a Feedforward Neural Network is to approximate some function f^*, such that the input x is linked to the output y through $y = f^*(\mathbf{x})$. The parameters of the network, θ, are adjusted in the learning process, in order to obtain the learned model defined by $f(\mathbf{x};\theta)$ that best approximates the true target function f^*. This relationships are defined by $f(\mathbf{x};\theta) = \hat{y} \approx f^*(\mathbf{x}) = y$.

Feedforward Neural Networks can be represented by the composition of different functions, which is why they are denoted networks. A directed acyclic graph is used to describe how these functions are composed together in a chain with different layers L, denoted mathematically by $f(\mathbf{x},\theta) = f^{(L)}(f^{(L-1)}(...f^{(2)}(f^{(1)}(\mathbf{x};\theta^{(1)})))$. For each layer in the network, the input data is transformed to an output by these functions. The depth of the model is defined by the overall length of the chain, or by the number of layers in the network.

For Neural Networks, one distinguish between *input layer*, *hidden layers* and *output layer*. The main function of the input layer is to pass the input data into the network, without making any transformations on the data. The input and output layer are connected by a sequence of hidden layers, $l = 1,2,...L-1$, where transformations are made on the data. The type of transformation made in each hidden layer depends on the *activation function* specified to used for the neurons in each layer. Examples of such activation functions are presented in section 4.4.3. The output layer is connected to the last hidden layer in the sequence of hidden layers, and its main function is to convert the signal to the final output, denoted $\hat{y}$.

The architecture of the Neural Network is defined by how the layers in the network are composed, such as the number of neurons in each layer. Weights or synapses connects the neurons in one layer, l_i, to the neurons in layer l_{i+1}, which are commonly stored in a layer specific matrix denoted $\mathbf{W}^{(l)}$. The neurons in a layer receives input data from the neurons in the preceding layer, where the input signal to a neuron is the output signal of the neurons in the preceding layers multiplied by their weights, with an added bias term. The input signal is defined mathematically as $z_i^{(l)} = \mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + b_i^{(l)}$, where $\mathbf{h}^{(l-1)}$ denotes the vector of signal outputs from the neurons in the preceding layer and $b_i^{(l-1)}$ denotes the bias term for neuron i. The input signal, $z_i^{(l)}$, received by the neurons are then transformed by an activation function σ, before the signal is passed on to the neurons in the next layer. Hence, each neuron has a specific output, $a_i^{(l)}$, which is calculated based on the input signal, weights, bias and activation function, defined mathematically by $a_i^{(l)} = \sigma(z_i^{(l)})$ (Goodfellow, 2017), (Ketkar, 2017), (Chollet, 2017). A Feedforward Neural Network with two hidden layers is illustrated in Figure 4.9, where the input layer and hidden layer have 3 neurons, and the output layer consists of a single neuron.

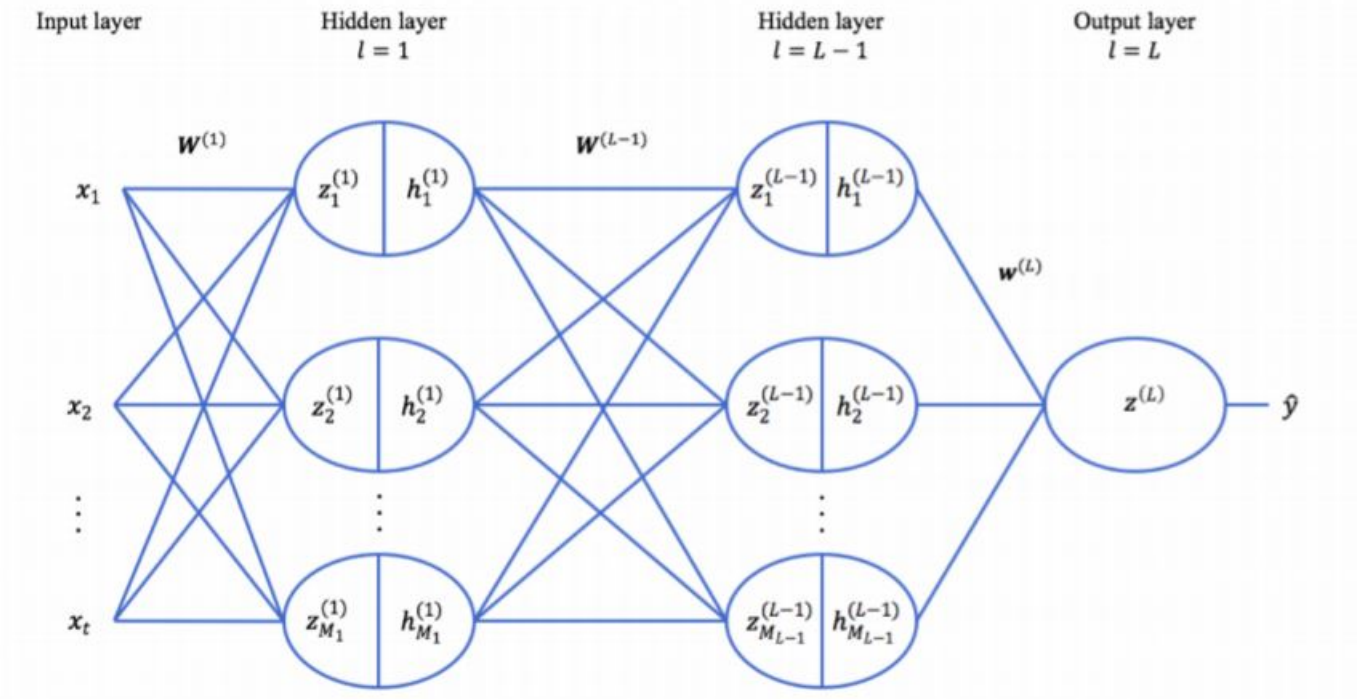


Figure 4.9: A Feedforward Neural Network with two hidden layers, demonstrating the concept of input signals, output signals and final prediction $\hat{y}$ (Goodfellow, 2017).

The name *feedforeward* originates from the basic concept of these models, where information flows through the network from the input layer, to the intermediate hidden layers and finally

to the output layer. Hence, there are not any feedback connections where the model's output is fed backwards in the model. Feedforward Neural Networks can be extended to include such feedback connections, such as for example a *Recurrent Neural Networks* (Goodfellow, 2017), (Chollet, 2017), Ketkar (2017).

Hyperparameter Optimization

Neural Networks have high flexibility, which is also one of their main drawbacks when it comes to hyperparameter optimization. The hyperparameters to optimize are related to three main factors, namely the architecture of the network, the activation function and the signal weights associated with each neuron in the network. In relation to the architecture, the number of hidden layers and the number of neurons in each hidden layer need to be optimized, which depends on the complexity of the task and the time and resources available. A Feedforward Neural Network with only one hidden layer can accomplish to model complex tasks, provided that each layer have enough neurons. However, a Neural Network with several hidden layers can model complex functions with exponentially fewer neurons in each layer, which has proven to reduce the time required to train the network. In addition, the choice of activation function relies to some extent on the problem at hand (Goodfellow, 2017), which is emphasized further in section 4.4.3.

In order to optimize the hyperparameters of a Neural Network, such that the predicted output $\hat{y}$ is as close as possible the true output y, a cost function, also known as loss function, is used to map the progress (Goodfellow, 2017). The loss function, $C(f(\mathbf{x};\theta), f(\mathbf{x};\theta))$, aims to evaluate the performance of a set of given hyperparameters, θ, such that the distance between the predicted output $\hat{y}$ and the true output y can be minimized. Furthermore, the loss function should be chosen depending on the type of problem at stake. For time series forecasting, also known as sequential modelling, the Mean Squared Error, the Root Mean Squared Error, the Mean Absolute Error, and the Mean Absolute Percentage Error are well known loss functions, as presented in section 4.3.4 (Goodfellow, 2017).

An epoch denotes one training round, where the parameters in the Neural Network are adjusted to minimize the loss function. As explained in section 4.3.2, there is always a trade of between optimization and generalization. As the number of epochs increase, the risk of overfitting the training set increases, whereas the risk of underfitting increases if the number of epochs is too low. In addition, computational resources and time also impacts the number of epochs used on training and optimization (Goodfellow, 2017) (Chollet, 2017).

For Feedforward Neural Networks, the information flows forward in the network, from the input

layer to the the output layer, as explained in section 4.4.3. Back-propagation on the other hand, refers to the concept of feeding information obtained from the loss function back into the Neural Network, with the aim of reducing the predicting error. For more advanced Neural Networks, such as Recurrent Neural Networks, back-propagation has proven to improve the performance significant. In order to perform back-propagation, the chosen activation function needs to be differentiable, such that gradients indicating the direction of the error can be obtained. This is referred to as *gradient-based methods*, where the gradient is used to minimize the loss by adjusting the weights in the Neural Network, depending in the direction of the error (Goodfellow, 2017) (Ketkar, 2017).

Activation Functions

The transformations made on the signals by each of the neurons in a Neural Network are performed by an activation function, $\sigma(x)$, as explained in section 4.4.3. Unfortunately, there exist no simple guide providing clear information on when to use a specific activation function, due to the fact that this is still an area of research. Hence, testing several activation functions and estimating the generated error between the predicted and true values is therefore the most common approach. However, there are several properties of interest for activation functions that should be taken into account. Theory claims that by choosing a non-linear activation function, a Neural Network consisting of only two layers will be able to approximate any function between the input and output signal, given by the fact that the hidden layer consists of a sufficient amount of neurons. Furthermore, choosing a continuously differential activation function is a prerequisite for gradients to be computed, such that back-propagation can be applied. In addition, activation functions with finite ranges have proven to give more stable performance when gradient-based methods are applied, and empirical evidence prove that smooth functions are preferred (Goodfellow, 2017).

The most commonly used activation functions are the *Linear*, *Sigmoid*, *Tanh* and *ReLU* functions, all having in common that they are continuously differentiable such that gradient-based learning can be applied (Goodfellow, 2017). A linear activation function is the simplest, transforming the input signal by the following equation;

$$\sigma(x) = ax, \tag{4.10}$$

where x denotes the input signal and a denotes a constant. Due to its simplicity, a linear activation function makes gradient-based learning simple (Ketkar, 2017).

The Sigmoid activation is often used to transform the signal in the output layer, due to the fact that the output is a scalar value often regarded as a probability (Ketkar, 2017), (Goodfellow, 2017), (Chollet, 2017). The following equation defines the Sigmoid activation function:

$$\sigma(x) = \frac{e^x}{1 + e^x} \tag{4.11}$$

The hyperbolic tangent activation function is commonly used for hidden layers and outputs a value between -1 and 1 (Ketkar, 2017), (Chollet, 2017), defined mathematically by:

$$\sigma(x) = \tanh(x) \tag{4.12}$$

The Rectified Linear Unit (ReLU) has recently been used as the default function for hidden layers, and results show that the ReLU activation function is specifically useful for gradient-based learning, due to the fact that the ReLU function gives large and consistent gradients (Ketkar, 2017). Furthermore, the ReLU function zeros out negative values, and is defined mathematically by:

$$\sigma(x) = \max(0, x) \tag{4.13}$$

4.4.4 Clustering Based Machine Learning Models

Clustering based models build on the principle of unsupervised learning, where the main objective is to classify the data at hand into separated groups with similar patterns, where each group is a cluster. As explained in section 4.2.2, unsupervised learning is usually performed to gain insight about the data at hand, classify and compress the data, without mapping the relationship between input features and a target variable. Since the early 1950s, a variety of clustering based algorithms have been proposed, which generally can be classified into two main groups; *hierarchical* and *partitional* clustering (Celebi, 2015). Clustering based on hierarchical algorithms work in a recursively manner, by finding nested cluster from a top-down or bottom-up approach. Partitional algorithms on the other, do not impose an hierarchical structure, meaning that all clusters are found simultaneously as a partition of the provided input data. Due to lower complexity, partitional algorithms are better suited to handle large data sets than hierarchical algorithms. The *K-means clustering algorithm* is the most widely used partitional clustering algorithm, mainly due to is simplicity and robustness (Wu, 2012). For the problem at hand, the K-means clustering algorithm can potentially contribute to mapping outliers and anomalies not located in a specific cluster.

The K-Means Clustering Algorithm

The K-Means clustering algorithm is attempting to find K non-overlapping clusters of data. Each of the K clusters are represented by a centroid, which generally is the mean of all samples gathered in one specific cluster, as illustrated in Figure 5.3. The data set to be clustered is defined by $D = \{x_i, ..., x_n\}$, where x_i represents one sample of measurement. The number of clusters K with individual centroids are specified by the user in advanced. The K-Means algorithm builds on the principle of assigning all sample measurements to the closest centroid. Once all samples are assigned to a centroid, the samples surrounding the centroid form a cluster. Through an iterative process, the centroid of each cluster is updated based on the assigned data, until no samples change cluster (Wu, 2012), (Celebi, 2015).

The K-Means clustering algorithm can be expressed by an objective function, which depends on the proximities of the data points belonging to the centroids of the clusters (Wu, 2012), expressed by the following equation;

$$min_{m_k, 1 \leq k \leq K} \sum_{k=1}^{K} \sum_{\mathbf{x} \in C_k} \pi_{\mathbf{x}} dist(\mathbf{x}, \mathbf{m}_k), \tag{4.14}$$

where $\pi_{\mathbf{x}}$ is the weight of the samples $\mathbf{x}$ and n_k is the number of samples assigned to cluster C_k. Furthermore, $\mathbf{m}_k = \sum_{\mathbf{x} \in C_k} \frac{\pi_{\mathbf{x}\mathbf{x}}}{n_k}$ denotes the centroid of cluster C_k, while K is a predefined number of cluster decided by the user. The distance between the samples $\mathbf{x}$ and the centroid $\mathbf{m}_k$ is computed by the "$dist$" function. There exist several options regarding the choice of function to compute this distance. However, the most commonly used function is the squared Euclidean distance, denoted by $\|\mathbf{x} - \mathbf{m}\|^2$ (Wu, 2012) (Celebi, 2015)

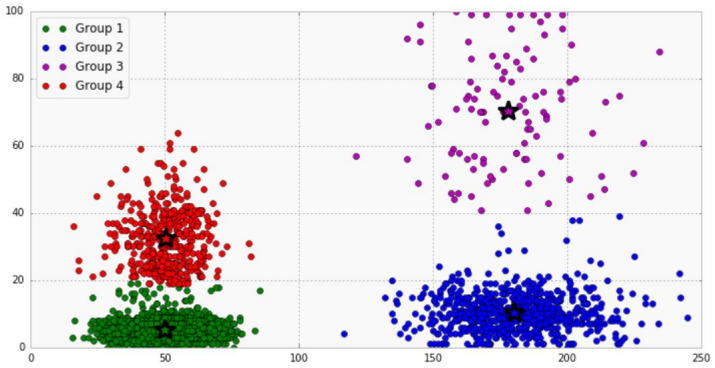


Figure 4.10: Illustrative example of clusters generated by a K-Means clustering algorithm (Trevino, 2016).

The K-Means algorithm is sensitive to outliers, due to the fact that the centroids are adjusted based on the samples assigned to each centroid through an iterative process (Trevino, 2016). However, the K-means algorithm is simple, robust and easy to implement, and are applicable for a variety of data types (Wu, 2012).

Chapter 5

Unsupervised Anomaly Detection Framework

In this chapter, two suggested work-flow-procedures for building unsupervised anomaly detectors based on machine learning are presented, namely the unsupervised anomaly detector based on residuals and the unsupervised anomaly detector based on clustering. The presented frameworks are based on reviewing the anomaly detection surveys presented by Chandola et al. (Chandola et al., 2009a), Sanz-Bobi (Sanz-Bobi, 2016) and Patcha and Park (Patcha and Park, 2007), and practical case studies implementing similar models. For the anomaly detector based on residuals, Ryan et al. (Ryan et al., 1998), Vasconcelos et al. (Vasconcelos et al., 1995), Markou and Singh (Markou and Singh, 2003b) and Augusteijn and Folkert (Augusteijn and Folkert, 2002) illustrate relevant examples, implementing Neural Networks to model the normal behavior of the system. For the anomaly detection based on clustering, the following case studies are weighted highly; Breunig et al. (Breunig et al., 2000), using the *local outlier factor*, Schölkopf et al. (Schölkopf et al., 2000), using *support vector machines*, and Bezdek et al. (Bezdek et al., 1984), applying a *K-means* clustering algorithms. In addition, the machine learning fundamental framework presented in Chapter 4 is included in the work-flow-procedure for anomaly detection, involving training, validation and testing the machine learning models (Chollet, 2017), (Ketkar, 2017), (Goodfellow, 2017).

5.1 Unsupervised Anomaly Detection Based on Residuals

For this approach, a supervised machine learning model is applied to model the normal behavior of the system, by approximating the relationship between a given target variable, such as for example the *Stage3-AirPressure*, and the input features. The established relationship make it possible to predict the target variable, only by considering newly available samples from the input features, as emphasised in Chapter 4. The anomaly detector can accordingly be established by calculating the residuals between the predicted and actual target variable, using the predicted variable as a reference describing normal behavior (Chandola et al., 2009a), (Sanz-Bobi, 2016), (Patcha and Park, 2007). In the following, the procedure for building an anomaly detector based on residuals is presented;

1. Model the normal behavior of the system based on a chosen machine learning model, such as the Decision Tree model, the Random Forest model or a Feedforward Neural Networks. This involves all steps associated with machine learning models aiming to predict the target variable based on a learned relationship with the input features, such as;
   (a) Preprocessing the raw data, involving cleaning, filtering, filling out missing values, denoising and possible scaling the data.
   (b) Identify time frames where the input data appears to follow normal operating behavior, usually done thorough data visualization, descriptive statistics and based on expert judgments.
   (c) Split the continuous time series data associated with normal operating behavior into a training, validation and testing data set, allowing for hyperparameter optimization on the validation set and testing the model performance on never-before-seen data.
   (d) Evaluate the model performance on both the validation and testing set, by considering appropriate performance metrics for regression tasks, such as the MAE, MAPE, MSE and RMSE. The objective is to build a model that generalize well, meaning a model that performs well on never-before-seen data.
2. Calculate the residuals between the predicted target variable by the machine learning model, denoted estimated output, and the actual target variable belonging to the testing period, denoted real measured output. In Figure 5.1, the calculation of residuals between the estimated and measured output is visualized. The magnitude of the residuals for each instance of time is used to determine if a sample is classified as normal or abnormal.
3. Perform descriptive statistics on the calculated residuals, in order to investigate the dis-

tribution of the residuals. In normal operating behavior, the expected and mean value of the residuals should approach zero, and in most cases, the residuals will approximate a normal distribution having a fairly low standard deviation.

4. Decide on a confidence interval to use for establishing an *upper control limit* and *lower control limit*, such that samples exceeding these limits in either direction are classified as anomalies (Sanz-Bobi, 2016), (Dunning and Friedman, 2014). Considerations for determining these confidence bounds for decision making is further emphasized in section 5.1.1.

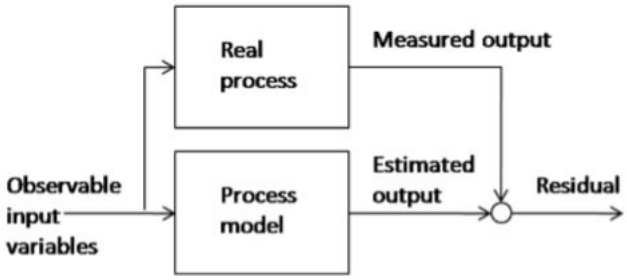


Figure 5.1: Residuals between the measured output and the estimated output from the machine learning model (Sanz-Bobi, 2016).

The main requirement for this approach is that the training and validation set are as representative as possible of the normal operating behavior of the system. Hence, the process of cleaning, visualizing and performing descriptive statistics on the input data is of high importance. If the algorithm is trained on data containing high levels of abnormal samples or patterns not corresponding to normal behavior, the anomaly detector will fail to classify these as anomalous when similar cases appear in new streaming data (Sanz-Bobi, 2016). If the learned relationship between the target variable and the input features in a normal situation stops yielding, evidence is indicating that something abnormal has occurred, argued as followed by Hawkins (Hawkin, 1980); "An outlier is an observation that deviates so much from other observations as to arouse suspicion that it was generated by a different mechanism.". In this case, the mechanism cause the relationship between the target variable and the input features to stop yielding.

5.1.1 Establishing Confidence Bounds for Decision Making

In order to determine if anomalous behavior is observed, an upper and lower control limit for the magnitude of the residuals need to be established, referred to as confidence bounds. The confidence bounds can be established based on the principles of *Statistical Process Control*,

where an upper and lower control limit are placed a specific number of standard deviations from the mean value in either direction, for example three standard deviations (Oakland, 2007). In Figure 5.2, an illustrative example of established control limits with three standard deviations from the mean value is visualized, where the population of samples is assumed to follow a normal distribution. For three standard deviations, there is a 99,7% chance that any new sample belonging to the same distribution falls within the upper and lower control limit, or there is 0,3% chance of finding a value beyond three standard deviations.

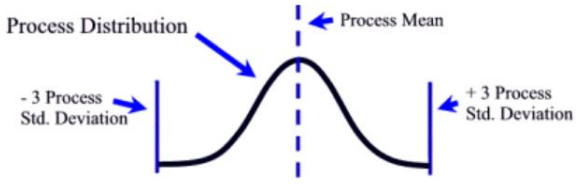


Figure 5.2: Illustrative example of established confidence bounds at three standard deviations from the expected value of the residuals (Oakland, 2007).

In order to established control limits that are suitable for this specific case, the following factors need to be considered;

1. Minimize the occurrence of false alarms and non-detection, which in statistical hypothesis testing is known as minimizing the occurrence of false positives, **type I error**, and false negatives, **type II error** (Lehmann, 2005), defined accordingly;

   (a) **type I error** refers to the rejection of the true null hypothesis, such that a false positive is detected. In this case, this implies that the target variable is characterized as abnormal when in reality the behavior is normal.

   (b) **type II error** refers to the failure of rejecting a false null hypothesis, such that a false negative is concluded. In this case, this implies that the target variable is characterized as normal when abnormal behavior should have been detected

2. False positives and false negatives are associated with both time, costs and risk, such that the following needs to be determined;

   (a) **The optimum amount of time** spent on inspection, taking into account that time spent on inspection increases as the number of false positives increases, which both can distract the operator and reduce the trustworthiness of the anomaly detector.

   (b) **The acceptable level of risk** associated with false negatives, taking into account both the costs associated with damaged equipment and production downtime if anoma-

lies not are detected. In addition, false negatives can potentially expose both the environment and humans to damage.

5.2 Unsupervised Anomaly Detection Based on Clustering

Clustering based on machine learning is categorized as unsupervised learning, where the model not relies on relevant examples demonstrating the relationship between the target variable and input features. For anomaly detection based on clustering, the aim is to classify the input data according to some reference pattern that are learned and extracted from the training data. The extracted pattern can either represent the entire vector of measurements for a variable or a reduced vector, generated through compression of the measurements. An anomaly detector based on clustering is said to be based on *information entropy reduction*, due to the fact that the input data is transformed into clusters of data having smaller dimensions than the original number of samples. In this way, observed sample redundancy is eliminated, while the discriminatory associated with the data is preserved (Sanz-Bobi, 2016). An anomaly detector based in clustering can be built based on the following procedure;

1. A sub set of the input data representing normal operating behavior is chosen, and a clustering algorithm is applied to group the input data into smaller groups with internal similarities. The generated clusters represents the pattern to expect in normal operating condition, and are used as *reference pattern*. The procedure of clustering depends on the chosen algorithm, for K-Means, the following yields;

   (a) Decide on the number of clusters, K, appropriate for describing the variance in the chosen variables. In order to determine K, the *Elbow Curve* is often used, which describes the score obtained by the clustering algorithm for different number of clusters. A sufficient score is given when the number of clusters accomplish to capture all the relevant patterns observed in the data (Wu, 2012), (Celebi, 2015).

   (b) Fit the K-Means clustering algorithm with sufficient number of K clusters to the data characterizing normal operating behavior, such that the reference pattern is created.

2. Once new data becomes available, the clustering procedure is again performed, and a *measured pattern* is generated. The anomaly detector is created by comparing the reference pattern, describing normal operating behavior, to the measured pattern, representing newly available data. The comparison of the reference and measured pattern is visualized in Figure 5.3. This comparison is usually performed based on a computation of

distance measurement, where the measured pattern is assigned to the reference pattern that are closest in distance.

3. Classification of anomalies is based on a predefined acceptance level of distance between the measured and preference pattern. The samples having the longest distance to any of the centroids in the reference pattern are classified as abnormal, depending on a predefined *outlier fraction*. For example, an outlier fraction of 0,02, classifies 2% of the samples with longest distance to any centroid as abnormal (Sanz-Bobi, 2016). The outlier fraction should be optimized such that the risk of false positives and false negatives are minimized, taking into account the considerations presented in section 5.1.1.

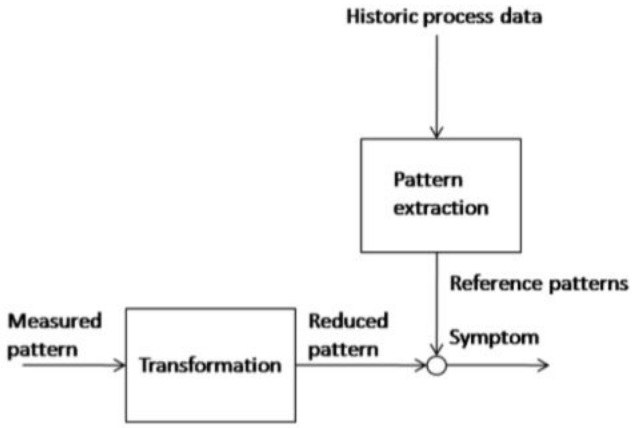


Figure 5.3: Comparison between the reference pattern representing normal behavior and measured pattern (Sanz-Bobi, 2016)

Similar as for the anomaly detector based on residuals, the main requirement for the detector based on clustering is that the reference pattern is created based on data describing normal operating behavior (Sanz-Bobi, 2016).

5.3 Comparison of the Unsupervised Anomaly Detectors

The main difference between the approach based on residuals and the approach based on clustering, is that the clustering approach is more limited by the number of features it considers. For regression tasks, such as the Decision Tree model, the Random Forest model and Feedforward Neural Networks, all the input features are examined, aiming to incorporating all variables affecting the target variable. For clustering approaches, a chosen number of variables are examined, often maximum two or three. However, the clustering approach requires little modeling effort, and are known to be relatively easy to build and implement (Sanz-Bobi, 2016).

According to the *Numenta Benchmark Requirements*, as presented in Chapter 3, the following constraints are weighted highly; i) the algorithm must have the ability to continuously learn in an adaptive manner, to capture seasonal and cyclic variations, and possible drifts, ii) anomaly detection must be made online and in real time, requiring the algorithm to fit the data in real time, iii) Anomalies should be detected as early as possible, based on decision rules minimizing the risk of false positives and false negatives (Lavin and Ahmad, 2015). In relation to the *Numenta Benchmark Requirements*, the two anomaly detectors fulfill different constraints. Both detectors are capable of detection anomalies at an early stage, to which degree, depends on the established confidence bounds and the defined outlier fraction. Furthermore, the Numenta Benchmark highlights the need for detecting anomalies in real time. The anomaly detector based on residuals has the advantage that the algorithm easily can be trained to capture seasonal and cyclic variations, where the relationship between the target variable and input features for different seasons is captured, assuming that the data is available. For clustering, which only considers a limited number of features, these variations are harder to capture, and one might need to create several reference patterns depending on the season (Chandola et al., 2009a). However, the presented work-flow-procedures demonstrate how unsupervised anomaly detectors for continuous time series data can be built, not relying on historical failures or labeling.

Chapter 6

Data Preprocessing

In the following chapter, the preprocessing performed on the raw data is presented, along with time series visualization and descriptive statistics performed on data characterizing normal operating behavior. In addition, the open source libraries and tools used to build the unsupervised anomaly detectors based on residuals and based on clustering are presented.

6.1 Data

As presented in Chapter 2, the compressor system is equipped with in total 21 sensors, sending automatic signals to the remote control system. The instrumentation equipment serves two main purposed; assist the automatic control system such that actions can be automatically taken if control limits are exceeded and monitor the system for detection of abnormal behavior. In Table 6.1, the sensors are presented with associated control limits established by the system manufacturer, Ingersoll Rand. Depending on the sensor, the following control limits are given; *Low trip*, *Low alarm*, *High alarm* and *High trip*. The trip control limits, either high or low, denotes the limit for which the compressor system is automatically switch of, argued by the fact that further use will damage the system equipment. The alarm limits, either low or high, indicates the level at which it is of serious belief that damage to the system equipment has occurred, argued by the fact that these limits are set further out than the trip limits. In cases where the system not shuts automatically down when the trip limits are exceeded, alarms will be given to inform the operator. For anomaly detection at an early stage, such as argued in the *Numenta Benchmark Requirements* in Chapter 3 (Lavin and Ahmad, 2015), these control limits fail to capture early degradation and small fluctuations that eventually can cause system damage. Hence,

in order to successfully implement a predictive maintenance strategy, an anomaly detector with finer granularity than these boundaries are required.

Table 6.1: Sensors with associated control limits provided by the equipment producer Ingersoll Rand

Sensor name	Unit	Sampling	Low trip	Low alarm	High alarm	High trip
Motor-TemperaturePhase1	C	1s	-	-	155	170
Motor-TemperaturePhase2	C	1s	-	-	155	170
Motor-TemperaturePhase3	C	1s	-	-	155	170
Bearing-TemperatureDE	C	1s	-	-	90	95
Bearing-TemperatureNDE	C	1s	-	-	90	95
InletValve-Position	%	1s	-	-	-	-
Stage1-Vibration	μm	1s	-	-	28	33
Stage1-AirTemperature	C	1s	-	-	46	49
Stage1-AirPressure	kPa	1s	-	-	-	-
Stage2-Vibration	μm	1s	-	-	24	29
Stage2-AirTemperature	C	1s	-	-	46	49
Stage2-AirPressure	kPa	1s	-	-	-	-
Stage3-Vibration	μm	1s	-	-	22	27
Stage3-AirTemperature	C	1s	-	-	190	195
Stage3-AirPressure	kPa	1s	-	-	-	-
BlowOffValve-Position	%	1s	-	-	-	-
Oil-Pressure	kPa	1s	140	155	-	-
VentFanTemperature	C	1s	-	-	-	50
Motor-Current	A	1s	-	-	-	-
Capacity	m3/min	1s	-	-	-	-
PowerConsumption	kW	1s	-	-	-	-

6.1.1 Raw Data

The system has been continuously monitored by the sensors since first day of operation, while the storing and collection of the data began in October 2018. Hence, only sensor data from the end of October 2018 to the end of January 2019 were available for the analyses. The sensor data is sampled with a granularity of 60 seconds, implying that the sensor data from each feature is extracted every 60 second. Furthermore, the data is provided as raw, unsorted and unfiltered

csv files, where one csv file contains the sensor measurements for all five compressors in the system, for one hour. Having a granularity of sixty seconds, the amount of data quickly grows big. Each hour, a new csv file is generated containing all samples from all compressors and their associated features, implying that 672 separate csv files are stored each month. For three months, this adds up to 2016 separate csv files, requiring in total 60 000 MB of memory capacity. In order for the data to be valuable, it must be preprocessed (Chollet, 2017).

6.2 Data Preprocessing

Data preprocessing is one of the fundamental steps involved in data analytics, independent of the chosen modelling approach. Hence, in order to build the unsupervised anomaly detectors, the raw data needs to be filtered, sorted and down-scaled as much as possible, such that noisy data impacting the performance of the algorithm and long running times are reduced (Chollet, 2017) (Ketkar, 2017). As presented in Chapter 2, only one of the five compressors are to be considered, namely *Compressor 31*, such that all the samples from the other compressors are treated as irrelevant. Based on the raw data structure, scale and content, the following procedure was used to preprocess the data;

1. **Append files**: All the csv files were appended together in one big file for the entire time period, allowing for reorganization, filtering and sorting the continuous time series data for the entire period at once.
2. **Split columns and reorganize**: In the appended csv file, all the sensor measurements were stored in one single column, making it impossible to sort and filter the data. Hence, this column was split, such that each feature with associated time stamp was given a unique column.
3. **Filter**: Only data from the features belonging to *Compressor 31* was kept, while the rest was filtered out to reduce the size of the file.
4. **Sort**: The data was sorted according to time and date, required both for continuous time series visualization and for machine learning based on regression.
5. **Missing values**: The data was investigated to detect single instances or time frames of missing values that needed to be filled in or filtered out.
6. **Re-sample**: The data was re-sampled for each minute, such that the average value for each minute where calculated based on the samples from each second, in order to reduce the amount of data.

6.2.1 Libraries and Tools Used for Analysis

The two presented anomaly detectors with all associated analysis are programmed in Python version 3.7.1. The code for descriptive statistics, machine learning models and the two unsupervised anomaly detectors are presented in Appendix B. In addition to well known open source libraries, such as *NumPy* and *Matplotlib*, the following libraries were used to preprocess the data, perform descriptive statistics and train, validate and test the machine learning models;

- **Pandas**: is an open source library providing easy-to-use data structures and data analysis tools for Python (McKinney, 2010). The Pandas library provides functions simplifying the process of data preprocessing, by storing the big amounts of data in *Pandas Dataframes*. For Pandas Dataframes, easy implementable functions for splitting, sorting, filtering, resampling, visualization and descriptive statistics are available.
- **Seaborn**: is a Python data visualization library, which provides a high level interface for informative, statistical graphics (Waskom, 2019). For the problem at hand, the Seabron library was used to create the heatmap presented in section 6.3.4, demonstrating the correlations between the input features.
- **Scikit-Learn**: is an open source Python library providing efficient tools for data mining and data analysis, and inbuilt functions for machine learning classification, regression and clustering. In addition, Scikit-Learn offers inbuilt performance metrics for the various machine learning models, such as MAP, MAPE, MSE, RMSE and a scoring metrics returning the accuracy (Pedregosa et al., 2011). For the problem at hand, Scikit-Learn was used to train, validate and test all the presented machine learning models, along with their associated performance metrics.

6.2.2 Data Transformations During Preprocessing

In the following, the steps involved with the data preprocessing are demonstrated, emphasizing the main transformations made on the data. As stated by several authors, data preprocessing often takes up 80 % of the total time spent on a machine learning project, which is also demonstrated by the weight it is given in this report (Chollet, 2017) (Goodfellow, 2017).

Raw Data Structure

The raw csv-files containing measurements from one hour each lack the structure required for analyzes, as visualized in the dataframe below. As seen, both the date, time, compressor num-

ber, sensor name and value are stored in the same column, named *"timestamp|device|tag|value"*. In order to save time on splitting, reorganizing, filtering and sorting the raw csv files, they were all appended together, and stored in one dataframe.

	timestamp|device|tag|value
0	2019-01-01T00:00:00.0000000Z|Compressor 11|Loa...
1	2019-01-01T00:00:00.0000000Z|Compressor 11|Unl...
2	2019-01-01T00:00:00.0000000Z|Compressor 13|Set...
3	2019-01-01T00:00:00.0000000Z|Compressor 13|Cap...
4	2019-01-01T00:00:00.0000000Z|Compressor 14|Set...
5	2019-01-01T00:00:00.0000000Z|Compressor 15|Set...
6	2019-01-01T00:00:00.0000000Z|Compressor 31|Set...
7	2019-01-01T00:00:00.0000000Z|Compressor 14|Sta...
8	2019-01-01T00:00:00.0000000Z|Compressor 31|Oil...
9	2019-01-01T00:00:00.0000000Z|Compressor 14|Sta...

Splitting, Filtering, Reorganizing and Sorting

The main part of the data preprocessing consisted of splitting, reorganizing, filtering and sorting the appended raw-data file. First, the data was split, such that time, date, compressor number, sensor name and values were stored in separate columns. Secondly, the data steaming from the irrelevant compressors was filtered out, such that only data from *Compressor 31* was kept, giving the result demonstrated in the dataframe below.

	Compressor	Measure	Value	Date	Time
6	Compressor 31	SetPoint	746.529900	2019-01-01	00:00:00
8	Compressor 31	Oil_Pressure	233.783300	2019-01-01	00:00:00
28	Compressor 31	Stage2_AirTemperature	11.153866	2019-01-01	00:00:00
29	Compressor 31	Stage3_AirTemperature	49.597940	2019-01-01	00:00:00
39	Compressor 31	Stage1_AirPressure	2.898613	2019-01-01	00:00:00
42	Compressor 31	Motor_TemperaturePhase2	51.632523	2019-01-01	00:00:00
56	Compressor 31	System_AirPressure	749.968600	2019-01-01	00:00:00
65	Compressor 31	BlowOffValve_Position	100.000000	2019-01-01	00:00:00
67	Compressor 31	Stage1_Vibration	0.269890	2019-01-01	00:00:00
70	Compressor 31	Motor_TemperaturePhase3	51.449980	2019-01-01	00:00:00

Furthermore, the filtered dataframe was reorganized, such that the samples from each specific sensor were listed in the same column. In order to easily access the sample date and time, the *Datetime* was set as the index in the dataframe. By setting *Datetime* as index, Pandas enables re-sampling of data, making it possible to easily reduce the amount of data. Hence, a new re-organized dataframe was created, where specific sensor values at specific dates and times were easily accessible, as displayed in the dataframe below for three of the sensors.

Datetime	Stage1_AirPressure	Stage1_AirTemperature	Stage1_AirVibration
2019-01-01 00:00:00	2.898613	11.916216	0.269890
2019-01-01 00:00:01	2.898613	11.916216	0.269890
2019-01-01 00:00:02	2.898613	11.916216	0.269890
2019-01-01 00:00:03	2.927958	11.923943	0.265956
2019-01-01 00:00:04	2.959737	11.923578	0.274159
2019-01-01 00:00:05	2.939639	11.927032	0.270537
2019-01-01 00:00:06	2.927373	11.943260	0.271669
2019-01-01 00:00:07	2.882026	11.925222	0.276107
2019-01-01 00:00:08	3.038895	11.924143	0.274634
2019-01-01 00:00:09	2.937302	11.923943	0.272421

Detecting Missing Values

Having an evenly spaced time index, where each feature have a valid value for all instances of time, is required for further analysis of the time series data (Chollet, 2017) (Goodfellow, 2017). Instances of missing values can be handled by varies techniques, among them *forward filling*, *backward filling* or simply by deleting the features containing numerous missing values (Ketkar, 2017). Descriptive statistics is performed on all the features for the whole time period, to identify time instances or time intervals containing missing values. As illustrated in Table 6.2, the following features contained only zero values; *Capacity*, *InletValvePosition*, *PowerConsumption*, and *MotorCurrent*. Consequently, they were removed from the data set. In addition, all the sensor measurements from November and October equaled zero. This might either be due to the fact that the compressor was switched off during this period, or that the measurements not were collected and stored in the right manner. Due to this, only data from December and January were available for the analysis.

Table 6.2: Features with missing values

Statistics	Capacity	InletValve Position	PowerConsumption	InletValvePosition
Count	1440	1440	1440	1440
Mean	0	0	0	0
Std	0	0	0	0
Min	0	0	0	0
25 %	0	0	0	0
50 %	0	0	0	0
75 %	0	0	0	0
max %	0	0	0	0

Re-sampling

In order to reduce the size of the dataframe, re-sampling was performed, by calculating the average value for each minute. As displayed in the dataframe below, the *Datetime* index now has a granularity of 1 minute instead of 1 second. The re-sampled version reduced the size and memory required to process the data by 60 %, which highly influenced the time spent on training, validating and testing the machine learning models.

Datetime	Stage1_AirPressure	Stage1_AirTemperature	Stage1_AirVibration
2019-01-01 00:00:00	2.904993	11.922803	0.268253
2019-01-01 00:01:00	2.906916	11.927518	0.270891
2019-01-01 00:02:00	2.910251	11.926206	0.268822
2019-01-01 00:03:00	2.892509	11.926567	0.269541
2019-01-01 00:04:00	2.867792	11.932126	0.269693
2019-01-01 00:05:00	2.881382	11.926801	0.269652
2019-01-01 00:06:00	2.882400	11.927943	0.271043
2019-01-01 00:07:00	2.873145	11.927783	0.270391
2019-01-01 00:08:00	2.888669	11.934224	0.268263
2019-01-01 00:09:00	2.888479	11.947697	0.271433

6.3 Mapping Normal Operating Behavior

In order to build the unsupervised anomaly detectors, the machine learning models need to be trained and optimized on data characterizing normal operating behavior, as clearly emphasized in Chapter 5. Through data visualization, time periods with normal and stable operating behavior can easily be identified. In addition, by performing descriptive statistics on time periods with normal and stable operating behavior, a good indication of what to expects when the system is in good health can be obtained.

6.3.1 Data Visualization

In the following, the preprocessed time series data for *Air Pressure*, *Air Temperature* and *Vibration* are visualized. These features are a good indication of the overall system health, as explained in Chapter 2, and are therefore potential target variables.

Time Series Visualization for the Whole Period

In Figure 6.1 and Figure 6.2, the *Air Pressure*for all three stages are visualized, similarly for *Air Temperature* in Figure 6.3 and Figure 6.4 and for *Vibration* in Figure 6.5 and Figure 6.6.

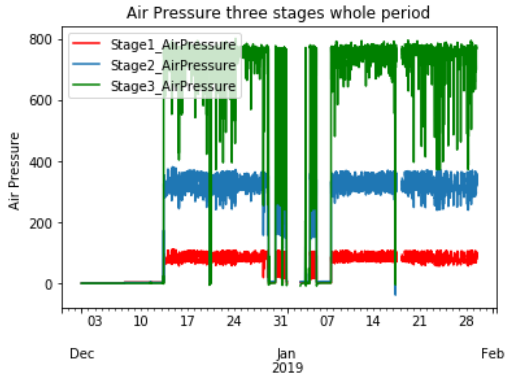


Figure 6.1: *Air Pressure* for all three stages from December and January.

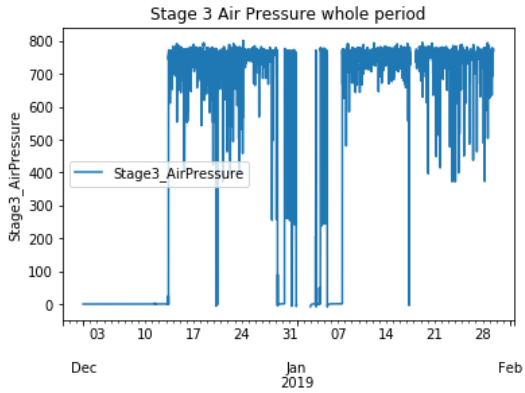


Figure 6.2: *Stage3-AirPressure* from December and January.

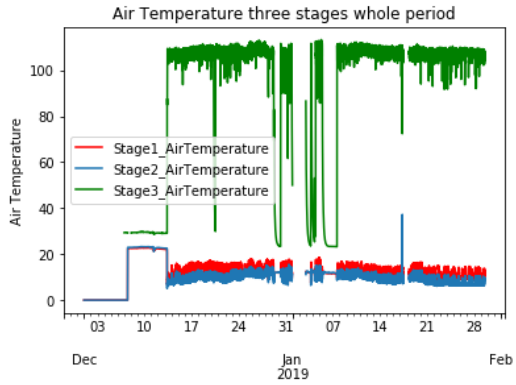


Figure 6.3: *Air Temperature* for all three stages from December and January.

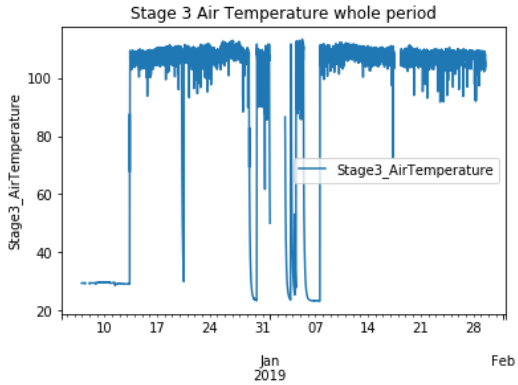


Figure 6.4: *Stage3-AirTemperature* from December and January.

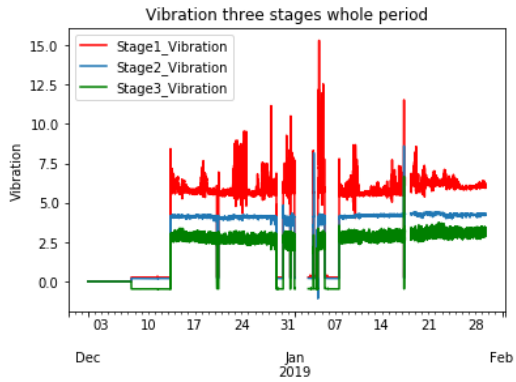


Figure 6.5: *Vibration* for all three stages from December and January.

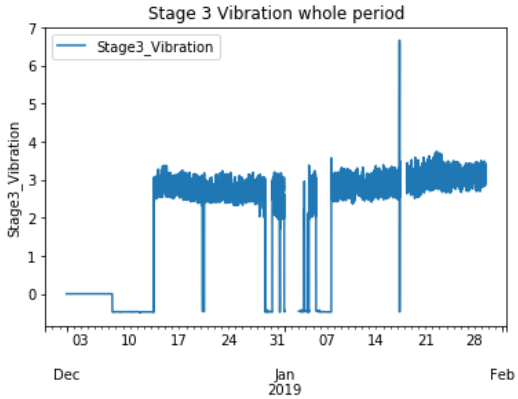


Figure 6.6: *Stage3-Vibration* from December and January.

By investigating the time series plot from beginning of December until the end of January, there

is clearly some stable and unstable time periods. From 14. - 30. December and 8.- 16. January, both the *Air Pressure*, *Air Temperature* and *Vibration* appear to be stable, while the other time periods seem to have irregularities. The irregular patterns could potentially be a result of the *load distribution principle* controlling the five compressors simultaneously. As explained in Chapter 2, this control system cause the compressors to be automatically switch off and on, such that the capacity is utilized to the full extent. However, due to lack of data, it is assumed that the identified stable periods characterize the normal operating behavior of the system. In order to get a deeper understanding of the characteristics of the normal operating behavior, the time period from 8. - 16. of January was investigated in more detail, both through visualization and through descriptive statistics.

Visualization of Normal Operating Behavior

In Figure 6.7 and Figure 6.8, the *Air Pressure* for all three stages characterizing normal operating behavior are visualized, similarly for *Air Temperature* in Figure 6.9 and Figure 6.10 and for *Vibration* in Figure 6.11 and Figure 6.12.

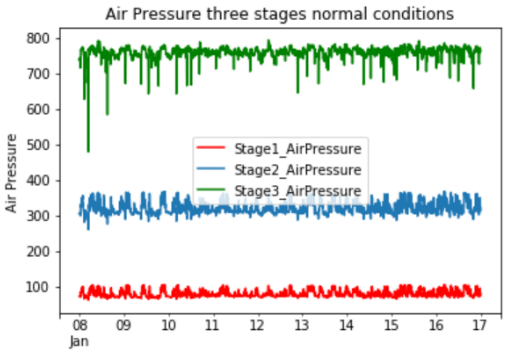


Figure 6.7: *Air Pressure* for all three stages characterizing normal behavior.

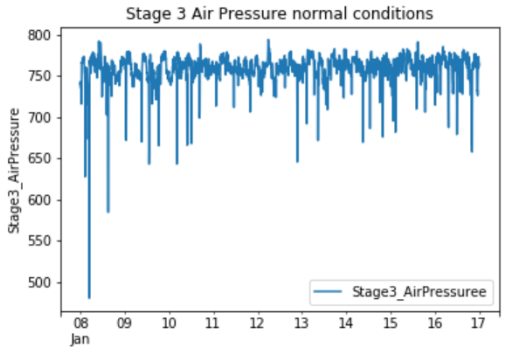


Figure 6.8: *Stage3-AirPressure* characterizing normal behavior.

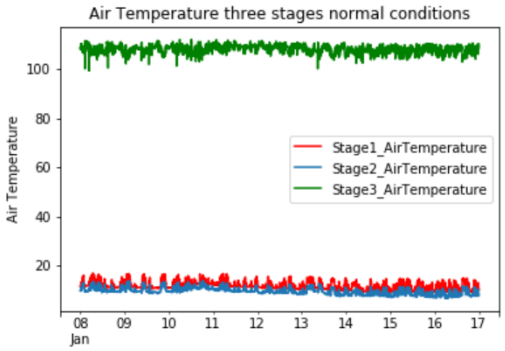


Figure 6.9: *Air Temperature* for all three stages characterizing normal behavior.

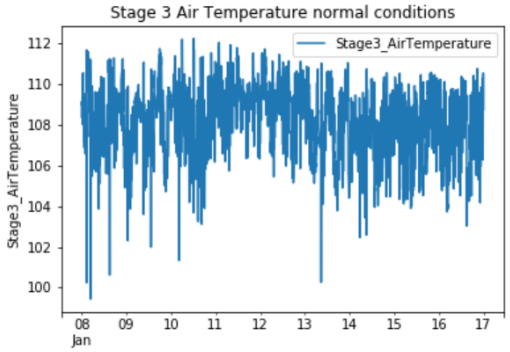


Figure 6.10: *Stage3-AirTemperature* characterizing normal behavior.

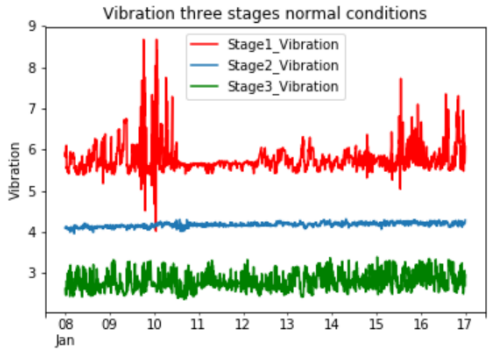


Figure 6.11: *Vibration* for all three stages characterizing normal behavior.

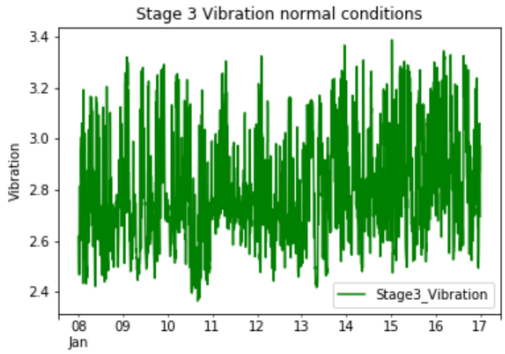


Figure 6.12: *Stage3-Vibration* characterizing normal behavior.

6.3.2 Time Series Data with Alarm Bounds

In addition, time series data for *Stage3-AirTemperature* and *Stage3-Vibration* with established alarm bounds are visualized, as seen in Figure 6.13 and Figure 6.14. Clearly, both the *High trip* and *High alarm* limits are set far from the normal operating behavior, making it hard to detect anomalies at an early stage, as explained in section 6.1.

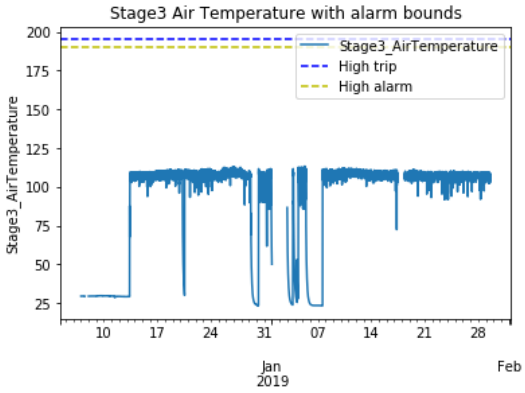


Figure 6.13: *Stage3-AirTemperature* from December and January with alarm bounds

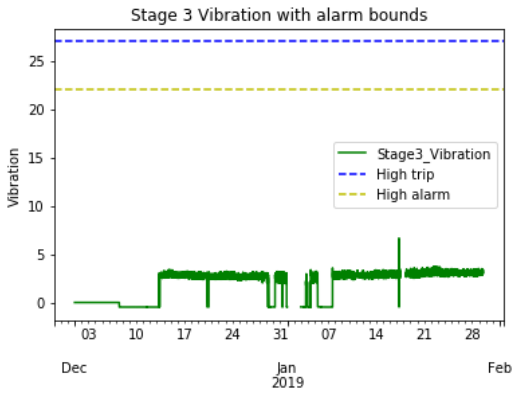


Figure 6.14: *Stage3-Vibration* from December and January with alarm bounds.

6.3.3 Descriptive Statistics

Descriptive statistics is generally known as brief descriptive coefficients summarizing a given data set, broken down to measures of variability and central tendencies. Measures of variabilities refers to measures such as standard deviations, variance, maximum and minimum sample, while measures of central tendencies refers to measures such as the mean and median of the samples (Goos, 2015). As presented in the dataframe below, descriptive statistics characterizing normal behavior for the *Stage3-AirPressure*, *Stage3-AirTemperature* and *Stage3-Vibration* are presented. These statistics are generated from the time period from 8. - 16. January. In addition, the histogram for each of these features characterizing the spread around the mean value are presented in Figure 6.15, Figure 6.16 and Figure 6.17.

	Stage3_AirPressure		Stage3_AirTemperature		Stage3_AirVibration
count	12961.000000	count	12961.000000	count	12961.000000
mean	758.982022	mean	108.251146	mean	2.794423
std	20.236454	std	2.117979	std	0.241546
min	371.251963	min	79.292979	min	2.317837
25%	752.976328	25%	106.959304	25%	2.612897
50%	760.328561	50%	108.746151	50%	2.737162
75%	768.434696	75%	109.744919	75%	2.953905
max	797.360843	max	113.058494	max	3.467410

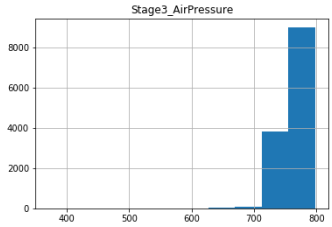


Figure 6.15: Histogram of the *Stage3-AirPressure* in normal operating behavior.

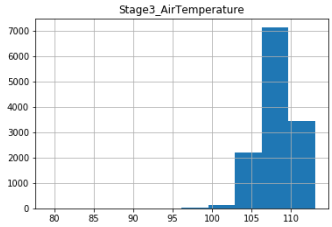


Figure 6.16: Histogram of the *Stage3-AirTemp* in normal operating behavior.

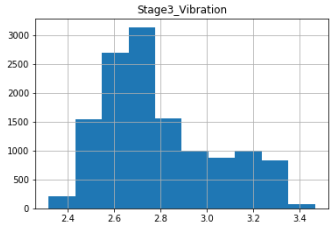


Figure 6.17: Histogram of the *Stage3-Vibration* in normal operating behavior.

As seen from both the time series visualization of the normal behavior in section 6.3.1 and the descriptive statistics presented in the dataframe, the selected features have a fairly low spread around the mean value. The *Stage3-AirPressure* has a standard deviation of 20,23, while the *Stage3-AirTemperature* and *Stage3-Vibration* have a standard deviation of 2,11 and 0,24 respectively. The selected period appears to have a stable operating behavior, and is therefore appropriate for modelling the normal behavior of the system.

6.3.4 Correlations Between Features

As explained in Chapter 4, machine learning regression builds on the principal of mapping the relationship between the target variable and the input features. Therefore, estimating the correlations between the features is of great interest, with the aim of understanding which features that will be used and weighted highly when the target variable is to be predicted.

Pearson Correlation Coefficient

Correlations between the features can be calculated in several ways, such as by the *Pearson*, *Spearman's* and *Kendall's Tau* Correlation Coefficient (Dalinina, 2017). The *Pearson Correlation Coefficient* is the most widely used coefficient, which measures the linear association between continuous variables. In other words, the degree to which the relationship between two continuous variables can be described by a line is quantified by the Pearsons Correlation Coefficient, denoted mathematically by;

$$p_{X,Y} = \frac{\sum (X_i - \overline{X})(Y_i - \overline{Y})}{\sqrt{\sum (X_i - \overline{X})^2 (Y_i - \overline{Y})^2}}, \tag{6.1}$$

where X and Y denotes two continuous variables and $\overline{X}$ and $\overline{y}$ denotes the mean value of these variables. As the Pearson Correlation Coefficient approaches 1, the more an increase in for example variable X will associate to an increase in variable Y. Hence, if the Pearsons Coefficient is close to 0, the variables are independent. However, the coefficient can be small even if the variables are strongly correlated (Dalinina, 2017).

In Figure 6.18, the correlations between all features are presented in a *heatmap*, ranging from -1 to 1, with an associated color to visualize the degree of correlation. These correlations are calculated based on the Pearson Correlation Coefficient, where a positive correlation implies that an increase in one variable associates to an increase in the other variable, while opposite for the negative correlations. In the Dataframes below, the *Top Positive Correlations* and the *Top Negative Correlations* between the features in normal operating behavior is presented.

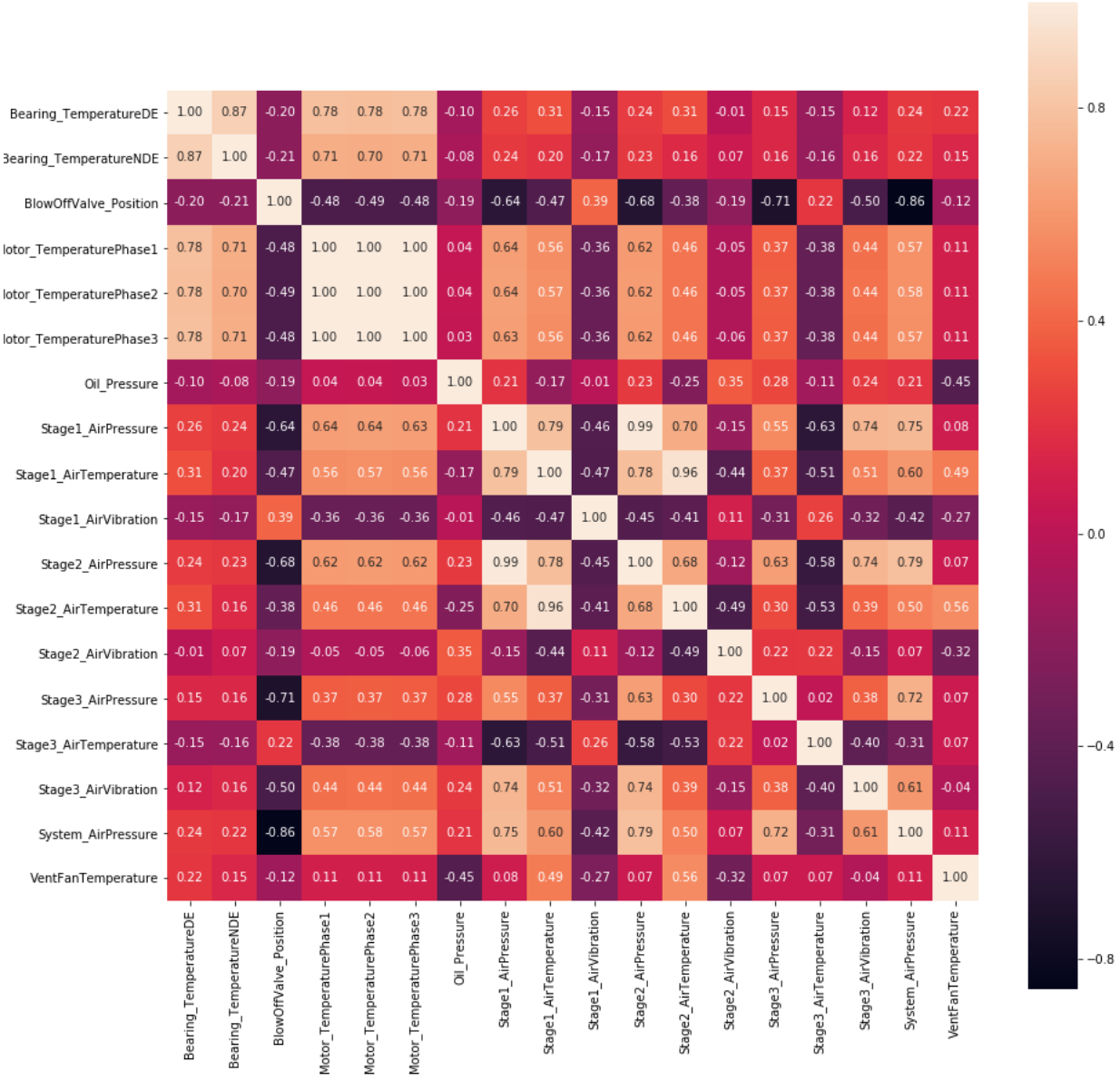


Figure 6.18: Heatmap ranging from -1 to 1, displaying the correlations between the features.

		Top Positive Correlations
Bearing_TemperatureDE	Bearing_TemperatureDE	1.000000
Motor_TemperaturePhase1	Motor_TemperaturePhase3	0.999866
	Motor_TemperaturePhase2	0.999794
Motor_TemperaturePhase3	Motor_TemperaturePhase2	0.999631
Stage1_AirPressure	Stage2_AirPressure	0.992661
Stage1_AirTemperature	Stage2_AirTemperature	0.955611
Bearing_TemperatureNDE	Bearing_TemperatureDE	0.866704
Stage1_AirPressure	Stage1_AirTemperature	0.794880
System_AirPressure	Stage2_AirPressure	0.788569
Bearing_TemperatureDE	Motor_TemperaturePhase3	0.784996
	Motor_TemperaturePhase1	0.782808
Stage2_AirPressure	Stage1_AirTemperature	0.777574
Motor_TemperaturePhase2	Bearing_TemperatureDE	0.775492
System_AirPressure	Stage1_AirPressure	0.753601
Stage2_AirPressure	Stage3_AirVibration	0.742864

		Top Negative Correlations
System_AirPressure	BlowOffValve_Position	-0.857139
Stage3_AirPressure	BlowOffValve_Position	-0.714155
BlowOffValve_Position	Stage2_AirPressure	-0.679315
	Stage1_AirPressure	-0.640748
Stage3_AirTemperature	Stage1_AirPressure	-0.633218
	Stage2_AirPressure	-0.584294
	Stage2_AirTemperature	-0.528234
Stage1_AirTemperature	Stage3_AirTemperature	-0.507298
BlowOffValve_Position	Stage3_AirVibration	-0.500736
Stage2_AirTemperature	Stage2_AirVibration	-0.494868
Motor_TemperaturePhase2	BlowOffValve_Position	-0.486654
BlowOffValve_Position	Motor_TemperaturePhase1	-0.483102
Motor_TemperaturePhase3	BlowOffValve_Position	-0.480565
Stage1_AirVibration	Stage1_AirTemperature	-0.472319
Stage1_AirTemperature	BlowOffValve_Position	-0.467950

Regression Line and Pearsons Correlation Coefficient

There is a clear connection between the slope of a line and the Pearsons Correlation Coefficient, such that the calculated correlation coefficients can be displayed through the slope of the line between the variables X and Y (Dalinina, 2017). In Figure 6.19, Figure 6.20, Figure 6.21, Figure 6.22, Figure 6.23 and Figure 6.24, the scatter plots between the *Stage3-AirPressure* and the *Stage1-AirTemperature*, *Stage2-AirTemperature*, *Stage3-AirTemperature*, *System-AirPressure* and *BlowOffValve-Position* are visualized.

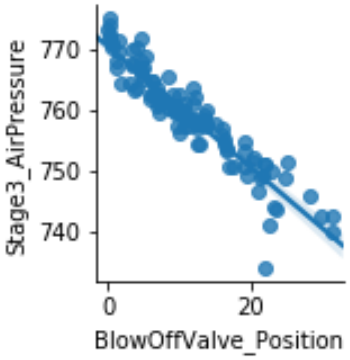


Figure 6.19: Correlation between *Stage3-AirPressure* and *BlowOffValve-Position*.

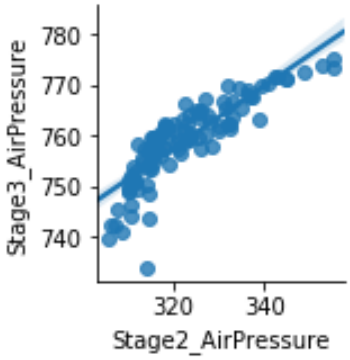


Figure 6.20: Correlation between *Stage3-AirPressure* and *Stage2-AirTemperature*.

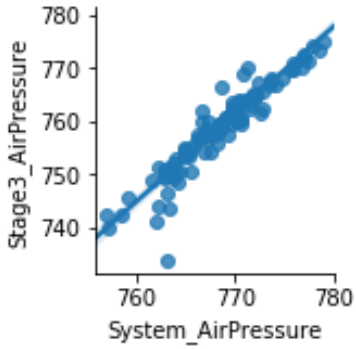


Figure 6.21: Correlation between *Stage3-AirPressure* and *System-AirPressure*.

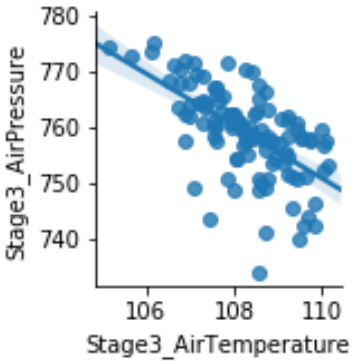


Figure 6.22: Correlation between *Stage3-AirPressure* and *Stage3-AirTemperature*.

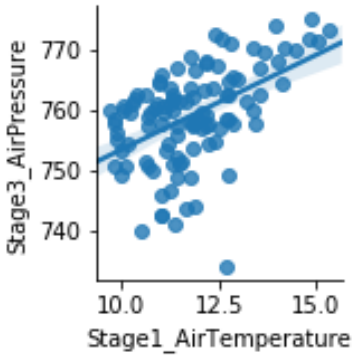


Figure 6.23: Correlation between *Stage3-AirPressure* and *Stage1-AirTemperature*.

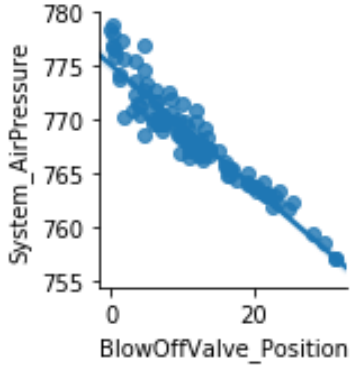


Figure 6.24: Correlation between *System-AirPressure* and *BlowOffValve-Position*.

As seen, the *Stage3-AirPressure* is strongly correlated with both the *Stage2-AirPressure* (and *Stage1-AirPressure*) and the System-AirPressure. This can logically be explained by the fact that for

each stage in the three-stage Centrifugal Air Compressor system, the air is compressed, implying that the *Stage3-AirPressure* strongly depends on the exit air pressure from stage 1 and 2. In addition, the air pressure is strongly correlated with the air temperature, which is explained mathematically by thermodynamic laws such as the *Ideal Gas Equation*, $pV = nRT$ (Giampaolo, 2010). Hence, as the pressure increases, the temperature will also increase. Furthermore, the level of openness in the *BlowOffValve* heavily affects the air pressure, argued by the fact that as the *BlowOffValve* opens, the compressed air is blown out over the roof, and the air pressure drops. This explains the negative correlation displayed in Figure 6.19 and Figure 6.24 between the *Stage3-AirPressure* and the *BlowOffValve-Position* and between the *System-AirPressure* and the *BlowOffValve-Position*.

6.4 Conclusion Normal Operating Behavior

From the analysis presented in this chapter, it can be concluded that the continuous time series data contains both stable and unstable time periods. The time periods from 14. - 30. December and 08. - 16. January were identified as stable, and assumed to characterize normal operating behavior. Through descriptive statistics on the data characterizing normal operating behavior, it was concluded that the standard deviations of the *Stage3-AirPressure*, *Stage3-AirTemperature* and *Stage3-Vibration* are fairly low. Furthermore, correlations between the features in normal operating behavior were presented, concluding that the *Stage3-AirPressure* is strongly correlated with both the air temperature, valve position and the system air pressure. This correlations were also presented in Chapter 2, assessed through system knowledge. The presented correlations give important guidelines to the relative importance of the features, which is highly valuable when the target variable is to be predicted. Based on the *Stage3-AirPressur's* strong correlations with the other sensors, and its ability to give an indication of the overall system health, the *Stage3-AirPressure* is chosen as the target variable to be predicted.

Chapter 7

Machine Learning Modelling

In the following chapter, the practical implementation of the *Decision Tree Model*, the *Random Forest Model* and a *Feedforward Neural Network* are presented. As explained in Chapter 4, these models are all capable of predicting a chosen target variable based on the input features, and once trained on a period with normal operating behavior, the predicted target variable can be used to build the anomaly detector based on residuals. The main focus in this chapter is to train, validate and test the three models, with the objective of building a model that performs well on *never-before-seen* data. The *Stage3-AirPressure* is chosen as the target variable to be predicted, while the remaining sensors are used as input features. Furthermore, the theoretical framework associated with the basic machine learning work flow was presented in Chapter 4, such that only the practical implementation will be presented here.

7.1 Creating Training, Validation and Testing Data Seta

In order to evaluate if the models perform well on *never-berfore-seen* data, a training set for training, a validation set for hyperparameter optimization and a testing set for evaluating the model's performance are established, as presented in Table 7.1. These sets are established based on the principal of *hold-out validation*, without random shuffling the data, which was explained in Chapter 4. The training set compromises approximately 60% of the data, while the data for validation and testing compromise approximately 20% each. There exists no such clear guide on the amount of data to use in each set, except that the process of training often requires more data than the process of validation and testing (Chollet, 2017). Most importantly, the training and validation sets are established based on time intervals where the *Stage3-AirPressure* appears to

follow normal operating behavior, which was identified in Chapter 6. Furthermore, the test set is allowed to contain more irregular and unstable patterns, in order to test if the models are capable of capturing also these. Both the Random Forest model, the Decision Tree model and the Feedforward Neural Network are trained, validated and tested on these established sets.

Table 7.1: Training, validation and testing data sets

	Training set	**Validation set**	**Test set**
Datetime	2019-01-08 00:00:00 - 2019-01-16 00:00:00	2018-12-14 00:00:00 - 2018-12-18 00:00:00	2019-01-22 00:00:00 - 2019-01-27 00:00:00
Percent	60 %	20 %	20%

7.2 The Decision Tree Model and The Random Forest Model

In the following, the hyperparameter optimization performed both on the Decision Tree model and the Random Forest model are presented, along with the model performance and feature importance for each of the trained models. As the Random Forest model compromise several independent decision trees, the two models are tightly connected and therefore also presented in parallel.

7.2.1 Hyperparameter Optimization for Decision Trees and Random Forest

The hyperparameters to optimize for one single decision tree include the maximum allowed depth of the tree, the minimum number of sample leaves, the minimum number of sample split, the maximum number of leaf nodes and maximum number of features that are evaluated for splitting at each node. For the Random Forest Model, the number of trees used to grow the forest and the maximum number of samples in each ensemble can be tuned in addition to the hyperparameters for one single decision tree (Géron, 2017). For decision trees, the *depth of the tree* appears to impact the performance significantly more than the tuning of the other hyperparameters, while for Random Forest, the *number of trees* have the highest impact on the performance (Géron, 2017). Evidently, these hyperparameters became the main focus during the optimization process. In Figure 7.1, the performance obtained on the validation set for the Decision Tree model with varying tree-depth is visualized. As seen, the best performance is obtained with **Max-depth = 8**, with Mean Absolute Error (MAE) equal to 2,38. For Random Forest,

the best performance on the validation set is obtained with **Number of trees = 25**, having Mean Absolute Error (MAE) equal to 2,18, as visualized in Figure 7.2. As emphasized in Chapter 4, the objective is to optimize the hyperparameters such that the effects of overfitting and underfitting is mitigated.

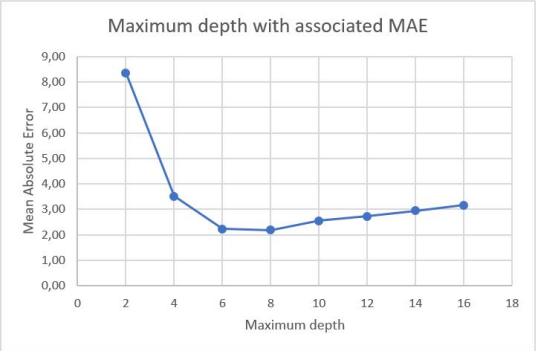


Figure 7.1: Optimizing the depth of the tree for the Decision Tree model

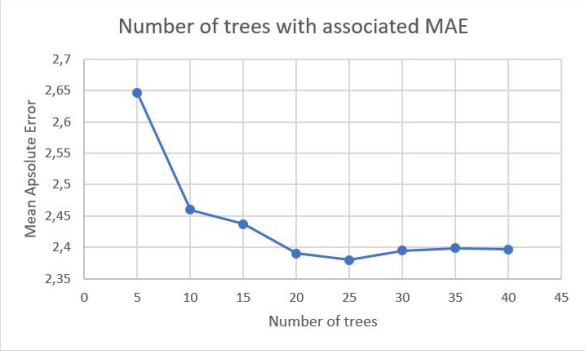


Figure 7.2: Optimize the number of trees for the Random Forest model

7.2.2 Model Performance - The Decision Tree and Random Forest Model

In order to evaluate the models performance, the Mean Absolute Error (MAE), the Mean Squared Error (MSE) and the Root Mean Squared Error (RMSE) are calculated based on the predicted target variable and the true target variable, which in this case is the *Stage3-AirPressure*. In addition, the *accuracy* is calculated based on an inbuilt score function in *Scikit-Learn*. In Table 7.2 and Table 7.3, the calculated performance metrics for the Decision Tree model, having a tree depth equal to 8, and the Random Forest model, with 25 trees, are presented.

Decision Tree Performance

Table 7.2: Performance of the Decision Tree model with **Max-depth = 8** on the validation and testing set

	Validation set	**Testing set**
Accuracy	0,982	0,976
Mean Absolute Error	2,38	3,58
Mean Squared Error	23,73	79,42
Root Mean Squared Error	4,87	8,91

Random Forest Performance

Table 7.3: Performance of the Random Forest model with **Number of trees = 25** on the validation and testing set

	Validation set	**Testing set**
Accuracy	0,984	0,980
Mean Absolute Error	2,18	3,49
Mean Squared Error	21,26	77,14
Root Mean Squared Error	4,61	8,78

For both models, the trained model performs better on the validation set than the testing set, which can be explained logically by the following; i) as the algorithm's hyperparameters were optimized based on the performance on the validation set, one would in general expect a higher performance on the validation set, due to the fact that information stored in the validation set leaks into the model during the optimization, ii) the validation set contains data belonging to normal operating behavior, while the testing set contains more irregular patterns that not were introduced during the training and optimization process. In cased where the gap between the validation and the testing error is too big, one risk to have overfitted the training and validation set, such that transformation only stored in these sets are weighted too high. Based on the obtained results, this seems not to be the case here, implying that the hyperparameters are appropriate for the problem at hand. Furthermore, the Random Forest model performs slightly

better than the Decision Tree model, which can be explained by the fact that the Random Forest model takes the average value of the predicted output of 25 trees, instead of trusting the output of one single decision tree.

Visualization of Predicted Target Variable - Random Forest model

In Figure 7.3 and Figure 7.4, the predicted *Stage3-AirPressure* by the Random Forest model and the actual *Stage3-AirPressure* are visualized. In addition, aligning and smoothing is performed on both the predicted and actual output, to reduce the noise associated with the data. As seen, the predicted pressure accomplishes to follow the actual measured pressure, which is also proven by the high performance given by the MAE, MSE and RMSE.

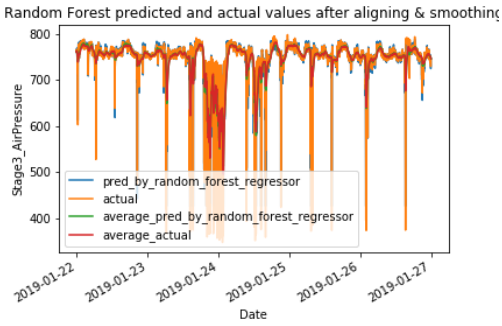


Figure 7.3: Predicted *Stage3-AirPressure* by the Random Forest model during the testing period.

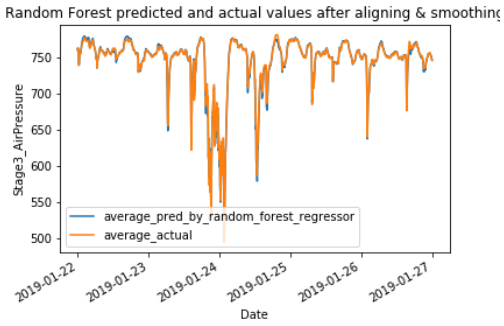


Figure 7.4: Predicted *Stage3-AirPressure* by the Random Forest model during the testing period.

7.2.3 Feature Importance Decision Tree and Random Forest model

In contradiction to statistical and physical models, machine learning models do not make any predefined assumptions about the relationship between the target variable and the input features. This implies that the weight given to each input feature in relation to the target variable not is given in advanced, nor is any direction on which features to use, as emphasized in Chapter 3. For Decision Tree models and Random Forest models, Scikit-Learn offers easy implementable functions to access the relative importance of each feature used when predicting the target variable. In Table 7.4, the features used to predict the *Stage3-AirPressure* are presented, along with their relative importance, which also are visualized in Figure 7.5. As seen, for both models, the *Stage2-AirPressure*, the *System-AirPressure* and the *BlowOffValve-Position* are the only features used to predict the *Stage3-AirPressure*. This result can be explained mathematically by the correlations calculated in Chapter 6, where the relationship between the *Stage3-AirPressure* and the

highly important features given here was close to linear. This information is extremely useful, serving two main purposed; i) enabling *feature engineering*, which refers to the concept of only letting the model train on input features that are known to add useful information, ii) deeper understanding the dynamics of the system, which is a requirement when diagnosis and actions are to be taken. For systems containing hundreds of sensors, feature engineering may be a pre-requisite, to reduce the scope and time used on training and testing (Chollet, 2017).

Table 7.4: Feature Importance for the Decision Tree and Random Forest model

Feature	**DT-importance**	**RF-importance**
Stage2-AirPressure	0,67	0,62
System-AirPressure	0,27	0,31
BlowOffValve-Position	0,06	0,06

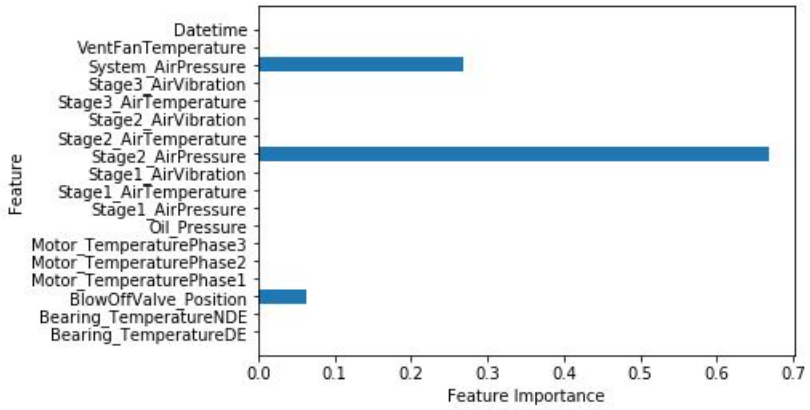


Figure 7.5: Feature importance for the Decision Tree and Random Forest model.

7.2.4 Visualizing Decision Tree

Finally, in order to illustrate how the *Stage3-AirPressure* is predicted, a decision tree with **max-depth = 3** is generated from the Decision Tree model, as seen in Figure 7.6. As for any decision tree, the feature having highest importance is located in the roof node, which in this case is the *Stage2-AirPressure*, with children nodes of lower importance, such as the *System-AirPressure* and the *BlowOffValve-Position*. Depending on the constraints presented in each node, the tree predicts the target variable based on the true or false statements associated with each constraint.

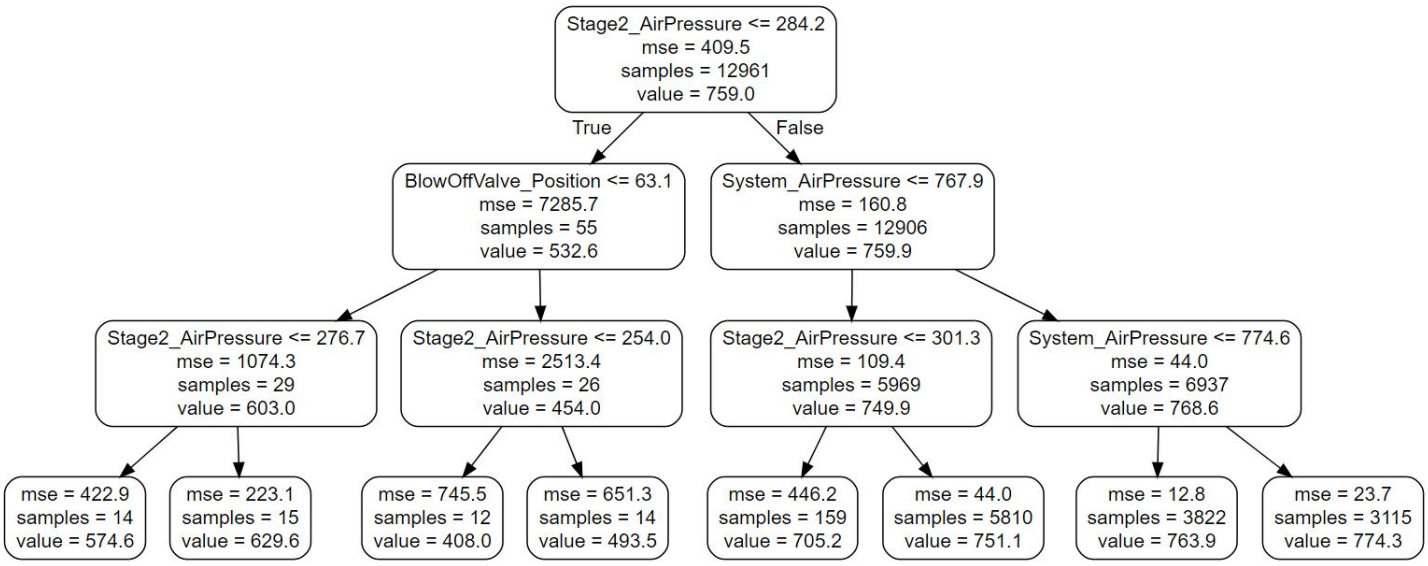


Figure 7.6: Visualization of a decision tree with **max-depth = 3**.

7.3 Feedforeward Neural Network

In the following, the *Feedforeward Neural Network* model is presented, including hyperparameter optimization, evaluating the model performance on both the validation and testing data set and visualizing the predicted output in relation to the actual measured output.

7.3.1 Hyperparameter Optimization

For Feedforward Neural Networks, the number of hidden layers, the number of neurons in each layer, and which activation function to use are among the most important hyperparameters to be optimized, as presented in Chapter 4. In contradiction to Decision Trees and Random Forest models, the optimization process is not straight forward, due to the fact that the combinations of hyperparameters to optimize approaches infinity. Hence, to most common approach is to test several models on the validation set, each having a unique combination of hidden layers, neurons and activation functions. As seen in Figure 7.7, 6 different models were evaluated on the validation set, where *Model 6* obtained the lowest MAE. This model had an input layer consisting of 4 neurons, one hidden layer consisting of 6 neurons and an output layer consisting of 4 neurons. The *ReLU* function was used as activation function, which is known to generally provide high performance, as emphasized in Chapter 4. The combinations of hyperparameters for the other models are presented in Appendix B.2.1.

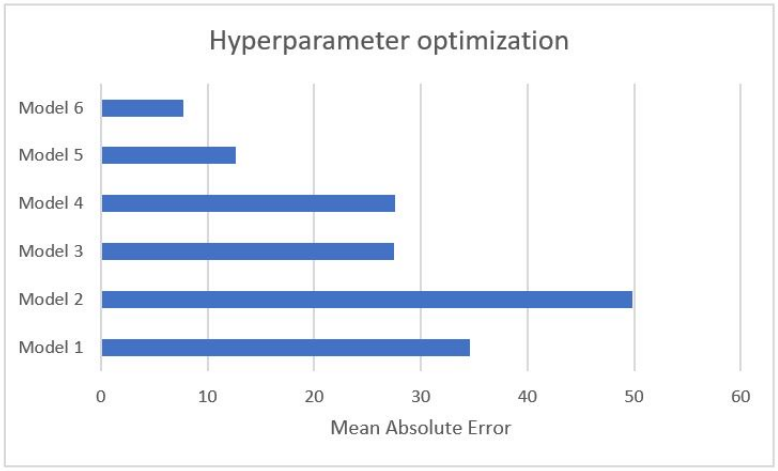


Figure 7.7: Hyperparameter optimization for the Feedforward Neural Network

7.3.2 Model performance - Feedforward Neural Network

Similarly as for the Decision Tree and Random Forest model, the MAE, MSE and RMSE are calculated to estimate the model's performance on the validation and testing set, along with the accuracy provided by Scikit-Learn. As seen in Table 7.5, the model's performance is higher on the validation set than on the testing set, which was explained to be logical in section 7.2.2. However, the gap between the validation and testing error is not as big as to rise suspicion about overfitting the training data. As seen, the Feedforward Neural Network struggles to predict the target variable, both on the validation and testing data set. This implies that the model is underfitting the data, and is struggling to capture the relationship it is attempting to learn. In order to reduce the problem of underfitting, other combinations of hyperparameters are required.

Table 7.5: Feedforward Neural Network, **Model 6**, performance on the validation and testing set

	Validation set	**Testing set**
Accuracy	0,54	0,42
Mean Absolute Error	7,73	13,70
Mean Squared Error	657,59	1956,23
Root Mean Squared Error	25,64	44,23

Visualization of Predictions - Feedforward Neural Network

As seen in Figure 7.8 and Figure 7.9, the Feedforward Neural Network struggles to predict the *Stage3-AirPressure*, as the actual value deviates from the predicted value. This is also highlighted through the calculated performance metrics presented in Table 7.5.

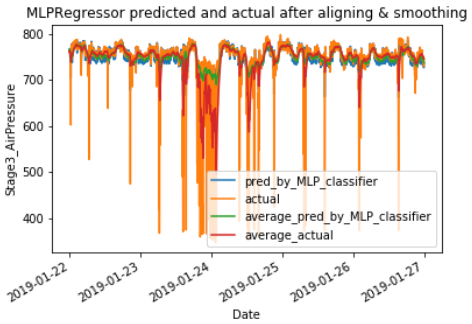


Figure 7.8: Predicted and actual *Stage3-AirPressure* by the Feedforward Neural Network during the testing period.

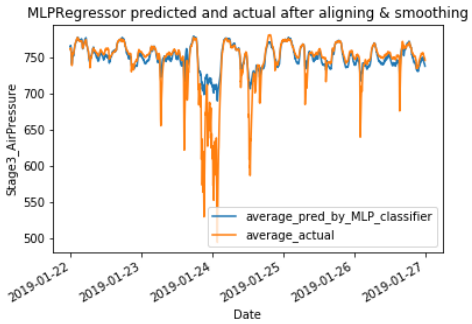


Figure 7.9: Predicted and actual *Stage3-AirPressure* by the Feedforward Neural Network during the testing period.

7.4 Model Benchmark

To conclude, to performance obtained on the validation and testing set for both the *Decision Tree model*, the *Random Forest model* and the *Feedforward Neural Network* is compared, as presented in Table 7.6 and Figure 7.10. As seen, the *Random Forest model* has the overall best performance, both on the validation set and the testing set, followed by the *Decision Tree model*. The *Feedforward Neural Network* struggles both on the validation and testing set, and suffers from underfitting the data. This can be explained by several reasons; i) the combinations of hyperparameters to tune in a Neural Network are higher, such that the model performance could have been improved if more combinations were tested, ii) Neural Networks are in general more sensitive to the scale of the data, and standardization and normalization of the data is often recommended (Chollet, 2017). Due to lack of time and competence, this was not performed. Based on the obtained results, the *Random Forest model* with 25 trees will be used to build the unsupervised anomaly detector based on residuals.

Table 7.6: Validation and testing set performance for the Decision Tree (DT), Random Forest (RF) and Feedforward Neural Network (MLP) model

	Validation set			**Testing set**		
	RF	**DT**	**MLP**	**RF**	**DT**	**MLP**
Accuracy	0,98	0,98	0,54	0,98	0,97	0,42
MAE	2,18	2,38	7,73	3,49	3,58	13,70
MSE	21,26	23,73	657,59	77,14	79,42	1956,23
RMSE	4,61	4,87	25,64	8,78	8,91	44,23

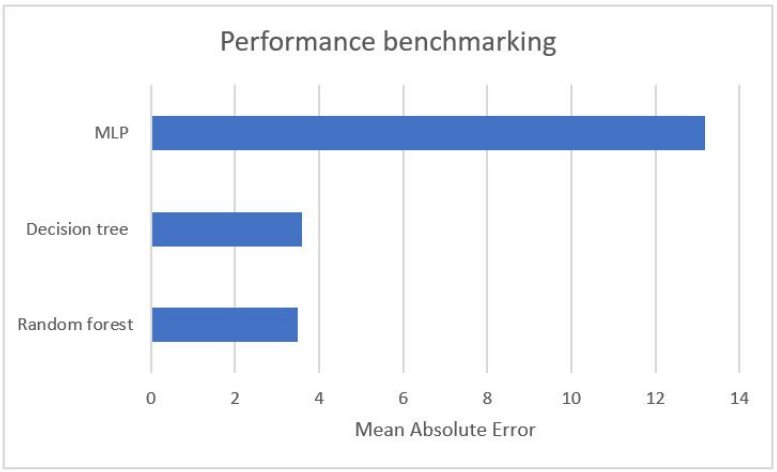


Figure 7.10: Mean Absolute Error obtained by the three models.

Comparison of the Predicted Target Variable

In Figure 7.11, Figure 7.12, Figure 7.13 and Figure 7.14, the predicted and average predicted *Stage3-AirPressure* generated by all three models are visualized, along with the actual and average actual *Stage3-AirPressure*. As seen, the Random Forest model manages to predict the *Stage3-AirPressure* with highest performance, followed by the Decision Tree model and the Feedforward Neural Network.

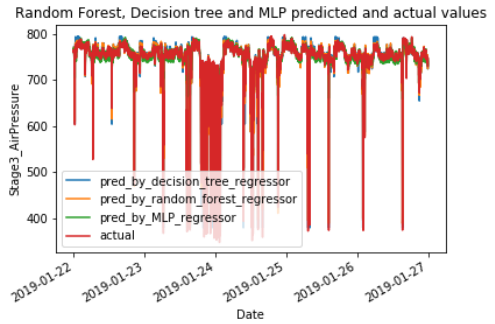


Figure 7.11: Predicted and actual *Stage3-AirPressure* by the three models for the testing period.

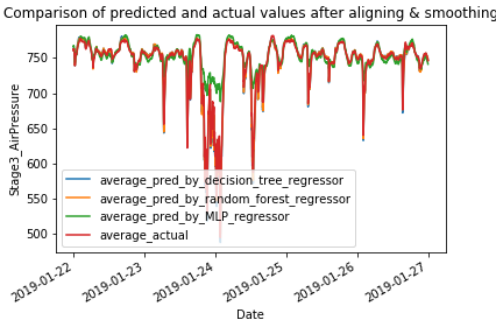


Figure 7.12: Predicted and actual *Stage3-AirPressure* by the three models for the testing period.

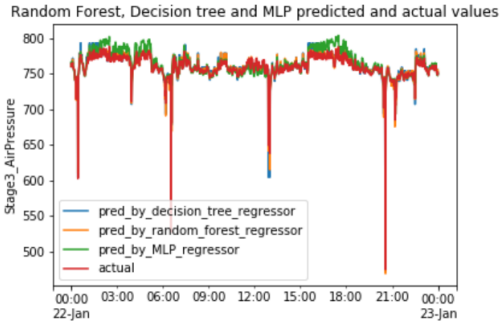


Figure 7.13: Close up one day, predicted and actual *Stage3-AirPressure* by the three models.

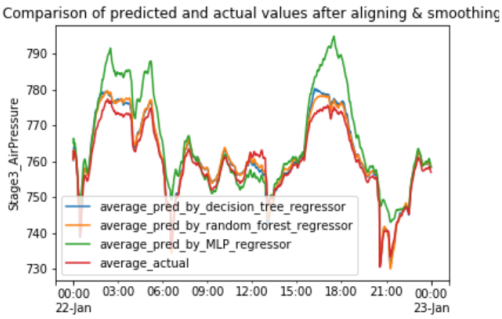


Figure 7.14: Close up one day, predicted and actual *Stage3-AirPressure* by the three models.

Chapter 8

Unsupervised Anomaly Detection Implementation

In this chapter, the *anomaly detector based on residuals* and the *anomaly detector based on clustering* are presented. The anomaly detectors are built based on the framework presented in Chapter 5 and relevant machine learning theory presented in Chapter 4, using the *Stage3-AirPressure* as target variable.

8.1 Anomaly Detection Based on Residuals

The procedure of building an unsupervised anomaly detector based on residuals can be summarized as follows; i) predict the target variable,*Stage3-AirPressure*, by a machine learning regression model, which is trained on normal operating behavior, ii) calculate the residuals between the predicted and the actual measured target variable, denoting the error of the predictions iii) establish confidence bounds for decision making, which balance the trade off between false positives and false negatives (Sanz-Bobi, 2016), (Chandola et al., 2009a). For a more detailed description of the steps involved, refer to Chapter 5.

8.1.1 Calculating the Residuals

The *Random Forest model* presented in Chapter 7 achieved the highest performance both on the validation and testing data set, and is therefore used to predict the *Stage3-AirPressure*. The residuals are calculated based on the following formula;

$$r_t = y_t - f_t, \tag{8.1}$$

where y_t denotes the actual value and f_t denotes the predicted value, which in this case is the actual and predicted *Stage3-AirPressure*.

The residuals denote the error between the predicted and actual *Stage3-AirPressure* for each instance of time. Calculated for the testing period from 22.- to 27. January, the results presented in Table 8.1 are obtained. As seen, the mean error is 0,41, which indicates that the predicted *Stage3-AirPressure* accomplishes to follow the actual *Stage3-AirPressure* in most instances of time. However, both the maximum value of the error, equal to 63,62, and the minimum value of the error, equal to −71,10, indicate that there are instances of time where the error is far from zero. In Figure 8.1 and Figure 8.2, the plot and the histogram of the residuals are visualized respectively. The residuals are assumed, after inspection of the histogram, to follow a normal distribution.

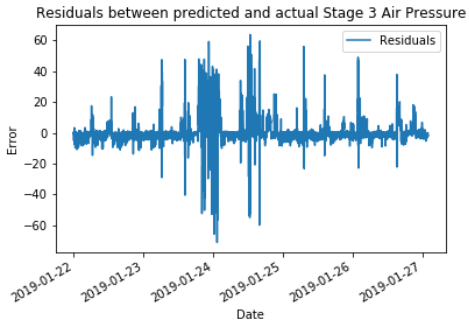


Figure 8.1: Calculated residuals between the actual and predicted *Stage3-AirPressure* during the testing period.

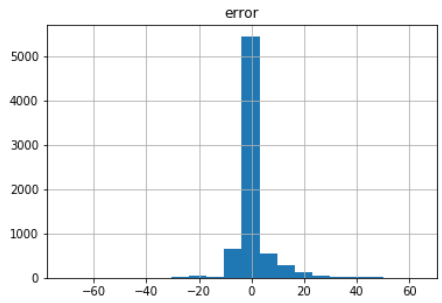


Figure 8.2: Histogram of the residuals between the actual and predicted *Stage3-AirPressure* during the testing period.

Table 8.1: Descriptive statistics on the residuals between the predicted and actual *Stage3-AirPressure*.

Descriptive statistics	**Value**
Count	7322,00
Mean	0,38
Std	7,59
Min	-71,10
25%	-2,23
50%	-0,69
75%	0,39
Max	63,63

8.1.2 Establishing Confidence Bounds for Decision Making

In order to determine if abnormal behavior is observed, an upper and lower control limit for the magnitude of the residuals need to be established, which is done by the principles of *Statistical Process Control*, as explained in Chapter 5. A sample is classified as abnormal if either the actual *Stage3-AirPressure* exceeds the *upper control limit* or if the actual *Stage3-AirPressure* is lower than the *lower control limit*. The upper control limit is established based on the predicted value plus the standard deviation of the residuals, multiplied by a chosen constant, while the lower control limit is established based on the predicted value minus the standard deviation of the residuals, multiplied by a constant. Mathematically, a sample is classified as abnormal if either;

$$y_t > f_t + x * \sigma, \tag{8.2}$$

$$y_t < f_t - x * \sigma, \tag{8.3}$$

where y_t denotes the actual value, f_t denotes the predicted value, σ denotes the standard deviation of the residuals and x denotes a constant.

Optimizing the Confidence Bounds

In order to minimize the risk of false positives and false negatives, the confidence bounds need to be optimized for the problem at hand. In Figure 8.3, the number of detected anomalies are plotted with respect to increased confidence bounds, ranging from an upper control limit at 1σ to 10σ and a lower control limit from -1σ to -10σ. As the confidence bounds are increased, the number of detected abnormal samples decrease. In addition, the residuals are visualized with upper and lower control limit equaling 3σ and -3σ respectively. Any samples crossing these control limits are then classified as anomalous. This can also be illustrated by plotting the predicted and actual $Stage3 - AirPressure$ with confidence bounds, as seen in Figure 8.5, where the green area denotes the confidence are. In this case, anomalies are detected when the actual $Stage3 - AirPressure$ starts moving outside the green confidence area.

Due to lack of historical data containing failures and anomalies, checking the occurrence of false positives and false negatives for difference confidence bounds is impossible. Therefore, it is assumed that the big spics exceeding the upper and lower control at 3σ are true anomalies that need to be detected. For an upper and lower control limit at 3σ, it can be read from Figure 8.3 that approximately 180 abnormal samples are detected during the testing period, out of in total 7322 samples. As seen in Figure 8.4, many of the abnormal samples occur in the same time interval, such that they most probably are caused by the same phenomena. In reality, these spics may be a result of the *load distribution principle*, causing irregular patterns that were excluded from the training data. In addition, the individual spices exceeding the control limits at the end of the testing interval can be a result of the phenomena of *lagging* between the sensors measurements. Lagging may occur when there is a delay between the measurements of correlated sensors, such that when for example the $Stage1 - AirPressure$ decreases, it takes some time before the $Stage3 - AirPressure$ sensor measures the decrease, causing deviations between the predicted and the actual $Stage3 - AirPressure$. Hence, more attention should be payed to samples that are continuously exceeding the control limits for several minutes or hours. However, these spics are treated as anomalies to demonstrate how an anomaly detector can be built based on residuals.

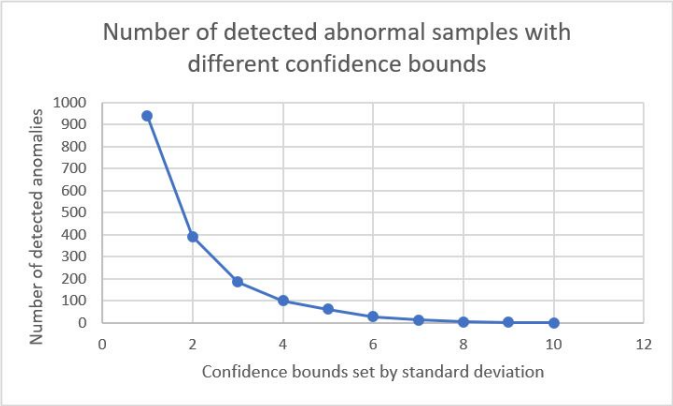


Figure 8.3: Number of detected abnormal samples during the testing period with different confidence bounds.

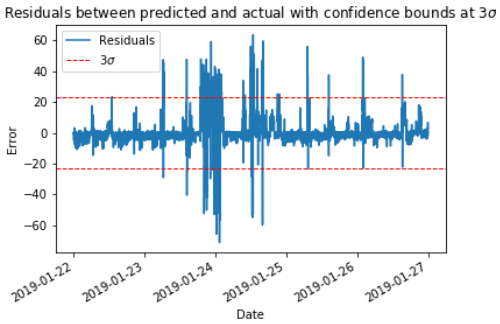


Figure 8.4: Residuals between the predicted and actual *Stage3-AirPressure* with confidence bounds at 3σ.

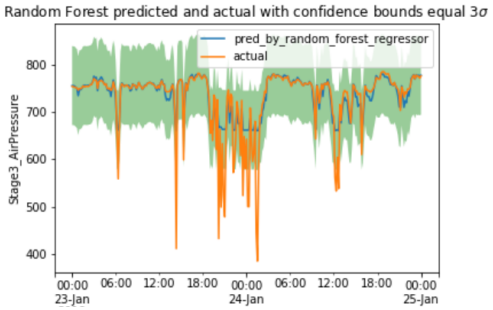


Figure 8.5: Predicted and actual *Stage3-AirPressure* with confidence bounds at 3σ.

8.2 Anomaly Detection Based on Clustering

The procedure of building an anomaly detector based on *K-means clustering* can be summarized as follows; i) determine the optimal number K clusters needed to capture the variance in the target variable, ii) create the *reference pattern* by fitting the K-means clustering algorithm to the training set characterizing normal operating behavior, iii) Fit the K-means algorithm to the testing set to create the *measured pattern*, and compute the distance between the reference and measured pattern, iv) decide on an *outlier fraction* that minimizes the risk of false positives and false negatives. For a more detailed description of the steps involved, refer to Chapter 5. For clustering based procedures, the data belonging the target variable itself is the main focus, yet other features can be included in the data set to assist describing the variance observed in the target variable. Hence, in addition to the target, $Stage3-AirPressure$, the

$Stage2 - AirPressure$ is included in the data set, argued by the fact that these features are strongly correlated, as proven in Chapter 6. Despite this, the time frame for training and testing is the same as established in Chapter 6.

8.2.1 Establishing the Reference Pattern

In order to establish the *reference pattern*, the number clusters K needs to be determined, which should be sufficiently high to capture the variance in the target variable, yet not higher than necessary Wu (2012). In order to determine the optimal number of K clusters, the K-means algorithm is fitted to the training data describing normal operating behavior, with K ranging from $1-20$. For each fitting, a score is calculated, indicating at which level the algorithm accomplished to assign all sampled in the training set to an established cluster. This optimization procedure is visualized by the *Elbow Curve*, describing the score obtained by the K-means algorithm for different clusters, as seen in Figure 8.6. From the Elbow curve, is is evident that the graph levels off after approximately 8 clusters, which implies that any additional cluster would not contribute further to describe the variance in the $Stage3 - AirPressure$. The K-means algorithm with 8 clusters is fitted to the training set containing data from both the $Stage3 - AirPressure$ and the $Stage2 - AirPressure$, as seen in Figure 8.7, where each color denotes a specific cluster. These clusters forms the *reference pattern*. The data here is scaled and normalized, explaining why the axis values in Figure 8.7 deviates from the true measured pressure values.

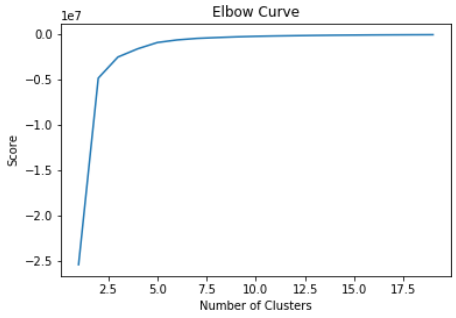


Figure 8.6: The Elbow Curve indicating the score for different clusters.

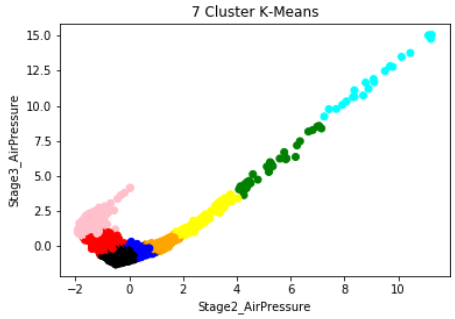


Figure 8.7: Visualization of the 8 clusters forming the *reference pattern*

8.2.2 Establishing the Measured Pattern

In order to detect anomalies, the K-means algorithm with 8 clusters characterizing the normal operating behavior is used as a reference pattern. The algorithm is fitted to the *never-before-*

seen samples in the testing set, where samples classified as normal will belong to any of the established 8 clusters, while samples classified as abnormal not will belong to any of the clusters. In order to determine if a sample can be assign to any of the clusters, the distances between each sample and its nearest cluster centroid is calculated. Based on a predefined $outlier\,fraction$, which needs to be optimized, the samples with the biggest distances to a centroid is classified as abnormal.

Optimizing the Outlier Fraction

The *outlier fraction* determines the percentage of samples that are classified as abnormal, among the samples with biggest distance to their closets centroid (Chandola et al., 2009a). As seen in Figure 8.8, the number of detected anomalies increase linearly as the outlier fraction increase, which logically can be explained by the percentage increase in the fraction of distances that are classified as cluster outliers. The outlier fraction needs to be determined such that the risk of false positives and false negatives are minimized, which also here requires historical data containing known failures and anomalies.

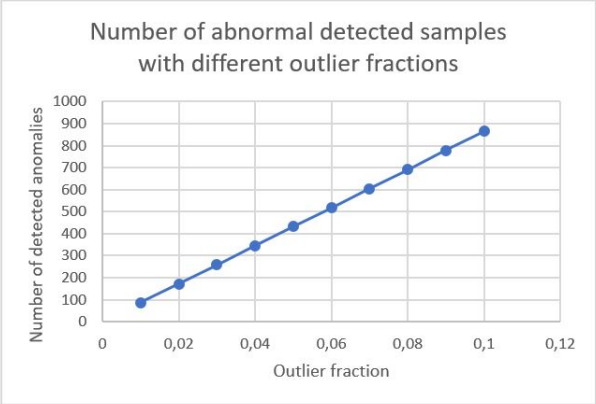


Figure 8.8: Number of detected abnormal samples during the test period with different outlier fractions

To illustrate the principle, the time series plot of the *Stage3-AirPressure* with detected anomalies for an *outlier fraction = 0,02* and *outlier fraction = 0,04* are visualized, as seen in Figure 8.9 and Figure 8.11. In addition, the histogram for each case representing the *Stage3-AirPressure* spread is visualized in Figure 8.10 and Figure 8.12, along with the samples classified as abnormal. Through visual inspection, it becomes evident that the big spikes are classified as anomalous, to which extent, depends on the predefined outlier fraction. For an *outlier fraction = 0,02*,

approximately 180 samples out of 7322 samples in total are classified as abnormal, while for an *outlier fraction = 0,04*, approximately 350 anomalies are detected.

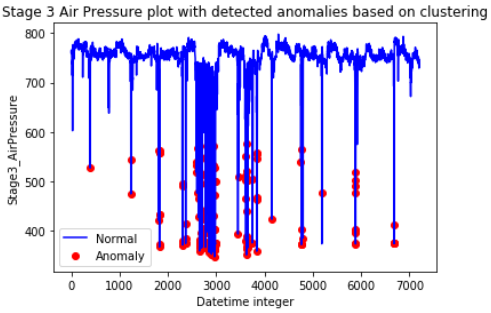


Figure 8.9: *Stage3-AirPressure* plot with detected anomalies with *outlier fraction=0,02*

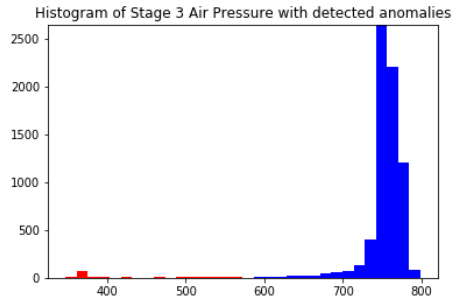


Figure 8.10: Histogram of *Stage3-AirPressure* with detected anomalies with *outlier fraction=0,02*.

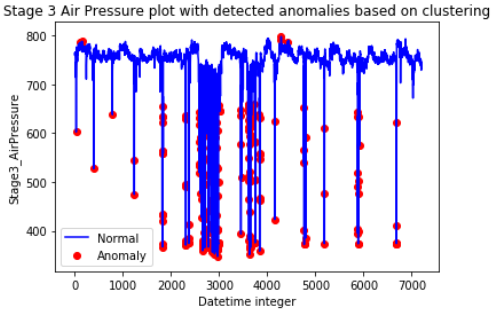


Figure 8.11: *Stage3-AirPressure* plot with detected anomalies with *outlier fraction=0,04*.

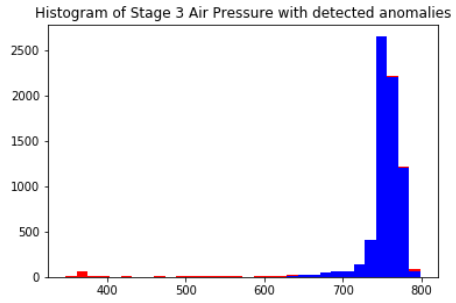


Figure 8.12: Histogram of *Stage3-AirPressure* with detected anomalies with *outlier fraction=0,04*.

8.3 Comparison of the Implemented Anomaly Detectors

For the anomaly detector based on clustering, an outlier fraction at 0,02 gives the result of 180 detected abnormal samples. Similar results were obtained for the anomaly detector based on residuals, with confidence bounds equaling 3σ, giving 180 detected abnormal samples, out of in total 7322 samples. By visual inspection of the residuals plot with confidence bounds in section 8.1.2, and the time series plot with detected anomalies in section 8.2.2, approximately the same samples are classified as anomalous by the two detectors. As emphasized in Chapter 2, *Low System Air Pressure*, and evidently deviations in the *Stage3-AirPressure*, may have the the following *root cause*; incorrect valve calibrations, valve malfunctions, air leakages, degraded or

damaged impellers, incorrect control calibration or high temperature air damaging the equipment. In relation to the fields of RAMS and Health Management, being familiar with such root causes is promoted as highly important, enabling efficient diagnosis and scheduling of predictive maintenance. However, the the big spics may be caused by the *load distribution principle* or be a result of *lagging* between the sensor measurements, as explained in section 8.1.2. In order to minimize the risk of false positives and false negatives, expert judgment or data including known anomalies should be considered. By only establishing the decision rules based on results revealed through modeling of what is assumed to be normal, one face the risk of establishing decision boundaries that are misleading.

For the Centrifugal Air Compressor system operating continuously during the year, *seasonal* and *cyclic* variations, and possible *drifts* may impact the data. For the presented anomaly detectors, only measurements from December and January were considered, which not may be representative of the spring, summer and fall season. Outside temperature and air humidity are examples of factors that may cause seasonal variations, whereas cyclic variations may be caused by economical variations impacting the production demand of compressed air. Furthermore, possible *drifts* may cause the normal operating characteristics to change with time. Environmental changes increasing the outdoor temperature permanently and permanently changes in the air composition are example of factors causing drifts. The air cooling system and the inlet filter are examples of sub-systems that are directly impacted these possible seasonal variations and drifts, which again impacts the other sub-systems of the compressor system as whole. As the characteristics of normal operating behavior change, the decision boundaries need to be updated, most preferably in an adaptive manner, reducing the risk of false positives and false negatives. According to the *Numenta Benchmark Requirements*, as presented in Chapter 3, the algorithm must have the ability to continuously learn in an adaptive manner, to capture seasonal and cyclic variations, and possible drifts (Lavin and Ahmad, 2015). The residuals based approach is better suited to handle such variations than the clustering based approach, which is limited by the number of features it considers, as emphasized in Chapter 5. In addition, the machine learning models used for the residuals based approach can accomplish to learn all this variations, whereas several reference patterns may need to be crated to capture the variations for the clustering based approach (Lavin and Ahmad, 2015) (Sanz-Bobi, 2016). However, for both the presented anomaly detectors, manual tuning may be required to optimize the decision boundaries if drifts occur, requiring expert and system knowledge.

Chapter 9

Conclusions, Discussion, and Recommendations for Further Work

In this final chapter, a summery and conclusion is provided, along with a discussion of the presented findings. In addition, recommendations for further work are provided.

9.1 Summary and Conclusions

The main objective for this master thesis was to build an unsupervised anomaly detector based on machine learning, capable of detecting abnormal behavior in continuous time series data, not including labeling or failure history. The system under study was a three-stage Centrifugal Air Compressor system, which is continuously monitored by 21 sensors. In order to reach the objective, several approaches to unsupervised anomaly detection were reviewed, concluding that anomalies can be detected either by calculating the residuals between the predicted target variable and the actual measured target variable, or by comparing new samples to an established reference pattern, generated through clustering. For both the approaches, it is a requirement that the machine learning model is trained and optimized on data characterizing normal operating behavior, such that any new sample deviating from this modeled behavior is classified as anomalous. The framework on how to build an anomaly detector for each of these approaches is presented in Chapter 5, which is used as a foundation for the practical implementation presented in Chapter 8. In contradiction to physical and statistical models, machine learning models do not not make any predefined assumptions about the relationship between the variables in the system, which for complex systems, may enable the incorporation of all the

variables affecting the dynamics of the system into the model. However, machine learning models may suffer from both overfitting and underfitting, requiring the models to be trained, tested and validated on three different data sets, as emphasized in the machine learning framework presented in Chapter 4. The main objective is to build a model that generalize well, referring to the model's ability to perform well on *never-before-seen* data, where hyperparameter optimization plays an essential role.

Three state-of-the art supervised machine learning models were reviewed and implemented on the compressor data, namely the *Decision Tree model*, the *Random Forest model* and a *Feedforward Neural Network*, along with the unsupervised machine learning model *K-Means clustering*, which all were presented in Chapter 4. In order to predict the target variable, *Stage3-AirPressure*, each model were trained, validated and tested based on the data sets established in Chapter 6, after visual inspection of the data at hand and the results from descriptive statistics. The performance benchmark presented in Chapter 7 revealed that the Random Forest model predicts the *Stage3-AirPressure* with highest performance, having an accuracy equal to 0,98, followed by the Decision Tree model and Feedforward Neural Network. Arguably, the Random Forest model was used to build the anomaly detector based on residuals. Using an upper control limit and lower control limit at respectively 3σ and -3σ, in total 180 samples out of 7322 samples were classified as anomalous. Similar results where obtained by the anomaly detector based on clustering, using an outlier fraction at 0,02. By comparing the two anomaly detectors, is it evident that both the detectors classify the large irregular patterns in the middle of the test period as anomalous, similar yields for the big spics at the end of the test period. As emphasized in Chapter 8, this can either be caused by the *load distribution principle* controlling the compressors, the phenomena of *lagging* between the sensor measurements or detection of true anomalies. In order to minimize the risk of false positives and false negatives, these decision boundaries for detection need to be optimized based on expert judgment or failure history containing known anomalies, which raise the need for converging the fields of Health Management and Artificial Intelligence.

According to the *Numenta Benchmark Requirements*, it was concluded that an unsupervised anomaly detector based on residuals is more capable of learning in an adaptive manner, capturing possible seasonal and cyclic variations, than the clustering based approach. However, both the presented unsupervised anomaly detectors accomplish to detect anomalies and at an early stage, without relying on labels or failure history, such that it can be concluded that the main objective with associated sub-objectives for this master thesis are met.

9.2 Discussion

The results from the descriptive statistics revealed that the *Stage3-AirPressure* is strongly correlated with both the *System-AirPressure*, *Stage3-AirTemperature*, *Stage2-AirTemperature*, *Stage2-AirPressure* and *BlowOffValve-Position*, which also was visualized by the linear regression line between the *Stage3-AirPressure* and these sensors in Chapter 6. Based on this, it can be discussed if a simple linear regression model could have been applied to model the normal behavior of the system, instead of a machine learning model. Similarly, a physical model, such as the ideal gas equation, $pV = nRT$, would most probability manage to capture the deviations seen in the *Stage3-AirPressure* if the temperature deviates. However, as highlighted in chapter 3, the main limitation of statistical and physical models, is that it is hard to determine if all the variables affecting the target variable actually are incorporated into the model. Establishing a regression model that incorporates all the 21 sensors are impractical, such that assumptions about which variables affecting the target variable the most would have been necessary. However, the feature importance presented from the Random Forest model and the Decision Tree model in Chapter 7 indicates that only these highly correlated sensors were used to predict the the *Stage3-AirPressure*. Hence, for the three-stage Centrifugal Air Compressor system, a statistical regression model including only these correlated sensors could have managed to capture the same dynamics as a machine learning model.

Form a Health Management and RAMS perspective, assessing potential failure modes and correlations based on system knowledge are in focus. Through the presented descriptive statistics, and based on the feature importance, the correlations and relationships between the system components were also revealed. From a data mining perspective, where the failure modes and internal system relationships not are assessed in advanced, it is questionable why such detailed examination of the system actually needs to be carried out. One could argue that the data analytics speaks for itself, reducing the need to truly understand the system in advance. However, by combining the two approaches, hypothesis may be proven twice, increasing the trustworthiness of the results. In addition, by only considering the data at hand, correlations between features could be found, which in the real world not are true correlations. Examples of such phenomena could be that the price in the stock market increases, evidently at the same time as birds return home after spending time in warmer countries. By only considering the results gained through data analytics, one could have been concluding that the stock price increases as the amount of birds increases, which would be a big mistake. This example strengthens the belief that an interdisciplinary collaboration between the fields of Health Management and IT is required, to maximizing the value that possible can be created through data analytics.

The same theory also yields for optimizing the decision rules used to classify samples as abnormal or normal. If no failure history is available, expert knowledge is required to determine if the algorithm actually manages to detect true anomalies. By only relying on the results believed to be normal from machine learning, the risk of misplacing the decision boundaries increases, which again increases the risk of false positives and false negatives.

9.3 Recommendations for Further Work

In the following, possible extensions and recommendations for future work are presented, with the aim of providing some guidance for further analysis and topics that needs to be addressed for successfully implementing an unsupervised anomaly detector.

- In order to develop an unsupervised anomaly detector capable of detecting anomalies in real time during the whole year, the machine learning model needs to be trained and validated on data for a longer time period. Seasonal and cyclic variations, and possible drifts, most probably impacts the normal operating behavior of the system, which needs to be incorporated into the model for online, real time detection.
- For Neural Networks, hyperparameter optimization is a complex task, due to the number of possible hyperparameter combinations. In order to improve the performance of the Feedforward Neural Network presented, more attention should be payed to the optimization process. In addition, Neural Networks with back-propagation can be considered, such as Recurrent Neural Networks, which assists the optimization process by giving indications on how the weights should be adjusted to obtain high performance.
- In order to establish decision rules that minimize the risk of false alarms and false positives, expert judgment is required. In addition, the anomaly detector should run in parallel with the system while operating, such that the decision rules for determining if a sample actually is abnormal can be tuned.
- Taking into account that only one out of five possible compressors was considered, an anomaly detector analyzing the data from all the compressors should be considered. The load distribution principle causes irregular patterns, such that by considering all the compressors simultaneously, one can verify or reject if the irregular patterns are caused by the load distribution principle, or if actual abnormal behavior is observed.

Appendix A

Acronyms

RUL Remaining useful lifetime

MTTF Mean time to failure

ETTF Estimation of time to failure

ML Machine Learning

AI Artificial Intelligence

RF Random Forest

DT Decision Tree

NN Neural Network

MLP Multilayer Perceptron

MFE Mean Forecast Error

MAE Mean Absolute Error

MAPE Mean Absolute Percentage Error

MSE Mean Square Error

RMSE Root Mean Square Error

Appendix B

Listings of Codes for the Presented Analysis

B.1 Descriptive Statistics on Normal Operating Behavior

Descriptive statistics on normal operating behavior

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

Data = pd.read_csv('JanuarDesember-sorted-appended.csv', index_col = 0, parse_dates=True
    ↪ )

x_col=Data.columns.values

#Timeinterval - Normal operating behavior
time_start_date = '2019-01-08 00:00:00'
time_end_date = '2019-01-16 00:00:00'

df=Data.loc[time_start_date:time_end_date,:]

#Calculate the correlations between all the features
Dataframecorr=pd.DataFrame(data=df.corr().unstack().sort_values().drop_duplicates())
```

```
Dataframecorr['Correlation']=Dataframecorr[0]
Dataframecorr=Dataframecorr.drop([0], axis=1)

#Create dataframes with top positive and top negative correlations
Topneg=Dataframecorr.head(15)
Topneg['Top␣Negative␣Correlations']=Topneg['Correlation']
Topneg=Topneg.drop(['Correlation'], axis=1)

Toppos=Dataframecorr.tail(15)
Toppos['Top␣Positive␣Correlations']=Toppos['Correlation']
Toppos=Toppos.drop(['Correlation'], axis=1)
Toppos.sort_values('Top␣Positive␣Correlations', inplace=True, ascending=False)

#Create heatmap for the correlations between all the features
fig, ax = plt.subplots(figsize=(15,15))
sns.heatmap(df.corr(), square=True, annot=True, fmt=".2f")
fig.savefig('snsheatmap.png')

#Create scatter plots between hihly correlated features

Correlation1= sns.pairplot(df, x_vars= ['BlowOffValve_Position'],
                           y_vars=['Stage3_AirPressure'], diag_kind="kde", kind="reg")


Correlation2= sns.pairplot(df, x_vars= ['Stage2_AirPressure'],
                           y_vars=['Stage3_AirPressure'], diag_kind="kde", kind="reg" )


Correlation3= sns.pairplot(df, x_vars= ['System_AirPressure'],
                           y_vars=['Stage3_AirPressure'], diag_kind="kde", kind="reg" )


Correlation4= sns.pairplot(df, x_vars= ['BlowOffValve_Position'],
                           y_vars=['System_AirPressure'], diag_kind="kde", kind="reg" )
```

```
Correlation5= sns.pairplot(df, x_vars= ['Stage1_AirTemperature'],
                                y_vars=['Stage3_AirPressure'], diag_kind="kde", kind="reg" )


Correlation6= sns.pairplot(df, x_vars= ['Stage3_AirTemperature'],
                                y_vars=['Stage3_AirPressure'], diag_kind="kde", kind="reg" )
```

B.2 Machine Learning Regression Models

Machine learning models

```
#Importing the relevant libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.neural_network import MLPRegressor
from sklearn import metrics

#Importing the appended and preprocessed CSV file
Data = pd.read_csv('JanuarDesember-sorted-appended.csv', index_col = 0, parse_dates=True
    ↪ )

#Prepairing Data for Machine learing
col = Data.columns.tolist()
Data_ts = pd.DataFrame(data=Data[col].values.tolist(), index=Data.index, columns=col)
Data_ts.sort_index(inplace=True)

Data_tmp = Data_ts.copy()
Data_tmp['Datetime'] = Data_ts.index
```

```
Data_tmp = Data_tmp.dropna()

def encode(x):
    if (x.dtype is np.dtype('O') and x.name != 'Stage3_AirPressure') or x.name == 'Datetime':
        return x.astype('category').cat.codes

    return x

Data_tmp = Data_tmp.apply(encode)

x_col = Data_tmp.columns.values[Data_tmp.columns.values != 'Stage3_AirPressure']

#Creating training, validation and test set
train_start_date = '2019-01-08 00:00:00'
train_end_date = '2019-01-16 23:59:00'
validate_start_date = '2018-12-14 00:00:00'
validate_end_date = '2018-12-18 23:59:00'
test_start_date = '2019-01-22 00:00:00'
test_end_date = '2019-01-27 00:00:00'


Xtrain=Data_tmp.loc[train_start_date:train_end_date,x_col].values
Ytrain= Data_tmp.loc[train_start_date:train_end_date,'Stage3_AirPressure'].values

Xvalidate=Data_tmp.loc[validate_start_date:train_end_date,x_col].values
Yvalidate= Data_tmp.loc[validate_start_date:train_end_date,'Stage3_AirPressure'].values

Xtest = Data_tmp.loc[test_start_date:test_end_date,x_col].values
Ytest= Data_tmp.loc[test_start_date:test_end_date,'Stage3_AirPressure'].values

#Training and Testing with optimized hyperparameters

#DecisionTreeRegressor:
dsr = DecisionTreeRegressor(random_state = 0, min_samples_split = 10, max_depth = 8)
dsr.fit(Xtrain, Ytrain)
pre_y_by_dsr = dsr.predict(Xtest)
```

```
#RandomForestRegressor:
rfr = RandomForestRegressor(n_estimators = 24, random_state =0)
rfr.fit(Xtrain, Ytrain)
pre_y_by_rfr = rfr.predict(Xtest)

#MLPClassifier – Feedforward Neural Network – Model 6 obtained best performance
mlpc = MLPClassifier(hidden_layer_sizes=(4, 6, 4), activation='relu', alpha=0.05, solver="adam
    ↪ ")
mlpc.fit(Xtrain, Ytrain)
pre_y_by_mlpc = mlpc.predict(Xtest)



#Extract featyre importance for Decision Tree and Random Forest
feature_importance_RF=rfr.feature_importances_
feature_importance_RF=dsr.feature_importances_

#Creating dataframe with predicted and actual values
Data_test=Data_tmp.loc[test_start_date:test_end_date,x_col]

df = pd.DataFrame(index=Data_test.index)

df['pred_by_decision_tree_regressor'] = pre_y_by_dsr
df['pred_by_random_forest_regressor'] = pre_y_by_rfr
df['pred_by_MLP_regressor']= pre_y_by_mlpc
df['actual'] = Ytest

#Smoothing and averaging the time series data
df['ewmaDT'] = df['pred_by_decision_tree_regressor'].ewm(com=25).mean()
df['ewmaRF'] = df['pred_by_random_forest_regressor'].ewm(com=25).mean()
df['ewmaMLP'] = df['pred_by_MLP_regressor'].ewm(com=25).mean()
df['actual_value_ewma'] = df["actual"].ewm(com=25).mean()

# Changing column names
df.columns = ["pred_by_decision_tree_regressor", "pred_by_random_forest_regressor", '
    ↪ pred_by_MLP_regressor', "actual", "average_pred_by_decision_tree_regressor", "
```

```
    ↪ average_pred_by_random_forest_regressor", "average_pred_by_MLP_regressor", "
    ↪ average_actual"]

#Calculate Performance Metrics
print('Mean␣Absolute␣Error␣DSR:', metrics.mean_absolute_error(Ytest, pre_y_by_dsr))
print('Mean␣Squared␣Error␣DSR:', metrics.mean_squared_error(Ytest, pre_y_by_dsr))
print('Root␣Mean␣Squared␣Error␣DSR:', np.sqrt(metrics.mean_squared_error(Ytest,
    ↪ pre_y_by_dsr)))

print('Mean␣Absolute␣Error␣RFR:', metrics.mean_absolute_error(Ytest, pre_y_by_rfr))
print('Mean␣Squared␣Error␣RFR:', metrics.mean_squared_error(Ytest, pre_y_by_rfr))
print('Root␣Mean␣Squared␣Error␣RFR:', np.sqrt(metrics.mean_squared_error(Ytest,
    ↪ pre_y_by_rfr)))

print('Mean␣Absolute␣Error␣MLP:', metrics.mean_absolute_error(Ytest, pre_y_by_mlpc))
print('Mean␣Squared␣Error␣MLP:', metrics.mean_squared_error(Ytest, pre_y_by_mlpc))
print('Root␣Mean␣Squared␣Error␣MLP:', np.sqrt(metrics.mean_squared_error(Ytest,
    ↪ pre_y_by_mlpc)))


# Plotting predicted and actual values
predictions_df_average = df[['pred_by_decision_tree_regressor', '
    ↪ pred_by_random_forest_regressor', "pred_by_MLP_regressor", 'actual']]
predictions_plot = predictions_df_average.plot(title='Random␣Forest,␣Decision␣tree␣and␣
    ↪ MLP␣predicted␣and␣actual␣values')
predictions_plot.set_xlabel("Date")
predictions_plot.set_ylabel("Stage3_AirPressure")
fig = predictions_plot.get_figure()
fig.savefig("RF−DT−MLP−actual33.png")

# Plotting avergae predicted and average actual values
predictions_df_average = df[['average_pred_by_decision_tree_regressor', '
    ↪ average_pred_by_random_forest_regressor', "average_pred_by_MLP_regressor", '
    ↪ average_actual']]
predictions_plot = predictions_df_average.plot(title='Comparison␣of␣predicted␣and␣actual␣
    ↪ values␣after␣aligning␣&␣smoothing')
```

```
predictions_plot.set_xlabel("Date")
predictions_plot.set_ylabel("Stage3_AirPressure")
fig = predictions_plot.get_figure()
fig.savefig("RF-DT-MLP-average-actual33.png")

#Ploting feature importance Random Forest
n_features = Data_ts.shape[1]
plt.barh(range(n_features), rfr.feature_importances_, align='center')
plt.yticks(np.arange(n_features), Data_tmp.columns.values[Data_tmp.columns.values != '
    ↪ Stage3_AirPressure'])
plt.xlabel('Feature␣Importance␣Random␣Forest␣Regressor')
plt.ylabel('Feature')
plt.show()
```

B.2.1 Hyperparameter Optimization - Feedforward Neural Network

Hyperparameter Optimization for Feedforward Neural Networks

```
#Hyperparameter Optimization - Feedforward Neural Network
#Model1
mlpc = MLPClassifier(hidden_layer_sizes=(20, 30, 20), activation='tanh', alpha=0.05, solver="
    ↪ adam")

#Model2
mlpc = MLPClassifier(hidden_layer_sizes=(10, 15, 10), activation='tanh', alpha=0.05, solver="
    ↪ adam")

#Model3
mlpc = MLPClassifier(hidden_layer_sizes=(5, 10, 5), activation='relu', alpha=0.05, solver="
    ↪ adam")

#Model4
mlpc = MLPClassifier(hidden_layer_sizes=(4, 8, 4), activation='relu', alpha=0.02, solver="adam
    ↪ ")

#Model5
```

```
mlpc = MLPClassifier(hidden_layer_sizes=(2, 4, 2), activation='relu', alpha=0.005, solver="
    ↪ adam")

#Model6 – Highest performance on validation and testing data set
mlpc = MLPClassifier(hidden_layer_sizes=(4, 6, 4), activation='relu', alpha=0.005, solver="
    ↪ adam")
```

B.3 Unsupervised Anomaly Detection Based on Residuals

Anomaly detector based on residuals

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor



Data = pd.read_csv('JanuarDesember-sorted-appended.csv', index_col = 0, parse_dates=True
    ↪ )

#Prepairing Data for Machine learing
col = Data.columns.tolist()
Data_ts = pd.DataFrame(data=Data[col].values.tolist(), index=Data.index, columns=col)
Data_ts.sort_index(inplace=True)

Data_tmp = Data_ts.copy()
Data_tmp['Datetime'] = Data_ts.index
Data_tmp = Data_tmp.dropna()

def encode(x):
    if (x.dtype is np.dtype('O') and x.name != 'Stage3_AirPressure') or x.name == 'Datetime':
        return x.astype('category').cat.codes

    return x
```

```
Data_tmp = Data_tmp.apply(encode)

x_col = Data_tmp.columns.values[Data_tmp.columns.values != 'Stage3_AirPressure']

#Creating testing and training data sets
train_start_date = '2019-01-08 00:00:00'
train_end_date = '2019-01-16 00:00:00'
test_start_date = '2019-01-22 00:00:00'
test_end_date = '2019-01-27 00:00:00'

Xtrain=Data_tmp.loc[train_start_date:train_end_date,x_col].values
Ytrain= Data_tmp.loc[train_start_date:train_end_date,'Stage3_AirPressure'].values

Xtest = Data_tmp.loc[test_start_date:test_end_date,x_col].values
Ytest= Data_tmp.loc[test_start_date:test_end_date,'Stage3_AirPressure'].values

Data_test=Data_tmp.loc[test_start_date:test_end_date,x_col]

df = pd.DataFrame(index=Data_test.index)

#Training and predicting with the RandomForestRegressor
rfr = RandomForestRegressor(n_estimators = 10, random_state =0)
rfr.fit(Xtrain, Ytrain)
pre_y_by_rfr = rfr.predict(Xtest)

df['pred_by_random_forest_regressor'] = pre_y_by_rfr
df['actual'] = Ytest

# Establishing confidence bounds to determine abnormalites
df['error'] = df['actual']- df['pred_by_random_forest_regressor']

df['Upperbound']= df['pred_by_random_forest_regressor'] + df['error'].std()*3

df['Lowerbound']= df['pred_by_random_forest_regressor'] - df['error'].std()*3

errorstd= df['error'].std()
```

```
errorinfo=df['error'].describe()

# Labeling the actual Stage3_AirPressure as abnormal or normal to count the number of
    ↪ detected anomalies
df.loc[df.actual > df.Upperbound, 'anomaly'] = 'True'
df.loc[df.actual < df.Lowerbound, 'anomaly'] = 'True'
df.loc[(df.actual > df.Lowerbound) & (df.actual < df.Upperbound), 'anomaly'] = 'False'

print(df['anomaly'].describe())
print(df['anomaly'].isnull().sum())

#Plotting the residuals between the predicted and actual Stage3_AirPressure with confidence
    ↪ bounds
ax = plt.gca()
df.plot(kind='line', y='error', ax=ax, title='Residuals␣between␣predicted␣and␣actual␣with␣
    ↪ confidence␣bounds␣at␣$3\sigma$')
ax.axhline(y=errorstd*3, color='r', linestyle='--', lw=1)
ax.axhline(y=-errorstd*3, color='r', linestyle='--', lw=1)
ax.legend(["Residuals", "3$\sigma$"])
ax.set_xlabel("Date")
ax.set_ylabel("Error")
plt.show()

#Histogram of the error between the predicted and actual Stage3_AirPressure
fig, ax = plt.subplots()
df.hist(column='error', bins=20, ax=ax)
fig.savefig('error-hist.png')


#Ploting the predicted and actual Stage3_AirPressure with confidence bounds to detect
    ↪ anomalies
predictions_df_average = df[['pred_by_random_forest_regressor','actual']]
predictions_plot = predictions_df_average.plot(title='Random␣Forest␣predicted␣and␣actual
    ↪ ␣with␣confidence␣bounds␣equal␣3$\sigma$')
predictions_plot.fill_between(df.index, df['pred_by_random_forest_regressor']+errorstd*2, df['
    ↪ pred_by_random_forest_regressor']-errorstd*2, facecolor='green', alpha=0.4)
```

```
predictions_plot.set_xlabel("Date")
predictions_plot.set_ylabel("Stage3_AirPressure")
fig = predictions_plot.get_figure()
```

B.4 Unsupervised Anomaly Detection Based on Clustering

Anomaly detector based on clustering

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import pylab as pl

df = pd.read_csv('JanuarDesember-sorted-appended.csv', index_col = 0, parse_dates=True)

# Standardize the features
data = df[['Stage3_AirPressure', 'Stage2_AirPressure']]
min_max_scaler = StandardScaler()
np_scaled = min_max_scaler.fit_transform(data)
data = pd.DataFrame(np_scaled)

#Creating training and testing data set
train_start_date = '2019-01-08 00:00:00'
train_end_date = '2019-01-16 00:00:00'
test_start_date = '2019-01-22 00:00:00'
test_end_date = '2019-01-27 00:00:00'

dftrain = df.loc[train_start_date:train_end_date,:]
dftest = df.loc[test_start_date:test_end_date,:]

#Standardized training and testing set
train = data.loc[train_start_date:train_end_date,:]
test = data.loc[test_start_date:test_end_date,:]
```

```
# Creating the Elbow curve to determine the number of clusters required to capture the variance
Y = train[['Stage3_AirPressure']]
X = train[['Stage2_AirPressure']]
Nc = range(1, 20)
kmeans = [KMeans(n_clusters=i) for i in Nc]
score = [kmeans[i].fit(Y).score(Y) for i in range(len(kmeans))]
plt.figure()
pl.plot(Nc,score)
pl.xlabel('Number␣of␣Clusters')
pl.ylabel('Score')
pl.title('Elbow␣Curve')
pl.show();

# Cluster the training data in 7 clusters with the K−means algorithm
n_cluster=7
kmeans = [KMeans(n_clusters=i).fit(train) for i in n_cluster]
df['cluster'] = kmeans[7].predict(train)
dftrain['principal_feature1'] = train[0]
dftrain['principal_feature2'] = train[1]
dftrain['cluster'].value_counts()


#plot the different clusters for the Stage3_AirPressure and Stage2_AirPressure
fig, ax = plt.subplots()
colors = {0:'red', 1:'blue', 2:'green', 3:'pink', 4:'black', 5:'orange', 6:'cyan'}
ax.scatter(dftrain['principal_feature1'], dftrain['principal_feature2'], c=dftrain["cluster"].apply
    ↪ (lambda x: colors[x]))
plt.title('7␣Cluster␣K−Means')
plt.xlabel('Stage2_AirPressure')
plt.ylabel('Stage3_AirPressure')
plt.savefig('K−meanscluster.png')
plt.show();

# Calulate the distance between each point in the testing set and the nearest centroid, the
    ↪ samples with biggest distance are classified as anomalies
```

```
def getDistanceByPoint(data, model):
    distance = pd.Series()
    for i in range(0,len(data)):
        Xa = np.array(data.loc[i])
        Xb = model.cluster_centers_[model.labels_[i]-1]
        distance.set_value(i, np.linalg.norm(Xa-Xb))
    return distance

outliers_fraction = 0.02
distance = getDistanceByPoint(test, kmeans[7])
number_of_outliers = int(outliers_fraction*len(distance))
threshold = distance.nlargest(number_of_outliers).min()

# Store the series classifying the samples as abnormal og normal (0:normal, 1:anomaly)
dftest['anomaly1'] = (distance >= threshold).astype(int)

#Convert the datetime to datetime integer
Getdates=dftest.index.tolist()
dftest['date_time']=Getdates
dftest = dftest.sort_values('date_time')
dftest['date_time_int'] = dftest.date_time.astype(np.int64)

#Visualisation of time series plot with anomalies
a = dftest.loc[dftest['anomaly1'] == 1, ['date_time_int', 'Stage3_AirPressure']] #anomalies

fig, ax = plt.subplots()
ax.plot(dftest['date_time_int'], dftest['Stage3_AirPressure'], color='blue', label='Normal')
ax.scatter(a['date_time_int'],a['Stage3_AirPressure'], color='red', label='Anomaly')
plt.xlabel('Date Time Integer')
plt.ylabel('Stage3_AirPressure')
plt.title('Stage 3 Air Pressure time series plot with detected anomalies based on
    ↪ clustering')
plt.legend()
plt.show();

#Visualization of anomalies in a histogram plot
```

```
a = dftest.loc[dftest['anomaly1'] == 0, 'Stage3_AirPressure']
b = dftest.loc[dftest['anomaly1'] == 1, 'Stage3_AirPressure']

fig, axs = plt.subplots()
axs.hist([a,b], bins=32, stacked=True, color=['blue', 'red'])
plt.title('Histogram␣of␣time␣series␣data␣with␣detected␣anomalies␣based␣on␣clustering')
plt.show();

#Count the number of anomalies for varying outlier fraction
print(dftest['anomaly1'].sum())
print(dftest['anomaly1'].count())
```

Bibliography

Abu-Mostafa, Y. S., Magdon-Ismail, M., and Lin, H.-T. (2019). *Learning From Data*.

Adhikari, R. and Agrawal, R. K. (2013). An introductory study on time series modeling and forecasting.

Augusteijn, M. and Folkert, B. (2002). Neural network classification and novelty detection. *International Journal of Remote Sensing*, 23(14):2891–2902. cited By 37.

Barros, A. (2018). Data driven prognostic and predictive maintenance. 1(2):1–59.

Baruah, P. and Chinnam, R. (2005). Hmms for diagnostics and prognostics in machining processes. *International Journal of Production Research*, 43(6):1275–1293. cited By 167.

Bezdek, J., Ehrlich, R., and Full, W. (1984). Fcm: The fuzzy c-means clustering algorithm. *Computers and Geosciences*, 10(2-3):191–203. cited By 2619.

Blanchet, M., Thaden, G. V., and Thieulloy, G. D. (2014). Industry 4.0: The new industrial revolution - how europe will succeed.

Breunig, M. M., Kriegel, H.-P., Ng, R. T., and Sander, J. (2000). Lof: Identifying density-based local outliers. *SIGMOD Rec.*, 29(2):93–104.

Brockwell, P. J. and Davis, R. A. (2002). *Introduction to Time Series and Forecasting*. Springer Texts in Statistics,. Springer New York, New York, NY.

Byington, C., Roemer, M., and Galie, T. (2002). Prognostic enhancements to diagnostic systems for improved condition-based maintenance. volume 6, pages 6–2815 to 6–2824.

Celebi, M. E. (2015). Partitional clustering algorithms.

Chandola, V., Banerjee, A., and Kumar, V. (2009a). Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3):15:1–15:58.

Chandola, V., Banerjee, A., and Kumar, V. (2009b). Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3):15:1–15:58.

Chollet, F. (2017). *Deep Learning with Python*. Manning Publications.

Dalinina, R. (2017). Introduction to correlation. *Data Science*.

Donges, N. (2018). The random forest algorithm. *Towards Data Science*.

Dunning, T. and Friedman, E. (2014). *Practical Machine Learning: A New Look at Anomaly Detection*. O'Reilly Media, Inc.

Gao, R., Wang, L., Teti, R., Dornfeld, D., Kumara, S., Mori, M., and Helu, M. (2015). Cloud-enabled prognosis for manufacturing. *CIRP Annals*, 64(2):749 – 772.

Giampaolo, T. (2010). Compressor handbook : principles and practice.

Goodfellow, I. (2017). Deep learning.

Goos, P. (2015). Statistics with jmp : graphs, descriptive statistics and probability.

Gottlieb, I. M. (1997). 1 - electric motor generalities. In Gottlieb, I. M., editor, *Practical Electric Motor Handbook*, pages 1 – 31. Newnes, Oxford.

Géron, A. (2017). Hands-on machine learning with scikit-learn and tensorflow : concepts, tools and techniques to build intelligent systems.

Hawkin, D. M. (1980). *Identification of Outliers, - (Monographs on applied probability and statistics)*. SPRINGER-SCIENCE+BUSINESS MEDIA, B.V, second edition.

Hyndman, R. J. and Athanasopoulos, G. (2018). *Forecasting: Principles and Practice*. OTexts, 2nd edition edition.

James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). *Tree-Based Methods*, pages 303–335. Springer New York, New York, NY.

Janacek, G. (2010). Time series analysis forecasting and control. *Journal of Time Series Analysis*, 31(4):303–303.

Jelaska, D. (2012). Introduction. In *Gears and Gear Drives*, pages 1–16. John Wiley & Sons, Ltd, Chichester, UK.

Joseph, L. (2001). Centrifugal air compressor basics. *Plant Engineering*.

Ketkar, N. (2017). *Deep Learning with Python: A Hands-on Introduction*. Apress, Berkeley, CA.

Lavin, A. and Ahmad, S. (2015). Evaluating real-time anomaly detection algorithms – the numenta anomaly benchmark. In *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, pages 38–44.

Lehmann, E. L. E. L. (2005). Testing statistical hypotheses.

Markou, M. and Singh, S. (2003a). Novelty detection: A review - part 1: Statistical approaches. *Signal Processing*, 83(12):2481–2497. cited By 826.

Markou, M. and Singh, S. (2003b). Novelty detection: a review—part 2:: neural network based approaches. *Signal Processing*, 83(12):2499–2521.

McKinney, W. (2010). Data structures for statistical computing in python. In van der Walt, S. and Millman, J., editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56.

Morel, M. (2017). Compressed air aftercoolers. *VMAC Air Inovated*.

Oakland, J. (2007). *Statistical Process Control: Sixth Edition*. Elsevier.

Olson, C., Judd, K., and Nichols, J. (2018). Manifold learning techniques for unsupervised anomaly detection. *Expert Systems With Applications*, 91:374–385.

Patcha, A. and Park, J.-M. (2007). An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer Networks*, 51(12):3448–3470.

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

Peng, Y., Dong, M., and Zuo, M. (2010). Current status of machine prognostics in condition-based maintenance: a review. *The International Journal of Advanced Manufacturing Technology*, 50(1):297–313.

Prisecaru, P. (2017). The challenges of the industry 4.0. *Global Economic Observer*, pages 66–72.

Ryan, J., Lin, M.-J., and Miikkulainen, R. (1998). Intrusion detection with neural networks. pages 943–949. cited By 210.

Sanz-Bobi, M. A. (2016). Review of analytics methods supporting anomaly detection and condition based maintenance. *iit.commillas, MonitorX framework*, (2.0):68.

Schölkopf, B., Williamson, R., Smola, A., Shawe-Taylor, J., and Platt, J. (2000). Support vector method for novelty detection. *Support Vector Method for Novelty Detection*, pages 582–588. cited By 23.

Si, X.-S., Wang, W., Hu, C.-H., and Zhou, D.-H. (2011). Remaining useful life estimation – a review on the statistical data driven approaches. *European Journal of Operational Research*, 213(1):1–14.

Sikorska, J., 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.

Trevino, A. (2016). Introduction to k-means.

Vachtsevanos, G., Lewis, F., Roemer, M., Hess, A., and Wu, B. (2006). Intelligent fault diagnosis and prognosis for engineering systems.

Vasconcelos, G., Fairhurst, M., and Bisset, D. (1995). Investigating feedforward neural networks with respect to the rejection of spurious patterns. *Pattern Recognition Letters*, 16(2):207–212. cited By 26.

Vasja, R., Maja, M., and Alojz, K. (2016). A complex view of industry 4.0. *SAGE Open*.

Waskom, M. (2019). Seaborn: statistical data.

Wee, D., Kelly, R., Cattel, J., and Breunig, M. (2015). Industry 4.0 - how to navigate digitization of the manufacturing sector. *McKinsey Digital*.

Wu, J. (2012). Advances in k-means clustering : A data mining thinking.