NTNU
Norwegian University of
Science and Technology

Master's thesis

2019

Halvor Reiten, Thomas Kleiven

Master's thesis

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

Halvor Reiten
Thomas Kleiven

Unsupervised Anomaly Detection on Streaming Data in the Petroleum Industry Using Deep Neural Networks

June 2019

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

Unsupervised Anomaly Detection on Streaming Data in the Petroleum Industry Using Deep Neural Networks

Halvor Reiten
Thomas Kleiven

Engineering and ICT
Submission date: June 2019
Supervisor: Jørn Vatn

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

Abstract

Modern petroleum production systems are intensively monitored by a high number of sensors and condition monitoring equipment, generating vast amounts of data that is continuously streamed to production data platforms. A fundamental capability of streaming analytics is to develop anomaly detection methods that can efficiently monitor critical equipment and warn about anomalous behavior, helping to prevent imminent equipment failures and reduce operational risk. As such, the main objective of this thesis is to demonstrate how to develop unsupervised, real-time anomaly detection algorithms for equipment in the oil and gas industry. We specifically look at a gas compressor operating at an oil rig in the North Sea, and propose two model-based anomaly detection methods that identify anomalies by comparing the true measurements to the predictions of a probabilistic model that simulate the normal behavior of the equipment.

An accurate predictive model is crucial to the success of such anomaly detection methods. Seeing *deep learning* and sequence specialized neural networks as tremendously successful in similarly complex modeling problems, this thesis focuses on researching the potential of deep learning as a modeling framework for the physical equipment. To this end, we implement deep neural networks of varying types and designs, as well as techniques to assess the predictive uncertainty of these models. In addition, a set of less advanced baselines are implemented to benchmarks their performances. We find that the best performance is achieved by an ensemble model consisting of a Long Short-Term Memory network and a feed-forward neural network, clearly outperforming the best benchmark model by 17.2%. Consequently, this model is employed in the anomaly detection algorithms.

The first of the proposed anomaly detection methods uses a static residual distribution to classify observations, while the other uses the prediction intervals of point-predictions to do the same. The quality of the methods is evaluated by examining whether their anomaly warnings conform to our expectations when applied to three arbitrarily chosen datasets with clear abnormal patterns. On this evaluation criterion, their performances are highly satisfactory and consistent with our expectations, and we find that the applied deep learning model works excellently in conjunction with the proposed anomaly detection methods. The implemented anomaly detection setups meet the high demands of a real-time streaming anomaly detection algorithm, they can process and evaluate incoming data in a fast, efficient and unsupervised manner, and show great potential to be utilized in an industrial setting.

Sammendrag

Moderne produksjonssystemer innen olje og gass er intensivt overvåket av et stort antall sensorer og tilstandsovervåkende utstyr. Dette generer enorme mengder data som kontinuerlig strømmes til produksjonsdataplattformer, noe som presenterer industrien med store muligheter og tekniske utfordringer. Et stort potensiale knyttet til dette ligger i å utvikle effektive metoder for å overvåke tilstanden til systemene slik at man kan avdekke når enkelte systemer avviker fra normalen for å kunne avdekke umiddelbare farer. Med dette i siktet er hovedmålet for denne oppgaven å utvikle algoritmer for avviksdetektering i sanntid for utstyr i olje- og gassindustrien. Vi fokuserer på en gasskompressor på en oljeplattform i Nordsjøen, der vi foreslår to modell-baserte algoritmer for å detektere avvik ved å sammenligne observerte målinger med prediksjoner fra en statistisk modell som simulerer normaltilstanden til systemet.

En presis prediktiv modell er avgjørende for suksessen av slike algoritmer. Ettersom *dyp læring* med sekvensbaserte nevrale nettverk har vist seg å være svært suksessfulle innen lignende problemstillinger, ser vi på potensialet for dyp læring som et rammeverk for å modellere fysiske system. Vi implementerer ulike typer dype nevrale nett, såfremt teknikker for å estimere den prediktive usikkerheten tilknyttet disse modellene. Videre implementerer vi et sett av mindre avanserte modeller som fungerer som standarder for å sammenligne ytelsen til de nevrale nettverkene. Våre resultater viser at den mest presise modellen er satt sammen av et Long Short-Term Memory nettverk og et feed-forward nevralt nettverk, som klart presterer bedre enn de beste standardene med en relativ forbedring på 17.2%. Dermed benytter vi denne modellen i algoritmene for å detektere avvik.

Den første algoritmen for å detektere avvik benytter seg av en statisk residualdistribusjon for å klassifisere observasjoner, mens den andre benytter seg av det punktvise prediksjonsintervallet. Kvaliteten av metodene er evaluert ved å anvende metodene på tre tilfdelig utplukkede datasett som innehar flere observasjoner som avviker fra normalen, der vi undersøker om avviksadvarslene stemmer overens med det vi forventer. Begge metodene presterer svært tilfredsstillende, der avviksadvarslene stemmer godt overens med våre forventninger, og vi finner at den prediktive modellen fungerer utmerket i forbindelse med algoritmene. Metodene implementert i denne oppgaven møter de høye kravene for avviksdetektering i sanntid, der de kan prosessere og evaluere data på en rask og effektiv måte, og dette viser stort potensiale for å benytte seg av slike modeller i industrien.

Preface

This thesis was carried out during spring of 2019 and concludes our Master of Science in Engineering and ICT at the Norwegian University of Science and Technology. We want to express our gratitude towards our supervisor, Professor Jørn Vatn, for valuable guidance during the work with this thesis. We would also like to thank the team at Cognite, represented by Alexander Gleim, Patrick M. Robertson and Peter Malec, for supplying us with a challenging problem and all their support.

Trondheim, June 2019

Thomas Kleiven

Halvor Reiten

Thomas Kleiven

Abbreviations

ADAM	=	Adaptive Moment Estimation
AdaGrad	=	Adaptive Gradient
ASV	=	Anti-Surge Valve
API	=	Application Programming Interface
CP	=	Coverage Probability
CART	=	Classification and Regression Trees
ElNet	=	Elastic Nets
FT	=	Discharge Flow
GBM	=	Gradient Boosting Machines
GMM	=	Gaussian Mixture Model
GRU	=	Gated Recurrent Units
KA	=	Power Indicator
LSTM	=	Long Short-Term Memory
MAR	=	Multivariate Autoregressive
MAE	=	Mean Absolute Error
ML	=	Machine Learning
MLP	=	Multilayer Perceptron
MVL	=	Minimum Validation Loss
OID	=	Open Industrial Data
PI	=	Prediction Interval
PT	=	Discharge Pressure
RF	=	Random Forest
RMSE	=	Root Mean Squared Error
RNN	=	Recurrent Neural Network
RSS	=	Residual Sum of Squares
RUL	=	Remaining Useful Life
SDK	=	Software Development Kit
SGD	=	Stochastic Gradient Descent
STV	=	Suction Throttle Valve
SVM	=	Support Vector Machine
TT	=	Discharge Temperature
ZT/ZI	=	Valve Indicators

Table of Contents

1 Introduction **1**

1.1 Background . . . 1

1.2 Problem Definition . . . 2

1.3 Objectives . . . 3

1.4 Contributions . . . 4

1.5 Limitations . . . 5

1.6 Outline . . . 5

2 Related Work **7**

2.1 Unsupervised Anomaly Detection . . . 7

2.2 Evaluation of Unsupervised Anomaly Detection Models . . . 9

3 Theoretical Framework **11**

3.1 Time Series . . . 11

3.1.1 The components of a time series . . . 12

3.1.2 Cross-correlation and autocorrelation . . . 12

3.1.3 Stationarity . . . 12

3.2 Machine Learning . . . 13

3.2.1 Linear regression models . . . 14

3.2.2 Tree based methods . . . 14

3.3 Deep Learning . . . 16

3.3.1 Feed-forward neural networks . . . 16

3.3.2 Activation functions . . . 18

3.3.3 Backpropagation & gradient-based learning . . . 19

3.3.4 Optimization in neural networks . . . 20

3.3.5 Generalization, overfitting and regularization . . . 21

3.3.6 Recurrent neural networks . . . 22

3.3.7 Model ensembling . . . 24

3.3.8 Uncertainty assessment of deep learning models 25
3.4 Anomaly Detection 28
3.4.1 Types of anomalies 28
3.4.2 Outputs of an anomaly detection algorithm 28
3.4.3 Precision and recall 29
3.4.4 Concept drift 29
3.4.5 Anomaly detection in streaming data environments 29

4 Data and Preprocessing **31**
4.1 The Gas Compressor and Physical Data Sources 31
4.1.1 Context 31
4.1.2 High level system description 32
4.2 Data Collection 33
4.3 Selecting Model Inputs and Outputs 33
4.3.1 Model outputs 34
4.3.2 Model inputs 35
4.3.3 Time frames, aggregates and other specifications 37
4.4 Data Partitioning 38
4.5 Data Preprocessing 38
4.5.1 Handling missing data 39
4.5.2 Handling outliers and extreme values 39
4.5.3 Scaling by standardization 40

5 Method **41**
5.1 Constructing the Predictive Model 41
5.1.1 Evaluation metrics 42
5.1.2 Dealing with the temporal aspect for forecasting 42
5.1.3 Benchmarks 43
5.1.4 Deep learning models 45
5.2 Anomaly Detection 50
5.2.1 Residual-based anomaly detection 50
5.2.2 PI-based anomaly detection 52
5.2.3 Evaluation of unsupervised anomaly detection models 54
5.3 Hardware and technical implementation 54

6 Results and Analysis **57**
6.1 The Predictive Model 57
6.1.1 Benchmarks 58
6.1.2 Deep learning 59

6.2 Anomaly Detection . . . 64
6.2.1 Selected data regions with anomalous behavior . . . 64
6.2.2 Method I: Residual based anomaly detection . . . 65
6.2.3 Method II: PI-based anomaly detection . . . 68
6.2.4 Comparison of the methods . . . 70

7 Discussion 71
7.1 Potential of Deep Learning as Predictive Models . . . 71
7.2 Anomaly Detection with Deep Learning . . . 72
7.2.1 Operational application of the methods . . . 73
7.3 Challenges and Practical Barriers . . . 74
7.4 Further work . . . 75

8 Conclusion 77

Bibliography 83

A Additional Theory 84
A.1 Data Preprocessing Techniques . . . 84
A.1.1 Boruta . . . 84
A.1.2 Amelia II . . . 84
A.2 Bayesian modelling . . . 85

B Preprocessing and Statistical Properties 87
B.1 Data Preprocessing . . . 87
B.1.1 Feature sparsity . . . 87
B.1.2 Correlation plots . . . 88
B.1.3 Feature selection results . . . 89
B.2 Summary Statistics and Distribution Plots . . . 90

C Deep Learning Models & Anomaly Detection 93
C.1 Training Histories and Validation Data . . . 93
C.1.1 LSTM . . . 93
C.1.2 GRU . . . 94
C.1.3 MLP . . . 94
C.2 Anomaly Detection . . . 95
C.2.1 Method I: residual based anomaly detection . . . 95

List of Tables

3.1 Overview of some common activation functions. 18

4.1 Selected output tags 34
4.2 Selected input tags. 37

5.1 Example of lagged variables 42

6.1 MAE of benchmarks: common-sense heuristics 58
6.2 MAE of benchmarks: simple machine learning 59
6.3 MAE of deep learning models 60
6.4 Comparison of best benchmark models to best deep learning model 61
6.5 Uncertainty assessment of deep learning models 62
6.6 Expected anomalies in the selected time periods 65

B.1 Statistics of the complete dataset before scaling. 90
B.2 Statistics of the complete dataset after scaling. 90

List of Figures

3.1 Illustration of a feed-forward neural network. 17
3.2 An unfolded recurrent neural network. 22

4.1 A schematic of the compressor 32
4.2 Sample data of the selected output tags. 35
4.3 Physical positions in the subsystem of the selected inputs and outputs 37
4.4 Example of data partitioning 38

5.1 Examples of the common-sense heuristics 44
5.2 Sketch of the LSTM network 47
5.3 Sketch of the ensemble model 48
5.4 Scheme for anomaly detection method I 51
5.5 Scheme for anomaly detection method II 53

6.1 Predictive distribution of a model for a single input 63
6.2 Example of discharge temperature predictions 63
6.3 Data regions used to evaluate anomaly detection methods. 64
6.4 Q-Q plot for validation residuals 65
6.5 Anomaly detection plot for December 2018: method I 66
6.6 Anomaly detection plot for April 2018: method I 67
6.7 Anomaly detection plot for July 2018: method I 67
6.8 Anomaly detection plot for December 2018: method II 68
6.9 Anomaly detection plot for April 2018: method II 69
6.10 Anomaly detection plot for July 2018: method II 69

B.1 Feature sparsity 87
B.2 Cross correlation plots of the data variables 88
B.3 Feature importance plot for discharge flow, temperature and pressure 89
B.4 Distributions of the training, validation and test data. 91

B.5 Scatter matrix plot of the data 92

C.1 Training history of the LSTM model. 93
C.2 Training history of the GRU model. 94
C.3 Training history of the MLP model. 94
C.4 Threshold analysis for method I 95
C.5 Residual distributions used in model I 95

Chapter 1

Introduction

1.1 Background

Modern oil and gas production is subject to intense monitoring by a magnitude of sensors and condition monitoring equipment. Vast amounts of sensor data are continuously streamed to production data platforms, presenting the industry with significant opportunities and technical challenges. Analyzing and contextualizing these data streams can provide valuable, actionable insights that can be used to support decision-making, production optimization, and smart maintenance strategies (Ahmad et al., 2017). However, the industry is far from exploiting the opportunities that come with the available data and data analytics today (Anand, 2015). In order to stay competitive in a market with increasingly slimmer margins, it is a necessity for petroleum production companies to leverage advanced analytics and exploit data-driven solutions to optimize production and maintenance strategies (Ileby and Knutsen, 2017). Reports state that the digitalization of the oil and gas sector collectively can unlock $1.6 trillion of value for the industry and that *data* is arguably the industry's most significant asset in the years to come (Anand, 2015).

A key capability for streaming analytics is to develop models for detecting unusual, anomalous behavior in critical equipment (Ahmad et al., 2017). The streaming data often arrives with high frequency and in vast amounts, making the task of manual inspection and analysis to look for potential faults a tedious one. It is therefore of great interest to develop semi-automatic, unsupervised procedures to automate such tasks. Not only can such models be more precise than a human resource, but they work continuously and can catch anomalous behavior in equipment as early as possible. Critical anomalies must be identified and acted upon promptly to avoid potentially costly consequences for the production and the equipment itself.

One way to implement anomaly detection for physical equipment is to develop statistical models that simulate the normal behavior of the equipment and compare the actual observed behavior to what the model predicts. It is, however, an interesting question how the behavior of physical equipment is best modeled. Each component is part of a complex production system and can have hundreds to thousands of potentially relevant sensors associated with it, that have interdependent relationships that can be hard to determine. In addition, the performance of the machinery can be highly dependent on exogenous features and the specific operational conditions. This is not easily incorporated into rigorous theoretical models or existing simulation tools. For these reasons, a data-driven modeling approach might be a more suitable framework for the task.

Recent years have seen a rise in the use of *machine learning* (ML), a subfield of *artificial intelligence* (AI), for addressing similarly complex and non-linear modeling problems. ML is a collective term of algorithms where models are trained rather than explicitly programmed, i.e. models learn statistical structures in the data by evaluating empirical examples (Chollet, 2018). These techniques have become a hot topic due to the immense success demonstrated on a range of problems, made possible by the rapid development in hardware, computational power, and big data (Chollet, 2018). *Deep learning* is arguably the biggest and most successful subfield of ML, which most commonly involves learning in the form of *neural networks* (Goodfellow et al., 2016). Deep learning has proved tremendously successful in a broad range of problems within AI, from self-driving cars to fraud detection, but also within time series modeling. With sequence-specialized neural networks such as *Recurrent Neural Networks* (RNNs), deep learning has shown to outperform classical models in the case of long, multivariate, interdependent time series analysis (Laptev et al., 2017), and has been established as state-of-the-art approaches to sequence-modeling (Chollet, 2018).

Seeing deep learning as a promising framework for solving a problem of this complexity, this thesis aims to show how an anomaly detection method can be built for critical equipment in the petroleum production industry using specialized sequential networks from the deep learning framework. Our results demonstrate the potential of deep learning as a framework for modeling the behavior of physical equipment, and how such predictive models can be implemented to discover anomalous behavior. In addition, we attempt to shed light on the practical hurdles of implementing data-driven solutions in this context, as well as the advantages and disadvantages automated anomaly detection setups can have in an operational setting.

1.2 Problem Definition

Specifically, the main objective of this thesis is to construct an algorithm to detect and warn for abnormal behavior in a *first stage gas compressor* operating on an oil rig in the North Sea[1]. Our focus is on model-based anomaly detection setups, which consists of two major components: a predictive model that can simulate the *normal* behavior of the compressor, and a method that compares the predictions of the model to the true observed measurements to classify abnormal behavior. The intuition of this setup is that the actual observed values might differ substantially

[1] Further details regarding the compressor and the surrounding system are found in chapter 4.

from the predictions when it operates anomalous, which can be exploited to uncover novelties. Hence, a precise predictive model is decisive for the success of such anomaly detection methods. In detail, the model should predict the *one-step-ahead* sensor measurements of the gas discharged from the compressor - the discharge *flow*, *temperature* and *pressure* - given a history of sensor measurements prior to the compressor, and a set of system control parameters.

There are many ways to construct such models, but we focus on machine learning techniques, and especially deep learning models. There exist engineering models and simulation tools for modeling the behavior of gas compressors, but we focus on models without applying laws of physics or domain knowledge about the equipment. Effectively, this ensures that the produced models are as generic and adaptive as possible. Moreover, in order to properly evaluate the performance of the complex deep learning models, they are benchmarked to a set of less advanced models. In general, we seek the least complex, most accurate model that solves the problem.

While deep learning has seen high success in similar modeling problems and is presumed a promising framework for modeling the behavior of the compressor, it is commonly criticized for its lack of interpretation and ”black box” characteristics (Chollet, 2018). Conventional neural networks are deterministic and can produce overly confident point-predictions without considering the inherent predictive uncertainty (Chollet, 2018). Seeing the *predictive confidence* as highly important if the models are to be of practical value for decision-support and anomaly detection, it is essential to assess the predictive reliability of the model. As such, an important part of the thesis involves researching techniques to assess the predictive uncertainty of deep learning models.

After defining the predictive models for the gas compressor, one question lingers: how should a new observation be classified as anomalous, and when should the setup report for abnormal behavior of the equipment? A major part of this thesis is concerned with researching how to best implement the predictive model in the anomaly detection method, and how anomalous behavior can be classified based on the comparison between the actual observed and the predicted value. Moreover, how can we evaluate the performance of the anomaly detection method, as we are dealing with an unsupervised problem with no labeled regions? Lastly, the practical barriers of developing data-driven models with industrial data must be thoroughly discussed. What are the main challenges tied to data processing, developing predictive models and anomaly detection for industrial equipment in the petroleum industry, and what are the advantages and disadvantages of data-driven models in this context?

1.3 Objectives

The following objectives are defined in order to address the aforementioned challenges:

- Review related work regarding unsupervised anomaly detection algorithms for streaming data.

- Implement and evaluate one or more deep learning models that can model the behavior of the equipment. Moreover, define a set of less advanced baselines that will be used to benchmark the performance of these models.
- Identify and implement techniques to assess the predictive uncertainty of deep neural networks.
- Develop an anomaly detection algorithm for the gas compressor that can compare true observed values to those predicted by the most accurate predictive model.
- Evaluate and discuss the quality of the anomaly detection method and whether deep learning models offer suitable frameworks for modeling the behavior of physical equipment in the petroleum industry. Moreover, discuss the hurdles of implementing data-driven models in such a practical, operational context.

By completing these objectives, we research how anomaly detection can be applied for real, operating equipment in the petroleum industry using relevant sensory data. We explore the performance of deep learning in modeling behavior of industrial equipment, how to assess the predictive uncertainty of the models, as well as exploring the main challenges in implementing data-driven models in such a context.

1.4 Contributions

This thesis is a practical demonstration of how it is possible to build model-based anomaly detection setups for critical equipment in the petroleum industry using deep learning as the modeling framework. As a result, the main contributions of this thesis are the following:

- A review of existing and popular approaches for unsupervised anomaly detection in streaming environments, and a demonstration of the practical implementation of model-based anomaly detection algorithms for real, operating equipment.
- A thorough investigation of the potential of deep learning as a modeling framework for complex, industrial equipment.
- A presentation and implementation of techniques to assess the predictive uncertainty of deep neural networks, and how the uncertainty estimates can be incorporated in anomaly detection setups.
- An analysis on the advantages and challenges of real-time anomaly detection in production systems, and an elucidation of the practical hurdles of implementing data-driven solutions in data-intensive, industrial settings.

These contributions represent a positive development in the major topic of digitalization in the petroleum industry; making use of the vast amounts of available production data. We argue that the methods and techniques implemented in this thesis are highly generic and that they easily can be adapted to other types of equipment. Moreover, the techniques used for time series forecasting can be extended to entirely different problems involving time series forecasting of multivariate, interdependent time series.

1.5 Limitations

The models implemented in this thesis are specifically designed for one gas compressor due to the limited availability of public production data for other types of equipment used in the petroleum industry. However, because of the general nature of our models and methodology, they can easily be extended to other production systems. We have a particular focus on model-based anomaly detection methods in this thesis, and we do not research any other anomaly detection methodologies, although other setups could have been equally successful. In addition, there is a limitation in terms of quality assessment of the anomaly detection methods. Due to limited access to experts and labeled data, the models have been reviewed on hand-picked datasets and subjective expectations. The true quality of the methods is therefore in need of expert evaluation before any conclusions can be made.

1.6 Outline

First, related work regarding unsupervised anomaly detection on streaming data is presented in chapter 2. Then, the theoretical foundations of the methods used in this thesis are given in chapter 3. Chapter 4 describes any performed data collection, feature selection and preprocessing steps, as well as a high-level description of the physical sources to the data; the compressor and the subsystem it is part of. Chapter 5 gives a detailed description of the methods utilized in this thesis, i.e. how the benchmarks are implemented, the design choices of the deep neural networks, how the models are evaluated, and how the anomaly detection methods are built. Subsequently, chapter 6 presents the results of the predictive models and the proposed anomaly detection methods. In light of the results, chapter 7 provides a discussion on the true potential of deep learning as a modeling framework for industrial equipment, the quality of the anomaly detection algorithms and the practical hurdles of implementing data-driven solutions in the industry. Finally, chapter 8 concludes the thesis, and presents a summary of how the objectives in this thesis were met.

Chapter 2

Related Work

2.1 Unsupervised Anomaly Detection

Anomaly detection in time series is an extensively studied topic within a broad range of research areas, including medical diagnosis, fraud detection, network intrusion and programming defects (Hayes and Capretz, 2015; Fawcett and Provost, 1997). Many conventional approaches exist, both for supervised and unsupervised problems, but most are unsuited for the demands of a real-time anomaly detection algorithm for streaming data. In general, supervised classification based methods (Görnitz et al., 2013) that require labeled data are not applicable for real-time anomaly detection of data streams (Ahmad et al., 2017), as it often involves large quantities of data in need of instant processing, and that is collected continuously without information on whether the observation is anomalous or not.

The majority of existing and commonly used unsupervised anomaly detection algorithms can be categorized into three main groups (Goldstein and Uchida, 2016): *Clustering methods* (Guha et al., 2000; Sequeira and Zaki, 2002), *nearest neighbour based techniques* (Shuchita and Karanjit, 2012; Bay and Schwabacher, 2003; Knorr et al., 2000) and *statistical model-based approaches* (L. Simon and Rinehart, 2014; Ahmad et al., 2017; Laptev et al., 2017). Methods commonly used in practice for anomaly detection in *streaming data environments* are normally simple, computationally lightweight statistical techniques, including *k-sigma* (Laptev et al., 2015), *exponential smoothing techniques* by Holt-Winters (Szmit and Szmit, 2012) and *abrupt change detection* (Basseville and Nikiforov, 1993). However, these methods are mainly concerned with spatial anomalies and struggle to incorporate contextual dependencies, limiting their usefulness in applications where context-based anomalies are important to discover (Ahmad et al., 2017).

Several authors suggest anomaly detection models tailored for real-time anomaly detection of streaming data (Hill and Minsker, 2010; Ahmad et al., 2017; Hundman et al., 2018). These are often referred to as *analytical redundancy models* because they classify anomalies based on a simulated data stream whose measurements are compared to the true measurements of the actual sensor (Hill and Minsker, 2010). Thus, these methods involve developing statistical models that reconstruct univariate or multivariate data streams, as well as the corresponding logic to how the predicted data stream should be compared to the true values in order to discover anomalies (Mehrotra et al., 2017).

Early work within model-based anomaly detection include Upadhyaya et al. (1990) and Upadhyaya and Skorska (1984) that developed multivariate autoregressive (MAR) models to estimate the next value in a data stream based on historical data. Another popular technique for time series modeling is *autoregressive integrated moving average* (ARIMA) (Bianco et al., 2001), which is known as an effective, general purpose model for detecting outliers in time series with regular temporal patterns. However, these models usually require manual tuning to assess seasonality, trends and other statistical parameters of the data, making them hard to generalize (Zhu and Laptev, 2017).

Recent years have seen a rise in the application of *machine learning* in developing regression models in model-based anomaly detection algorithms (Ahmad et al., 2017; Bhattacharyya and Kalita, 2013). In particular, sequence specialized neural networks such as Recurrent Neural Networks (RNN), Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) and Gated Recurrent Units (GRU) (Chung et al., 2014) have been established as state-of-the-art approaches to sequence-modeling, and have garnered increasing attention due to their flexibility, ease of incorporating exogenous variables and ability to model highly complex, non-linear patterns (Zhu and Laptev, 2017). Recent papers (Laptev et al., 2017) show that deep neural networks can exceed the performance of classical time series models in the modeling of long, interdependent time series.

There are numerous examples of successful implementations of deep learning in an anomaly detection context. Nairac et al. (1999) employs an artificial neural network to predict the vibration data stream of a jet engine and to report sudden transitions in the engine when the prediction error exceeds a predetermined threshold. Yuan et al. (2016) fits a LSTM model to estimate the Remaining Useful Life (RUL) of an aero engine, which in turn is used to decide maintenance actions. Malhotra et al. (2016) also utilizes a LSTM model for multi-sensor anomaly detection and Malhotra et al. (2015) uses stacked LSTM networks for anomaly detection in time series.

Analytical redundancy models are dependent on a logic that classifies abnormal behavior based on the comparison between the prediction and observed value. We find that the majority of related work uses two methods when it comes to the classification of new observations. The first method detects anomalies based on the prediction error and a distribution of historical residuals (Ahmad et al., 2017; Hundman et al., 2018; Malhotra et al., 2015). The residual distribution is obtained by estimating the parameters of a Gaussian distribution based on the k most recent residuals. New observations are classified by calculating the probability that the residual of a new observation and the predicted belongs to the residual distribution. Other articles, such as Nairac et al. (1999), define what is considered *normal* behavior based on the results on the training data, indicating the use of a *static* distribution that is never updated. Common for most articles that utilize residual distributions to detect anomalies is the assumption that the distribution can be modeled as a Gaussian distribution.

The second approach detects outliers by using the prediction intervals of the models (Hill and Minsker, 2010; Zhu and Laptev, 2017). In order to create the prediction intervals, these methods rely on either stochastic models that produce non-deterministic predictions (Zhu and Laptev, 2017), or on the variance of the prediction errors of a deterministic model on some dataset where the behavior is considered normal (Hill and Minsker, 2010). This method registers an anomaly if the observed values are outside of the prediction interval.

2.2 Evaluation of Unsupervised Anomaly Detection Models

Evaluating the quality of unsupervised anomaly detection algorithms is not as straightforward as in the supervised case where the data contains labeled regions. Labels make it possible to calculate *precision and recall*-scores (Hundman et al., 2018; Lee et al., 2018), which are measurements of how many anomalies the model classified versus how many of them were correctly classified anomalies[1]. The precision and recall score is a direct measure of quality of a supervised anomaly detection procedure, and can therefore be used to compare methods to one another (Goldstein and Uchida, 2016). However, such labels are not available in unsupervised problems, and the evaluation of the effectiveness of unsupervised anomaly detection models remain challenging (Campos et al., 2016).

A typical procedure in the literature to bypass this problem is to partially convert the problem to a supervised or semi-supervised one; consulting experts to obtain labels for specific data sets, and use this to calculate precision and recall scores (Hundman et al., 2018; Lee et al., 2018; Goadrich et al., 2004). However, this manual procedure is labor intensive for a human resource and not particularly reliable (Campos et al., 2016). Other articles have a more intuitive approach and label the data manually based on deviations from observed patterns and logical reasoning. As an example, Malhotra et al. (2016) attempts to find anomalies in a power demand dataset, and observes a regularly occurring pattern that the first five days of the week have a high power demand. Any week where the power demand is low for one or more of the first five days is labeled as anomalous, which enables the use of precision and recall-scores as a measure of quality.

Another way to assess the effectiveness of an unsupervised procedure is to test the algorithm on publicly available benchmark datasets and compare its performance to acknowledged baselines (Goldstein and Uchida, 2016). However, this requires highly generic models that work for the benchmark data set as well as the specific case where the method should be applied (Campos et al., 2016). Moreover, the results are not directly transferable to how well the method works

[1] Precision and recall is described with more detail in section 3.4.3.

on the specific case. Thus, this evaluation technique is limited when it comes to implementations where models are made to work in specific applications.

Chapter 3

Theoretical Framework

The following section provides a brief overview of the theory applied in this thesis. This includes basic definitions of methods within time series, time series modeling, machine learning, and anomaly detection. A large part of this chapter is dedicated to the explanation of fundamental principles of deep learning and neural networks.

3.1 Time Series

A time series can be defined as a sequence of data points x observed over time t (Brockwell and Davis, 2016). Time series are found everywhere, e.g. in financial data, climate data, biological data, and time series analysis is therefore a broadly researched field (Hastie et al., 2009). Time series analysis and modeling are techniques to draw inferences from time series data. A time series model can be defined as a model that approximates the joint distribution of which the observed set of data points have been generated from (Hyndman and Athanasopoulos, 2018). The forecasting of time series is the ability to use time series models to predict the likely future development of a time series.

A time series that only consists of one variable is a *univariate* time series, whereas a *multivariate* time series is one that consists of one or more variables. The time series is *continuous* if the series is measured for every time instance, and *discrete* if it is only recorded at discrete time steps, usually with a constant interval between the readings (e.g., every minute, every hour, etc.). Usually, discrete time series are a result of the discrete measurements of continuous processes (Hastie et al., 2009).

3.1.1 The components of a time series

In general, time series can be decomposed into three major components: *Seasonality*, *Trend* and *Cycles* (Brockwell and Davis, 2016). The trend component usually captures both the cycle and trend components, and it is therefore referred to as the cycle-trend component. The trend can be described as the general, long-term change of the time series, the cycle describes cyclical variations around the trend, and the seasonality term is periodical variations in the time series (Brockwell and Davis, 2016). A time series can therefore be broken down to the following three components: a trend-cycle component (T_t), a seasonal component (S_t) and the remainder (R_t) which denotes the stochastic variations not captured by the former components.

The components can be combined in different ways. An *additive decomposition* yields $y_t = S_t + T_t + R_t$, whereas a *multiplicative decomposition* gives $y_t = S_t \cdot T_t \cdot R_t$. In practice, the multiplicative model is more popular, but what works best is dependent on the nature of the time series (Brockwell and Davis, 2016).

3.1.2 Cross-correlation and autocorrelation

Correlation is a general term used in statistics to measure how strong a relationship is between two variables. There are many ways to quantify the correlation between variables, such as *Spearman rank correlation*[1] and *Kendall Tau correlation* [2], but a popular measure, and the one applied in this thesis, is the *Pearson correlation coefficient*[3]. It is a measure of linear correlation, and can be defined with the following equation:

$$\rho_{X,Y} = \frac{\text{cov}(X,Y)}{\sigma_X \sigma_Y} \tag{3.1}$$

In other words, the Pearson correlation factor is the covariance of two variables divided by the product of their respective standard deviation.

Autocorrelation is the correlation of a variable with itself over time. The analysis of a variable's autocorrelation is used to infer any temporal patterns and trends in the data and is usually applied to uncover characteristics regarding stationarity.

3.1.3 Stationarity

A time series is said to be *stationary* if the statistical properties of the series do not vary with time, meaning that the mean, variance, and covariance of the data are independent of time (Brockwell and Davis, 2016). The stationarity of a series is usually uncovered by *auto-correlation plots* and *run sequence plots*. For a non-stationary time series, the auto-correlation plot will decrease slowly

[1] Defined in Zar (1972)

[2] Defined in Bolboaca and Jäntschi (2006)

[3] Defined in Benesty et al. (2009)

with an increasing lag, while it will drop quickly to zero in the case of a stationary time series (Brockwell and Davis, 2016).

Many of the classical time series models only work with stationary data, even though this is an unreasonable assumption for most real-world applications. A technique to make a non-stationary series stationary is *differencing*. By differencing, one computes the change between consecutive observations in the series, $y'_t = y_t - y_{t-1}$, with the aim of stabilizing the mean of the series (Brockwell and Davis, 2016). If the resulting difference y'_t is white noise[4], the equation becomes $y'_t = y_{t-1} + \epsilon_t$. If first order differencing does not stabilize the series, then other types of differences might be used such as *second-order differencing* given by $y''_t = y'_t - y'_{t-1}$, or *seasonal differencing* that computes changes from one season to another, $y'_t = y_t - y_{t-m}$. Sometimes, combinations of these methods are necessary (Brockwell and Davis, 2016).

3.2 Machine Learning

Machine learning is a collective term of algorithms that are able to learn from data without being explicitly programmed (Mostafa et al., 2012). On a general basis, machine learning algorithms are split into two main categories: *supervised learning* - learning problems where the target variables are known - and *unsupervised learning* - problems where the targets are unknown. The objective of supervised machine learning models is to approximate the true relationship between a set of inputs and their corresponding target variables - the function f that has generated y given X.

It is assumed that there exists an unknown underlying function f that maps the input features $\mathbf{x}$ to the output targets $\mathbf{y}$ such that $y = f(x) + \epsilon$ where ϵ is a randomly sampled, independent error term that is normal distributed with a mean of zero (Goodfellow et al., 2016). The objective of any machine learning problem is to approximate this function such that the outputs of the approximated function (the predictions) are as close as possible to the true targets, i.e. $f^*(\mathbf{x}) = \hat{y} \approx f(\mathbf{x}) = y + \epsilon$, where f^* is the estimated function and $\mathbb{E}[\epsilon] = 0$. The total bias of a model consists of a *reducible* and *irreducible* error term. While the former can be mitigated by using other statistical learning models or optimizing model parameters, the latter represents inherent random noise in the data that cannot be predicted. The learning process is essentially a problem of minimizing the reducible error term.

This section will elaborate on fundamental theory on machine learning. First, simpler machine learning techniques such as linear regression models and tree-based methods are described, before moving on to deep learning and more advanced topics in the next section.

[4] This is the case when the differences are independent and identically distributed with zero mean.

3.2.1 Linear regression models

Ordinary least squares linear regression

Ordinary least squares linear regression finds the linear combination of the input variables such that the sum of squared differences of the observed and predicted values by the model is minimized (Hastie et al., 2009). Given the inputs $\mathbf{x}$ and output y, the objective is to infer the weights $\mathbf{w}$ such that $y \approx f^*(x, w) = w_0 + \sum_{i=1}^{k} w_i \cdot g(x_i)$. Mathematically this can be defined as minimizing the cost function $C = (y - x\beta)^T (y - x\beta)$, where β are the estimated parameters and y the observed values. In essence, β defines the parameters of a hyperplane that is fitted to the data such that the total sum of squared errors is minimized.

Shrinkage methods

Shrinkage methods are *biased learning regression methods* that introduce a penalty term to the minimization objective (Hastie et al., 2009). They are essentially least squares linear regression with an added regularization term $\Omega(\beta)$. The cost function to be minimized is then least squares *plus* the penalty: $C = (y - x\beta)^T (y - x\beta) + \alpha \cdot \Omega(\beta)$. The effect of the regularization term is that it *shrinks* the estimated coefficients closer towards zero (Mostafa et al., 2012). As with any other regularization technique, the main motivation is to reduce the generalization error by reducing the variance of the model. This, however, comes at the cost of a slightly increased bias, but the decrease in variance outweighs the slight decrease in bias such that the total generalization error is lowered (Mostafa et al., 2012).

Ridge regression and *LASSO*[5] are two shrinkage methods that introduce regularization (Hastie et al., 2009). With ridge regression, the penalty term is set equal to the L2-norm ($\Omega(\beta) = \beta^T \beta$), whereas LASSO sets the penalty term to the L1-norm ($\Omega(\beta) = |\beta|$). The parameter α sets the *degree of regularization* and must be optimized in the implementation of these models. $\alpha = 0$ means ordinary least squares regression with no regularization.

Elastic Nets are a hybrid between Ridge regression and LASSO, where both L1-norm and L2-norm regularization is introduced (Hastie et al., 2009). Here, there are two parameters that must be optimized, namely the degree of regularization by α, and the fraction of L1 and L2 regularization by λ. Setting $\lambda = 1$ implies using only the LASSO penalty, $\lambda = 0$ only the Ridge penalty, and between 0 and 1 implies a combination of the two.

3.2.2 Tree based methods

Tree-based methods comprise models that use *decision trees* to produce predictions (Mostafa et al., 2012). The fundamental building blocks of tree-based methods are Classification and Regression

[5] *Least Absolute Shrinkage and Selection Operator*

Trees (CART). They are popular because they are straightforward to implement, computationally cheap and easy to interpret. A CART is a directed acyclic graph that consists of internal nodes, leaves and edges (Hastie et al., 2009). Each internal node corresponds to one of the input variables, and each split is a Boolean test that is either true or false. The path between one node and another is called an edge, and the leaves corresponds to one of the target variables. The trees are grown using a top-down, greedy algorithm called *recursive binary splitting* (Hastie et al., 2009). The algorithm splits a feature space into J distinct, non-overlapping regions, and it is completed once the subset of each node contains all the target variables. In regression problems, the splits are commonly made such that the split maximizes the reduction in the *residual sum of squares* (RSS) (Mostafa et al., 2012). While these trees are easy to understand and implement, they are rarely used as standalone models due to having relatively poor predictive capabilities.

Bagging and Boosting

One of the obvious disadvantages with single decision trees is their high sampling variability and comparably poor predictive abilities (Mostafa et al., 2012). *Bagging* and *Boosting* are methods designed to improve the accuracy and stability of such trees by combining the results of many of them (thereby they are commonly called *ensemble* methods) (Hastie et al., 2009).

Bagging, short for *bootstrap aggregation*, is a technique that samples B samples of n observations from the dataset, fits a decision tree for each of the data samples and averages the outcome (Hastie et al., 2009). The trees are grown independently, and all the input features are used as split-candidates in the tree-generating process. The final prediction is the average of the ensemble of trees. In theory, this reduces the variance of the predictions, which overall reduces the generalization error of the model.

Boosting are methods that combines many weak learners into a strong one (Hastie et al., 2009). Here, the trees are grown sequentially using information from previously grown trees. More specifically, the residuals from the previously grown tree are used as the response variable when growing the next. The trees will only have a small number of splits each, denoted d. This yields a sequence of B trees, where each tree accounts for some variation in y that was not captured by the previous trees. The boosting method is a slow learner, and will, in general, require a small learning rate and many trees to get good results.

Random Forest

In Bagging, the trees are grown on randomly sampled subsets of the data using all p predictors in the data set as split-candidates. However, if the bagged trees are highly correlated and produce similar outcomes, the improvement will not be substantial (Mostafa et al., 2012). This will, for instance, be the case if the data set has one or more dominant predictors that most grown trees consistently will use as a split-criterion (Hastie et al., 2009). A Random Forest is a special case of Bagging that aims to fix this problem. Random Forests decorrelates the trees by randomly sampling which predictors to consider every time a tree is to be grown (Hastie et al., 2009). A Random Forest will therefore draw B samples from the data as well as m predictors to create the trees, and average the outcome. Notice that a Random Forest is the same as Bagging for $m = p$, where p is the total number of input features.

Gradient Boosting Machines

Gradient Boosting Machines are methods that use boosting in combination with steepest gradient descent optimization. With a defined loss function, each tree is fit on the steepest descent of the gradient (the negative gradient). For in-depth explanations on gradient boosting machines, the interested reader is referred to Hastie et al. (2009) on page 349.

3.3 Deep Learning

Deep learning is a subfield of ML, usually in the form of *neural networks*, which emphasizes the learning of successive layers of increasingly meaningful representations (Chollet, 2018). Deep learning has garnered increasing attention in recent years due to the tremendous successes demonstrated by the framework in a range of challenging tasks within artificial intelligence. Also in time series- and sequence modeling, deep learning has become an increasingly popular approach. Recurrent Neural Networks, Long Short-Term Memory (Hochreiter and Schmidhuber, 1997) and Gated Recurrent Neural Networks (Chung et al., 2014) have been established as state-of-the-art approaches to sequence-modelling and point-forecasting (Assaad et al., 2008). These are specialized neural networks that are especially suited for sequence-based data such as text, speech and time series, and they have proved superior to classical time series modeling techniques in the case of multivariate, interdependent time series (Zhu and Laptev, 2017).

This section gives an introduction to the fundamentals of deep learning, and especially the methods applied in this thesis. First, key concepts and the building blocks of deep learning models, the feed-forward neural networks, are described. Further on, the theory on other types of neural networks is given, such as Recurrent Neural Networks. The theory presented is mostly inspired by two textbooks on deep learning, Chollet (2018) and Goodfellow et al. (2016).

3.3.1 Feed-forward neural networks

Feed-forward artificial neural networks, also known as *multilayer perceptrons*, are the very foundation of deep learning models (Goodfellow et al., 2016). As for any machine learning algorithm, the objective of a neural network is to infer the underlying function of the data-generating process, $y = f(\mathbf{x})$, for a given set of inputs and outputs $\mathbf{x}$ and $\mathbf{y}$. The parameters of the network, θ, are adjusted such that the learned model $f(\mathbf{x};\theta)$ best approximates the true target function f, i.e. $f(\mathbf{x};\theta) = \hat{y} \approx f(\mathbf{x}) = y$ (Chollet, 2018).

The feed-forward neural network can be interpreted as a series of nested functions that transform the input data to an output. Each step in the chain of functions is denoted as *layers*, and each layer performs a non-linear transformation to the data. The model can have an arbitrary number of layers, denoted L, and the number of layers defines the *depth* of the model. The composition and structure of these layers define the *topology* of the neural network. Thinking of the multilayer perceptron as a series of consecutive transformations, the learned function f can be defined as $f(\mathbf{x},\theta) = f^{(L)}(f^{(L-1)}(...f^{(2)}(f^{(1)}(\mathbf{x};\theta^{(1)}))))$, where $f^{(l)}$ is the transformation through layer l. The combination of functions constitutes the hypothesis space $\mathcal{H}$, the set of solutions (or hypotheses) h that it's possible for the model to choose from when fitting the model.

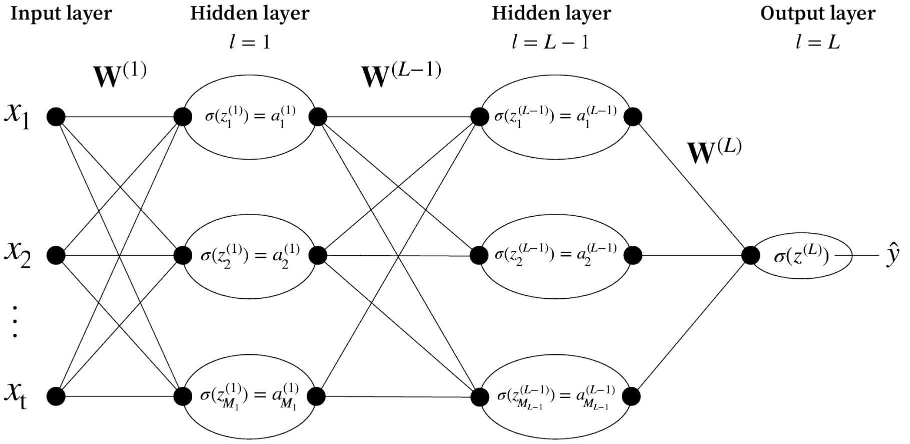

Input layer
Hidden layer
$l = 1$
Hidden layer
$l = L - 1$
Output layer
$l = L$
$\mathbf{W}^{(1)}$
$\mathbf{W}^{(L-1)}$
$\mathbf{W}^{(L)}$
x_1
x_2
x_t
$\sigma(z_1^{(1)}) = a_1^{(1)}$
$\sigma(z_2^{(1)}) = a_2^{(1)}$
$\sigma(z_{M_1}^{(1)}) = a_{M_1}^{(1)}$
$\sigma(z_1^{(L-1)}) = a_1^{(L-1)}$
$\sigma(z_2^{(L-1)}) = a_2^{(L-1)}$
$\sigma(z_{M_{L-1}}^{(L-1)}) = a_{M_{L-1}}^{(L-1)}$
$\sigma(z^{(L)})$
$\hat{y}$

Figure 3.1: Illustration of a feed-forward neural network. The input layer takes the input $[x_1, ..., x_t]$. Two hidden layers follow, with m neurons in each. The synapses between the layers have weight matrices $\mathbf{W}$. The neurons calculate the weighted sum of the outputs of the preceding layer and transform it with a function h. The output layer is a single neuron, making the final prediction $\hat{y}$.

There are three types of layers in a neural network, namely the *input layer*, the *output layer* and *hidden layers*. The input layer is responsible for the entry and through-putting of the input data and does not transform the data in any way. Following the input layer comes a sequence of hidden layers $l = 1, ..., L - 1$. Each layer, l_i, will transform the data in a particular way, depending on the activation function of the neurons comprising the layer. The final hidden layer is connected to the output layer, which converts the signal to make the final prediction $\hat{y}$.

Each of the layers consists of *nodes* or *neurons*. Each of the neurons in one layer, l_i, are fully connected to the neurons the next, l_{i+1}, by *synapses* or *weights*. These weights are commonly given in the layer-specific weight matrix $\mathbf{W}^{(l)}$. The role of a neuron in a layer is simple: It receives the input signals from the neurons in the preceding layer (multiplied by their respective weights), adds a bias term, and transforms the signal through an activation function σ and passes the transformed signal on to the next layer. The output of a single neuron, $a_i^{(l)}$ can then be calculated as follows:

$$\begin{aligned} z_i^{(l)} &= \mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + b_i^{(l)} \\ a_i^{(l)} &= \sigma(z_i^{(l)}) \end{aligned} \tag{3.2}$$

where $\mathbf{W}^{(l)}$ is the weight matrix for layer l, $\mathbf{h}^{(l-1)}$ is the vector of outputs from the preceding layer $l-1$, $b_i^{(l)}$ a bias term for neuron i in layer l, and $h_i^{(l)}$ the output of neuron i in layer l through the activation function σ. The combination of many non-linear transformations across neurons and layers makes it possible to approximate approximate highly complex, non-linear functions.

3.3.2 Activation functions

Each of the neurons in the neural network transforms the input with a non-linear function. These functions are denoted as *activation functions*. The activation functions are necessary to enable the models to learn other than linear patterns. The hypothesis space consisting of linear transformations of the input data is too restricted, and the model would not be able to learn more complex patterns. Moreover, deeper models with exclusively linear activation functions would not benefit from adding more layers (i.e., making the model deeper), as linear transformations of linear transformations do not extend the hypothesis space (Chollet, 2018).

The choice of when to use which activation functions is still an active area of research (Goodfellow et al., 2016), and there does not yet exist many theoretically guided principles. Which activation function to use when is therefore subject to the trial-and-error procedure. Testing must be performed with what activation functions work best for a specific problem. Some of the most common activation functions are given in the table 3.1.

A sigmoid function outputs a scalar that can be interpreted as a probability on the interval $[0, 1]$, whereas a ReLU (Rectified Linear Unit) zeros out negative values (Chollet, 2018). The Tanh function returns a value on the interval $[-1, 1]$. Literature suggests that the ReLU activation function provides an excellent default choice that usually works well with most problems (Goodfellow et al., 2016; Chollet, 2018). This is often used in combination with the sigmoid function in the output layer, but this detail is dependent on the nature of the problem and the function of the network.

Name	**Function**
Linear	$\sigma(x) = ax$
Sigmoid	$\sigma(x) = \frac{e^x}{1+e^x}$
Tanh	$\sigma(x) = tanh(x)$
ReLU	$\sigma(x) = max(0, x)$

Table 3.1: Overview of some common activation functions.

All the activation functions listed in table 3.1 are *differentiable* functions. This is important in order to use them with *gradient based learning*, where the partial derivative of all functions is needed.

3.3.3 Backpropagation & gradient-based learning

The neural network learns by adjusting the parameters of the network, θ, such that the sample inputs presented to the network are mapped correctly to their associated targets. However, a network can have millions of these parameters, and finding the specific combination that minimizes the prediction error seems like a daunting task.

Initially, the weights are set to random, small values, resulting in poor initial performance of the model (Chollet, 2018). Next, a loss function is defined, $C(f(\mathbf{x};\theta), f^{*}(\mathbf{x}))$, that is used to evaluate how well a set of parameters θ performs by calculating the distance between the predictions $\hat{\mathbf{y}}$ and the true values $\mathbf{y}$. The neural network will iteratively update the weights in the network by processing sampled training data[6], attempting to minimize the loss function. In other words, for every data point, the weights in the network are adjusted to decrease the loss score. Each round of training is called an *epoch*, and one epoch is completed once each observation in the sampled data has been processed. Naturally, the more epochs, the better the model will perform on the in-sample training data. However, increasing the number of epochs comes at a computational cost, as well as the increased risk of *overfitting*[7]. Specifying the number of epochs is therefore a trade-off between computational resources, achievable accuracy, and generalization.

Goodfellow et al. (2016) describes how the weights in the network are updated by the use of *backpropagation* and optimization algorithms: When the network produces a prediction $\hat{y}$, the error of the prediction is calculated using the loss function. This error is *fed backward* through the network to calculate each weight's contribution to the error. The weights are then modified proportionally to their contribution. More specifically, this contribution is found by computing the gradient of the loss function with respect to each of the weights in the network, i.e. $\nabla_{\theta} C(f(\mathbf{x};\theta), f^{*}(\mathbf{x}))$. The gradient contains information about all the partial derivatives of the error function concerning each of the weights in the network, which an optimization algorithm uses in combination with a specified *learning rate* or *step rate* to update the weights. The gradient of the loss function concerning a specific set of parameters $\theta_0 \in \theta$ describes the *curvature* of the loss function around this particular set of parameters, and the weights can be updated by moving slightly in the direction where the negative derivative is the largest. That is, $\theta_1 = \theta_0 - \lambda \cdot \nabla_{\theta_0}$, where λ is the learning rate. This is also the origin of the commonly used name of this method, *gradient descent* - moving the weights in the direction where the gradient declines.

[6] N samples will be drawn from the training data, where N is the size of the data set

[7] The concept of overfitting is described in section 3.3.5

3.3.4 Optimization in neural networks

As described in the previous section, the neural network is dependent on some optimization algorithm to calculate the magnitude and direction of the parameter update. The optimization algorithms that are mostly used in deep learning are referred to as *mini-batch* methods, where they use ”mini-batches” of the training data at a time (not all the data at once, with a minimum of one data point). The fact that there is randomness in the sampling of small batches of data, makes these methods *stochastic*. A larger batch size is desirable because each update is more accurate as more data is used in the training process (thus improving the accuracy of the gradient estimation), but it comes at a computational cost. In contrast, a smaller batch size can have a regularizing effect on the model (Goodfellow et al., 2016). It is common practice to use batch sizes in the power of 2, such as 16, 32, 64, etc.

A family of common optimization techniques used in neural networks is *stochastic gradient descent* (SGD). These methods are stochastic as they sample random mini-batches from the data. For each mini-batch, the gradient of the loss function with respect to a given set of parameters is computed. This is used to update the parameters in the direction the descending gradient, thereby the name of *gradient descent*.

Given a loss function L, a batch of sampled data points $B \subset D$, D being the whole data set, and a set of weight parameters θ, the gradient, ∇L is calculated as follows:

$$\nabla L = \frac{1}{B} \nabla_\theta \sum_{b=1}^{B} L(\hat{y}_{b;\theta}, y_b) \tag{3.3}$$

Then, given a learning rate λ_k for iteration k, the new parameters are adjusted in the direction of the negative gradient proportional to the learning rate.

$$\theta_{\text{new}} = \theta_{\text{old}} - \lambda_k \cdot \nabla L \tag{3.4}$$

It is important that the learning rate is set appropriately, such that the model converges. If set too high, the gradient descent can overshoot the minimum, fail to converge or even diverge. The lower the learning rate, the slower the convergence, hence it involves an increased computational cost. *Adaptive learning* aims to adjust the learning rate in an adaptive manner during training, and introduces unique rates for the individual weights that are dependent on the previously computed gradients. Additionally, concepts such as *momentum* and *decay* can speed up the convergence of SGD methods. Common choices for optimization techniques that employ adaptive learning are ADAM (Adaptive Moment Estimation) and AdaGrad (Adaptive Gradient). ADAM has proved to work well in practice, a safe choice, and most often favored over other optimization algorithms (Goodfellow et al., 2016).

3.3.5 Generalization, overfitting and regularization

Overfitting is an essential concern in any machine learning problem. It is a phenomenon that occurs when the model is beginning to fit the training data too well, i.e., the model is learning patterns that are specific to the training data, but that is irrelevant when it comes to new, unseen data observations (Chollet, 2018). As a result, the out-of-sample error increases as the in-sample error decreases. Likewise, the model is said to be *underfitting* when there are still relevant patterns to learn in the training data that it has not learned yet, and it is beneficial to continue the learning process. For neural networks, and for any other machine learning method, there is a trade-off between *optimization* and *generalization*. Optimization is the process where the model is adjusted to get the best possible performance on the training data (i.e. minimizing the prediction error on the data it is trained on), whereas generalization refers to how well the model can predict unseen data that has not been part of the training process (Chollet, 2018).

The best solution to improve the generalizing properties of the model is to supply more data to the model (Ketkar, 2017). However, this is often not possible, and other measures must be taken. The process of fighting overfitting is commonly called *regularization* (Goodfellow et al., 2016). Regularization is a technique to constrain the model in terms of complexity, and how much of what information the model can process, with the purpose to reduce the generalization error.

Early stopping is a regularization measure that aims to stop the training process once the performance on the validation set has not improved by a threshold over a specified number of iterations (Goodfellow et al., 2016). This affects the number of epochs the network is allowed to train, effectively stopping the process once the out-of-sample loss does not improve. The danger of this method is when the loss has reached a local optimum, and the training is stopped too early.

Another approach is to reduce the trainable parameters of the network by reducing the number of layers and/or nodes in the network, i.e., performing a *complexity reduction* (Goodfellow et al., 2016). However, there are few theoretical principles to guide how this should be best done, so it is a trial-and-error based approach in need of manual tuning.

Weight regularization is another traditional approach to regularization of neural networks, such as L1 and L2 norm penalties. These methods work by adding a *regularization term*, $\Omega(h)$, to the loss function, effectively adding a handicap to the minimization problem (Mostafa et al., 2012). Instead of minimizing the loss function alone, one minimizes a combination of the loss function and the regularization term. The new cost function that is sought minimized can therefore be defined as $C(\hat{y}, y, \theta) = L(\hat{y}, y) + \Omega(\theta)$. This effectively awards simpler models, and penalizes more complex model. L1 and L2 regularization are regularization by adding the L1 norm ($\Omega(\theta) = \|w\|_1$) and L2 norm ($\Omega(\theta) = \frac{1}{2}\|w\|_2^2$).

Dropout is arguably the most popular regularization technique in neural networks (Goodfellow et al., 2016), due to being simple to implement, computationally cheap and efficient. It works by randomly deactivating nodes in the network, usually a share between 20 - 50% of the nodes in the layer (Chollet, 2018). The dropout technique induces the effect of randomness in the training process, which has positive effects on the generalizing properties of the model. Furthermore, it speeds up the training process and forces the network not to rely on a few important features and prohibits the nodes in the network to co-adapt too much (Srivastava et al., 2014). Hence, this is a technique we employ extensively in our models.

3.3.6 Recurrent neural networks

Recurrent Neural Networks are networks specialized for treating sequential data. Traditional neural networks (MLPs) learn sequences by processing the whole sequence at once, treating each temporal instance as a separate feature (Chollet, 2018). In contrast, RNNs process sequences sequentially by iterating over each element while keeping a state in-between the elements, which can be thought of a form of *memory* of previously seen data (Goodfellow et al., 2016). This enables the network to be able to recognize temporal patterns and generalize across sequences. Intuitively, RNNs can be thought of consecutive copies of the same network, each passing information to a succeeding network through the state (Olah, 2015). The prediction of a RNN is therefore not only based on the final input vector (as a traditional network is), but also on the state which is a product of calculations based on the preceding input vectors. This makes RNNs particularly well suited for solving sequence-based problems such as natural text processing, speech recognition, and time series modeling, where temporal ordering matters.

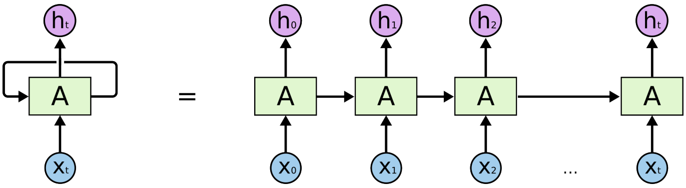


Figure 3.2: An unfolded recurrent neural network. Source: Olah (2015).

RNNs handle data sequentially by iterating over the elements in the sequence, while in between each element, a state is calculated based on the information it has previously seen (Chollet, 2018). Consider figure 3.2, an illustration of an unfolded RNN: For each iteration, a piece of a neural network, A, takes a new input $\mathbf{x}_t$ and outputs a value h_t, representing the state of the network at time t (Olah, 2015). The updated state can be calculated based on the previous state, $h_{(t-1)}$, the vector of inputs up to time t, $\mathbf{x}_t$, and the parameters of the network, θ:

$$h_t = f(h_{t-1}, \mathbf{x}_t, \theta), \tag{3.5}$$

The function $f(\cdot;\theta)$ represents the transformation performed by the neural network A, and this is held constant (with a constant set of parameters θ) throughout the processing of the sequence.

The unfolding of a RNN, illustrated in figure 3.2, shows that a RNN is equivalent to the repeated application of some function f for each time step, where only the previous state, h_{t-1}, is updated between each iteration.

The training of a RNN is done by applying backpropagation[8] to the unrolled network, better known as *backpropagation through time* (BPTT) (Goodfellow et al., 2016). The only difference from standard backpropagation is that the outputs at every timestep will have an impact on the loss, so the gradient is calculated in a recursive manner.

RNNs can be designed for a wide variety of purposes, and can even handle varying input sizes; useful for instance when treating textual data with a varying number of words. However, we use RNNs with fixed input sizes to make predictions for each sample in the dataset. Similar to traditional neural networks, RNNs can benefit from more advanced architectures where layers are stacked, but they are less certain to do so (Chollet, 2018).

Long Short-Term Memory and Gated Recurrent Units

One of the main appeals of RNNs is the idea that information can persist through time. In theory, RNNs are certainly capable of modeling long-term dependencies (Goodfellow et al., 2016). However, as the sequence length grows, the task of connecting information from one time step to information many steps before becomes increasingly difficult. The reason for this is the *vanishing gradient problem*, an effect that occurs when the network becomes very deep (Chollet, 2018). When training the network using gradient-based learning, the weights in the network are updated proportionally to the gradient of the error function. In deep networks, such as unfolded RNNs processing long sequences, this update will become vanishing small, effectively prohibiting the network from learning further (Goodfellow et al., 2016). For this reason, simple RNNs are almost never used as standalone models for solving real-world problems due to being too simplistic models (Chollet, 2018).

The theoretical reasons for the vanishing gradient problem were studied in detail by Hochreiter and Schmidhuber (1997), who designed alternatives to the simple recurrent unit to help solve the problem - the *Long Short-Term Memory* (LSTM). Like the simple RNN, the LSTM uses recurrence to generalize a model to sequences of variable length. Simple RNNs have a basic structure in the recurrent unit, usually a single neural layer. In contrast, the LSTM has four neural layers interacting in a special way, specifically designed to carry information over long sequences. The usual LSTM layer consist of a *cell*, a *forget* gate, an *input* gate and an *output* gate (Chollet, 2018). The three added gates control the flow of information in and out of the cell, and effectively decide what information the network should use, or not use, for a given input (Hochreiter and Schmidhuber, 1997). This enables information to persist over long periods while avoiding the problem of

[8] The concept of backpropagation is explained in section 3.3.4.

vanishing gradients. The interested reader is referred to Hochreiter and Schmidhuber (1997) or Goodfellow et al. (2016) for an in-depth technical description of the LSTM network.

While LSTMs are essential to the modern success of RNNs, they are expensive models to train. As a more effective alternative to the LSTM structure, Chung et al. (2014) introduce the *gated recurrent unit* (GRU). Different studies have shown that the GRU network in some cases can outperform the LSTM networks in terms of computational time and generalizing abilities (Chung et al., 2014), but, in general, the LSTM networks have more representative power, albeit more computationally demanding (Goodfellow et al., 2016). Similar to the LSTM layer, a GRU layer decides how much information to pass on to a cell by the use of sequential gates. The GRU only has two of these gates, commonly known as an *update* gate and a *reset* gate. The update gate controls how much of past information needs to be carried on to the next cell. The reset gate controls the opposite, e.g. how much of the information that can be forgotten. The interested reader is referred to Chung et al. (2014) for more information on GRUs.

LSTM and GRU networks are considered state-of-the-art when it comes to sequence processing techniques due to their unique ability to model long-term dependencies.

3.3.7 Model ensembling

Model ensembling is a powerful technique to help reduce the generalization error in machine learning tasks. Ensembling consists of pooling the predictions of many models together in order to make a better prediction as a whole (Chollet, 2018). A good ensemble relies on diversity and the assumption that the constituent models are good for *different reasons*. The main idea is that different models utilize different aspects of the data to make their predictions, thus extracting part truths of the data, but not the whole truth (Chollet, 2018). The combination of these perspectives leads to a better description of the data as a whole. However, if the models are similar to each other, they will be biased in the same way, and the ensemble will necessarily retain this bias. However, if they are biased in different ways, the biases will partly cancel each other out, and the ensemble will produce more accurate predictions than any of its constituent models can alone (Chollet, 2018).

Arguably, the easiest way to combine models is by averaging their predictions at inference time (Chollet, 2018). This is a good option if the models are more or less equally good, but can lead to worse accuracy if one model is significantly worse than the others. A smarter ensemble method is to use a weighted average, where each model is assigned a weight that is learned on the validation data. Even better results can be achieved by training a separate model that learns the best combination of the models in order to reduce the prediction error (Goodfellow et al., 2016).

3.3.8 Uncertainty assessment of deep learning models

The theory presented in the following section is mostly based on a doctoral thesis examining this topic (Gal and Ghahramani, 2015), and two studies that successfully implements the methods (Zhu and Laptev, 2017; Kendall and Gal, 2017).

Deep learning can achieve remarkable accuracy in modeling complex, high-dimensional relationships. For regression problems, the quality of the model is often given by an accuracy score. This score is often accepted blindly and the mapping is assumed to be accurate, even though this might not be the case (Kendall and Gal, 2017). The accuracy score does not necessarily reflect what the model *does not* know - the inherent uncertainty of the model. Traditional deep learning are deterministic models that produce point estimates, but fundamental questions like *how confident* the model is of its estimate remain unanswered (Gal and Ghahramani, 2015). Assessing the predictive uncertainty of models can be highly useful in an operational context, where the uncertainty information can be cardinal for a decision based on a prediction.

The uncertainty of a trained neural network $f^{\theta}(\cdot)$, where θ is the vector of learned weights, is quantified by its variance in predicting an output $\hat{y}^*$ for an input x^*. Using the law of total variance, this can be decomposed as follows:

$$\begin{aligned} \text{Var}(y^*|x^*) &= \text{Var}\left(\mathbb{E}\left[y^*|\theta, x^*\right]\right) + \mathbb{E}\left[\text{Var}(y^*|\theta, x^*)\right] \\ &= \text{Var}(f^{\theta}(x^*)) + \sigma^2 \end{aligned} \tag{3.6}$$

We see that the variance can be decomposed to two main terms: $\text{Var}(f^{\theta}(x^*))$ represents the variance of the predictive model and is referred to as *epistemic uncertainty*, while σ^2 is the inherent noise in the data-generating process, called the *aleatoric uncertainty*. In order to fully describe the uncertainty of the model, these two terms must be estimated. The remainder of this section explains these sources of uncertainty, and how they can be modeled for deep learning models.

Epistemic uncertainty

Epistemic uncertainty, also referred to as *model uncertainty*, captures the uncertainty regarding the specification model parameters, which for a neural network is the learned weights (Gal and Ghahramani, 2015). This source of uncertainty can be mitigated by feeding the model with more data samples, but the modeling of this uncertainty has traditionally been difficult for deep learning models (Kendall and Gal, 2017). However, Gal and Ghahramani (2015) identified a remarkably simple approach to estimate the epistemic uncertainty for neural networks, with no changes required to the model architecture or the learning process in general.

We are interested in the *expected model output* given our input - the predictive mean $\mathbb{E}(y^*)$ - and a *measure of confidence* of the prediction - the predictive variance $\text{Var}(y^*)$. Gal and Ghahramani

(2015) found that the application of stochastic dropout in the network in both the training and prediction phase is equivalent to a Gaussian process approximation. By applying random dropout in the network and standard Bayesian modeling techniques, the parameters of the predictive distribution can be estimated.

Key to estimating the parameters of a Gaussian distribution is the posterior distribution, $P(\theta|\mathbf{X}, \mathbf{Y})$. The posterior is the distribution of the most likely function parameters given the observed data $\mathcal{D} = \{\mathbf{X}, \mathbf{Y}\}$, and its deduction is a process known as *Bayesian inference* (Gal and Ghahramani, 2015). Given the posterior distribution, it can be used to calculate the output of a new input point x^*:

$$P(\mathbf{y}^*|\mathbf{x}^*, \theta, \mathcal{D}) = \int P(\mathbf{y}^*|\mathbf{x}^*, \theta)P(\theta|\mathcal{D})d\theta \tag{3.7}$$

The expected value of y^* is called the *predictive mean* of the model, and the variance is the predictive uncertainty (Gal and Ghahramani, 2015). In other words, if we can solve for the posterior distribution, we can also model the Gaussian distribution of the model for an input x^*.

While it is possible to evaluate the posterior analytically for simple models such as linear regression models, this is not the case for highly non-linear, high-dimensional models such as deep neural networks. It is therefore necessary to approximate the posterior. Common approximation methods are *Variational Inference* or *Markov Chain Monte Carlo*, better described in appendix A.2, but these techniques are not possible to scale to the vast number of parameters usually found in neural networks (Zhu and Laptev, 2017). Gal and Ghahramani (2015) found that performing stochastic dropout at inference time is equivalent to Bayesian approximation. By applying stochastic dropouts in each of the hidden layers in the network, a single output from the model can be viewed as a random sample from the posterior predictive distribution. The parameters of the predictive distribution of the model can therefore be approximated by running the same input x^* through the network many times and calculating the sampled mean and variance from the resulting set of predictions.

The parameters of the predictive distribution can therefore be estimated by predicting an output $\hat{y}^*$ for an input x^* B times through a network that applies stochastic dropout in each layer at inference time. This results in a sequence of B predictions for the input, $[\hat{y}_1, \hat{y}_2, ..., \hat{y}_B]$, of which we apply the following equations to obtain the predictive mean and variance:

$$\begin{aligned} \hat{y}^* &= \frac{1}{B}\sum_{b=1}^{B} \hat{y}_b^* \\ \eta_1^2 = \hat{\text{Var}}(f^\theta(x^*)) &= \frac{1}{B}\sum_{b=1}^{B} (\hat{y}_b^* - \hat{y}^*)^2 \end{aligned} \tag{3.8}$$

Notice that we denote the epistemic uncertainty as η_1^2.

Aleatoric uncertainty

Aleatoric uncertainty captures uncertainty regarding the irreducible noise that is present in the data, the σ^2-term in equation 3.6. A common way to estimate this is by evaluating the prediction error on an independent, held-out validation set (Zhu and Laptev, 2017; Kendall and Gal, 2017).

Let $\mathcal{D}_v$ define a separate, held-out validation set that contains inputs $\{x_1, ..., x_V\}$ and outputs $\{y_1, ..., y_V\}$. Given a network, $f^{\hat{\theta}}$, where $\hat{\theta}$ is the learned weights, the aleatoric uncertainty is estimated by evaluating the residual sum of squares of the network on the validation set:

$$\eta_2^2 = \hat{\sigma}^2 = \frac{1}{V} \sum_{v=1}^{V} \left(y_v - \mathbb{E}\left[f^{(\hat{\theta})}(x_v) \right] \right)^2 \tag{3.9}$$

Notice that we denote the aleatoric uncertainty as η_2^2. The validation set is independent of the learned function $f^{\hat{\theta}}(\cdot)$, and if we assume that $f^{\hat{\theta}}(\cdot)$ is an unbiased estimate of the true model, we have that

$$\begin{aligned} \mathbb{E}[\hat{\sigma}^2] &= \sigma^2 + \frac{1}{V} \sum_{v=1}^{V} \mathbb{E}\left[f^{\hat{\theta}}(x_v) - f^{\theta}(x_v) \right]^2 \\ &= \sigma^2 + \text{Var}_{\text{train}}(f^{\hat{\theta}}(x_v)) \end{aligned} \tag{3.10}$$

where $\text{Var}_{\text{train}}$ is the variance of the fitted model of the training data, which goes to 0 as the number of samples N in the training data goes to ∞. Thus, we see that $\hat{\sigma}^2$ is an asymptotically unbiased estimate of the true noise in the data. We also see that $\hat{\sigma}^2$ has σ^2 as lower bound, which in practice means that a finite estimation will overestimate the true value.

Total uncertainty

In order to calculate the total uncertainty of the network for a given input x^*, we estimate the aleatoric uncertainty and the epistemic uncertainty separately by equation 3.8 and equation 3.9, respectively, and combine these terms by equation 3.6. Denoting the epistemic uncertainty η_1 and the aleatoric uncertainty η_2, the total uncertainty, η, can be calculated by

$$\eta = \sqrt{\eta_1^2 + \eta_2^2} \tag{3.11}$$

where we take the root to obtain the uncertainty as standard deviation.

3.4 Anomaly Detection

Anomaly detection refers to the problem of finding patterns in data that do not conform to expected behavior (Chandola et al., 2009). Anomaly detection methods aim to identify the *normal* and *abnormal* state of the data (Mehrotra et al., 2017). Anomaly detection is usually categorized into two main types - *supervised* and *unsupervised* anomaly detection - based on the availability of labels in the data. *Supervised anomaly detection* describes a setup where the training and testing data consist of fully labeled samples (Goldstein and Uchida, 2016). The practical usefulness of supervised detection is limited because the anomalies in the data are already known and labeled, which is not the case in most real-world applications. *Unsupervised anomaly detection*, which is the area of focus in this thesis, is the process of identifying abnormal patterns in completely unlabeled data (Goldstein and Uchida, 2016).

3.4.1 Types of anomalies

Point anomalies are often extremely high or low values that deviate significantly from the norm (Mehrotra et al., 2017). *Spatial anomalies* are single data instances that are considered abnormal from the rest of the data independent of where it occurs (Ahmad et al., 2017). Anomalies can also be *temporal* or *contextual* if it is considered anomalous because of the context where it occurs (Goldstein and Uchida, 2016). Temporal anomalies are often harder to identify as they are more subtle, but they are important to acknowledge as they may provide useful, actionable information about potential problems of a given system (Ahmad et al., 2017). Furthermore, it is common in literature to separate between *global anomalies* - instances considered unusual as compared with the whole dataset - and *local anomalies* - instances that only are unusual from the perspective of a subset of the data (Mehrotra et al., 2017).

3.4.2 Outputs of an anomaly detection algorithm

There are in general two possible outputs of an anomaly detection algorithm (Goldstein and Uchida, 2016). The first possibility is a *label* that is a boolean[9] value that states if the instance is an anomaly or not. The second possible output is an *anomaly score* that represents how abnormal the observation is, which can be more informative than only a label (Mehrotra et al., 2017). In supervised problems, it is common to use the label exclusively, where the anomaly score is more common in the unsupervised case. With the anomaly score, it is common to either set appropriate thresholds that determine when to classify instances as anomalous, or to rank the anomalies and only report the most important ones.

[9] Boolean values are either true or false.

3.4.3 Precision and recall

An ideal anomaly detection algorithm should correctly classify *all* the observations in the data stream (Mehrotra et al., 2017). This is, however, a trade-off because if the algorithm should detect all the anomalies, then the parameters are often set liberally such that it is easier to identify all the anomalies. Similarly, if the algorithm should avoid to falsely classify a normal observation as an anomaly, the parameters are often set more conservative. The precision and recall score evaluate how well the algorithm captures existing anomalies, and how many of these are falsely classified. When the algorithm falsely classifies a sample as anomalous it generates a *false positive (fp)* or a type I error. Similarly, if an anomaly is not identified by the algorithm, it generates a *false negative (fn)* or a type II error. Correct classifications are known as *true positives (tp)* and *true negatives (tn)*.

Precision is a measure of how well the algorithm manages to identify the *true* anomalies, while *recall* measures how well the algorithm captures *all* anomalies. They can be defined as follows:

$$\text{Precision} = \frac{tp}{tp + fp} \quad \text{and} \quad \text{Recall} = \frac{tp}{tp + fn}. \tag{3.12}$$

The precision and recall metrics are not directly applicable in unsupervised anomaly detection problems, but they are important evaluation metrics that are widely utilized.

3.4.4 Concept drift

The behavior considered *normal* in a production system is likely to change over time. This problem is commonly referred to as *concept drift* (Ahmad et al., 2017; Pratama et al., 2016; Gama et al., 2014). For example, maintenance actions, upgrades and change of configurations are likely to occur in a system; hence, the anomaly detection method should be able to adapt to such cases. In real-world scenarios, the environments are often non-stationary and dynamically changing, i.e., the relationship between the input data and the target might change. Moreover, one of the challenges of concept drift handling is to distinguish the true drift from an anomaly or random noise. This is especially difficult in noisy systems. Gama et al. (2014) highlights some critical aspects of concept drift and suggests that both retraining and incremental learning, in addition to online learning is some of the techniques that might be useful for anomaly detection models to adapt to dynamic environments.

3.4.5 Anomaly detection in streaming data environments

Streaming applications involve analyzing a continuous sequence in real-time. The data is arriving continuously with high frequency, and in many applications, the data streams need to be properly processed and classified in a matter of milliseconds (Hundman et al., 2018). The data streams are possibly infinite, resulting in that an algorithm that attempts to store the stream in memory will fail (Tan et al., 2011). Also, uncertainties in the data stream caused by measurement errors or corrupt sensor readings are hard to distinguish from true measurements, making it difficult to discover the true source of anomalies. As a consequence, it is necessary to develop effective methods to process the data streams, identify uncertainties in them, and determine anomalies accurately (Ahmad et al., 2017).

The streaming data is commonly processed and analyzed in two ways: (1) the system classifies and learns in online fashion (Ahmad et al., 2017), or (2) the system trains in batches of a subset of the data in an offline-fashion and classifies new observations in real-time (Hundman et al., 2018). Inspired by Ahmad et al. (2017), we define the following characteristics for an ideal anomaly detection algorithm for real-world streaming data:

1. Predictions are made online, i.e. the model classifies each sample in the stream as it arrives before receiving and processing the next sample.
2. The model should either learn in a safe and liable online fashion, or offline in batches.
3. The model should operate in an unsupervised fashion since no labels are available.
4. The scenario where testing samples come from a different population than the training set should ideally be taken into account since the underlying statistics of the data stream is often non-stationary (see section 3.4.4 on concept drift).
5. An anomaly model should warn about anomalous behavior as reliably and early as possible.
6. The false positives and false negatives rates should be minimized.

Chapter 4

Data and Preprocessing

The data used in this thesis originates from a gas compressor operating on an oil platform in the North Sea. The dataset consists of hundreds of sensor data sources, maintenance logs, P&ID diagrams and production data for the entire lifetime of the equipment. Section 4.1 gives a high-level description of the gas compressor, the surrounding system as a whole, and the physical sources to the data. Section 4.2 explains how the raw data was retrieved, including all necessary steps to repeat the process for anyone interested. The model inputs and outputs are described in section 4.3. Section 4.4 explains the process of partitioning the data into training, validation and test sets, before describing any preprocessing done to the data in section 4.5[1].

4.1 The Gas Compressor and Physical Data Sources

4.1.1 Context

Cognite is a modern IT company that develops a platform to collect, process, analyze and visualize data that originates from various data-intensive industries. Utilizing and contextualizing extensive amounts of hitherto unused data, they implement data-driven solutions that can be used to aid decision-making, production efficiency and maintenance planning. AkerBP is one of Europe's leading oil and gas producers, and is one of Cognite's largest customers. AkerBP operates on several oil fields in the North Sea, amongst them the Valhall platform where the compressor is installed. Together with Cognite, AkerBP has decided to share real data from the production at the platform, making it publicly available online[2]. The project is called ”Open Industrial Data” (OID), and aims to open source industrial data for educational and research purposes, pushing for industrial innovations, and inspiring other players in the industry to do the same.

[1] All code used to extract and preprocess the data can be found in our public Github repository at https://github.com/reitenhalvor/master-thesis.

[2] The online link to the project is: https://openindustrialdata.com/

4.1.2 High level system description

The compressor is an electrically driven, fixed-speed, centrifugal compressor installed on the Valhall platform operated by AkerBP[3] in the North Sea. The equipment is responsible for the first of four stages of compression of natural gas on the platform. The main purpose of the compressor is to increase the pressure of the gas, preparing it for further processing and, finally, export from the production facilities. It receives gas from separators at approximately 3 barg and compresses the gas to approximately 12 barg.

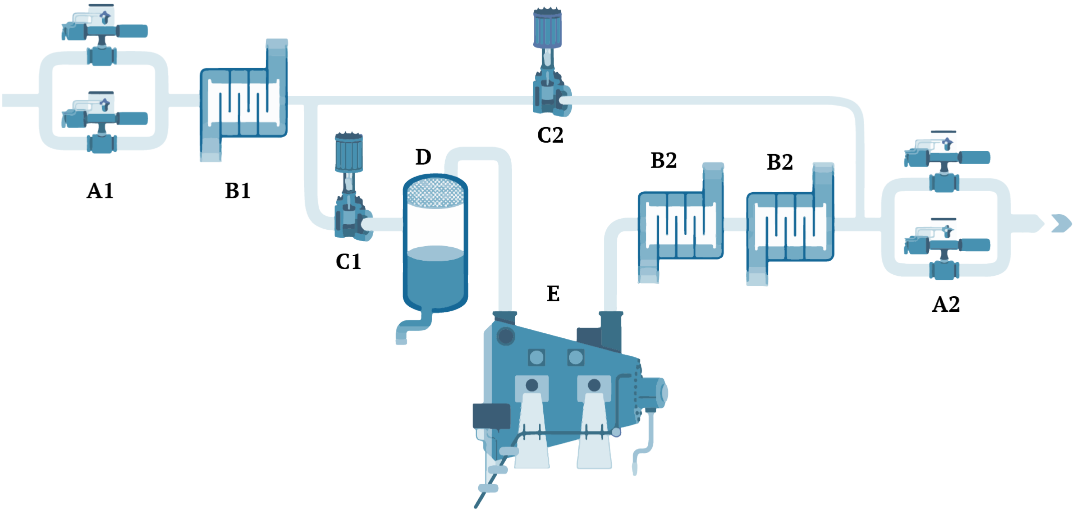


Figure 4.1: A schematic of the compressor and surrounding equipment responsible for the first compression stage. A) Upstream- and downstream valves. B) Heat-exchangers. C1) Suction throttle valve (STV). C2) Anti-surge valve (ASV). D) Suction scrubber. E) The compressor.

Figure 4.1 illustrates the relevant subsystem of the compressor:

A The upstream and downstream valves control the flow of gas in and out of the system. During stand-stills and in the case of incidents, these valves will isolate the system. If an incident occurs, then this stage may be isolated from other stages to avoid cascading errors, backflow through the compressor and breakdown.

B Suction and discharge coolers lower the temperature of the gas. These are shell-and-tube heat-exchangers that are regulated with cooling medium. The gas needs to be cooled down before the suction scrubber stage as well as after the compression stage, where the temperature increases because of the increase in density.

[3] More information on Valhall is found at https://www.akerbp.com/en/our-assets/production/valhall/

C Control valves

- **C1** The suction throttle valve regulates the pressure and volume of the gas being sent through the scrubber and the compressor.
- **C2** The anti-surge valve protects against surge[4] by regulating the amount of fluid to recycle. Note that the gas flows from *right to left* through this valve, which is reversed compared with the other components in the schematic.

D The suction scrubber removes excess liquid droplets from the gas in order to protect the compressor.

E The compressor compresses the gas, increasing density and pressure.

In addition, there are a number of utility systems related to the compressor. This is for instance the lubrication system for the compressor, the dry gas seal system that prevents gas inside the compressor from escaping, as well as the condition monitoring system (CMS). The CMS measures the vibrations and temperature in the motor, gearbox and compressor.

4.2 Data Collection

The data is available through a public Application Programming Interface (API) provided by Cognite. In order to access the API, one must first retrieve the proper credentials, where the procedure is explained in detail at the Open Industrial data homepage[5]. To ease the data manipulation process, Cognite has developed a Software Development Kit (SDK) in the Python language. The SDK is arguably the easiest way to access and download the data, although it is possible to use the API directly through HTTP-requests and any preferred framework that supports this. In addition, the entire dataset is available on the Cognite Operational Intelligence platform[6], making it easy to search for specific tags and view plots of the available time series in a web browser.

4.3 Selecting Model Inputs and Outputs

In total, there are more than 350 available time series for the subsystem of the compressor. Of these data sources, we must choose which ones to include as input features and output targets for the model. For the purpose of this thesis, the data sources can effectively be divided into three main categories: *input sensors*, *output sensors* and *controls*.

[4] Surge is a phenomenon occurring at low mass flows characterized by large amplitude oscillations in the flow and pressure of the fluid, with a potential reversal of flow inside the compressor. This can cause serious harm to the compressor itself and other system components (Helvoirt, 2007).

[5] https://openindustrialdata.com/

[6] https://opint.cogniteapp.com/

The *input sensors* are time series selected as sensory inputs to the model. These sensors are exclusively placed upstream of the compressor, which is *before* the compression takes place. That being said, any numerical data source before compression is a potential input feature to the model.

The *controls* are time series that contain information on the control parameters of the system, such as compressor engine running speeds, supplied power to the engine, and valve opening positions. These are not sensory data per se but are essential in explaining the system behavior.

The *output sensors* are the target variables that the model should predict. These are selected sensor sources placed directly downstream of the compressor, measuring physical parameters of the compressed gas.

The naming of the sensors/tags follows industry standards. The relevant tags in this thesis are ”PI/PT/PDT” (pressure), ”TI/TT/TIC” (temperature), ”FI/FT” (flow), ”ZI/ZT” (valve indicators) and ”KA” (power indicators).

4.3.1 Model outputs

The outputs of our models are defined as the sensors that measure specific parameters of the gas directly downstream of the compressor. Considering the ideal gas equation,

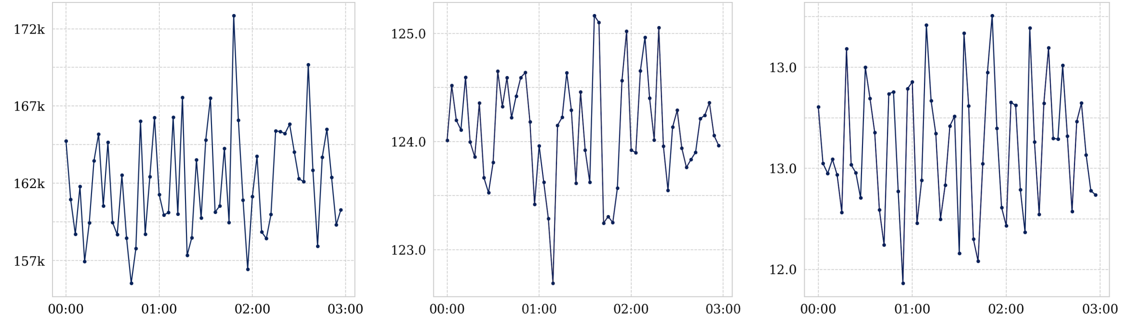
$$PV = nRT \tag{4.1}$$

where P is the pressure, V is the volume, n is the number of moles, R is the universal gas constant and T is the temperature, we see that *temperature*, *flow* (by mass or volume) and *pressure* is sufficient to adequately describe a gas. We, therefore, choose the first sensors measuring the temperature, flow, and pressure after the compression stage, listed in table 4.1 with their respective units. Figure 4.2 shows samples of these data sources.

Tag	**Description**	**Unit**
VAL_23-FT-92537-01:X.Value	Compressor discharge mass flow	kg/h
VAL_23-TT-92539:X.Value	Compressor discharge temperature	degC
VAL_23-PT-92539:X.Value	Compressor discharge pressure	barg

Table 4.1: The tags selected as outputs for the model, which the model should try and predict given its inputs. See figure 4.3 for their physical placement in the system.

It should be noted that there are other sensors that measure the discharge mass flow in other units, such as `VAL_23-FT-92537-04:X.Value`, measuring the volume flow. However, seeing that the mass flow is independent of temperature and pressure while this is not the case for the volume flow (Helvoirt, 2007), the mass flow measurement sensor is preferred.



(a) Discharge flow $[kg/s]$ **(b)** Discharge temperature $[{}^{\circ}C]$ **(c)** Discharge pressure $[barg]$

Figure 4.2: Four hours of sample data for the selected target tags. The samples are extracted from the 24th of December 2017.

4.3.2 Model inputs

The inputs to the model are a combination of input sensors and controls. As there are a a high number of data sources available, we first need to extract the ones that are relevant as model inputs, before analyzing the subset of features to find which ones are beneficial to include in the model.

Finding relevant inputs

The total number of available tags count to approximately 350, but many of these sensors are either placed downstream of the compressor, or they are non-numerical features, disqualifying them as possible candidates as inputs to the model. By consulting industry experts with domain knowledge and by analyzing P&IDs[7], we reduced the number of potential data sources from 350 to 146. However, most of these features are characterized by high sparsity, meaning that they contain a high number of undefined or missing values in the chosen period. Based on the plot in figure B.1, we set a percentage threshold for maximum sparsity at 2.5 percent, and removed any features exceeding this threshold. This effectively reduces the number of potential inputs to 32.

Feature selection

The remaining 32 features are not all equally informative. Some are noisy and only pollute the training process, many of them are highly correlated and contain mutually overlapping information, others are nearly constant variables across the whole dataset with approximately zero variance, while some are simply irrelevant. *Feature selection* is the process of selecting the most

[7] Piping & Instrumentation Diagrams

influential input features with respect to the target features (Chollet, 2018). The feature selection process results in a dataset with reduced dimensionality which is beneficial for a number of reasons; the datasets are more comprehensible and easier to work with, they are less demanding computationally to process and analyze, and will most likely lead to increased performance of the machine learning models (Goodfellow et al., 2016; Aha et al., 1996).

Beginning with analyzing the cross-correlation between the features in the dataset, we see from figure B.2a that there are, in general, high linear correlation between many of the variables. Specifically, we see high correlation between similar tags, such as sensors measuring pressure, temperature and flow, respectively, at different positions in the subsystem. This is not surprising, as a pressure measurement at one point is expected to be relatively equal to the next sensor measuring the pressure further down the pipeline. In essence, two highly correlated features will contain much of the same information, and keeping both can be considered redundant (Mostafa et al., 2012). However, Guyon and Elisseeff (2003) argue that high variable correlation does not necessarily mean absence of variable complementarity, but considering the aforementioned advantages of dimensionality reduction, we see it as beneficial to remove highly correlated features. If any two predictors have a Pearson correlation coefficient greater or lower than ± 0.95, we selected one of them arbitrarily. This results in a reduction of potential inputs to 21.

As a final step of the feature selection, we apply the Boruta algorithm for each target using the remaining 21 input features. Degenhardt et al. (2017) carried out an extensive comparison of feature selection techniques and found Boruta to be the most powerful technique on high-dimensional datasets. This feature selection algorithm is briefly described in appendix A.1.1. Figure B.3 shows the relative importance of each input feature with respect to each target variable. For the discharge flow, we see that the algorithm classifies *all* variables as relevant, implying that all variables are considered more descriptive than random noise. However, two features are considered considerably more important in predicting the discharge flow, hence we select these two features as predictors for our model. Similarly, we perform the same procedure for the discharge temperature and pressure, which results in a subset of six input features.

Finally selected inputs

The selected features are listed in table 4.2 along with their respective units, where *degC* is temperature in Celcius, *barg* is bar gauge pressure[8] and *mbar* is millibar. Figure 4.3 shows an overview of both the input and output sensors and their physical positioning in the subsystem of the compressor.

[8] The bar gauge pressure is the pressure in bar above or below the atmospheric pressure.

Tag	Description	Unit
VAL_23_TT_92532:Z.X.Value	Compressor suction temperature	degC
VAL_23-PT-92523:X.Value	Pressure in suction scrubber	barg
VAL_23-PDT-92534:X.Value	Differential pressure between separator and compressor suction side	mbar
VAL_23_ZT_92543:Z.X.Value	Anti-surge valve position	%
VAL_23-TIC-92504:Z.X.Value	Temperature out of suction cooler	degC
VAL_23_KA_9101_M01_62C:Z.X.Value	Power supplied to the compressor	kW

Table 4.2: The tags selected as inputs for the model, which are a combination of input sensors and controls.

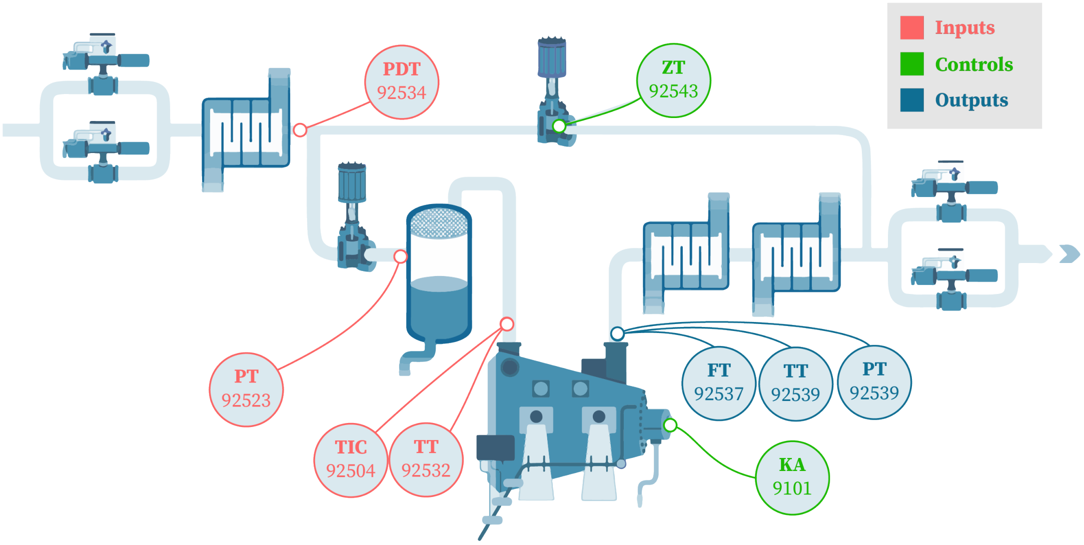


Figure 4.3: An overview of the selected tags and their physical positioning in the system. The red represent input sensors, the green are controls, while the blue are the output targets. For an explanation of the other components in the schematic, see figure 4.1 and the corresponding text.

4.3.3 Time frames, aggregates and other specifications

Although approximately six years of data is available, we only train the model on a fraction of this. In order to reconstruct the normal behavior of the compressor, the model needs to train on data where it behaves non-anomalous. A two-month window of seemingly normal operation was found, from the 24th of December 2017 to the 24th of February 2018. While the sampling rate of most of the selected sensors is on second-basis, implying 60 recordings per minute, we aggregate the measurements to one sample per minute by taking the average. The aggregating function is performed on consecutive time windows that do not overlap. One sample for each feature every minute in this period yields a dataset of 88685 observations for each of the six input features and three targets.

4.4 Data Partitioning

The downloaded data is split into a training set (60% of the data), a validation set (20%) and a test set (20%).

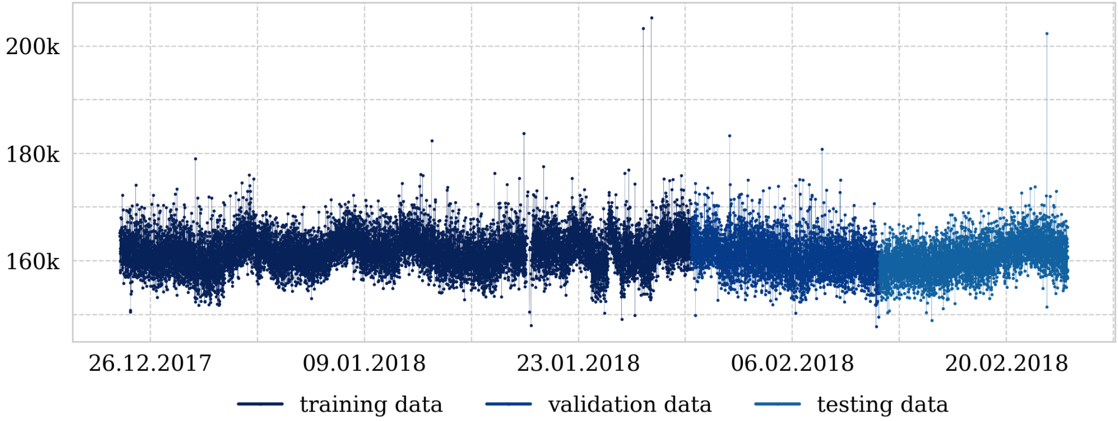


Figure 4.4: Defined training (dark blue), validation (blue) and test (light blue) regions of the dataset, here illustrated by the discharge flow.

It is common practice in traditional machine learning problems to partition the data by random sampling such that each subset contains a representative sample of the data (Goodfellow et al., 2016). This is, however, not the case for time series modeling, where the temporal aspect matters. As recommended by Chollet (2018), we do not shuffle the data but define consecutive intervals that define the training, validation and test regions, respectively. This is illustrated in figure 4.4.

The training set is used to train the model, while the validation data is used for model selection; the process of finding the best parameters, architecture and composition of a model. However, since the validation set is actively used in the model selection process, we use the test set to obtain a final realistic estimate of the generalization error.

4.5 Data Preprocessing

The raw data fetched from Cognite's API is not directly applicable in a learning context. As such, several preprocessing steps must be performed such that the data is transformed into a format that is understandable for the machine learning models. In general, this includes processes such as *data cleaning*, *handling missing values*, *outlier handling* and *data scaling* (Goodfellow et al., 2016). For time series specifically, it is important that the dataset contains one row per consecutive time step, one column per input variable, and one column per output variable (Brockwell and Davis, 2016). The rest of this section explains the data preprocessing steps that have been performed on the data.

4.5.1 Handling missing data

The dataset cannot contain missing data, i.e. each feature must have a defined value for every time observation. The average sparsity of each feature in the dataset for the selected period is approximately 0.4%, with a total sparsity of 3.8% for the entire dataset (3361 out of 88685 instances missing). When the preceding and succeeding value exist, we use linear interpolation to impute the values. This effectively reduces the total sparsity percentage in the dataset across all variables to 0.6% (552 missing observations). The remaining missing data are imputed using the Amelia II algorithm, which is briefly described in appendix A.1.2.

4.5.2 Handling outliers and extreme values

An outlier is a point or an instance that is distinctly different from the norm of the data it is part of (Chandola et al., 2009). Such instances are rare events that are usually either extremely high or low values that may obscure the training process (Goodfellow et al., 2016). The aim is to model the normal, stable behavior of the compressor. Therefore, anomalous data should ideally be removed from the dataset to capture the general case.

It is a non-trivial task to identify anomalies in the dataset. Specifically, it is hard to distinguish whether the anomaly is an actual anomaly, or if it stems from malfunctioning sensors or noise in the measurements. The most straightforward approach for identifying outliers in a dataset, regardless of their source, is to compare the specific instance to the variables mean and standard deviation. A data point is considered normal if it falls within the limits of $x = \mu \pm k \cdot \sigma$, where μ is the mean, k a constant[9] and σ the standard deviation. After identifying the outliers, a common approach is to remove all values for this time step. However, this is problematic for time series because the temporal continuity is disrupted if any instances are removed.

We recognize that there exist several approaches to handle outliers, such as *capping*[10] and *re-imputation*[11], but we do not find any of these to fit our problem, as we essentially replace one source of uncertainty with another. Consequently, we do *not* omit outliers in this thesis.

[9] Normally, k is set to equal 3.

[10] Capping involves setting any values that exceed a threshold to the maximum or minimum of this threshold

[11] Re-imputation is a technique where outliers are set to undefined values, before re-imputing them using an appropriate algorithm.

4.5.3 Scaling by standardization

Many machine learning methods are sensitive to the format of the input data (Chollet, 2018). Especially deep learning models have been found to perform better when continuous numerical variables have similar scales (Goodfellow et al., 2016). There are two common approaches to data scaling, namely *standardization* - transforming each feature to zero mean and unit variance - and *normalization* - transforming the data in the range from 0 to 1. Normalization is the simpler procedure of the two but is highly sensitive to outliers. Potentially, if the dataset has single large outliers, the normal instances will be squashed together in a small sub-range. Standardization is less affected by outliers, but is generally more effective when the features fit a Gaussian distribution. As we see in figure B.4, the variables in the dataset seem to follow a Gaussian distribution to a reasonable degree, hence we use standardization as the scaling method of choice for the data. The mean and standard deviation of every variable in the training data are used to scale the data in the training, validation and testing set. The transformation is given by

$$x_{scaled} = \frac{x - \mu}{\sigma} \tag{4.2}$$

where μ and σ is the variable's mean and standard deviation, respectively.

Chapter 5

Method

This chapter outlines the methods used to implement the predictive models and anomaly detection algorithms. We begin by defining a set of objective evaluation metrics that are used to evaluate the performance of the implemented models. This makes it possible to compare the performance of the deep learning models to each other, but also to compare them less advanced baselines such that we can assess the relative added value that follows with the increase in complexity. To this end, we implement a set of common-sense heuristics and simpler machine learning techniques. In addition, we define techniques to assess the predictive uncertainty of the deep learning models. With the evaluation metrics, benchmarks and uncertainty assessment methods in place, we experiment extensively with deep learning models of varying types and designs, before selecting a final set of promising models. The most accurate model is used in the anomaly detection methods, which we apply to three datasets that show clear abnormal patterns to evaluate the effectiveness of the methods. The results are used to thoroughly investigate the potential of deep learning for modeling the compressor, and whether the proposed anomaly detection methods are suited in this context.

5.1 Constructing the Predictive Model

The first building block of our anomaly detection methods is a predictive model that reconstruct the normal behavior of the compressor. Similar to Ahmad et al. (2017) and Hill and Minsker (2010), we construct models that predict the *one-step-ahead* outputs of the compressor. We are interested in finding the model with the best performance in simulating the behavior of the compressor, since a model with higher prediction accuracy will generate fewer false anomaly warnings with the model-based anomaly detection methods we implement.

5.1.1 Evaluation metrics

Evaluation metrics quantify the distance between the observed and predicted values. In the case of regression problems, common distance metrics are the *Root Mean Squared Error* (RMSE) or *Mean Absolute Error* (MAE) (Chollet, 2018). RMSE and MAE, given the predictions ($\hat{\mathbf{y}}$) and true targets ($\mathbf{y}$), are defined as:

$$\text{RMSE}(\hat{\mathbf{y}}, \mathbf{y}) = \sqrt{\frac{1}{N}\sum_{n=1}^{N}(\hat{y}_n - y_n)^2} \tag{5.1}$$

$$\text{MAE}(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{N}\sum_{n=1}^{N}|\hat{y}_n - y_n| \tag{5.2}$$

Both measure the average prediction error, they are non-negative and negatively-oriented (i.e., lower values are better scores). The RMSE takes the square of the errors before they are averaged, thereby giving more substantial biases a higher weight. The MAE calculates the average value of the bias regardless of direction, and every observation will then be weighted equally. This makes the RMSE relatively more suited than the MAE if large errors are particularly unwanted. However, we see the modeling of such spikes important in the modeling of the compressor, and as such, we apply MAE as our choice of performance metric. Hence, the MAE defines the loss function that the machine learning models try to minimize in the training process. While the inclusion of RMSE as an additional performance metric could have allowed for a deeper understanding of the models' strengths and weaknesses, we choose to omit it altogether for the sake of brevity.

5.1.2 Dealing with the temporal aspect for forecasting

x_t^1	x_t^2	x_t^3	y_t	y_{t+1}	y_{t+2}
1	10	3	1	2	3
8	13	8	2	3	4
90	39	16	3	4	5
35	33	13	4	5	-
7	14	14	5	-	-

Table 5.1: The concept of *lagged variables*. The target rows are shifted $k = [0, 1, 2]$ rows such that the inputs at time t corresponds to the output at time $t + k$.

The model should predict the *one-step-ahead* outputs of the compressor, y_{t+1}, given the inputs at the current step, X_t. Consequently, we introduce the concept of *lagged variables*. For a given target y_t, a lagged target y_{t+k} is generated by shifting the observations k rows such that the inputs at time t corresponds to the outputs at time $t + k$. This is exemplified in table 5.1, where the variable y is shifted two time steps. Notice that the inputs ($\mathbf{x}$) remain untouched, and it is only the target variables ($\mathbf{y}$) that are shifted. As a result, the inputs at time t corresponds to the outputs at time $t + k$, where k is the lag constant. As can be seen from the table, the introduction of lagged variables effectively means that the k last number of observations cannot be used as the lagged variables have undefined values. This results in the total number of observations in the dataset being reduced by k. In this thesis, we set the lag constant to $k = 1$.

5.1.3 Benchmarks

The benchmarks serve as a sanity check that the more complex deep learning models have to outperform in order to demonstrate usefulness. We develop benchmarks of two categories, namely common-sense heuristics and simpler machine learning techniques by linear models and tree-based methods. Heuristics are naive methods that follow strict, hard-coded rules to make predictions; hence, they reflect how well the compressor can be modeled without actually building a model. The machine learning baselines are shallow methods that are much easier to implement than the deep neural networks, even though they require much of the same data preprocessing steps. Common for all of the baselines is that they are statistically simple, computationally lightweight methods, and their implementation requires little in terms of technical proficiency.

Common-sense heuristics

We implement three heuristics. The first heuristic estimates the value at time t based on the observed preceding value. Hereafter, this heuristic is referred to as ”Prev”, and can more formally be defined as:

$$\hat{y}_t = y_{t-1} \tag{5.3}$$

where $\hat{y}_t$ is the prediction for target y at time t, and y_{t-1} is the true value of the sequence in the previous time step. Naturally, this heuristic will become less accurate the more time steps ahead the model predicts. This thesis is focused with predicting *one-step-ahead*, and as such, this benchmark will form a strong baseline as there is no reason to expect either of the output parameters to make big jumps from one time step to the next with a fine granularity of 60 seconds.

The second heuristic is one that constantly predicts the variable’s mean in the training data, hereafter referred to as ”Mean”. This can define a strong benchmark if the target variable only oscillates around the mean of the training data. More formally, this heuristic is defined as

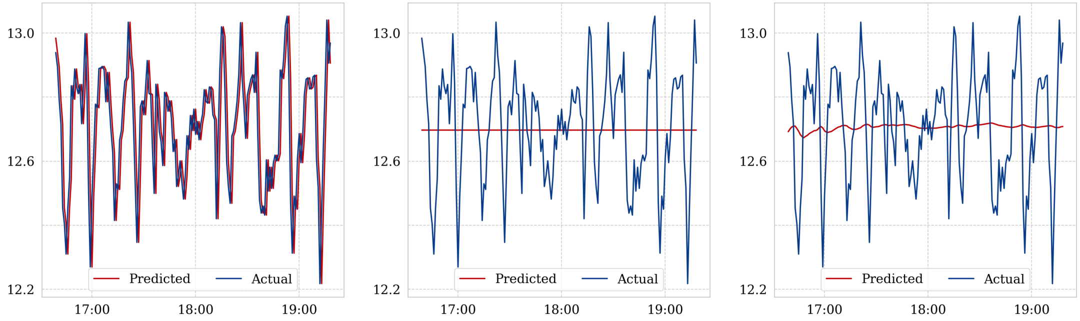
$$\hat{y}_{T+k} = C = \frac{1}{T}\sum_{t=0}^{T} y_t \tag{5.4}$$

where y_{T+k} is the prediction at time step $T + k$, T being the last time step included in the training data and k a constant > 0.

The third and final heuristic predicts the moving average of the data seen so far, and is hereafter referred to as ”MA”. Formally, this heuristic can be described as follows:

$$\hat{y}_{t+1} = \frac{1}{t}\sum_{i=0}^{t} y_i \tag{5.5}$$

As this heuristic incorporates the target sequence in its calculations, it is expected that this heuristic gives a stronger baseline than what ”Mean” can. The three heuristics are illustrated in figure 5.1.



(a) Heuristic: ”Previous” **(b)** Heuristic: ”Mean” **(c)** Heuristic: ”Moving Average”

Figure 5.1: Samples of the common-sense heuristics in predicting the discharge pressure; ”Prev” in (a), predicting the next value of the sequence to equal the previously observed value; ”Mean” in (b), predicting the values constantly to equal the mean of the training data; ”MA” in (c), predicting the previous value of the moving average of the sequence.

Simple machine learning models

Our set of benchmarks also includes linear models, which can be solid performers if there are strong linear dependencies between the inputs and outputs. We implemented ordinary least squares linear regression and shrinkage methods, where the theory for these techniques is described in section 3.2. Specifically, we implemented *ridge regression* (ridge), *least absolute shrinkage and selection operator* (LASSO), and *Elastic Nets* (ElNets) - a hybrid between ridge and LASSO regression. These are models that include regularization terms in their minimization objectives, and the degree of regularization is adjusted with tuning parameters λ and α. Optimal values are found by values are found by hold-out validation using an exhaustive grid search, where we experiment with values in the range $\lambda, \alpha = [0.01, 0.02, ..., 1.0]$.

To account for possible non-linear relationships in the data, we implement two tree-based methods called Random Forest (RF) and Gradient Boosting Machines (GBM). Tree-based methods may potentially learn statistical structures that the linear models cannot and are therefore considered stronger baselines. Random Forest and Gradient Boosting Machines combine many decision trees to make a prediction as discussed in greater detail in section 3.2.2. The essential tuning parameter for these models is the number of trees used to fit the models. We let the number of trees vary from 100 to 3000 with an increment of 100 to train a model for each configuration and evaluate it on the validation set. We choose the parameter that produces the overall lowest MAE on the validation set.

It should be noted that the simple machine learning models are not supplied with historical variables or any temporal context. Thus, the baselines are trained to map the inputs to their corresponding outputs without any historical information. The historical inputs could potentially have been included in the dataset as additional variables, and could possibly have strengthened the models' predictive performance, but would result in a big expansion of the dimensionality of the dataset. Considering this, and that we want to keep the technical implementation of these models as simple as possible, we have chosen to omit the inclusion of historical variables for these models.

5.1.4 Deep learning models

This section outlines the implementation and design choices of the deep learning models used to predict the *one-step-ahead* sensor outputs of the compressor. Inspired by related work, we explore three types of deep learning networks: A feed-forward neural network, a Long Short-Term Memory network and a Gated Recurrent Unit network.

Hyperparameters, topology, and architecture of a network

Neural networks have many adjustable *hyperparameters*, e.g. activation functions, learning rates, layer initialization functions, and optimization algorithms. Ideally, one would perform an exhaustive grid search to find the optimal hyperparameters of a network: training and evaluating a model for each unique configuration in a vast space of potential parameters. However, the computational resources needed to train a single network involve significant training times, making this type of hyperparameter tuning intractable in practice. Alternative approaches such as *random sweeps*[1] exist, but can be problematic for the same reason.

It is also necessary to specify the *topology* of the model, i.e. the model's building blocks and functional components. For a traditional feed-forward neural network, this is the number of layers in the network and the number of nodes in each layer. Together, the topology and hyperparameters of a network constitute its *architecture*. While exhaustive searches can be applied to find suitable architectures, these suffer from the same problems that they are notoriously expensive to train. Because of this, a more intuitive approach is taken when finding suitable hyperparameters and

[1] A random sweep entails training multiple networks with random parameter configurations from a system-defined parameter range, and choosing the best configuration based on the performance on the validation set (Goodfellow et al., 2016).

design architectures. Inspiration is gathered from the literature and previous implementations, and iterative, manual experimentation is performed with various configurations.

General design choices

All of the implemented networks have some general design choices in common:

- **Error metric**: The Mean Absolute Error, as defined in section 5.1.1, is used as the loss function.
- **Optimizer**: The ADAM optimizer is used. Literature states this as an effective choice that works well in most cases using the default values (Goodfellow et al., 2016; Chollet, 2018). We tested the RMSprop optimizer but found it inferior to ADAM.
- **Learning Rate**: The learning rate is initially set to 0.001, which is the default of the ADAM optimizer in Keras[2]. We also specify a procedure such that the learning rate is halved if the training loss has not improved in five or more iterations.
- **Number of epochs**: The number of epochs, i.e. the number of training rounds of the network, is set to 50. This is limited by computational resources and convergence rates of training and validation errors. 50 epochs is a reasonable compromise which allows for a low final training error and a convergence of the validation error.
- **Regularization techniques:** Regularization is implemented by *dropout*, a very common form for regularization that helps with overfitting and speeds up the training process (Goodfellow et al., 2016). As per the research of Gal and Ghahramani (2015), both a time-constant dropout and a recurrent dropout is applied both in the training *and* prediction phase of the recurrent neural network. For the standard neural network, only time-constant dropout is applied. The dropout rates are set between 10% to 30% for both time-constant dropout and recurrent dropout and are applied for every layer.
- **Batch size**: Instead of using the whole dataset at once when training the model, a batch size is specified such that the model only trains on one batch at a time. The batch size is set constant to 256.
- **Lookback**: The lookback parameter defines the fixed window-size of observations when making the one-step-ahead prediction. This parameter is set to 360. With a granularity of 60 seconds, the network will use the previous six hours of data in order to make a prediction. The standard feed-forward neural network does not utilize historical variables, so the lookback parameter is only relevant for the recurrent neural networks.

[2] https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam

- **Activation functions**: The ReLU-function is used as an activation function wherever relevant, which Goodfellow et al. (2016) argues is a suitable choice.

Final model designs

Even though our experimentation resulted in tenfolds of candidate models, we only present the most successful here for the sake of brevity. Inspired by related work, we design sequence specialized deep networks in the form of LSTM and GRU networks that have seen immense success in similar modeling tasks. A GRU network is a simplified version of the LSTM, and is known to require less in terms of computational resources at the cost of having less predictive power (Chung et al., 2014). As such, we experiment with various architectures using LSTM and GRU to select which one suits our problem best. Our final sequential deep networks are simple implementations that contain a single hidden layer (a GRU or LSTM-layer) with 128 nodes. We experiment with bigger networks with a more advanced topology, but find that going deeper than one layer, or wider than 128 nodes in the hidden layer, has a negligible or worsening effect. In addition, we research the effect of increasing or decreasing the window size, but find that the specified six hours of prior data is an acceptable setting.

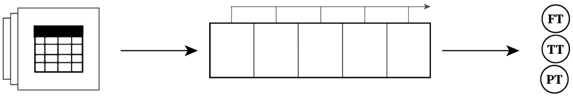


Figure 5.2: Sketch of the LSTM model. The network consists of one hidden layer of 128 nodes with 30% dropout, where both time-constant and recurrent dropout is applied. The model takes an input sequence consisting of a window of six hours of prior data (360 observations) to make a prediction.

In addition, we implement standard feed-forward neural networks. These networks are not supplied with the historical data, hence they are trained to transform the inputs to the outputs at a specific time. We explore how the inclusion of temporal variables affect the model, but find a decreasing performance of the neural networks as the dimensionality increases. We try many different network architectures, but similar to the sequential models we found that small, simple networks work best. The final MLP consists of a single hidden layer with 1024 nodes and time-constant dropout of 30% is applied for the hidden layer.

We also evaluate the performance of *ensembles* of some of the aforementioned models. The concept of *model ensembling* is explained in section 3.3.7, and represent techniques to combine the predictions of many models to potentially get more accurate predictions. A good ensemble relies on diversity, where the constituent models should be as different and as accurate as possible (Dietterich, 2000). Hence, we look at combinations between the feed-forward neural network and recurrent neural networks, which arguably will have highly different hypothesis spaces based on their inputs and architectures.

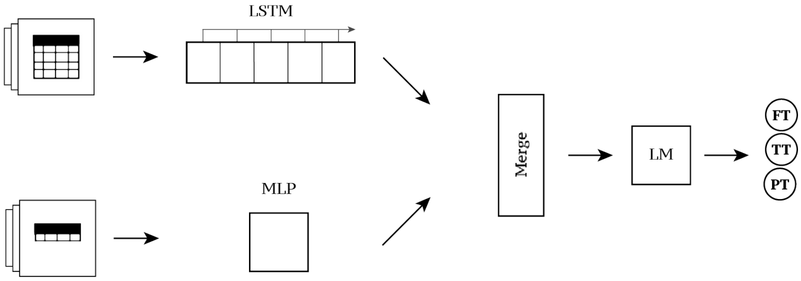


Figure 5.3: Sketch of the LSTM/MLP ensemble model. The MLP makes predictions based on inputs at the preceding time step, while the LSTM uses a window of the 360 prior inputs to make a prediction. The predictions of both models are combined through a linear regression model, that weights the respective predictions and produces the final prediction for the three targets.

The most straightforward way to combine models is to weight their predictions equally by averaging (Chollet, 2018). Arguably, a better technique is to use a weighted average of the predictions and feed the inputs of the ensemble models to a new, separate model that learns how to combine them in an optimal way. We apply the latter of these methods, where we feed the predictions of the constituent models to a linear regression model that learns how to weight the respective models such that it minimizes the validation error. Specifically, we combine the predictions of the LSTM network and the feed-forward neural network with a linear regression model and explore the potential performance gains from this ensemble.

Uncertainty assessment

For any input to a model, we are interested in the predictive mean, $\hat{y}_t^*$, and the predictive uncertainty, η_t. The pseudocode in algorithm 1, which is inspired by Zhu and Laptev (2017), explains how these are estimated. It can be seen as a three-step process.

The first step is to estimate the model uncertainty (epistemic uncertainty). The network applies Monte Carlo dropout in every layer at inference time, and each prediction of the model can be seen as a stochastic realization of the predictive distribution (Gal and Ghahramani, 2015). For an input x^*, the network makes B predictions, resulting in a sequence of predicted outputs $[\hat{y}_{(1)}^*, \hat{y}_{(2)}^*, ..., \hat{y}_{(B)}^*]$. Assuming a Gaussian distribution, the sampled variance can be calculated by $\frac{1}{B}\sum_{b=1}^{B}(y_{(b)}^* - \bar{y}^*)^2$. This represents the model uncertainty, denoted η_1^2.

The second step is to estimate the noise in the data (aleatoric uncertainty). This is done by evaluating the residual sum of squares of the network on the validation data set. For each set of inputs in the validation data, we calculate the mean prediction of the network and evaluate the residual sum of squares. The result represents the aleatoric uncertainty, denoted η_2^2.

The third and final step is to combine the epistemic uncertainty and the aleatoric uncertainty to obtain the final uncertainty estimate. This is given by equation 3.11 and this measure can be used to calculate prediction intervals for the specific input.

Algorithm 1 Uncertainty inference of a neural network with MC Dropout

```
Input: input point $x^*$, a trained prediction network $f^{(\theta)}(\cdot)$, number of predictions $B$
Output: mean prediction $y^*$, prediction uncertainty $\eta$
// estimate mean prediction and epistemic uncertainty
for $b = 1$ to $b = B$ do
    $\hat{y}^*_{(b)} \leftarrow f^{(\theta)}(x^*)$
end for

// prediction
$\hat{y}^* \leftarrow \frac{1}{B}\sum_{b=1}^{B} \hat{y}^*_{(b)}$

// epistemic uncertainty
$\eta_1^2 \leftarrow \frac{1}{B}\sum_{b=1}^{B} (y^*_{(b)} - \hat{y}^*)^2$

// estimate aleatoric uncertainty on validation set
for $x^{'}_v$ in $\{x^{'}_1, ..., x^{'}_V\}$ do
    // get mean prediction for every $x^{'}_v$
    $y^{'}_v \leftarrow \frac{1}{B}\sum_{b=1}^{B} f^{(\theta)}(x^{'}_v)$
end for
$\eta_2^2 \leftarrow \frac{1}{V}\sum_{v=1}^{V} (\hat{y}^{'}_v - y^{'}_v)^2$

// total prediction uncertainty
$\eta \leftarrow \sqrt{\eta_1^2 + \eta_2^2}$
return $\hat{y}^*, \eta$
```

An important remark is that the uncertainty estimates implemented in this thesis are *point-specific*. The noise estimations are global, but the model uncertainty is only defined for specific input points. Moreover, expected values can be calculated across multiple input points, but these are not necessarily informative of the general model uncertainty. Uncertainty measurements are therefore treated on a point-by-point basis in this thesis.

Evaluating model architectures

Minimum Validation Loss (MVL) is a popular method to reduce the effect of overfitting in the model selection process (Chollet, 2018). Whenever the validation loss improves during the training process, the weights are stored such that they can be loaded at a later stage. Consequently, the model is not forced to use the weights from the *last* training round, but rather the weights that produced the minimum validation error during the training process. The validation error is a biased estimate of the true generalization error, as the model is slightly overfitting to the validation set because it has been actively used in the model selection process. To get a more realistic out-of-sample performance estimate, final model architectures are evaluated on the test set. The test error is the metric that ultimately is used to compare a model to another.

5.2 Anomaly Detection

Automated monitoring of hundreds of unlabeled streaming observations requires a fast, unsupervised approach for determining if observed values should be characterized as abnormal. Anomaly detection on streaming data is challenging since data arrives with high velocity and requires instant processing and analysis.

Our anomaly detection models receive a continuous stream of data on the form

$$..., \vec{X}_{t-2}, \vec{X}_{t-1}, \vec{X}_t, \vec{X}_{t+1}, \vec{X}_{t+2}, ...$$

where $\vec{X}_{t+k}$ for $k \in \mathbb{Z}$ denotes the model inputs at time $t+k$. The observation at time $t+k$ should be classified before time $(t+k)+1$, and since the predictive models implemented in this thesis are trained in an offline fashion, this is the only decision that needs to be made at each time step.

Inspired by the findings in the literature, we propose two anomaly detection methods: residual-based and prediction interval-based anomaly detection. Both of these methods apply a predictive model that reconstructs the normal behavior of the compressor, and identifies abnormalities as described in chapter 2. The remainder of this section describes the two anomaly detection methods and how they are evaluated.

5.2.1 Residual-based anomaly detection

The first method is mainly inspired by Ahmad et al. (2017) and is hereafter referred to as *method I*. This method calculates the residuals between the observed and the predicted output values for the compressor, and identifies anomalies based on a comparison between the residual and a predefined residual distribution built on the validation set.

The schematic in figure 5.4 presents the overall architecture *method I*. Given a trained model $f^{(\theta)}(\cdot)$, the residuals on the validation set can be calculated by $\mathbf{e}^{(V)} = \mathbf{y}^{(V)} - \hat{\mathbf{y}}^{(V)} = [e^{(t-V)}, ..., e^{(t)}]$, where V is the number of observations in the validation set and $\hat{y_t}$ is the prediction of $f^{(\theta)}$ with the corresponding input x_t. The resulting sequence of V residuals is then smoothed using a simple moving average window of size $m = 5$ in order to dampen spikes in the residuals that are caused by particularly noisy data (Shipmon et al., 2017). Hence, the smoothed sequence of residuals can be defined as

$$\mathbf{e}_s^{(V)} = [e_s^{(t-V+m)}, ..., e_s^{(t)}]. \tag{5.6}$$

The sequence of smoothed residuals is modeled as a normal distribution, where the parameters of the distribution are estimated using Maximum Likelihood Estimation. This distribution defines the *residual distribution*, which is a key component for this method. For each new observation, we first calculate the residual between the observed and the predicted value. Subsequently, we

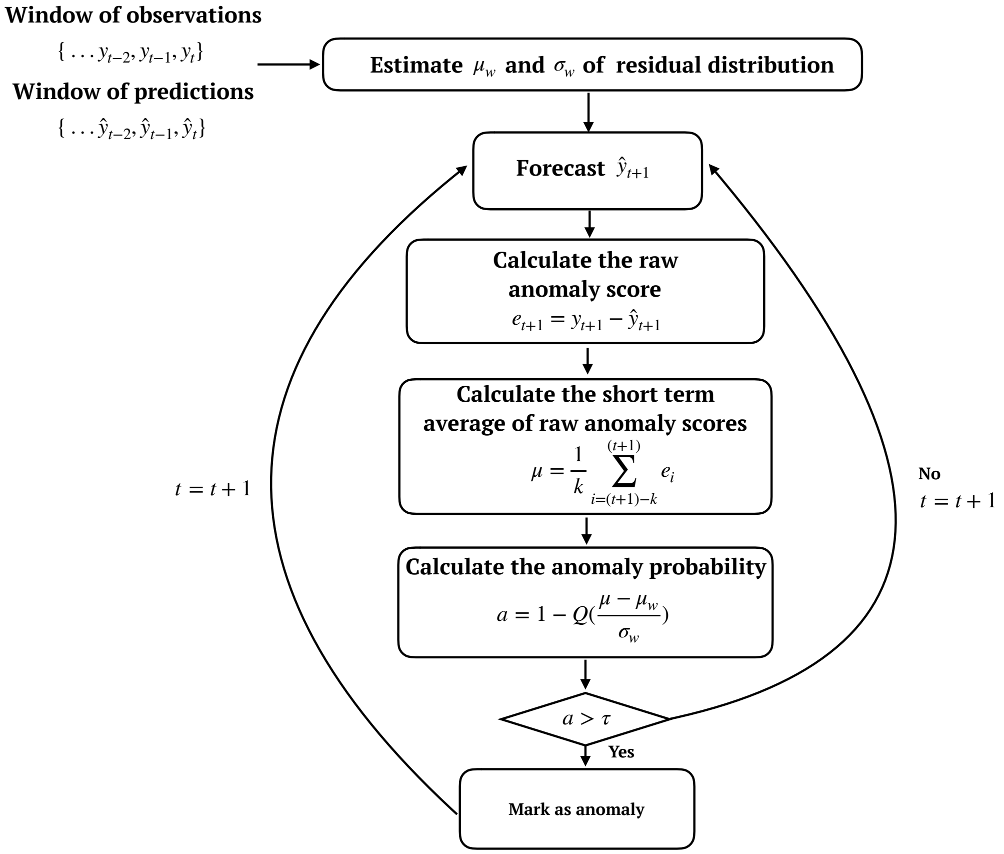

Window of observations
$\{\ldots y_{t-2}, y_{t-1}, y_t\}$
Window of predictions
$\{\ldots \hat{y}_{t-2}, \hat{y}_{t-1}, \hat{y}_t\}$
Estimate μ_w and σ_w of residual distribution
Forecast $\hat{y}_{t+1}$
Calculate the raw anomaly score
$e_{t+1} = y_{t+1} - \hat{y}_{t+1}$
Calculate the short term average of raw anomaly scores
$\mu = \frac{1}{k} \sum_{i=(t+1)-k}^{(t+1)} e_i$
Calculate the anomaly probability
$a = 1 - Q(\frac{\mu - \mu_w}{\sigma_w})$
$a > \tau$
Yes
Mark as anomaly
$t = t + 1$
No
$t = t + 1$

Figure 5.4: Schematic of the architecture of *method I*

compute a short-term average, μ, of the m most recent residuals, and compute the *probability* that μ comes from the residual distribution.

Consistent with Ahmad et al. (2017), we define the probability of an anomaly as the complement of the tail probability (Q):

$$a = 1 - Q(\frac{\mu - \mu_w}{\sigma_w}) \tag{5.7}$$

where $Q(\cdot)$[3] is defined as

$$Q(z) = \frac{1}{2}\text{erfc}(\frac{z}{\sqrt{2}}) \tag{5.8}$$

where

$$\text{erfc}(z) = \frac{2}{\sqrt{\pi}} \int_z^\infty \exp{(-x^2)} dx \tag{5.9}$$

[3] The tail probability $Q(\cdot)$ is described in detail by Karagiannidis and Lioumpas (2007)

is the complementary error function.

We find the log value more useful both for visualization and thresholding; hence, a log scale representation of the probability is calculated as

$$a_{ln} = \frac{\ln((1.0 + 10^{-10}) - 2 \cdot |0.5 - a|)}{\ln(1.0 - 10^{-10})} \tag{5.10}$$

where $\ln(\cdot)$ is the natural logarithm. If $a_{ln} \geq \tau$, then the corresponding observation is classified as abnormal, where τ is a constant. We set the threshold $\tau = 0.8$ by trial and error, such that the anomaly detection method warns for anomalies where we expect it to do so.

This method classifies a new observation based on a *static* residual distribution built on the residuals of the validation set. In contrast, Ahmad et al. (2017); Hundman et al. (2018) continuously updates the residual distribution in order to take into account the most recent history, and emphasizes that if there is a shift in the behavior of the system, the prediction error will be high at the point of the shift, but will automatically degrade to zero as the model adapts to the "new normal". However, we believe that the prediction error for our model is not guaranteed to be high if there is a shift in the behavior of the system. For example, for a gas compressor where the performance is slowly degrading it is not desired that the model should adapt to the "new normal". In fact, we believe that such behavior could lead to potentially costly consequences. Moreover, Hundman et al. (2018) emphasizes the fact that the Gaussian assumption of the distribution of past smoothed residuals is problematic if the parametric assumptions are violated when updating the distribution. With these arguments, we keep the residual distribution statically defined, and only update it whenever the predictive model is in need of retraining due to changes in the system characteristics. For our application in particular, we believe that the residuals obtained on the validation data reflects the normal behavior of the system.

5.2.2 PI-based anomaly detection

The second method is referred to as *method II*, and is mainly inspired by Zhu and Laptev (2017). This approach exploits the fact that the predictive models are probabilistic, and utilizes the pointwise predictive distributions of the models to identify anomalies. A new observation is classified based on whether it is within the bounds of the prediction interval of the model. Figure 5.5 presents the overall architecture of this method.

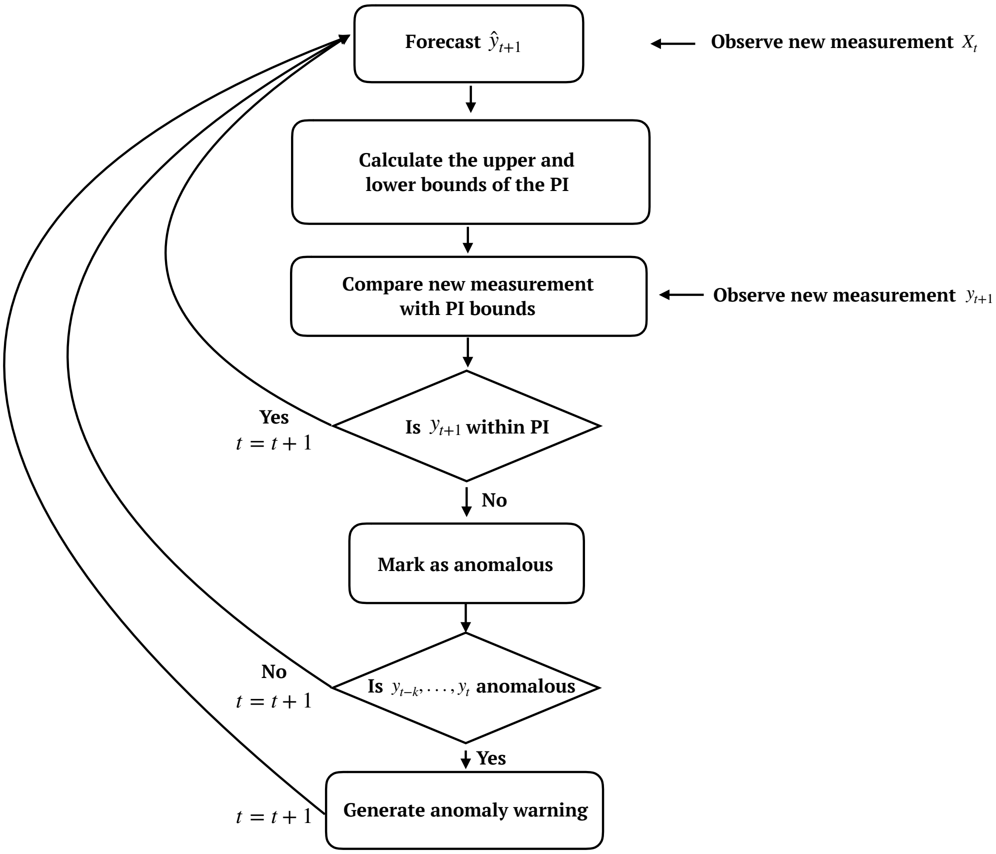

Forecast $\hat{y}_{t+1}$
Observe new measurement X_t
Calculate the upper and lower bounds of the PI
Compare new measurement with PI bounds
Observe new measurement y_{t+1}
Yes
$t = t + 1$
Is y_{t+1} within PI
No
Mark as anomalous
No
$t = t + 1$
Is $y_{t-k}, \ldots, y_t$ anomalous
Yes
$t = t + 1$
Generate anomaly warning

Figure 5.5: Schematic of the architecture of method II

.

As described in section 5.1.4, we have identified and implemented methods that can estimate the point-predictive distributions of the networks. We denote the predictive mean of the networks $\hat{y}^*$, and the total uncertainty regarding this prediction as η. Consequently, we can compute the upper and lower bounds of the prediction intervals for a prediction $\hat{y}^*$ by

$$[\hat{y}^* - \eta \cdot z_{\alpha/2}, \hat{y}^* + \eta \cdot z_{\alpha/2}] \tag{5.11}$$

where $z_{\alpha/2}$ is the upper $\alpha/2$ quantile of the standard normal.

By trial and error, we find that a 99 percent confidence interval is suitable. However, this parameter is easily adjustable based on preferences regarding type I and type II errors. A higher value for α implies a wider prediction interval, resulting in a generally higher acceptance and less anomaly warnings. Similarly, a smaller value indicates in a tighter interval, and the method will easier identify more observations as anomalous.

We find it useful for this method to generate an anomaly warning once a consecutive sequence of $n = 5$ or more points outside of the prediction interval occurs. This is motivated both by Lee et al. (2018) and Ahmad et al. (2017) because anomalous behavior often occurs in sequences and because inherently noisy systems often cause instantaneous predictions to be inaccurate.

One of the big advantages of this method compared to a static k-sigma model where an arbitrary threshold is set by specifying k, is that the prediction interval is dynamically changing based on the certainty of the model (Hill and Minsker, 2010). The width of the interval is determined by the degree of predictive confidence, which is reflected by the uncertainty, η, of the model for an input point.

5.2.3 Evaluation of unsupervised anomaly detection models

In contrast to supervised anomaly detection methods, there is no straightforward way to assess the quality of unsupervised methods. A common approach in related work (chapter 2), is to consult experts in order to obtain enough labeled data such that a precision and recall score can be calculated. Unfortunately, this is a manual, labor intensive method in need of expert knowledge, and it is not a feasible option in this thesis. Instead, we take a more intuitive approach when it comes to evaluating the quality of the proposed anomaly detection methods. Similar to Malhotra et al. (2016), we demonstrate the performance of the methods on hand-picked datasets that exhibit clear abnormal behavior, and assess the quality of the methods based on whether they can identify the unusual patterns. There are many examples of data with abnormal patterns in the available data on the compressor, and we select one dataset for each target, from different months. Remark that the selected datasets are completely independent of the training, validation or test sets, and they are obtained for up to a year after the training data. We analyze the datasets and define time intervals where we expect a good anomaly detection algorithm to warn for anomalies, and observe how the methods conform to our expectations. Of course, considering our lack of expertise or domain knowledge, we need to consult experts for verification of the results and to ensure that the identified areas of anomalies are indeed anomalous.

5.3 Hardware and technical implementation

The open source programming language Python v3.6.7 is used for all data processing and implementations. The main code is written in the form of *Jupyter Notebooks* which makes the code easily readable through markdown and latex report writing combined with code blocks. All the code written for this project thesis is found at `https://github.com/reitenhalvor/master-thesis`, of which the main notebooks are found in the `src/`-folder. The code is well documented, and the Jupyter platform makes the code and its outputs easy to review. All the results are 100% reproducible, as there have been set seeds where necessary to ensure this.

The most important libraries used for general data processing were `Numpy` v1.15.2 and `Pandas` v0.23.4. For the implementation of the benchmarks, the `sklearn` v0.20.1 was extensively used. The deep learning models were implemented in `Keras` v2.2.4, with `Tensorflow` v1.12.0 backend. `Keras` offers an easy-to-use library that is well documented. There are many alternatives to `Keras` and `Tensorflow`, such as `MXNet` and Facebook's `PyTorch`, but these introduce more flexibility, complexity and arguably unnecessary functionality. Keras offers the simplest implementation of all available libraries but still requires technical knowledge in terms of programming skills and machine learning.

In terms of hardware, the models were trained using a desktop with a 6-core, Intel i7-8700 CPU at 3.20GHz CPU[4] and 32GB RAM. Unfortunately, the GPU[5] on the desktop was not supported in Tensorflow, which could have had significant impacts on the training times of the networks. Instead, the CPU was used to train the models. With a dataset of almost 90.000 observations of 6 input features, the training times of a single epoch was approximately 3 minutes or more. As each model was trained for at least 50 epochs, this corresponds to a minimum of 2.5 hours of training time per model.

[4] Central Processing Unit

[5] Graphical Processing Unit

Chapter 6

Results and Analysis

This section begins by presenting the results regarding the modeling of the normal behavior compressor. The performance of the deep learning models are benchmarked with the proposed baselines, and their strengths and weaknesses are discussed in section 6.1. The best predictive model is utilized for anomaly detection on the gas compressor, of which the results are presented in section 6.2[1].

6.1 The Predictive Model

We investigate the performance of a model when predicting each of the target variables on the test data, where the average MAE across all targets reflects the overall performance of the model. Both the scores on the validation and test set are supplied, but it is the MAE on the test set that is the best estimate of the true generalization error. Hence, the average score on the test set is the measure that is ultimately used to compare models to one another. The performance metrics are *standardized*, meaning that they do not reflect the error with the variable's true unit, but they are rather scaled corresponding to the scaling of the data. This makes it harder to conceptualize the error metric, but it has the advantage of making each target-specific error metric comparable to each other regardless of their original units. Moreover, the training, validation and test sets are the same for all models. This ensures that we are able to compare models based on statistical learning models and functional differences, and not the data they have been trained on.

For the sake of brevity, the target variables have been renamed such that the result tables are easier to read. The discharge flow, discharge temperature, and discharge pressure have been shortened to ”FT”, ”TT” and ”PT”, respectively.

[1] All code written to obtain the results presented in this chapter is available in our Github repository at `www.github.com/reitenhalvor/master-thesis`. The results are reproducible, as seeds have been set where necessary to ensure this.

6.1.1 Benchmarks

Common-sense heuristics

The results of the common-sense heuristics are summarized in table 6.1. The *Prev* heuristic refers to the heuristic that predicts the next value to equal the preceding, the *Mean* heuristic predicts the mean of the training data, and the *MA* heuristic predicts the moving average of the data seen so far. Despite the simple and naive nature of these models, the result demonstrates that they can offer powerful baselines.

	Validation				**Test**			
	FT	TT	PT	Avg	FT	TT	PT	Avg
Prev	0.9073	0.2212	0.5586	0.5624	0.7840	0.1830	0.5156	0.4942
MA	0.8342	0.7102	0.8331	0.7925	0.7439	0.7147	0.7606	0.7397
Mean	0.8573	1.3854	0.8328	1.0252	0.8106	1.6929	0.7612	1.0882

Table 6.1: The standardized MAE of the implemented common-sense heuristics, sorted in ascending order from best to worst with respect to the average loss on the test set. ”FT” is discharge flow, ”TT” is discharge temperature and ”PT” is discharge pressure.

The overall best heuristic, by the average test MAE, is the *Prev* heuristic. The model performances differ mostly in predicting the discharge temperature and discharge pressure, where the *Prev* heuristic is considerably better than the other two. This indicates that a large portion of the *one-step-ahead* temperature and pressure can be explained by the preceding value. This makes intuitive sense, as there is no reason to think that the target signals deviate significantly from one time step to another with the defined granularity of one minute.

Linear models and simple machine learning

Table 6.2 summarizes the results of the linear models and tree-based methods. For each model, only the most accurate based on the validation score is presented. Whenever relevant, the optimal parameters of a model are found using hold-out validation or cross-validation. For the random forest and GBM, separate models are implemented for each of the target variables, because validation scores indicated that this was beneficial. Furthermore, the optimal number of trees for the random forest was found to be 3000[2], whereas 10000 trees were used for the GBM models. The optimal values for the regularization parameters of the shrinkage methods were found to be $\alpha = 0.01$ and $\lambda = 0.01$, i.e. approximately no regularization. Because of this, we see that the Ridge regression,

[2] We varied the number of trees from 100 to 3000 with an increment of 100.

	Validation				**Test**			
	FT	TT	PT	Avg	FT	TT	PT	Avg
Linear	0.6987	0.2902	0.5101	0.4997	0.6158	0.2524	0.4907	0.4530
Ridge	0.6987	0.2902	0.5101	0.4997	0.6158	0.2524	0.4907	0.4530
ElNet	0.6986	0.2960	0.5104	0.5017	0.6141	0.2606	0.4905	0.4551
GBM	0.6565	0.3229	0.5107	0.4967	0.5904	0.2692	0.5084	0.4560
LASSO	0.6975	0.3011	0.5116	0.5034	0.6112	0.2708	0.4912	0.4578
RF	0.6677	0.3392	0.5222	0.5097	0.6048	0.2924	0.5127	0.4699

Table 6.2: The standardized MAE of the implemented simple machine learning methods, sorted in ascending order from best to worst with respect to the average loss on the test set. ”FT” is discharge flow, ”TT” is discharge temperature and ”PT” is discharge pressure.

ElNet, and the ordinary linear regression model are equivalent, or marginally different, in terms of validation and test performance.

As expected, the baselines presented in this section outperform the common-sense heuristics. The ordinary linear regression model triumphs as the best of the implemented baselines, with an average test MAE of 0.453. Still, all the models have similar performance, both on average and for each target individually. The best of the tree-based methods is GBM, reaching an average test MAE of 0.456. It is consistently better than the random forest in predicting all targets.

A surprising result is that the linear regression models outrank the tree-based methods. As discussed in the previous chapter, the tree-based methods are able to capture non-linear dependencies in the data, and that this presumably should have a positive impact on their performance. This result can likely be explained by the strong linear dependencies between some of the inputs and outputs, as seen by the cross-correlation plot in figure B.2b. Nonetheless, by inspecting the scores for each target individually, we see that the tree-based methods are generally better at modeling the discharge flow, while it is opposite for the discharge temperature and pressure.

6.1.2 Deep learning

We implement three separate types of deep neural networks as described in section 5.1.4. Two of them are recurrent neural networks, namely a LSTM network and a GRU network, while the third is a standard feed-forward neural network. These are hereafter referred to as ”LSTM”, ”GRU” and ”MLP”, respectively, and their designs are described in section 5.1.4. To further enhance the performance, we implement an ensemble model, where the predictions of the LSTM and the MLP are combined through a linear regression model that weights the respective models in order to minimize prediction error on the validation set. The ensemble model is denoted ”LSTM/MLP”.

	Validation				**Test**			
	FT	TT	PT	Avg	FT	TT	PT	Avg
LSTM/MLP	0.5811	0.2534	0.4035	0.4127	0.5123	0.2264	0.3862	0.3749
LSTM	0.6042	0.3246	0.4286	0.4525	0.5422	0.3179	0.4153	0.4251
GRU	0.6169	0.3355	0.4454	0.4660	0.5526	0.3302	0.4383	0.4403
MLP	0.6492	0.3126	0.5063	0.4894	0.5895	0.2638	0.4919	0.4484

Table 6.3: The standardized MAE of the deep learning models, sorted in ascending order from best to worst with respect to the average loss on the test set. ”FT” (discharge flow), ”TT” (discharge temperature) and ”PT” (discharge pressure).

Recall that the models implemented are *probabilistic*; hence, we average the predictions 300 times in order to get a reliable estimate of the predictive mean.

Table 6.3 summarizes the performance of the deep learning models. Some findings are common for all models. First, the performance on the validation set is consistently better compared to those of the test set. This is normally not the case in machine learning problems since the model is partly biased towards the validation set as it has been actively used in the model selection process. However, this phenomenon can likely be explained by the test data being statistically more similar to the training data than the validation data[3]. Consequently, it is easier for the model to predict using the test data because the statistical properties are similar to the training data.

Second, we see that all deep learning models outperform the benchmarks with respect to the average test score. In fact, we see an overall improvement of 17.2% from the best benchmark to the best deep learning model in terms of average test MAE. This is a substantial improvement that showcases the value of deep learning in this modeling problem. In particular, we see a considerable improvement in modeling the discharge flow and pressure.

Third, we see vast gains by *model ensembling*. This is especially apparent on unseen data, where we see an improvement of 11.8% on average test MAE compared to the performance of the most accurate of the individual models included in the ensemble. As discussed in theory section 3.3.7, a good ensemble relies on diversity, where the constituent models should be as different and as accurate as possible (Chollet, 2018). We combine models that process the input differently, and this results in differences regarding performance in the modeling of different targets. While the LSTM is best at predicting the flow and pressure, the MLP is considerably better at predicting the temperature. In the ensemble, we combine the strengths and weaknesses of these two models, resulting in a major overall improvement for all targets. Moreover, the huge positive impact seen

[3] This can be seen by inspecting figure B.4 which shows the variable distributions for the training, validation and test regions of the data.

from this ensemble may indicate that even bigger and more advanced ensembles could further improve the performance.

The results presented in table 6.3 show that the sequence specialized networks are the most accurate models. This indicates that there is information in the temporal context of the time series that the RNNs exploits, which improve their predictions compared to the non-sequential networks. The LSTM is the most accurate network and this model obtains an average MAE of 0.425. Moreover, we see that the LSTM outperforms the GRU network in predicting all targets.

The MLP is the least accurate of the deep learning models, obtaining an average test MAE of 0.448, but we see that the MLP is more accurate than the sequential models when modeling the discharge temperature, obtaining a test MAE of 0.264. However, the *Prev* heuristic has a final MAE of 0.183 when predicting the discharge temperature, which is more accurate than any of the deep learning models. In fact, many of the linear models outperform the non-ensemble deep learning models when predicting the temperature in terms of the test MAE. As stated in the previous section, this indicates that there are strong linear dependencies between the input variables and the temperature, as can be verified by inspecting the cross-correlation plot in figure B.2b.

The LSTM/MLP ensemble is the overall best deep learning model with an average test MAE of 0.375. This corresponds to a $12-16$ percent improvement from the other deep learning implementations. The ensemble is better on average and offers considerable improvements for all targets. Comparing it to the individual performance of the constituent models, we see particular gains in predicting the temperature, with a relative improvement of up to 30%. Furthermore, the test MAE for predicting the flow and pressure is improved by up to 13.6% and 22.3%, respectively.

Objective	Best Benchmark	Benchmark MAE	LSTM/MLP MAE	Improvement (%)
FT	GBM	0.5904	0.5123	13.23%
TT	Predict previous	0.1830	0.2264	-23.72%
PT	ElNet Regression	0.4905	0.3862	21.26%
Overall	Linear Regression	0.4530	0.3749	17.24%

Table 6.4: Best benchmarks compared to the best deep learning model - the LSTM/MLP ensemble - in predicting each target.

Table 6.4 compares the LSTM/MLP ensemble model to the best benchmark for each target. We see that LSTM/MLP offers a 17.2% overall improvement of the average test MAE compared to the linear regression model. Moreover, we see similar numbers when predicting the discharge flow and pressure, with improvements of approximately 13.2% and 21.6%, respectively. The LSTM/MLP is outperformed by the *Prev* heuristic when predicting the discharge temperature, despite this being the target that the model predicts most accurate. While the LSTM/MLP does not outperform the best benchmark for temperature, it is consistently better for the other targets. Hence, we acknowledge that the LSTM/MLP model offers a considerable improvement in general.

Uncertainty assessment

Recall from section 5.1.4 that the deep learning methods apply Monte Carlo dropout at inference time, resulting in stochastic behavior of the networks. Instead of predicting deterministic point-wise estimates, a model will output samples from its predictive distribution. Moreover, as discussed in section 3.3.8, the predictions follow a normal distribution, and the parameters of the distribution can be estimated by empirically drawing sufficiently many samples.

		FT	TT	PT	Avg
Validation	**Aleatoric** (η_2)	0.5949	0.0964	0.2755	0.3223
	Epistemic (η_1)	0.1861	0.1429	0.1770	0.1687
	Total (η)	0.7948	0.3454	0.5549	0.5650
Test	**Aleatoric** (η_2)	0.4436	0.0813	0.2461	0.2570
	Epistemic (η_1)	0.1971	0.1625	0.1900	0.1832
	Total (η)	0.7968	0.3539	0.5587	0.5698

Table 6.5: Average estimates of epistemic (η_1), aleatoric (η_2) and total uncertainty (η) of the LSTM/MLP ensemble model on the test data. Results are averaged over 300 iterations. Note that the values are averaged across all inputs, although the uncertainty really is a point-wise estimate.

The normality of the point-wise predictive distributions of the LSTM/MLP-model can be verified by inspecting figure 6.1, which shows the distribution of the predictions of a model for a single input from the test set. A Shapiro-Wilk test[4] for the specific predictive distribution yields a result of 0.983, clearly indicating the data is normally distributed, which also can be visually confirmed by the fitted normal distribution, the dark blue line in the figure.

Table 6.5 summarizes the uncertainty estimates for the LSTM/MLP-model for all targets on the test set. An important remark is that the numbers presented are average values across all the observations in the test set, even though the uncertainty really is a point-wise estimate. That is, for each set of inputs in the test set, the point-wise uncertainty (η_t) is estimated empirically with 300 samples, and the numbers presented in table 6.5 are the average values across all estimates. For this reason, the relationship between η_1, η_2 and η is not preserved for the values presented in the table. Nonetheless, the numbers can give an idea to the general confidence of the network for the test dataset.

[4] The Shapiro-Wilk test tests the null hypothesis that the data is drawn from a normal distribution. Small values closer to 0 rejects the hypothesis, while larger values indicate a larger confidence that the data comes from a normal distribution (Shapiro and Wilk, 1965).

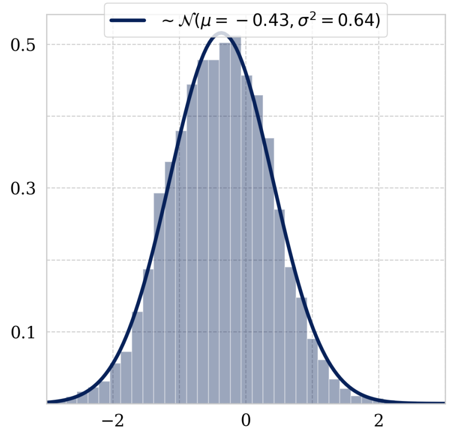


Figure 6.1: Predictive distribution of the ensemble model for a single input in the test set, where 300 samples are drawn. The fitted normal distribution, the dark blue line in the plot, corresponds neatly with the observed samples.

Figure 6.2 illustrates the uncertainty of the LSTM/MLP model in practice. The gray lines represent stochastic realizations of the model, while the red represents the predictive mean of the model. The spread of the gray lines for a single input reflects the variance of the model for the input, and higher spread indicates less confidence in the prediction.

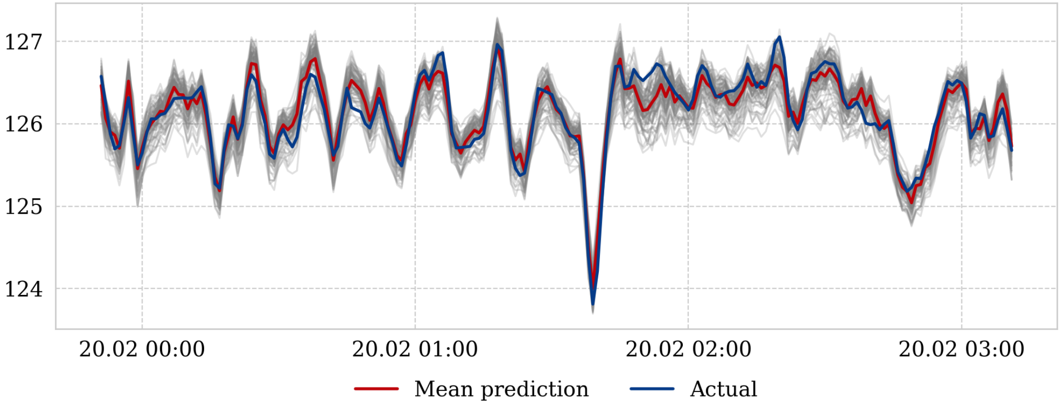


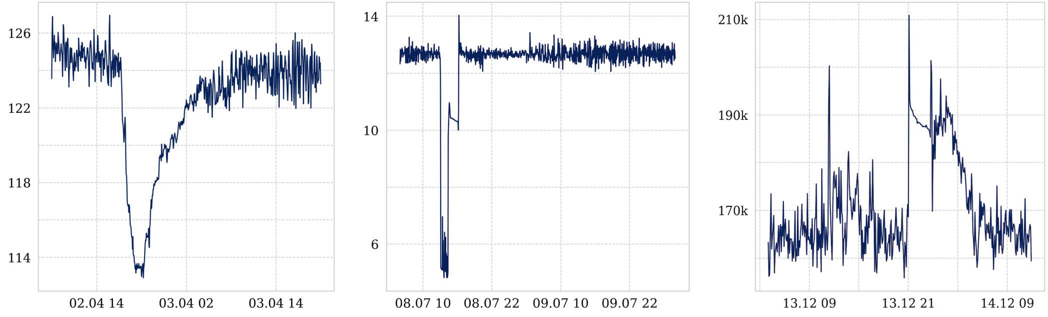
Figure 6.2: Discharge temperature predictions versus the actual temperature of the MLP/LSTM model. The gray lines represent different realizations of the model, and the red line is the mean of these.

6.2 Anomaly Detection

The LSTM/MLP model shows a major improvement in terms of accuracy compared to the other implementations. As the proposed anomaly detection methods are dependent on an accurate and reliable predictive model to identify anomalies, we apply the LSTM/MLP as the predictive model in both methods.

The two anomaly detection methods (method I and II) are demonstrated on data acquired up to a year after the training period. We specifically select data where the outputs of the compressor look highly abnormal, where we expect the anomaly detection methods to generate anomaly warnings. The remainder of this section demonstrates the performance of the anomaly detection methods on the chosen periods, and we assess the quality of the methods based on the results. An important assumption we make is that the *normal* state of the compressor, which is learned using data from December 2017 to February 2018, is still valid in the extracted periods. Section 6.2.1 describes the extracted time periods, while section 6.2.2 and 6.2.3 presents the results of the residual-based and prediction interval-based methods, respectively.

6.2.1 Selected data regions with anomalous behavior



(a) 2nd - 4th of April 2018 **(b)** 8th - 10th of July 2018 **(c)** 13th - 15th of December 2018

Figure 6.3: Selected data regions with anomalous patterns used to evaluate the proposed anomaly detection methods. Discharge temperature in (a), discharge pressure in (b) and discharge flow in (c).

We look at two-day intervals in April, July and December 2018, which are plotted in figure 6.3. Table 6.6 shows where we expect to find anomalies in the datasets. The results of the anomaly detection methods are compared to our expectations, which again will define the suitability of the methods.

Month	From time	To time	Cause
April	02.04 16:00	03.04 04:00	Rapidly decreasing, unusually low temp.
July	08.07 12:00	08.07 17:00	Sudden drop in pressure, low values.
December	13.12 11:00	13.12 12:00	Sudden spike in flow.
December	13.12 20:00	14.12 02:00	Spikes and high flow.

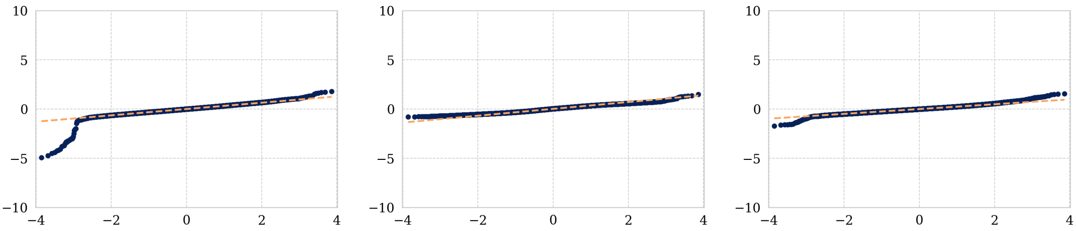
Table 6.6: Expected anomalies in the selected time periods.

6.2.2 Method I: Residual based anomaly detection

Verifying the normality of the residual distribution

It is assumed that the residuals obtained during the validation of the final predictive model follow a Gaussian distribution. This is verified by the quantile-quantile (Q-Q) plots in figure 6.4, which shows the quantiles of the estimated residual distributions for each target compared to the quantiles of the standard normal distribution. Provided that the residuals of the final model are normally distributed, the points in the Q-Q plot will fall on the line $y = x$, which is the orange reference line. It is seen that the points fall on the line for the central and the right region, but they deviate from the reference line closer to the left tail for the discharge flow and pressure. This deviation is apparent at $x \approx -3$, which indicates that the residuals of the final models have a thicker tail on the left-hand side than those of the standard normal. This is expected, considering that we do not remove outliers and extreme values from the dataset, as discussed in section 4.5.2. However, it is expected that this will not affect the validity and accuracy of method I in any major way.

Plots of the residual distributions for the discharge flow, temperature and pressure are shown in appendix C.5.



(a) Discharge flow **(b)** Discharge temperature **(c)** Discharge pressure

Figure 6.4: Q-Q plot for the three targets discharge flow, temperature, and pressure. Theoretical quantiles are on the x-axis and the sample quantiles are on the y-axis.

Demonstration and results

Figure 6.5 shows method I applied to the December dataset, where the bottom plot is the corresponding short-term average log anomaly probability plot. The red areas in the top plot indicate where the model has detected anomalous behavior, of which we find two regions; a short sequence on the 13th from 11:00 to 12:00, and a longer sequence starting later the 13th at approximately 20:00 to 03:00 the following day. Thus, the method roughly identifies all the expected anomaly regions for December. Significant deviance between the predicted and the actual discharge is observed for both anomaly sequences, indicating a highly abnormal relationship between the input variables and the target output in the periods. Furthermore, we see that the model is relatively accurate outside the anomalous intervals, which suggests that the model works as expected.

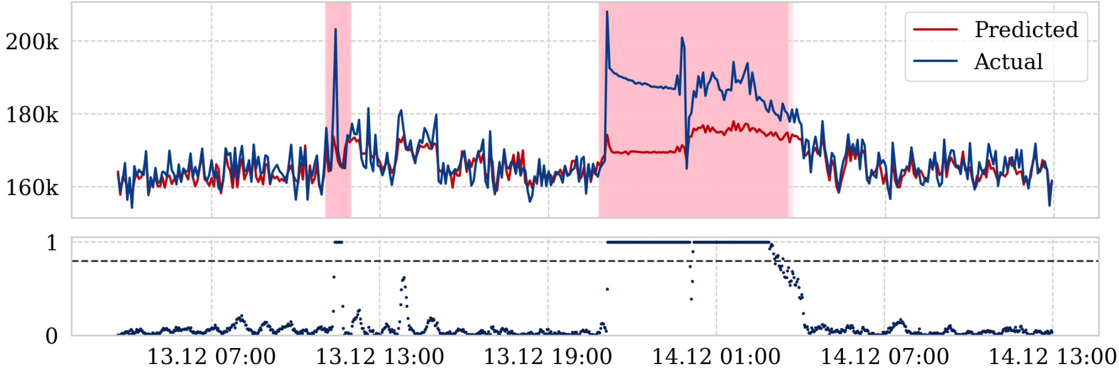


Figure 6.5: Anomaly detection plot for December 2018 (flow $[kg/h]$): method I. The top plot shows the predicted and the actual values, and the red areas indicate anomaly warnings. The bottom plot is the corresponding point-wise short-term log anomaly probability, where the anomaly threshold $\tau = 0.8$.

Figure 6.6 shows method I applied to the April dataset. We see that the model, in general, can accurately predict the development of the temperature, except for a couple of hours during the evening of April the 2nd until the morning of the following day. More specifically, an anomalous sequence is found between approximately 18:30 and 21:30 the 2nd. We see that the temperature is decreasing rapidly from 16:00 and that the predictive model manages to follow this descent to a reasonable degree for a little while, suggesting that this is expected behavior of the compressor. The temperature becomes unusually low around 18:00, and the predictive residuals increase gradually after this, which in turn increases the log probability. By inspecting the log probability plot, we see a gradual increase from 17:30 until the threshold is reached around 18:30, marking the beginning of the anomalous sequence. After 21:30, minor deviance is observed between the predictive model and the true measurements, but no anomaly is warned for due to the anomaly score being too small.

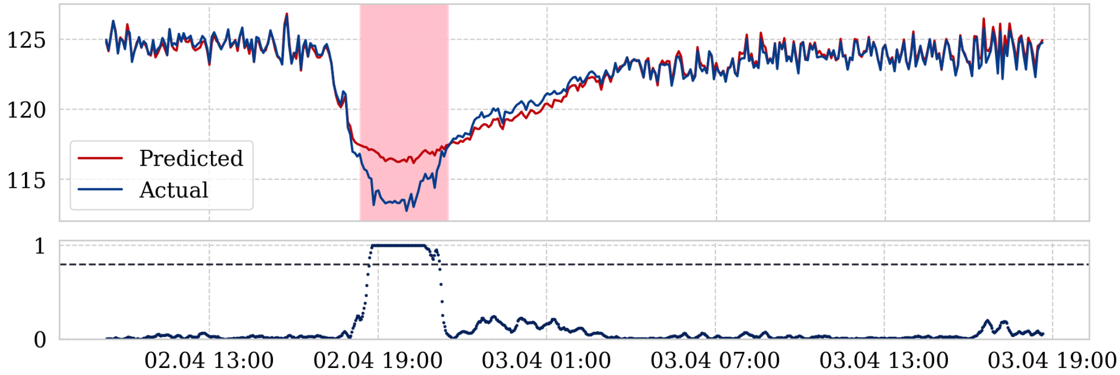


Figure 6.6: Anomaly detection plot for April 2018 (temperature $[^\circ C]$): method I. The top plot shows the predicted and the actual values, and the red areas indicate anomaly warnings. The bottom plot is the corresponding point-wise short-term log anomaly probability, where the anomaly threshold $\tau = 0.8$.

Figure 6.7 shows the method applied to the July dataset. We observe that a several hour long window is identified as anomalous, indicated in the figure approximately from 12:00 to 17:00 on July the 8th.

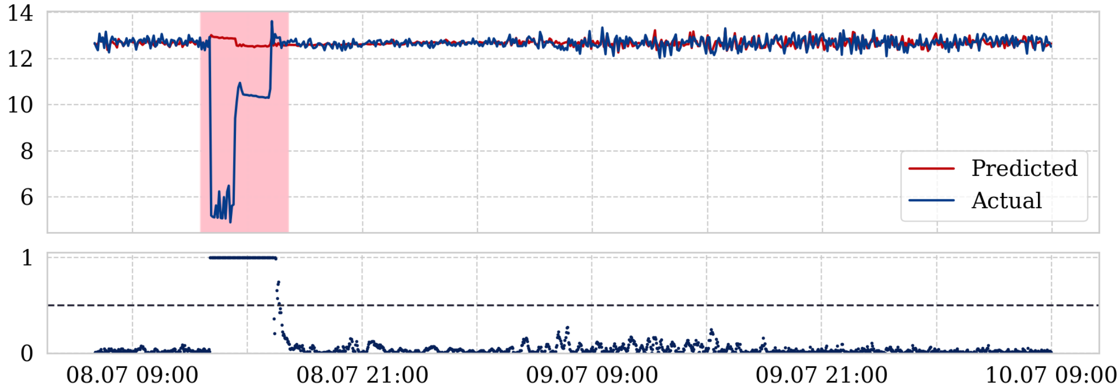


Figure 6.7: Anomaly detection plot for July 2018 (pressure $[barg]$): method I. The top plot shows the predicted and the actual values, and the red areas indicate anomaly warnings. The bottom plot is the corresponding point-wise short-term log anomaly probability, where the anomaly threshold $\tau = 0.5$.

We see a rapid decrease in the discharge pressure to approximately 6 barg, and that the predictive model does not follow along with this development. This is not surprising, as the predictive model is trained on data where the discharge pressure only oscillates between 12 and 14 barg[5]. The large residuals between the predictions and the true measurements effectively cause the method to warn about unusual behavior in the system. The corresponding log anomaly plot shows that

[5] See table B.1 for statistics of the training data.

it is confident in this warning, i.e., the log anomaly probability is close to 1 and well above the specified threshold. Otherwise, we see that the predictions and the true measurements correspond better, resulting in low log anomaly probability and no anomaly warnings.

6.2.3 Method II: PI-based anomaly detection

Demonstration and results

Figure 6.8 shows method II applied to the December dataset. We see that the identified anomalous regions correspond roughly to the ones marked by method I for the same dataset; a short window on the 13th from 11:00 to 12:00 with a significant spike in the flow, and a longer sequence from approximately 20:00 the 13th to 01:30 the 14th with severe spikes and particularly high flow measurements. The data in these regions exceed the 99% confidence threshold, which results in anomaly warnings of the method. We conclude that all expected anomaly regions for December are roughly identified also by method II. By inspecting the confidence threshold, we see that the width is increased during the anomalous data regions, indicating that the model is less confident about its predictions there. Another remark for this plot is that many of the true measurements are close to the upper and lower bounds of the prediction intervals, which illustrates the importance of setting thresholds appropriately, such that non-anomalies are not marked falsely.

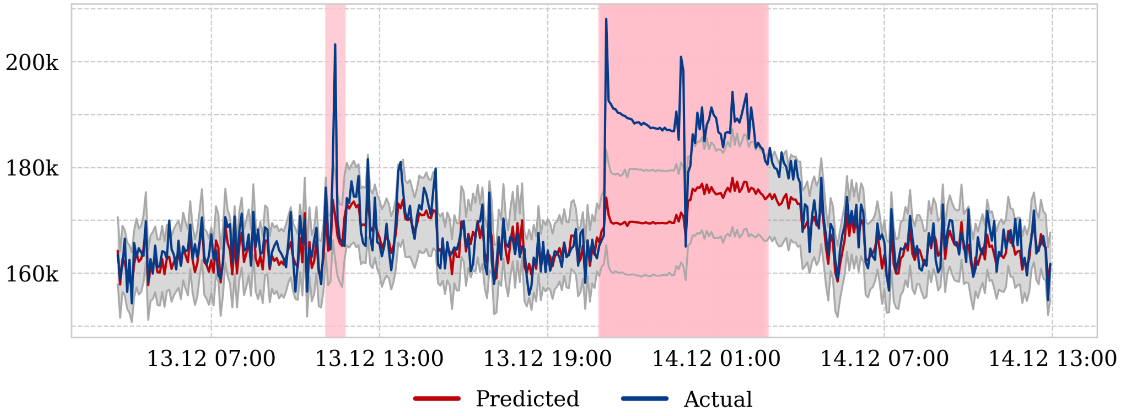


Figure 6.8: Anomaly detection plot for December 2018 (flow $[kg/s]$): method II. The grey area indicates the 99%-prediction interval, and the red areas indicate anomaly warnings.

Figure 6.9 shows method II applied to the April dataset. One region is identified as anomalous, approximately between 18:30 and 21:30 on April the 2nd, which corresponds to the interval identified by method I. In the anomalous region, the width of the prediction interval increases notably and that the observed measurements are well below the lower threshold. The increased width of the interval indicates that the predictive model is less confident about its predictions in this interval, and the poor predictions indicate that there is an abnormal relationship between the input variables and the target. Furthermore, we observe that the width of the prediction interval decreases gradually after the anomalous sequence, indicating increasing confidence of the model. We observe that the predictions follow the observed temperature excellently both early on the 2nd and on April the 3rd.

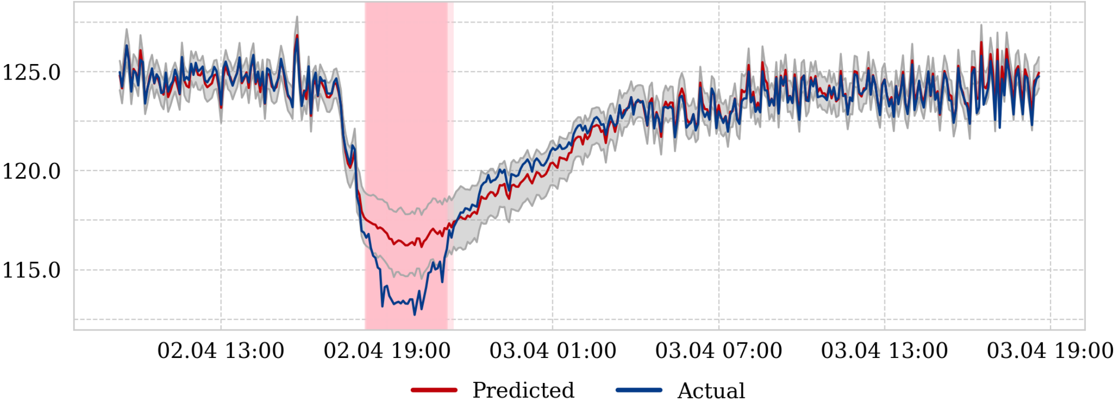


Figure 6.9: Anomaly detection plot for April 2018 (temperature $[^{\circ}C]$): method II. The grey area indicates the 99%-prediction interval, and the red areas indicate anomaly warnings.

Figure 6.10 shows method II applied to the July dataset. Similar to method I, one region is identified as anomalous, approximately from 12:00 to 17:00 on July the 8th. We observe that the discharge pressure drops below the lower bound of the prediction interval on July the 8th and consequently, an anomaly warning is given.

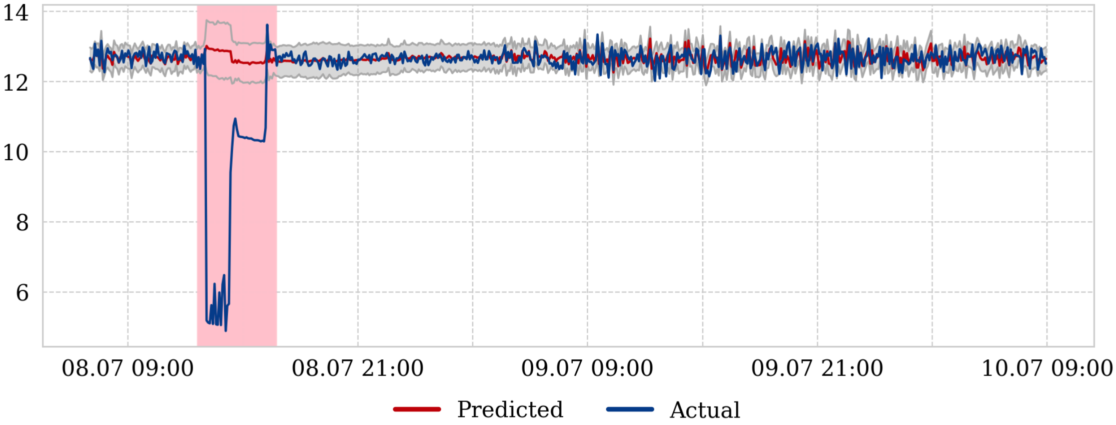


Figure 6.10: Anomaly detection plot for July 2018 (pressure $[barg]$): method II. The grey area indicates the 99%-prediction interval, and the red areas indicate anomaly warnings.

Moreover, the width of the prediction interval increases notably directly after the anomalous sequence on July the 8th, indicating that the model is unsure about the predictions of the discharge pressure. Additionally, we observe that the prediction of the discharge pressure corresponds well with the observed discharge pressure prior to the anomalous sequence on July the 8th and from 09:00 on July the 9th.

6.2.4 Comparison of the methods

The performance of the two methods is broadly similar. They both identify, wholly or partly, all of the regions listed in table 6.6. The only interval where there is partial deviance from our expectations is in the April dataset, where we had expected a *longer* anomaly interval than those given by the respective methods. Specifically, we defined the expected anomalous range of April to start 16:00 the 2nd and end 04:00 the next day, but both methods warned for anomalies approximately between 18:30 and 21:30 the 2nd. However, the minor deviance between the observed and predicted values outside the identified anomalous intervals could indicate that we on beforehand had overestimated the length of the anomaly interval. Meanwhile, the log anomaly probabilities on the night of April the 3rd fluctuate between 0 and 0.4, which indicates somewhat unusual behavior.

Both methods identify the sudden spike in the discharge flow on December the 13th between 11:00 and 12:00. However, it is hard to say whether the observed behavior is indeed anomalous or if it is a result of measurement errors of the sensors. The two methods differ slightly on the longer anomalous sequence in December (13.12 20:00 to 14.12 02:00): method I marks a slightly longer window than that identified by method II. However, this is dependent on the specified anomaly thresholds and is easily adjustable. The methods identify the same anomaly ranges in both the July dataset and the April dataset.

All in all, the results indicate that both methods are well suited to serve as a warning system in an industrial setting if the thresholds are set appropriately. However, the methods are so far only evaluated based on subjective expectations on arbitrarily picked datasets. Experts are needed to assess the quality of the models properly and to determine whether their performance is satisfactory.

Chapter 7

Discussion

In light of the results in chapter 6, we start this chapter by providing a deeper understanding of the true potential of deep learning as a modeling framework for industrial equipment. We focus specifically on whether the improvement in accuracy is justified by the increase in complexity and ways to enhance the performance of the models further. Further, in section 7.2, we discuss the quality of the anomaly detection methods, whether they are suitable algorithms in a practical setting and ways to improve their performance further. Section 7.3 outlines the practical hurdles of implementing data-driven models and anomaly detection algorithms for complex industrial equipment before we present natural avenues for further work in section 7.4.

7.1 Potential of Deep Learning as Predictive Models

The results show that it is possible to accurately model the normal behavior of the gas compressor using data-driven approaches, without the use of expert domain knowledge or by applying any laws of physics. The deep neural networks are the most accurate models, which shows promise for deep learning in this modeling problem. However, the results also indicate that the compressor can be modeled with reasonable accuracy by less complex models, such as ordinary least squares linear regression. Arguably, the relative improvement of 17.2% on held-out data is moderate considering the vast increase in complexity that follows with deep learning. The question is whether the improved accuracy is strictly necessary and if the improved performance justifies the increase in complexity. From an industrial perspective, it can be natural to use the least advanced model that meets the demands in terms of accuracy. However, we consider the improvement as a verification of the potential of deep learning, even though there are some non-negligible challenges to implementing these models.

The less complex models are straightforward to implement mainly because they have few parameters to tune, and they require little in terms of training times and computational resources.

However, this implies that the results obtained by these models are as good as they get, i.e. the simple models are nearly, or completely, at their full potential. In contrast, several parameters can be tuned in deep neural networks, and there is no reason to believe that we have achieved the optimal performance of these models.

Due to limitations in time and resources, we had to make some simplifying design choices in the implementation of the networks. We have presented some promising examples of network architectures, but there are undoubtedly other configurations that can improve the neural networks further. A thorough search through the vast space of possible configurations is likely to lead to more effective network designs. Specifically, we could see significant improvements by fine-tuning hyperparameters such as the learning rate of the optimizer and the initialization functions of specific layers. Similarly, introducing alternative types of regularization, increasing the number of training epochs, or exploring the specification of other parameters such as the window-size of the sequential models, could have a positive effect. In addition, the results showed a considerable improvement by model ensembling, and even bigger and more diverse ensembles could likely lead to better results. We implemented ensembles consisting solely of deep learning models, but we did not experiment with ensembles of deep and shallow models, which Chollet (2018) states can be beneficial. Furthermore, we acknowledge that there exists other types of ensemble techniques that are not covered in this thesis that could have had a positive impact. Moreover, the networks will become better with more data (Chollet, 2018). We constrained the data to a two-month interval, and although we found this to be sufficient for the model to learn the statistical characteristics of the system, more data is more than likely to have a positive effect on the model performances. However, there are some challenges tied to this, which are described in section 7.3.

7.2 Anomaly Detection with Deep Learning

The performance of the proposed anomaly detection methods is consistent with our expectations on the selected data regions, and the results indicate that there is a big potential for model-based anomaly detection for complex equipment in the oil and gas industry. The proposed anomaly detection methods are dependent on a robust, accurate and reliable predictive model. To this end, we find that the applied deep learning model performs excellently in conjunction with the anomaly detection methods: the accuracy is high when the compressor operates normal, but the model fails to predict seemingly anomalous sequences, resulting in, to the best of our judgment, accurate anomaly warnings.

The two anomaly detection methods have similar performance which makes it hard to compare the two models purely based on when they identify anomalous behavior. We find that both methods have advantages and disadvantages. Method I outputs a log anomaly probability for each observation. This can be regarded as a measure of abnormality and is more informative than just a label, which is the output of method II. Besides, the anomaly probability is easy to visualize, which can be helpful when tuning the anomaly threshold. On the other hand, method II has the advantage that the anomaly threshold is easier to tune since it only impacts the width of the prediction interval.

A key difference between the two models lies in how they classify new observations. Method I identifies anomalies based on a static residual distribution built on the validation data, whereas method II utilizes a prediction interval for each incoming observation. The width of the prediction interval is dependent on the variance of the predictive model, which naturally will increase when the model is less confident in its predictions. Consequently, method II will increase the width of the prediction interval when incoming observations have unusual or abnormal patterns, resulting in higher acceptance thresholds due to the increase in uncertainty. Thus, inherently anomalous observations can be classified as normal because of the uncertainty of the predictive model. This flexibility might seem counter-intuitive since increased uncertainty (i.e. less confidence) could be an indicator of abnormal behavior. We recognize that this is a potential drawback of the PI-based anomaly detection method.

In general, we find that both methods fulfill all but one of the requirements outlined in section 3.4.5 for an ideal real-time streaming anomaly detection algorithm: the methods are unable to adapt to scenarios where there are significant changes in the underlying statistics of the system (requirement 4). This can, for instance, be a result of changes in the configurations that alter the behavior of the system, and thus, the definition of normal behavior. In order to adapt to such scenarios, the predictive model must be retrained. However, we assume that a change in configurations is rare events and that the proposed methods are suitable for this reason. In fact, we demonstrate that the anomaly detection algorithms work well on datasets obtained almost a year after the data that the predictive model was trained on.

7.2.1 Operational application of the methods

Manual inspection and contextualization of multiple time series to uncover potential faults for equipment is an infeasible task for a human resource. Accurate and reliable anomaly detection methods may function as an early warning system and can lessen the monitoring burden placed on operations engineers. Moreover, it can reduce operational risk, and ultimately lead to reduced downtime. Our model is, however, not ready to be employed in production yet, as fine-tuning of thresholds and a more thorough analysis of the methods' behavior is required. Ideally, this is done by experts that have a solid understanding of the operation of the equipment, such that the anomaly detection methods generate warnings in accordance with desired levels of risks for type I and type II errors.

7.3 Challenges and Practical Barriers

A machine learning model is only as good as the data it is trained on, and as such, comprehensive data sets of high-quality features and lots of observations are a requisite (Chollet, 2018). To facilitate this, appropriate data pipelines must be developed before deep learning can be applied in an operational context. In addition, deep learning is notoriously demanding in terms of training times and computational resources. Hardware and software can set restrictions on what can be included in the models, and how long the models can train. Furthermore, as we have experienced in this thesis, this also affects to what degree it is possible to explore different configurations of respective networks, i.e. optimizing the parameters of the models. Another challenge of operational deep learning is therefore to facilitate sufficient computational resources.

With the data pipelines in place, there are still several challenges tied to the nature of the data in this modeling problem. First, the proposed anomaly detection methods rely on the predictive models being accurate when the equipment behaves normally, but inaccurate otherwise. Thus, a requisite of the data used to train the models is that it reflects the normal operation of the compressor. However, it is not easy to define rules for what ”normal” behavior is, though we have tried to the best of our ability to identify regions of the data that represents the general case for the behavior of the equipment. Although the results show that the models have managed to capture the normal characteristics to a reasonable degree, the practical data selection process requires expert domain knowledge. Furthermore, we want to feed the model with as much data as possible, but it is difficult to uncover consecutive intervals of data where the compressor operates normally without any anomalies. Finding general procedures to process anomalous sequences in the data could help with this, such that the models have access to more data.

The second challenge with the data is the lack of labels. As a result, it is difficult to evaluate the quality of the anomaly detection methods, as there is no objective way to know if the anomaly warnings are correct. As explained in section 5.2.3, we evaluate the methods by subjective expectations and how they fulfill these, but whether the anomaly warnings are correct requires expert knowledge. Another challenge is tied to concept drift, where the statistical properties of the data change over time. This can cause our pre-trained models to lose accuracy over time, which naturally affects the performance of the anomaly detection methods. Nevertheless, we show that our predictive model is maintaining a high prediction accuracy even for data that is obtained up to a year after the initial training process, which indicates that our model has learned the underlying statistics of the system well.

We see that the predictions of the model are highly affected when there are long anomalous sequences included in their window of inputs. Recall that the model uses six hours of data to make a single prediction, and we see a decrease in accuracy in the subsequent hours of a long anomalous sequence. This behavior can potentially lead to a several hour long window where the predictive model is unable to monitor the state of the system accurately. A way to address this problem is to modify the network to handle varying input sizes, where only the observations *after* the anomalous sequence and up to a maximum window size are used to make a prediction. Another possibility is to swap the actual observed discharge values with the corresponding predicted discharge for all the observations identified as anomalous. Although we recognize the problem, we have not found any appealing techniques to address it adequately.

Deep learning can achieve remarkable results, but they are inherently complex and demanding in terms of technical expertise. Open source libraries such as Keras and Tensorflow greatly simplify the implementation process, but technical skills and proficiency within machine learning and its fundamental principles are a requirement for success. However, an advantage is that this expertise is most needed in the development phase of these models, where the operational use and keeping the models updated is not as demanding. Similarly, the computational resources required to use the models to make predictions are negligible compared to what is required in the training phase.

It must be mentioned that deep learning in an industrial setting is a fine mix between data science and engineering. A challenge is therefore to combine the two, where engineers are needed to understand the problem, and data scientists are needed for the technical implementation. However, we show in this thesis that we can model complex equipment with data-driven methods and minimal domain knowledge.

7.4 Further work

We have demonstrated how model-based anomaly detection methods can be built for a gas compressor. However, certain aspects can be improved, and others that we have not explored yet. One of these aspects is the practical application of the methods, as they potentially can be used to predict maintenance events and making decision rules. How this is done is still an open question and a natural goal for further research. Moreover, while the predictive models predict three variables simultaneously, the anomaly detection methods only consider one variable at a time. Currently, the methods generate anomaly warnings for the compressor at time t if anomalies are identified for one of the targets, but a smarter and more robust detection method can be built by considering the classification of all three targets at once. We see this is as a natural extension of the work in this thesis.

The predictive model is trained in an offline fashion on a fixed training set and needs to be re-trained if the underlying statistics of the system change. However, other authors have suggested online learning setups where the models are constantly updated using a sliding window of observations. We have not explored such setups due to the argument that the training data should reflect normal equipment behavior, but the potential of online learning in this problem would be interesting to review.

We have implemented methods to assess the epistemic and aleatoric uncertainty of our deep learning models, but we have not addressed the uncertainty resulting from predicting samples with drastically different patterns compared to the data used in training. Including this in the total uncertainty assessment could result in more robust uncertainty estimates, and could benefit the implementations in this thesis.

Lastly, the predictive models and anomaly detection methods may improve by identifying *system states* and including this information as additional features in the dataset. For instance, the compressor is turned off in the case of production shutdowns, and because such events are not present in the training data, the current implementations will identify this as anomalous behavior. This could be avoided by taking advantage of the system states.

Chapter 8

Conclusion

The vast amounts of available sensor data in modern production systems present the industry with big opportunities. Automated procedures to identify unusual, anomalous behavior for equipment in complex production systems can be of high value to any production company, as they can be used as early warning systems to avoid imminent equipment failures and, ultimately, reduce production downtime. As such, the objective of this study has been to demonstrate how to implement unsupervised, real-time anomaly detection algorithms for critical equipment in the oil and gas industry. We have focused on model-based approaches, where the classification of a measurement is based on the difference between the model prediction and the sensor measurement. Using probabilistic sequential neural networks as predictive models, we propose two robust anomaly detection methods that conform to the high demands of a real-time, streaming data anomaly detection algorithm. They perform a fast and efficient evaluation of the incoming data in real-time, and they work in an unsupervised fashion.

The predictive models are crucial to the success of the model-based anomaly detection methods. Seeing deep learning as a highly successful framework in similarly complex modeling problems, this thesis has been particularly devoted to exploring the potential of deep neural networks in predicting the behavior of the equipment. We find that the implemented networks provide highly accurate and reliable predictions of the normal behavior of the compressor. After implementing a range of different networks of different types and designs, we concluded that the best performance was achieved by an ensemble model consisting of a LSTM network, using a history of sensor data to make its prediction, and a feed-forward neural network, with no historical variables in its inputs. We implemented a set of common-sense heuristics and less advanced machine learning models to benchmark the performance. The deep learning ensemble outperformed the best benchmark model, which was ordinary least squares linear regression, with a relative improvement of 17.2%. Moreover, most of the implemented benchmarks are already performing at their full potential, while there is no reason to believe that this is the case for the deep learning models; fine-tuning hyperparameters, exploring alternative architectures or including more data is likely to improve the performance even further. We argue that the results show that there is a big potential for deep learning in modeling the behavior of physical equipment in the petroleum industry.

We have developed two model-based anomaly detection algorithms. The first classifies anomalies by comparing the residuals of a new observation to a static residual distribution built on the validation set, while the second identifies anomalies by checking if it is within the bounds of the prediction interval of the predictive model. We assessed the quality of the two methods by applying them to three arbitrary months of data that, to the best of our knowledge, contained clear anomalous behavior. We found that both methods generated anomaly warnings where it was expected and that the performance was highly satisfactory. However, the quality assessment is based on subjective expectations, and their actual performance must be evaluated by experts. Moreover, the anomaly thresholds of the respective methods must be fine-tuned before they can be used in an operational setting.

Bibliography

D. W. Aha, R. L. Bankert, D. Fisher, and H.-J. Lenz. *A Comparative Evaluation of Sequential Feature Selection Algorithms*, pages 199–206. Springer New York, New York, NY, 1996. doi: 10.1007/978-1-4612-2404-4_19. URL https://doi.org/10.1007/978-1-4612-2404-4_19.

S. Ahmad, A. Lavin, S. Purdy, and Z. Agha. Unsupervised real-time anomaly detection for streaming data. *Neurocomputing*, 262:134–147, 2017.

M. Anand. Digital transformation in the oil and gas industry: Drill, data, drill. Technical report, Cisco Blog, The Platform, 2015. URL https://blogs.cisco.com/news/digital-transformation-in-the-oil-gas-industry-drill-data-drill.

M. Assaad, R. Boné, and H. Cardot. A new boosting algorithm for improved time-series forecasting with recurrent neural networks. *Inf. Fusion*, 9(1):41–55, Jan. 2008. ISSN 1566-2535. doi: 10.1016/j.inffus.2006.10.009. URL http://dx.doi.org/10.1016/j.inffus.2006.10.009.

M. Basseville and I. Nikiforov. *Detection of Abrupt Change Theory and Application*, volume 15. PTR Prentice-Hall, 04 1993. ISBN 0-13-126780-9.

S. D. Bay and M. Schwabacher. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 29–38. ACM, 2003.

J. Benesty, J. Chen, Y. Huang, and I. Cohen. Pearson correlation coefficient. In *Noise reduction in speech processing*, pages 1–4. Springer, 2009.

D. K. Bhattacharyya and J. K. Kalita. *Network Anomaly Detection: A Machine Learning Perspective*. Chapman & Hall/CRC, 2013. ISBN 1466582081, 9781466582088.

A. Bianco, M. Garcia Ben, E. J. Martínez, and V. Yohai. Outlier detection in regression models with arima errors using robust estimates. *Journal of Forecasting*, 20, 12 2001. doi: 10.1002/for.768.

C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. Weight Uncertainty in Neural Networks. *ArXiv e-prints*, May 2015.

S.-D. Bolboaca and L. Jäntschi. Pearson versus spearman, kendall's tau correlation analysis on structure-activity relationships of biologic active compounds. *Leonardo Journal of Sciences*, 5(9): 179–200, 2006.

P. J. Brockwell and R. A. Davis. *Introduction to Time Series and Forecasting*. Springer, 2016.

G. O. Campos, A. Zimek, J. Sander, R. J. G. B. Campello, B. Micenková, E. Schubert, I. Assent, and M. E. Houle. On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study. *Data Mining and Knowledge Discovery*, 30(4):891–927, Jul 2016. doi: 10.1007/s10618-015-0444-8. URL https://doi.org/10.1007/s10618-015-0444-8.

V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.

F. Chollet. *Deep Learning with Python*. Manning Publications, 2018.

J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014. URL http://arxiv.org/abs/1412.3555.

F. Degenhardt, S. Seifert, and S. Szymczak. Evaluation of variable selection methods for random forests and omics data sets. *Briefings in bioinformatics*, 20(2):492–503, 2017.

T. G. Dietterich. Ensemble methods in machine learning. In *Proceedings of the First International Workshop on Multiple Classifier Systems*, MCS '00, pages 1–15, London, UK, UK, 2000. Springer-Verlag. ISBN 3-540-67704-6. URL http://dl.acm.org/citation.cfm?id=648054.743935.

T. Fawcett and F. Provost. Adaptive fraud detection. *Data mining and knowledge discovery*, 1(3): 291–316, 1997.

Y. Gal and Z. Ghahramani. *Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning*. PhD thesis, University of Cambridge, June 2015.

J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia. A survey on concept drift adaptation. *ACM computing surveys (CSUR)*, 46(4):44, 2014.

M. Goadrich, L. Oliphant, and J. Shavlik. Learning ensembles of first-order clauses for recall-precision curves: A case study in biomedical information extraction. In *International Conference on Inductive Logic Programming*, pages 98–115. Springer, 2004.

M. Goldstein and S. Uchida. A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. *PloS one*, 11(4):e0152173, 2016.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

N. Görnitz, M. Kloft, K. Rieck, and U. Brefeld. Toward supervised anomaly detection. *J. Artif. Int. Res.*, 46(1):235–262, Jan. 2013. ISSN 1076-9757. URL http://dl.acm.org/citation.cfm?id=2512538.2512545.

S. Guha, R. Rastogi, and K. Shim. Rock: A robust clustering algorithm for categorical attributes. *Information systems*, 25(5):345–366, 2000.

I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar):1157–1182, 2003.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, 2 edition, 2009.

M. A. Hayes and M. A. Capretz. Contextual anomaly detection framework for big sensor data. *Journal of Big Data*, 2(1):2, 2015.

J. Helvoirt. Centrifugal compressor surge:modeling and identification for control. *Annali Di Matematica Pura Ed Applicata - ANN MAT PUR APPL*, 01 2007.

D. J. Hill and B. S. Minsker. Anomaly detection in streaming environmental sensor data: A data-driven modeling approach. *Environmental Modelling & Software*, 25(9):1014–1022, 2010.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8), 1997.

J. Honaker, G. King, M. Blackwell, et al. Amelia ii: A program for missing data. *Journal of statistical software*, 45(7):1–47, 2011.

K. Hundman, V. Constantinou, C. Laporte, I. Colwell, and T. Soderstrom. Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 387–395. ACM, 2018.

R. J. Hyndman and G. Athanasopoulos. *Forecasting: Principles and Practice*. OTexts, 2 edition, May 2018. URL https://otexts.org/fpp2/. Publicly available at https://otexts.org/fpp2/.

M. Ileby and E. Knutsen. Data-driven remote condition monitoring optimizes offshore maintenance, reduces costs. Technical report, World Oil, 2017.

G. K. Karagiannidis and A. S. Lioumpas. An improved approximation for the gaussian q-function. *IEEE Communications Letters*, 11(8):644–646, 2007.

A. Kendall and Y. Gal. What uncertainties do we need in bayesian deep learning for computer vision? *CoRR*, abs/1703.04977, 2017. URL http://arxiv.org/abs/1703.04977.

N. Ketkar. *Deep Learning with Python: A Hands-on Introduction*. Springer, Bangalore, Karnataka, India, 2017.

E. M. Knorr, R. T. Ng, and V. Tucakov. Distance-based outliers: algorithms and applications. *The VLDB Journal—The International Journal on Very Large Data Bases*, 8(3-4):237–253, 2000.

S. Kullback and R. A. Leibler. On information and sufficiency. *Ann. Math. Statist.*, 22(1):79–86, 03 1951. doi: 10.1214/aoms/1177729694. URL https://doi.org/10.1214/aoms/1177729694.

M. B. Kursa, W. R. Rudnicki, et al. Feature selection with the boruta package. *J Stat Softw*, 36(11): 1–13, 2010.

D. L. Simon and A. Rinehart. A model-based anomaly detection approach for analyzing streaming aircraft engine measurement data. *American Society of Mechanical Engineers: New York*, 6, 06 2014. doi: 10.1115/GT2014-27172.

N. Laptev, S. Amizadeh, and I. Flint. Generic and scalable framework for automated time-series anomaly detection. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pages 1939–1947, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3664-2. doi: 10.1145/2783258.2788611. URL http://doi.acm.org/10.1145/2783258.2788611.

N. Laptev, J. Yosinski, L. E. Li, and S. Smyl. Time-series extreme event forecasting with neural networks at uber. *Time Series Workshop 2017*, 3, 2017.

T. J. Lee, J. Gottschlich, N. Tatbul, E. Metcalf, and S. Zdonik. Precision and recall for range-based anomaly detection. *arXiv preprint arXiv:1801.03175*, 2018.

P. Malhotra, L. Vig, G. Shroff, and P. Agarwal. Long short term memory networks for anomaly detection in time series. In *Proceedings*, page 89. Presses universitaires de Louvain, 2015.

P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. Shroff. Lstm-based encoder-decoder for multi-sensor anomaly detection. *arXiv preprint arXiv:1607.00148*, 2016.

K. G. Mehrotra, C. K. Mohan, and H. Huang. *Anomaly Detection Principles and Algorithms*. Springer Publishing Company, Incorporated, 1st edition, 2017. ISBN 3319675249, 9783319675244.

Y. S. A. Mostafa, M. Magdon-Ismail, and H.-T. Lin. *Learning From Data*. AMLbook.com, 2012.

A. Nairac, N. Townsend, R. Carr, S. King, P. Cowley, and L. Tarassenko. A system for the analysis of jet engine vibration data. *Integrated Computer-Aided Engineering*, 6(1):53–66, 1999.

C. Olah. Understanding lstm networks, 2015. URL https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

M. Pratama, J. Lu, E. Lughofer, G. Zhang, and S. Anavatti. Scaffolding type-2 classifier for incremental learning under concept drifts. *Neurocomputing*, 191:304–329, 2016.

K. Sequeira and M. Zaki. Admit: anomaly-based data mining for intrusions. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 386–395. ACM, 2002.

S. S. Shapiro and M. B. Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 52(3/4):591–611, 1965.

D. T. Shipmon, J. M. Gurevitch, P. M. Piselli, and S. T. Edwards. Time series anomaly detection; detection of anomalous drops with limited features and sparse examples in noisy highly periodic data. *arXiv preprint arXiv:1708.03665*, 2017.

U. Shuchita and S. Karanjit. Nearest neighbour based outlier detection techniques. *International Journal of Computer Trends and Technology*, 3:299–303, 01 2012.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15: 1929–1958, 2014. URL `http://jmlr.org/papers/v15/srivastava14a.html`.

M. Szmit and A. Szmit. Usage of modified holt-winters method in the anomaly detection of network traffic: Case studies. *Journal of Computer Systems Networks and Communications*, 5, 05 2012. doi: 10.1155/2012/192913.

S. C. Tan, K. M. Ting, and T. F. Liu. Fast anomaly detection for streaming data. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

B. Upadhyaya, O. Glockler, and J. Eklund. Multivariate statistical signal processing technique for fault detection and diagnostics. *ISA transactions*, 29(4):79–95, 1990.

B. R. Upadhyaya and M. Skorska. Sensor fault analysis using decision theory and data-driven modeling of pressurized water reactor subsystems. *Nuclear Technology*, 64(1):70–77, 1984.

M. Yuan, Y. Wu, and L. Lin. Fault diagnosis and remaining useful life estimation of aero engine using lstm neural network. In *2016 IEEE International Conference on Aircraft Utility Systems (AUS)*, pages 135–140. IEEE, 2016.

J. H. Zar. Significance testing of the spearman rank correlation coefficient. *Journal of the American Statistical Association*, 67(339):578–580, 1972.

L. Zhu and N. Laptev. Deep and Confident Prediction for Time Series at Uber. *ArXiv e-prints*, Sept. 2017.

Appendix A

Additional Theory

A.1 Data Preprocessing Techniques

A.1.1 Boruta

Boruta is a novel feature selection algorithm for finding *all relevant variables*. The algorithm is named after the God of the forest in Slavic mythology and serves as a wrapper around a Random Forest classifier (Kursa et al., 2010). In essence, the main goal of the Boruta algorithm is to find all the important and interesting features in the dataset for a given outcome variable by comparing the importance of the real predictor features to those of so-called shadow features using statistical testing and several iterations of Random Forest classifiers. The shadow features are generated by permuting the original values across observations, thus dismantle the relationship with the outcome. In general, the algorithm is based on the same idea that forms the foundation of the Random Forest classifier, that by collecting results from the ensemble of randomized samples one can reduce the misleading impact of random fluctuations and correlations (Kursa et al., 2010).

A.1.2 Amelia II

Under the assumption that the complete data (that is, both observed and unobserved) is multivariate normal and that the observations are *missing at random* (MAR), the Amelia II algorithm can impute the incomplete data set so that analyses which require complete observations can appropriately use all the information present in a dataset. Consequently, biases and incorrect uncertainty estimates that can result from dropping all partially observed observations from the analysis are avoided (Honaker et al., 2011). In essence, the algorithm utilizes an Expectation-Maximization with Bootstrapping (EMB) to perform multiple imputations where the missing values are filled in with a distribution of imputations that reflect the uncertainty of the missing data (Honaker et al., 2011).

A.2 Bayesian modelling

Let the training data $\mathcal{D}_t$ be defined by a set of inputs $\mathbf{X} = \{\mathbf{x_1}, ..., \mathbf{x_N}\}$ and their corresponding outputs $\mathbf{Y} = \{\mathbf{y_1}, ..., \mathbf{y_N}\}$. Bayesian modeling aims to find the parameters θ of the function $\mathbf{y} = f^{\theta}(\mathbf{X})$ that are likely to have generated the output $\mathbf{Y}$ (Gal and Ghahramani, 2015). In plain English, the question is: ”How can the parameters of the function f^{θ} be set such that it *likely* maps the input to the output?”.

To answer this question using Bayesian statistics, a prior distribution for the parameters $P(\theta)$ is assumed, representing the prior beliefs of which parameters that have generated the data before any data samples have been observed. Further, a likelihood distribution $P(\mathbf{y}|\mathbf{x}, \theta)$ is defined, better described as the probability of the features $\mathbf{x}$ generating the outputs $\mathbf{y}$ given a set of parameters θ. For regression problems, a Gaussian distribution is typically assumed:

$$P(\mathbf{y}|\mathbf{x}, \theta) \sim \mathcal{N}(\mathbf{y}; f^{\theta}(\mathbf{x}), I)$$

Invoking Bayes’ theorem, the posterior distribution of the parameters θ can be defined as follows:

$$P(\theta|\mathbf{X}, \mathbf{Y}) = \frac{P(\mathbf{Y}, \mathbf{X}, \theta)P(\theta)}{P(\mathbf{Y}|\mathbf{X})}$$

The posterior is the distribution of the most likely function parameters given the observed data $\mathcal{D} = \{\mathbf{X}, \mathbf{Y}\}$ (Gal and Ghahramani, 2015). This posterior can be be used to calculate the output of a new input point x^*:

$$P(\mathbf{y}^*|\mathbf{x}^*, \theta, \mathcal{D}) = \int P(\mathbf{y}^*|\mathbf{x}^*, \theta)P(\theta|\mathcal{D})d\theta$$

Approximate inference

An important part of the posterior is the *model evidence* $P(\mathbf{Y}|\mathbf{X})$ (Gal and Ghahramani, 2015). It is the marginalization of the likelihood over θ, i.e. the weighted average of the probability with respect to all possible values in θ. It can be calculated with the following integral:

$$P(\mathbf{X}|\mathbf{Y}) = \int P(\mathbf{Y}|\mathbf{X}, \theta)P(\theta)d\theta$$

The marginal likelihood is possible to evaluate analytically for simple models such as linear regression models. Unfortunately, the analytic solution quickly becomes intractable for more complex models (Gal and Ghahramani, 2015). Particularly in the context of deep learning, representing a class of models characterized by high non-linearity and non-conjugacy, the true posterior cannot be solved analytically (Zhu and Laptev, 2017). As an alternative, it is possible to approximate the posterior.

There are two well studied approaches to deal with this problem - Markov Chain Monte Carlo (MCMC) and Variational Inference (VI). MCMC will perform successive random sampling from the target distribution (the posterior) which in effect forms a Markov Chain. It can be viewed as an intelligent way to draw samples from a high-dimensional distribution. VI aims to approximate the posterior by defining a *variational distribution* $q_\omega(\theta)$ that is parameterized by ω and is much easier to evaluate. MCMC can converge at a more accurate result than the VI often can, but it is significantly more computationally expensive. For this reason, it is VI that is most often used.

To make the approximate distribution as similar to the true posterior as possible ($q_\omega(\theta) \approx P(\theta|\mathbf{X}, \mathbf{Y})$), one finds the parameters of ω that minimize the Kullback–Leibler (KL) divergence (Kullback and Leibler, 1951)[1] between the approximate distribution and the true posterior. Srivastava et al. (2014) shows that this is equivalent to maximizing the *evidence lower bound* with respect to θ:

$$\begin{aligned}\omega^* &= \underset{\omega}{\operatorname{argmin}} \, \mathrm{KL}[q_\omega(\theta)||P(\theta|\mathcal{D})] \\ &= \underset{\omega}{\operatorname{argmin}} \, \mathrm{KL}[q_\omega(\theta)||P(\theta)] - \mathbb{E}_{P(\theta|\omega)}[\log P(\mathcal{D}|\theta)]\end{aligned} \tag{A.1}$$

The exact solution to this minimization problem is intractable, so optimization techniques such as gradient descent and other approximations are applied. A common approach is to use Monte Carlo sampling to approximate the integral: Instead of regarding all possible set of parameters θ_0^∞, a finite sample of possible parameters are evaluated for each iteration. The interested reader is encouraged to read more on variational inference in Blundell et al. (2015). Variational inference approximates the intractable Bayesian marginalisation procedure as an optimization problem, replacing calculations of integrals with derivatives, which are a lot easier to evaluate.

[1] The Kullback-Leibler divergence is a measure of similarity between two probability distributions

Appendix B

Preprocessing and Statistical Properties

B.1 Data Preprocessing

B.1.1 Feature sparsity

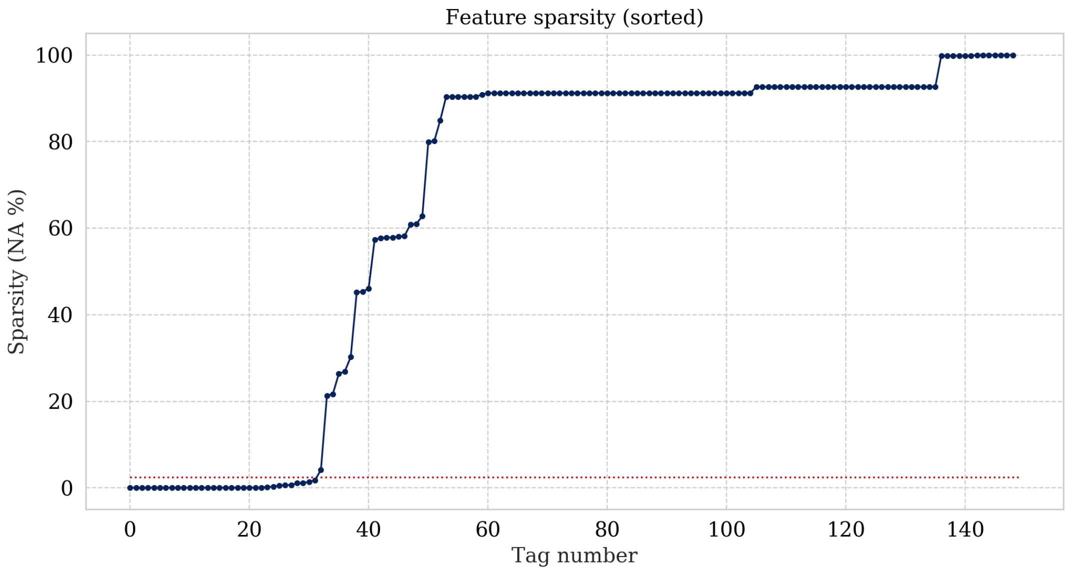
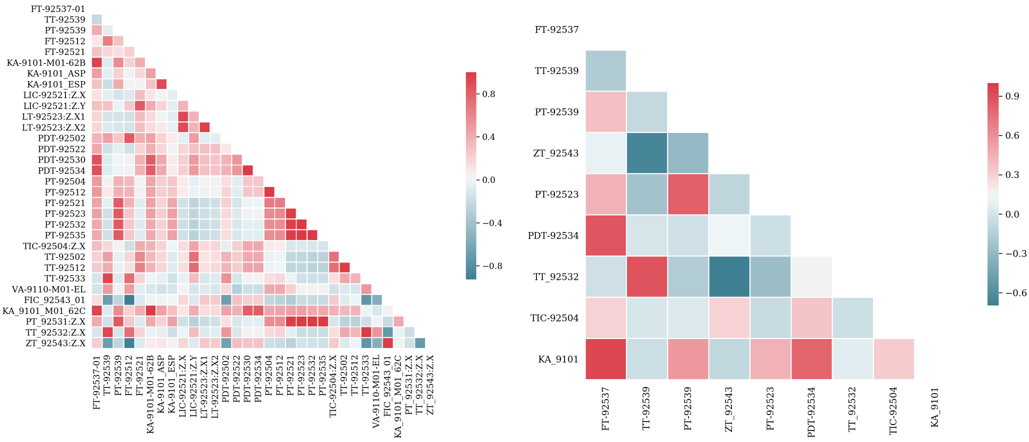


Figure B.1: Feature sparsity

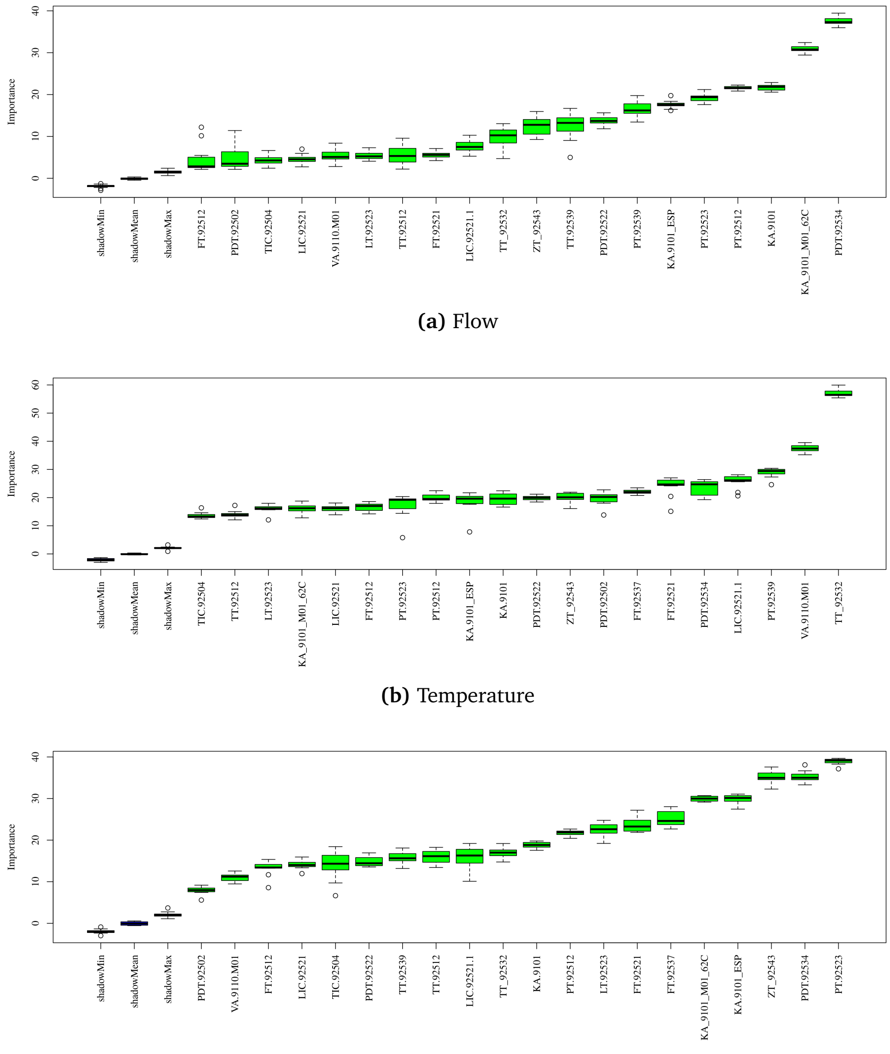
B.1.2 Correlation plots



(a) Cross-correlation matrix of the 32 filtered variables. **(b)** Cross-correlation matrix of the 9 finally chosen variables.

Figure B.2: Cross correlation plots of the data variables

B.1.3 Feature selection results



(a) Flow

(b) Temperature

(c) Pressure

Figure B.3: Box plot of the feature importance for the target variables flow, temperature, and pressure. Green variables are considered important and blue variables corresponds to the *shadow* features, i.e., the randomly shuffled features.

B.2 Summary Statistics and Distribution Plots

	Mean	Median	Std	Max	Min	1st Qu.	3rd Qu.	NAs
FT-92537	161417.2	161285.9	3451.3	224396.1	147442.2	159105.7	163550.6	18.0
TT-92539	125.1	125.1	1.2	130.6	119.5	124.3	125.9	1227.0
PT-92539	12.7	12.7	0.2	14.7	11.8	12.5	12.9	21.0
ZT_92543	36.4	36.4	4.2	60.9	16.5	33.3	39.5	1006.0
PT-92523	2.9	2.9	0.1	4.0	2.6	2.8	2.9	31.0
PDT-92534	103.2	103.0	4.1	157.1	86.0	100.4	105.7	20.0
TT_92532	33.4	33.4	1.2	40.0	27.9	32.5	34.2	960.0
TIC-92504	39.7	39.7	1.3	45.1	35.1	38.8	40.4	50.0
KA_9101	9026.3	9026.8	163.3	11349.9	8230.1	8917.4	9134.4	28.0

Table B.1: Statistics of the complete dataset before scaling. The first three rows are the target variables, and the remaining rows are the input features.

	Mean	Median	Std	Max	Min	1st Qu.	3rd Qu.	NAs
FT-92537	-0.115	-0.154	1.022	18.536	-4.254	-0.800	0.516	0.0
TT-92539	0.603	0.589	1.213	6.225	-5.099	-0.215	1.425	0.0
PT-92539	0.003	0.022	0.992	7.917	-3.816	-0.721	0.718	0.0
ZT_92543	-0.579	-0.564	1.198	6.386	-6.214	-1.460	0.301	0.0
PT-92523	0.031	-0.077	1.010	14.442	-3.743	-0.694	0.650	0.0
PDT-92534	-0.113	-0.166	1.005	13.174	-4.357	-0.806	0.512	0.0
TT_92532	0.607	0.681	1.176	7.239	-4.816	-0.213	1.433	0.0
TIC-92504	-0.011	0.066	1.283	5.217	-4.396	-0.796	0.660	0.0
KA_9101	-0.073	-0.070	1.008	14.272	-4.988	-0.745	0.594	0.0

Table B.2: Statistics of the complete dataset after scaling. The first three rows are the target variables, and the remaining rows are the input features.

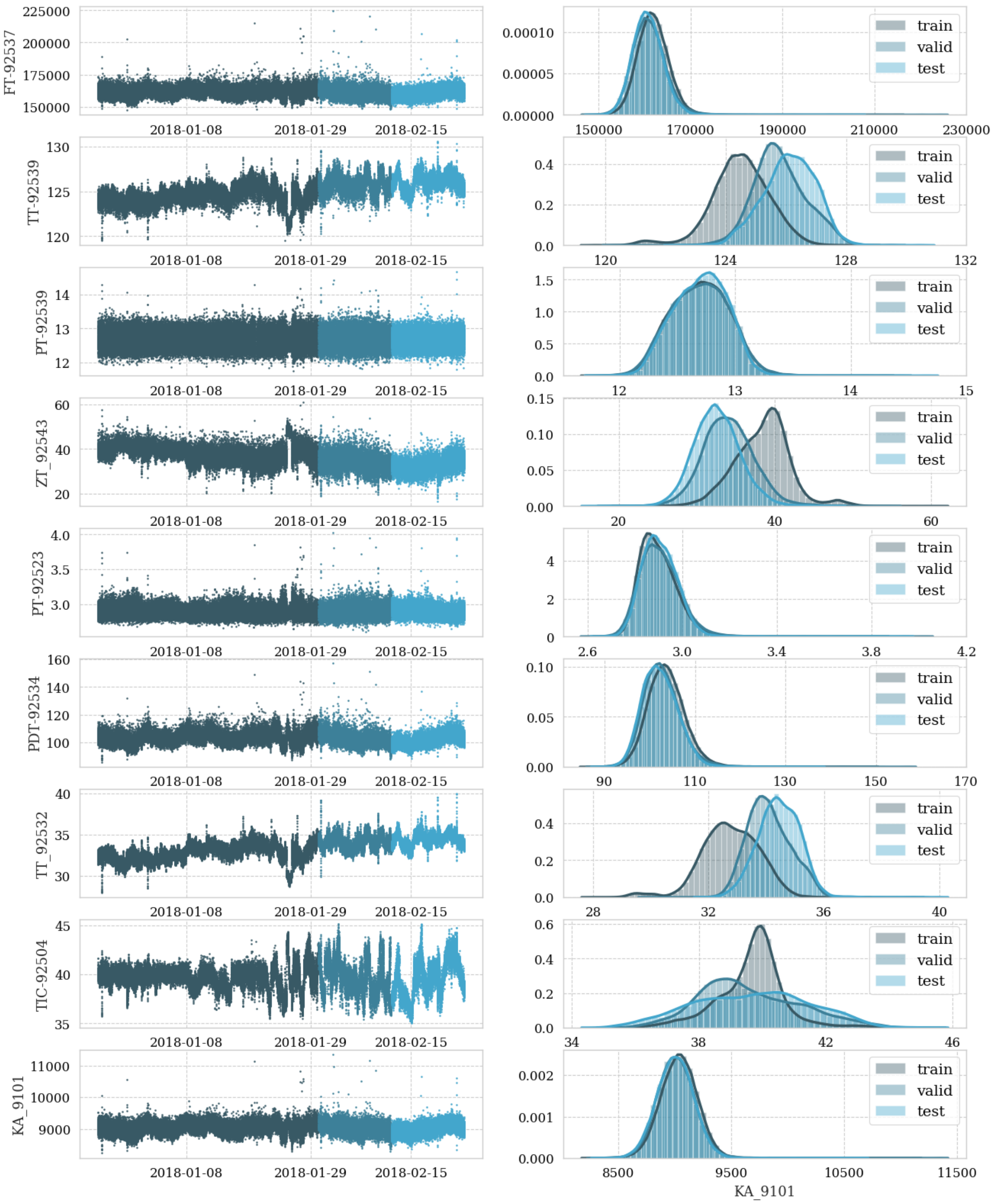


Figure B.4: Distributions of the training, validation and test data.

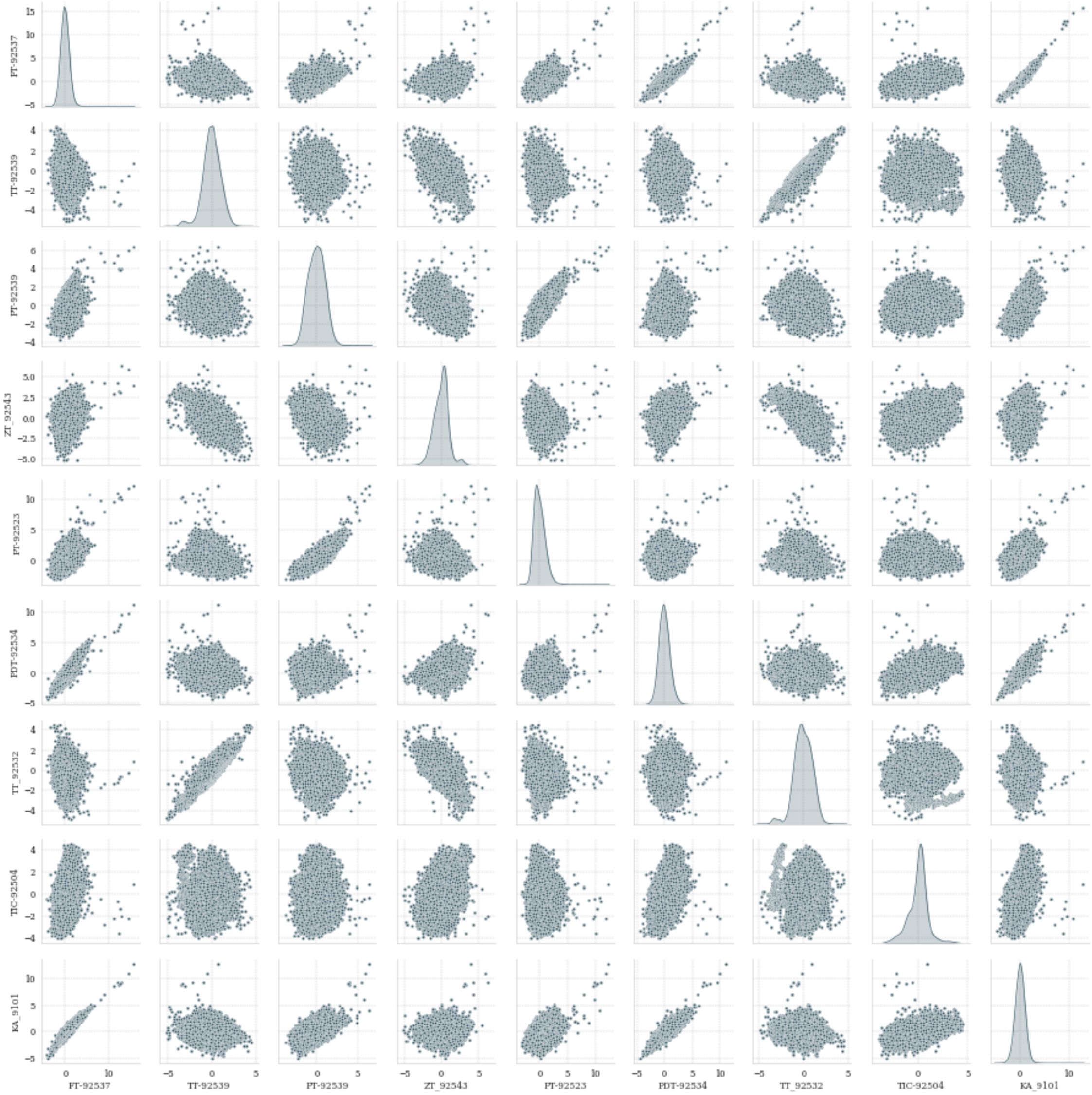


Figure B.5: Scatter matrix plot of the data with Kernel Density Estimation in the diagnoal

Appendix C

Deep Learning Models & Anomaly Detection

C.1 Training Histories and Validation Data

C.1.1 LSTM

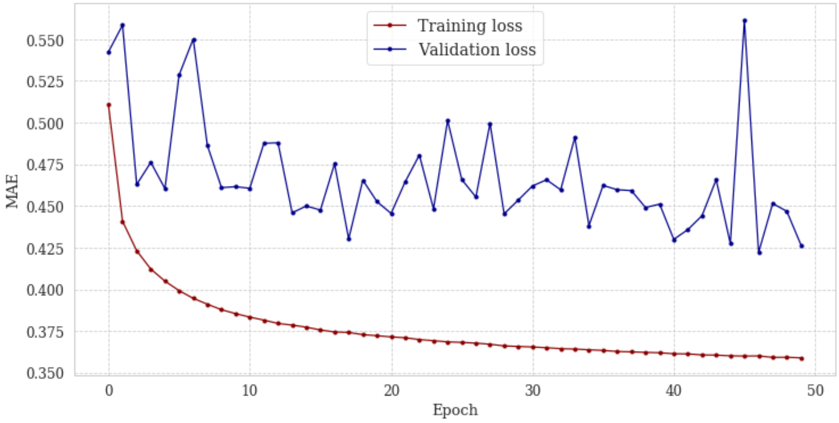


Figure C.1: Training history of the LSTM model.

C.1.2 GRU

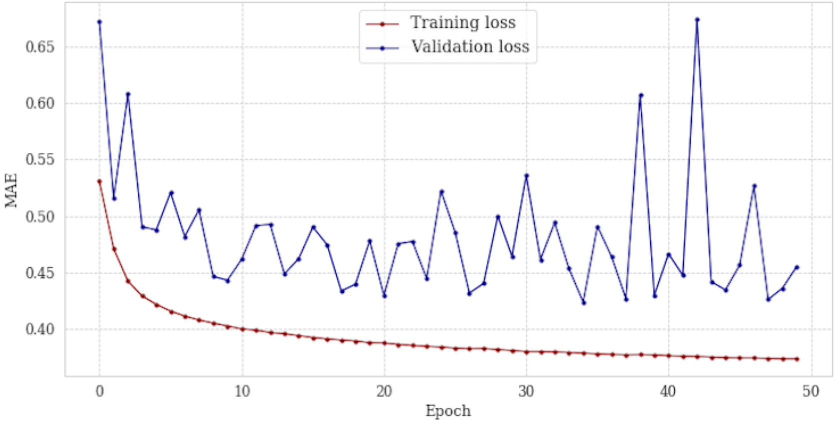


Figure C.2: Training history of the GRU model.

C.1.3 MLP

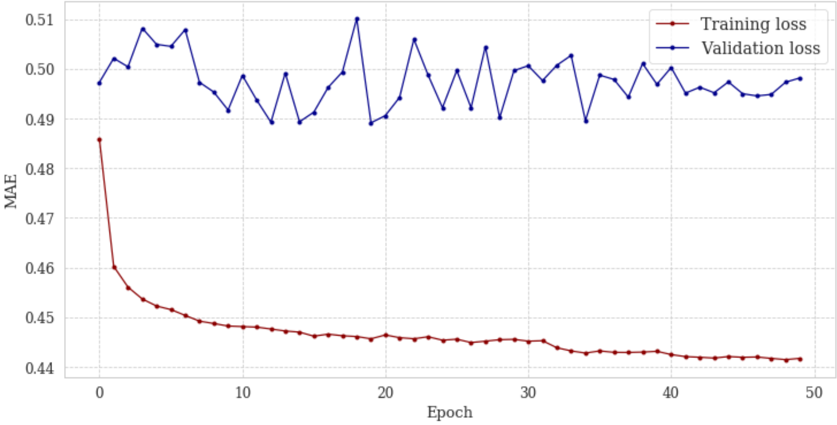


Figure C.3: Training history of the MLP model.

C.2 Anomaly Detection

C.2.1 Method I: residual based anomaly detection

Threshold analysis

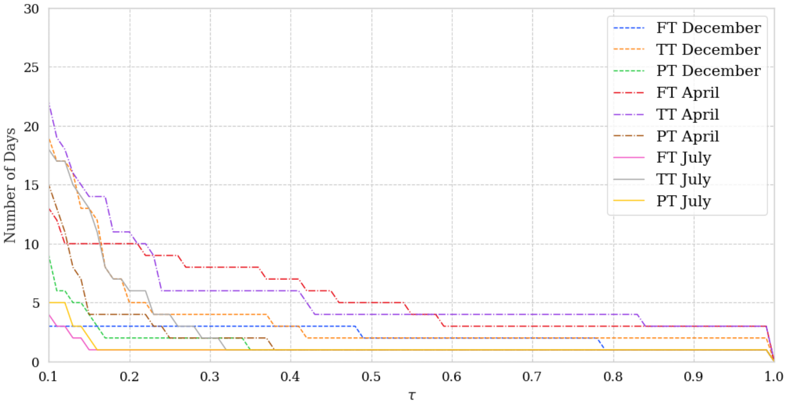
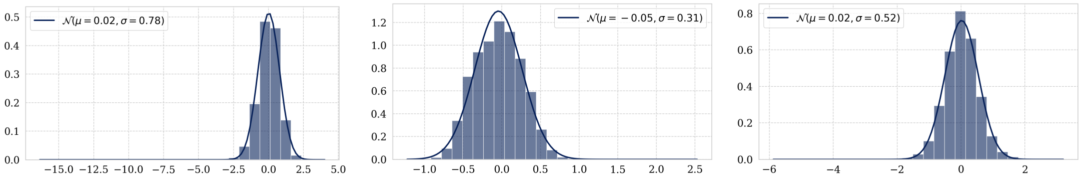


Figure C.4: Number of days where method I identifies anomalous behavior with increasing threshold, τ.

Residual distributions



(a) Flow residual dist.

(b) Temperature residual dist.

(c) Pressure residual dist.

Figure C.5: The residual distributions used in model I