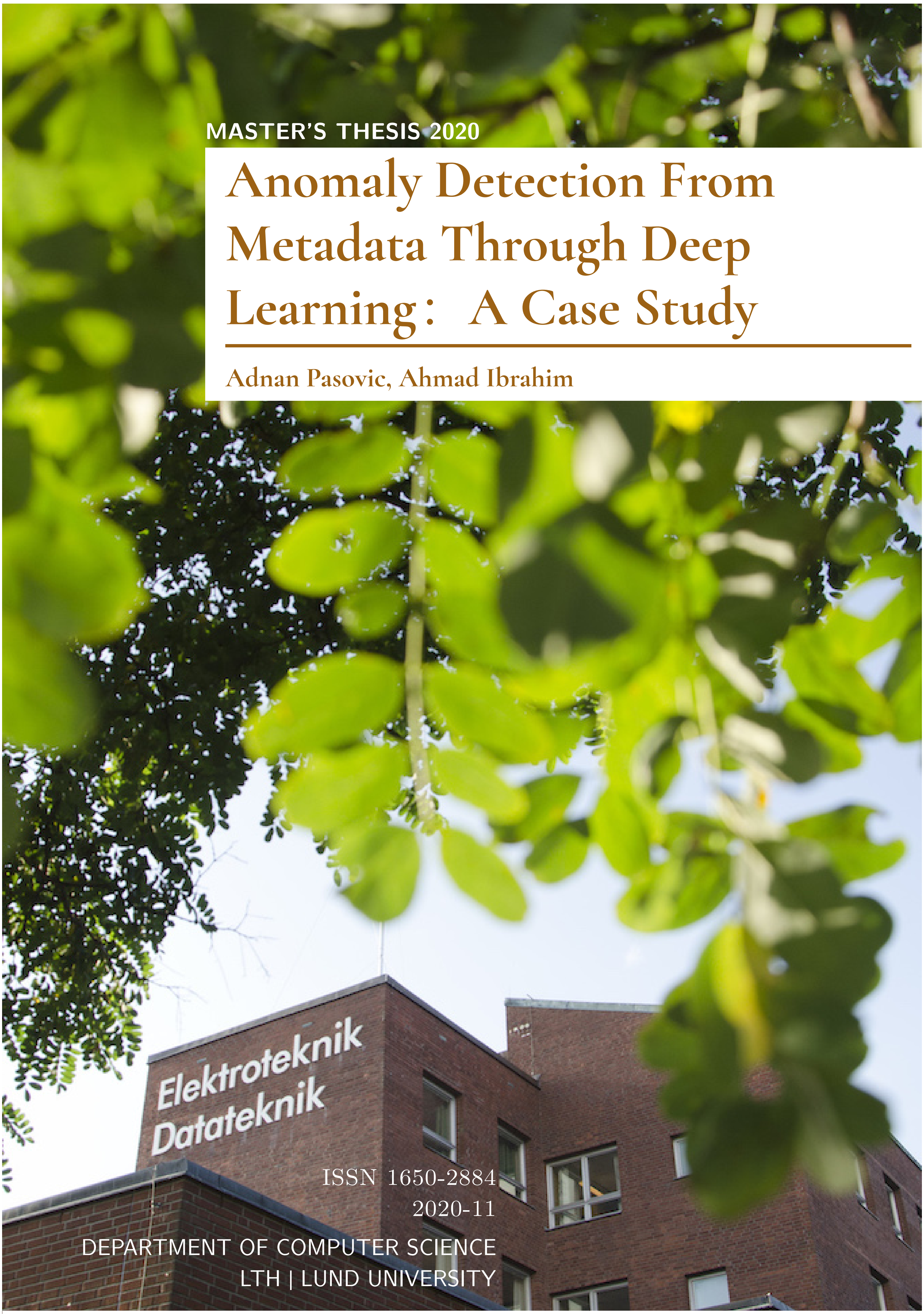

MASTER'S THESIS 2020

Anomaly Detection From Metadata Through Deep Learning: A Case Study

Adnan Pasovic, Ahmad Ibrahim

ISSN 1650-2884
2020-11

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

2020-11

Anomaly Detection From Metadata Through Deep Learning：A Case Study

Adnan Pasovic, Ahmad Ibrahim

Anomaly Detection From Metadata Through Deep Learning: A Case Study

Adnan Pasovic
dat15apa@student.lu.se

Ahmad Ibrahim
dat15aib@student.lu.se

March 27, 2020

Master's thesis work carried out at Sigma IT AB.

Supervisors:
Per Andersson, per.andersson@cs.lth.se
Emma Sjögren, emma.sjogren@sigma.se
Johan Wredin, johan.wredin@sigma.se
Filip Titoff, filip.titoff@sigma.se

Examiner: Flavius Gruian, flavius.gruian@cs.lth.se

Abstract

This thesis forms a case study where we implement anomaly detection for processes that transport and transform data in data warehousing. Anomaly detection is performed by utilizing metadata, more specifically process execution time and the amount of processed data. We implement three recurrent neural networks, a vanilla recurrent prediction network, a stacked Long-Short-Term-Memory (LSTM) prediction model, and a LSTM autoencoder, then evaluate and compare their performance.

Our results show that recurrent neural networks are suitable structures for anomaly detection in time series data, where we obtain F1-scores in similar ranges as those found in related work. Our tests do not definitively show one model outperforming the others in the general case, however our results indicate that certain models perform best for individual anomaly types.

We found that our approach of using data embodying normal process behavior for anomaly detection works relatively well, despite difficulties in fully separating anomalous and normal data. We conclude that the adopted approach would have been more suitable in a setting where normal and anomalous data could be more clearly defined.

Keywords: Machine Learning, Deep Learning, Anomaly Detection, RNN, LSTM, Autoencoder, Time Series.

Acknowledgements

First and foremost we would like to thank our supervisor Per Andersson for his invaluable input and guidance, which helped us considerably in our work. We then extend our gratitude to Sigma and our supervisors on site, for giving us the opportunity to perform this case study and for their support during our work.

Contents

Chapter 1
Introduction

The ability to detect anomalies in data processing tasks can reduce the number of costly and time consuming errors and process interruptions. Anomalies can be very subtle and therefore hard to detect with the naked eye, yet their effects can be dramatic, especially if they persist over a longer period of time. An example anomaly in this particular context would be a process that does not mark processed batches of data as completed, but instead reprocesses older data repeatedly. As time progresses such a process will have an increase in both execution time and amount of processed data, this increase may however be very subtle and hard to detect when observing a shorter time span. Deep learning approaches have become increasingly popular in the field of anomaly detection, often outperforming traditional anomaly detection models [1]. Deciding on what technique to adopt for a specific problem is a challenge on its own, as there is a broad spectrum of available techniques and numerous ways to approach each specific case.

In this chapter we introduce anomaly detection and describe the context in which this case study has been conducted. We introduce the Data & Advanced Analytics department of Sigma IT Consulting and define the goals of this thesis.

1.1 Anomaly detection

Anomaly detection is the activity in which we are trying to identify samples or patterns in data that deviate from normal behaviour. Anomaly detection is applied in various domains, such as fraud detection, intrusion detection, network event detection, and system monitoring. Anomalies can be divided [2] into three categories:

- If a single data instance diverges from the rest of the data, it is considered a *point* anomaly. As an illustrative example, imagine anomaly detection in credit card use where we only take the debited amount into consideration. An amount significantly higher than any other previous purchases could be flagged as an anomaly.

- If a data instance displays behaviour that is anomalous when observed in the given context, but that behaviour could be fully normal in another context, it is seen as an *contextual* anomaly. An example would be a data point in a seasonal time series, where that point value is not abnormal, yet in the context of seasonality the observed value deviates from the pattern. These types of anomalies require that we can define contexts in a meaningful way.
- If a set of interrelated data instances display anomalous behaviour relative to the rest of the data set, but each individual instance is not a point anomaly nor a contextual anomaly, it is regarded as a *collective* anomaly. An example of such an anomaly would be if the price of a company's stock would stagnate over several time steps, or if the residents of an entire neighborhood decided to move on the same day. The observations may not be anomalous individually, it may not be so strange that one of your neighbors moves, but it would be odd if all your neighbors decided to move on the same day.

A characteristic unique to time series are change points [3], which are points in time where the structure of the time series abruptly changes due to external or internal factors. Examples of such change points could be sudden changes to the trend of the time series, or changes in the state of the time series leading to significantly different behaviour. Change points do not always indicate the presence of an anomaly in the time series.

Anomaly detection requires that we first define normal behaviour, a task that is non-trivial. Real world data is often prone to noise and missing data, and labeled real world data sets are scarce. Training an anomaly detection model on such data can be challenging, as we often first must ensure that the training data embodies the normal behavior of the system we are trying to model. Additionally, anomalous events might be sparse or hard to detect, often resulting in a deficient number of events to be used for training.

1.2 Purpose

The purpose of this thesis is of two dimensions. Firstly, we aim to determine and choose what machine learning techniques that are most suited for anomaly detection with time series data in this particular case (see 1.3). Secondly, we will try to evaluate how generic the selected models are by evaluating them against different industry data sets.

1.2.1 Research Questions

We will limit the scope of the thesis by selecting two machine learning techniques that we deem most suited for this specific case. The selected models will be tested on four processes originating from different data sets. This is summarized by the following research questions.

1. Which machine learning techniques are suitable for anomaly detection using time series data?
2. How generic are the selected machine learning techniques?

   The second research question relates to the generic nature of models, where we mean to examine if any model clearly outperfoms the rest in the general case.

1.3 Case

The Data & Advanced Analytics department of Sigma IT Consulting AB (hereinafter referred to as "Sigma") offers consulting services in the areas of data integration, data warehousing, data analysis, and machine learning. With the relatively recent world wide digitisation, in addition to the current and projected increase in IoT-connected devices [4], there is a need among businesses to process and analyze large amounts of data efficiently.

Sigma offers clients a custom platform for data integration and data warehousing solutions. The platform consists of several *Extract Transform Load* and *Extract Load Transform* processes, which transform and transport data through the various stages and databases of a data warehouse. Each such process generates metadata which is stored in a specific database in the data warehouse. Metadata generated by the processes include, but are not limited to, source and destination information, timestamps, process status, and the number of rows affected. SQL Server Management Studio was used to develop the platform, which is a database management system provided by Microsoft, running the Transact-SQL extension.

The platform deployed by Sigma has a *Single Tentant Architecture*, meaning each implementation of the platform houses data from one client only. Processes are executed periodically, outside of office hours. Each implementation of the platform is unique, no two clients have the same platform, i.e the same underlying processes. All implementations do however handle metadata in the same way and they conform to the same metadata database structure. The metadata database structure can be seen in figure 1.1.

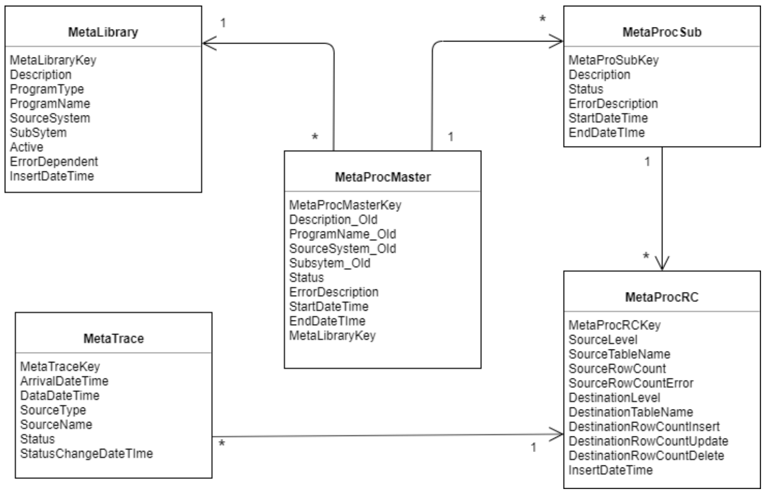

Figure 1.1: ER-model of metadata structure

The processes form a hierarchy where there are two types of processes, master processes and subprocesses. Master processes consist of zero or more subprocesses, and subprocesses have no subprocesses of their own. Master processes have a fixed start time and call their respective subprocesses, which are often interdependent and can therefore not have fixed start times. Figure 1.1 illustrates how master processes relate to subprocess, more importantly to the record count table MetaProcRC which was used to extract a representation of the amount of processed data. These processes are scheduled to not overlap to the extent possible, however when a process executes past its assigned time window it will execute in parallel with the next process. Parallel execution of processes can have a profound impact on process execution time since the underlying software can assign resources in arbitrary ways.

1.4 Contributions

This case study explores possible solutions to a domain specific problem within anomaly detection. An objective of this thesis has been to determine machine learning techniques suitable for the specific problem. Through our literature study we find several candidate techniques and implement a subset of these techniques. Our work shows that useful results can be obtained in a new domain with the selected techniques, further increasing their credibility as appropriate techniques for anomaly detection in time series data. Our work has not been able to identify a specific process as the most appropriate in the general case. The results do however indicate that certain techniques may perform better for specific anomalies.

Our approach is based on the assumption that networks trained on data portraying normal behavior will produce poor output when faced with input that deviates from the norm. A challenge in our work was to fully separate normal and anomalous data, leading to networks being trained on data that does not exclusively embody normal behavior. Our work shows that our selected approach can still yield useful results even when normal data is moderately polluted with anomalous instances.

1.5 Division of Work

Both authors have participated equally in the programming aspect of this case study, often dividing the work into similar tasks. Author A.P. has taken a lead role in the formulation, composition, and analytical aspects of the paper. Author A.I. has taken responsibility for the tests, and performed them independently. Both authors have participated in all activities.

Chapter 2
Theory

In this section we give an introduction to theory relevant to our thesis. We begin by describing time series, then introduce the fundamentals of Artificial Neural Networks. We will then briefly present more complex types of neural networks relevant to our thesis.

2.1 Time Series

A discrete time series is a sequence of data points, indexed in temporal order [5]. If the data points of a time series are measured with a regular interval, the time series is said to be *equidistant*. Any arbitrary number of variables that change over time can be viewed as a time series. A time series that depends on several variables is known as a *multivariate time series*. Time series can have several characteristics, one of which is *seasonality*, where the time series experiences periodic variation. Another characteristic is *stationarity*, where the mean value and variance are independent of time. There has been extensive research on the field of time series forecasting [6], where most of the traditional approaches are based on statistical analysis. Statistical analysis approaches often fall short when the complexity in the underlying data increases, and when data volumes reach the ranges of gigabytes [1].

2.2 Artificial Neural Networks

An Artificial Neural Network (ANN) is a machine learning model that tries to mimic the learning process of biological neural networks, such as those found in human beings [7]. Such a biological neural network consists of interconnected cells, called neurons, that communicate by electrical and chemical means. We will throughout this thesis refer to *artificial neural networks* as neural networks, not to be confused with biological ones.

The most trivial types of neural networks are the perceptrons, which consist of a single neuron [7], capable of learning linearly separable functions. In contrast to the perceptron, neural networks with multiple layers have several computational units and hidden layers [7]. When a neural network has several layers, the network is said to be *deep* [8]. By using several layers the model can extract and learn features of different levels of abstraction, where the abstraction level increases with the number of layers. A classical example is the image classification task, where the raw input is a matrix of pixel values. The first layer may abstract pixels into edges, the second layer may group edges together, a third layer could perhaps learn to identify facial features such as eyes, and a final fourth layer could identify a face in a picture. The network learns these features, and in which level to place them, on its own without the help of feature engineering from operators. This automatic feature extraction is a major advantage compared to traditional machine learning models where feature engineering is considered to be one of the most challenging tasks when building machine learning models [9].

2.2.1 Activation Functions

Activation functions are applied to the output of neurons with the purpose of introducing non-linearities in the output, which allows the network to learn and perform complex non-linear tasks [8, 10]. There exists a variety of different activation functions used in artificial neural networks, used in different context. In this thesis we will only use the *hyperbolic tanget* (tanh) and the *rectified linear unit* (ReLU), which are further described in [7].

2.2.2 Loss Functions

In neural networks, the *loss function* is used to calculate the difference between the actual output and the predicted output [7]. Networks are trained to minimize the loss function by optimizing the weight matrices in the neural network. When the weights of the network are updated, the loss function is calculated in order to determine how close the network comes to outputting the labels. There are several types of loss functions, an example is the *mean squared error* (MSE), often used in regression problems [7].

2.2.3 Optimization algorithms

The fundamental method by which neural networks learn is *gradient descent*, through which weights are altered as to minimize the loss function [11, 7]. The elemental gradient descent algorithm has some shortcomings, one of which are the oscillations that occur in the descent process and cause a higher number of steps to the local minima. To overcome such issues several *optimization algorithms* have been developed, one of which the *Adam* optimization algorithm [12], used in this thesis.

2.3 Recurrent Neural Networks

Imagine that we are trying to develop a model that predicts tomorrows stock price of a company share. It would be reasonable to assume that knowledge of prior prices would be beneficial in predicting tomorrows price, if there has been an upwards trend in the price we could assume that tomorrows price would be higher. This is because there exists a *temporal dependency* between different observations of the stock price, just like a time series. In situations such as this, our model would benefit from having access to prior observations. A class of neural networks that utilize information from previous time steps are the *recurrent neural networks*, which are a fundamental part of this thesis.

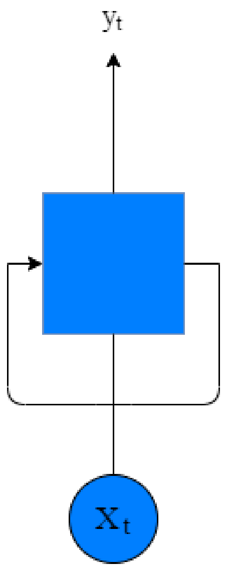

Figure 2.1: A recurrent neural network where the output of the current time step is fed back into the network at the next time step. The square represents an arbitrary network structure and the subscript *t* represents time.

Information from past time steps is stored in the *hidden state* of the RNN. The RNN is able to capture dependencies between sequences of data, and have been successfully applied in fields such as handwriting recognition, speech recognition, and language modeling [7] [13]. Recurrent neural networks are particularly suitable when working with sequences where observations are not independent of each other, in contrast to traditional neural networks where this independence is assumed [14]. Simple recurrent neural networks suffer from the explodin and vanishing gradient problem [15] [16], causing a weak ability to detect long-term dependencies in data. To overcome these issues more complex recurrent neural networks have been developed. One such model is the *long short-term memory* model, utilizing logical gates to control information flow [17, 15]. A similar solution is the *gated recurrent unit* (GRU) [18], which is less complex compared to the LSTM due to the reduced number of gates. In this thesis we choose to incorporate LSTM cells in our networks, since the available literature on LSTM networks outweighs the literature relating to GRU networks.

2.3.1 Autoencoders

An *autoencoder* is a neural network used for unsupervised learning where the aim is to reconstruct the input as the output [7]. This may seem trivial as one would assume that the network could just copy the input over to the output, however restrictions are placed on the network that prohibits it from using this approach. The network consists of an *encoder* that compresses the input into a latent space representation, the *decoder* then tries to reconstruct the input sequence by using the compressed version of the input [19]. By calculating the reconstruction error we can evaluate how well the model reconstructs the original input.

Typically the network is restricted from learning the identity function by having fewer nodes in the hidden layers than in the input layer. Figure 2.2 shows an autoencoder with a single hidden layer, the x_i are variables of the input sequence and the $\hat{x}_i$ are variables of the reconstructed sequence. Other implementations of autoencoders allow for hidden layers with more neurons than the input layer, and instead impose other limitations on the network. An example of such an architecture would be the *sparse autoencoder*, where a sparsity parameter is used to determine how often a node should be allowed to be active [20]. By imposing sparsity, the network is not allowed to use all neurons simultaneously, forcing it to learn a compressed version of the input. In this thesis we will utilize the sparse autoencoder structure.

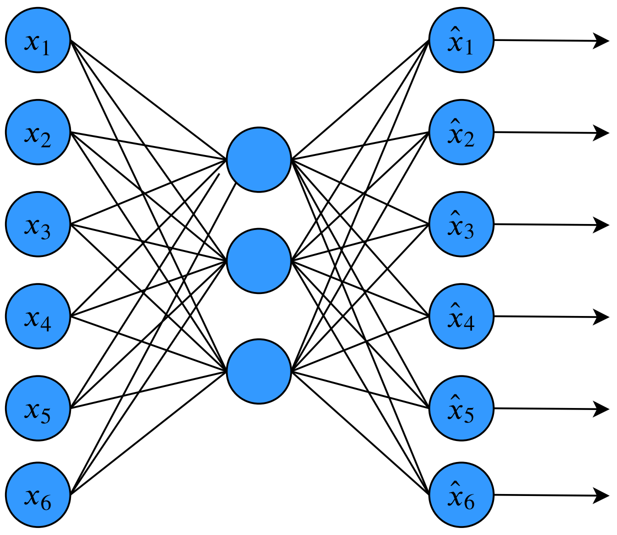

Figure 2.2: A simple autoencoder with a single hidden layer.

2.3.2 Evaluation metrics

Classification tasks in machine learning are a problem domain where we try to partition a data set into two or more categories. Evaluating such a model is often not as trivial as determining how many classifications were correct. For example, classifying an error as normal behavior in a nuclear power plant may be disastrous, also known as a *false negative*. A metric for the number of false negatives is *recall* (1), where the number of correctly identified items are divided by the total number of items that belong to the class.

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives} \tag{1}$$

In contrast, the nature of our justice system favors not processing innocent individuals over releasing a potential criminal, a *false positive* is a greater concern. A metric for the number of false positives is *precision* (2), where the number of correctly identified items is divided by the number is divided by the number of items put into the category.

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives} \tag{2}$$

A model that achieves perfect recall has a recall value of 1, synonymously a model that achieves perfect precision has a precision value of 1. The recall-precision metric is particularly useful in cases when the number of items in each class is greatly disproportionate, where both metrics are required to properly evaluate the model performance.

A metric that takes both recall and precision into consideration equally is the *F1 score* (3), which is the *harmonic mean* between recall and precision.

$$F1\ score = 2 * \frac{Recall * Precision}{Recall + Precision} \tag{3}$$

Depending on the problem domain, models can be tweaked to favour either recall, precision, or F1 score. In the case of nuclear power plant monitoring we would favour high recall to minimize the number of possible accidents. Our justice system however favours precision, minimizing the number of innocent people processed. For this particular case, F1-score is a good evaluation metric as we want to detect as many anomalies as possible without submerging operators in false anomaly reports.

Chapter 3
Research Method

In this section we present the methodology used for conducting this case study. This case study has drawn inspiration from the guidelines developed by Runeson and Höst [21].

3.1 Literature Review

The first step in our work is to determine techniques in the field of machine learning that are suitable for our particular case. To get an orientation of current literature and to examine if possible approaches to the problem exist, we performed a *literature review* following the guidelines of Randolph [22]. The literature review aims not only to help identify promising approaches, but also to help identify approaches that will not yield desirable results.

We started by using online search engines such as Google, Google Scholar, DuckDuckGo, and later expanded to include scientific digital libraries such as IEEE Xplore and LUBsearch. By entering keywords and combinations of keywords such as "anomaly detection in time series" and "anomaly detection machine learning", we were able to find some relevant literature. When the search results obtained by using keywords were exhausted, we began reading the references of the papers we had found, which lead to the discovery of several relevant papers. The literature study was brought to an end when relative results started to decrease with new queries, but also due to the available time frame. By the end of our search effort we had found approximately 30 papers related to our case. These were studied in detail and we chose to include the eight most relevant papers in the related work section. Each of the eight papers was summarized, highlighting results, the implemented model, and domain settings similar to those found in our own particular case. We then proceeded with a selection process where chose which models to implement for this thesis. Models were ranked based on the related work similarity to our own case, the availability of relevant literature, and the inherent model complexity.

3.2 Empirical Research

The case study performed in this thesis does not share all of the characteristics of a case study conducted within the field of software engineering as described by Runeson and Höst, however there are more similarities than differences.

The experiments in this thesis form an empirical study, where the first step was to analyse the obtained industry data. A small subset of the data were chosen for further analysis and experimentation. Data collection in case studies, particularly in the field of software engineering as described by Runesson and Höst, is an essential phase. For this case study the data collection phase was eliminated as all data were readily provided. Data analysis has instead played a vital part. Three machine learning models were then implemented and trained, tested, and optimized on the chosen data. Data was analysed from both a quantitative and qualitative perspective as explained by Runesson and Höst. The statistical data presented in the results section form the quantitative aspect of our research, while the discussion and conclusion sections form the qualitative aspect of our case study.

Chapter 4
Related Work

In this section we present the papers that directly relate to this thesis and which form the basis for our own work. Papers are divided into three categories based on the type of model described in the work.

4.1 LSTM Prediction Models

Malhotra et al. [23] present their results for anomaly and fault detection in time series data using a stacked Long Short Term Memory predictive model. The traditional LSTM architecture is known for its ability to learn sequences with long term patterns as a consequence of its long term memory. By stacking recurrent hidden layers the authors show that the model gains the ability to learn higher level temporal features. The model is trained using four different data sets, and the results are compared to the performance of a traditional Recurrent Neural Network. The results show that the stacked LSTM performs equally well as the RNN model when the data set only contains short term temporal dependencies. The stacked LSTM model outperforms the RNN model significantly when the data set contains both long-term and short-term temporal dependencies. The method proposed by the authors requires training data consisting only of normal behavior, meaning the training data need to be free of anomalies and errors. This makes the model suitable in scenarios where normal behaviour occurs in abundance, but anomalous behavior is sparse. The proposed model has two fully connected hidden layers containing LSTM units, where the number of hidden units in each layer is optimized for each data set. The number of hidden units varies between 20 and 35, and the same number of units is used in both hidden layers. In our work we adopt the same approach where the number of hidden units varies, and the same number of units is used in all hidden layers. The paper is relevant to our own work with regards to evaluating the generic nature of models, as several data sets from different domains are used in the experiments. The data sets are however univariate, as opposed to our own multivariate time series.

Nanduri and Sherry [24] investigate how anomaly detection can be performed on multivariate time series data from simulated aircraft flight data recorders. The authors compare different implementations of Recurrent Neural networks to Multiple Kernel Based Detection (MKAD) models. The first RNN model proposed is a deep neural network consisting of one or more LSTM layers. The second architecture examined in the study is a deep neural network consisting of *Gated Recurrent Units*, which are special implementations of LSTM units. The authors argue that the RNN is superior to traditional methods in the sense that they overcome obstacles such as weak ability to detect anomalies in short term dependencies and in underlying features. For the RNN models, the authors tested several model configurations, where the number of hidden layers varied between one and two layers. The input layer consisted of 21 fully connected neurons, one for each feature in the data set. In the case of the LSTM models, the hidden layers were fully connected and contained LSTM units. The authors tested hidden layers with either 30 or 60 units, and in the case of two hidden layers both layers had the same number of units. Dropout layers were added after each hidden layer, with a constant dropout rate of 0.2. The authors utilize the Adam optimization algorithm with default parameters when training the network. The data set included a training set describing normal behaviour, and a test set containing 11 anomalies and 11 normal data points. The work relates to our own in the sense that different models and anomalies are incorporated in the experiments, and a multivariate time series is used for testing. The data set was however obtained by simulating flight data recorders, allowing the authors to manipulate the data set and introduce anomalies. The traditional methods for anomaly detection were able to detect 6 of 11 anomalies, while both the RNN models were able to detect 9 anomalies.

Chauhan and Vig [25] propose a recurrent deep neural network using LSTM cells for anomaly detection in Electrocardiography (heartbeat monitoring) time series data. There exists many types of arrhythmia, or irregularities in heartbeats, due to the complexity of the heart organ. These irregularities can be of both long-term and short-term character. The architecture suggested by the authors is a deep RNN with stacked LSTM units which allows the model to capture several time scales of the input resulting in a wider range of temporal features. The authors approach the problem by training their model on normal data, and expecting the model to perform poorly for anomalous input. Other approaches to the same problem context often require feature engineering and labeled training data, or are computationally expensive. The authors show that not only is the proposed architecture a viable candidate for anomaly detection in ECG time series, the model also performs well in identifying all four types of arrhythmia contained in the data set, indicating that the network achieves high precision in the general case of identifying different types of arrhythmia. The paper examines several anomalies individually, similar to our own approach. The data set used forms a univariate time series where all anomalous instances are labeled, unlike the multivariate time series used in our own work. The proposed network architecture is a stacked LSTM network with two fully connected hidden layers. Each hidden layer had a fixed number of 20 units. The authors use early stopping as a regularization technique and utilize a linear activation function for the output layer.

Filonov, Lavrentyev, and Vorontsov [26] adopt a LSTM-based neural network approach for fault monitoring and detection in industrial multivariate time series data. The authors created a Modelica [27] model for simulation of a gasoil plant, in which they introduced logical model faults. This allowed access to a data set with labelled anomalies for validation, while the model was trained using data that exhibits exclusively normal behaviour. The data set used is similar to our own data set as both form multivariate time series, a difference being that their data are simulated and anomalous instances are introduced by the authors. This allows for labeled anomalous instances, where normmal data can be fully separated from anomalous data. In our own work we do not have access to pre-labeled anomalous observations, instead we rely on domain expertise to identify anomalous instances. The model is trained by making predictions on training data, and treating the mean squared error between the predicted value and the actual value as a loss function. Inputs are classified as anomalous when the forecast error between the predicted value and the actual value exceeds the predefined threshold. The authors point out that the threshold level can be used as a tuning parameter for the values of precision, recall, and F1 score. In particular, this parameter can be tuned in order to regulate the rate of false positives in a given system. The authors show that their proposed model can perform on par or better than well known and established approaches in the problem domain. The model architecture consists of two hidden layers with LSTM units, however the authors do not disclose the number of units in each layer. The authors use dropout for regularization, where the dropout probability can assume the values of 0.5, 0.1, and 0.01. The authors make use of the RMSprop optimization algorithm when training the network, and deploy a linear activation function for the output layer.

4.2 Autoencoder Models

Malhotra et al. [19] propose a Long Short Term Memory based Encoder-Decoder scheme for anomaly detection. According to the authors, the model is particularly applicable in situations where external factors make the time series naturally unpredictable. Traditional mathematical models or prediction models fall short in these types of situations where stationarity in the time series is weak and making a prediction is difficult. The model takes a sequence of multiple variable readings as input, and attempts to reconstruct that same sequence as the output. The reconstruction error for each point in the time series can then be used to calculate an anomaly score. The likelihood that a point is displaying anomalous behavior increases with greater anomaly score. The LSTM based Encoder-Decoder is dependent on having access to training data embodying the normal behaviour of the system. The model was compared to the one proposed by Malhotra et al. [23] using the same data sets, and the authors show that the LSTM-based Encoder-Decoder performs better for sequences with more unpredictable data, while the LSTM prediction model performs better for data sets where sequences are more predictable. The setup is similar to our own where several models are evaluated and compared on the same data sets, although the work is divided across two papers.

Zhu and Laptev [28] introduce a deep learning network consisting of an LSTM Encoder-Decoder framework to capture the internal shape of the time series, and a vanilla deep neural network as a prediction model. The Encoder-Decoder framework acts as a feature extractor, capturing both normal and uncommon input. When the LSTM Encoder-Decoder has been trained, the state of the last LSTM cell in the Encoder is given as input to the prediction model. An uncertainty value is estimated for each prediction. The combination of a prediction value and an estimated uncertainty of that prediction results in more accurate anomaly detection. i.e. lower false anomaly alarms. The model proposed by the authors takes uncertainty into account while at the same time providing scalability, which other traditional approaches fail to do according to the authors. The authors show that their approach provides satisfactory prediction accuracy in addition to providing a higher anomaly detection accuracy. A multivariate time series is used, with a sliding window size of 28, similar to our own setup. The authors also compute an uncertainty value for each observation, something that we do not do in this thesis.

4.3 Convolutional Models

Wen and Keyes introduce [29] an approach for time series segmentation based on convolutional neural networks for anomaly detection. The architecture developed by the authors following the design of U-net was for a time series with length 1024 and C channels which exist between the encoding layers and decoding layers in order to prevent information loss along deep sequential layers by concatenating high-level features and low-level features. The model is encoded by five sections of convolution layers. The authors state in the paper that the snapshots were taken regularly on the latest batch of data form the streaming environment to run the model on the taken snapshots. They recommend that the frequency of taking snapshots should be high at least as the length of snapshot. Doing this according to the authors, allows the model to evaluate every time point at least once and then returns a probability of anomaly. The approach introduced by the authors was tested on a univariate task with sufficient data and 36 anomalies were detected of 39 known anomalies, a multivariate task with sufficient data and here only one anomaly was missed of 18 testing sequences, a univariate task with insufficient data with transfer learning and the model found 71.95 % of the anomalies, and on a multivariate task with insufficient data with transfer learning where the model found 64.10 [30] with Tensorflow as a backend.

Zhang et al. [31] introduce the Multi-Scale Convolutional Recurrent Encoder-Decoder (MSCRED) framework for anomaly detection and identification. The proposed framework not only aims to provide accurate anomaly detection, but also to evaluate the severity of each anomaly. The method utilizes signature matrices that are created from the time series, and that are resilient to noise according to the authors. The signature matrices are then fed to a convolutional encoder, and the idea is that a convolutional decoder will struggle to reconstruct the encoded matrices which demonstrate behaviour that has not been seen before by the model. Previous work in the field of anomaly detection do not simultaneously consider, according to the authors, temporal dependencies, resistance to noise, and the severity of the detected anomaly, contrary to MSCRED which demonstrates all of these characteristics. The framework is evaluated based on several metrics, namely precision, recall, and F1 score. The framework was tested on a synthetic data set and a real world power plant data set, and compared to other baseline methods on the same data sets. Several multivariate time series are used in the experiments, which compare the performance of different models across data sets, similar to our own study. A simple anomaly scoring system is used where the duration of the anomalous behavior is directly proportionate to the severity of the anomaly. We do not implement a scoring system in our work, we do however discuss the importance of the anomaly duration and how it affects operators. The authors show that the MSCRED framework is able to outperform all the baseline methods included in this particular study.

4.4 Literature Discussion

The related works listed in this section fall under the categories of unsupervised and semi-supervised machine learning techniques, meaning that unlabeled data is used or unlabeled data is used in conjunction with some labeled data. The absence of labeled data or sparsity of anomalous events for anomaly detection is a recurring issue in real industry settings, which is illustrated in [25, 29, 31, 26], leading to difficulty in implementation of supervised machine learning techniques [32]. We experience the same difficulty in this thesis as the data are provided unfiltered and unlabeled, with a very limited number of known anomalies. We will therefore need to adopt a model that does not require supervised training as creating such labels for normal behaviour would be tedious and time consuming work, and creating labels for anomalous data will be fruitless due to the sparsity of anomalous events.

When labeled data is not available, an alternative approach is to train the model on normal behaviour [23, 19, 24, 25, 26]. Using this approach, the model learns the normal state of a system, enabling operators to determine if the input to the network is anomalous by examining the output generated by the network. Seeing as we do not have access to labeled data sets, we will adopt this approach of training our models on normal behaviour. In industry settings the availability of normal data may be the opposite for that of anomalous data, where normal data is available in abundance [23] [32]. The challenge here is instead to differentiate and isolate normal behaviour in an adequate amount from the rest of the data set. This is made further difficult by the volume and complexity of the data, and domain expertise is required to extract normal sequences from the data set [29].

Chapter 5
Data

In this chapter we present how both test and train data was collected and prepared for our study. We describe the data sets and the operations made to manipulate the data.

5.1 Overview

The data sets used in this thesis are unlabeled, meaning that observations are not marked as anomalous or normal. To overcome this obstacle we will rely on domain experts and try to train our networks on normal data. The intuition is that networks will adapt well to the normal input data, however when anomalous data is presented as input the networks will produce a poor output. Domain experts at Sigma provided us with four processes (see 5.4) that exhibit abnormal behaviour. Domain experts then identified periods of abnormal behavior. This is a crucial step in our approach where we rely on an "oracle" to identify what constitutes anomalous behaviour, allowing us to label instances as anomalous in collaboration with domain experts. Using domain expertise, we can identify anomalous time steps and utilize this information to evaluate model performance. By training our models on normal behaviour, the assumption is that the models will produce a poor output when encountering abnormal input.

5.2 Data Collection

In this thesis we have used two industry data sets provided to us by Sigma. The data sets were provided as database backup files, allowing us to create and modify entities as we pleased using SQL. The data made available to us were *secondary data sets* [33], meaning data were collected by others for other purposes than this thesis. The obvious advantage of using a secondary data set is that no time or resources need to be spent to collect the data. A possible limitation of secondary data sets is that preprocessing of data may be more complex and time consuming since data may have been intended for other domains and purposes.

In the particular case of this thesis, there was no other option than to use secondary data sets. Due to legislation concerning data privacy and proprietary data rights, there was no way for us to gain access to any data other than the metadata provided by Sigma. The databases consisted exclusively of metadata describing the behaviour and state of various processes, no other types of data were made available to us at any point.

The data bases provided to us contained metadata for numerous processes. Depending on the granularity level of queries, approximately 1.6 million distinct rows could be obtained. Processes execute periodically and the metadata of each such execution is one observation, several such observations ordered in time form a time series. An example of a process could be a series of operations that extract data from some unstructured data source and then store them in a initial staging area of the data warehouse, where they await further processing.

5.3 Data Preprocessing

The backup files were restored in Microsoft SQL Server Management Studio (SSMS), an environment for working with SQL used by Sigma. Since all the data contained in the databases was not of interest, we created a database view in which we selected what data to include in our model. We then connected Python, our programming language of choice, with SSMS by using the pyodbc module for open database connectivity. We could now query our database directly from Python, enabling us to select rows from our view and perform data preprocessing inside Python.

5.3.1 Feature Engineering

The databases contained approximately 40 columns, where several columns could be eliminated from start. Columns such as "ProgramName" and "Description" are time invariant for individual processes and were removed immediately. Columns containing time stamps were transformed into new columns containing the year, month, day of the month, and weekday of the original time stamp.

The database did not contain the size of the data handled by a process. Through discussion with supervisors at both Sigma and LTH we decided that the record count information was representative of the amount of data processed. This record count information was available for each individual subprocess, but not for the master process in total. We created a new column containing the sum of inserted rows for all subprocesses belonging to a specific execution of a master process.

From the columns containing the start time and end time datetimes of master process, we created an additional column containing the execution time in seconds. This was done by calculating the time difference between the two datetimes. By looking at the data and through conversation with domain experts at Sigma, we determined that it was most appropriate to measure execution time in seconds. Few processes had execution times less than a second, and even fewer had execution times that lasted several minutes.

A number of columns consisted of *categorical* values, or categorical features [34] that can not be measured in a meaningful way. An example is the weekday column, containing the limited range of string values representing the 7 weekdays. The models can not understand the string "Monday", therefore the column must be transformed into something that is understandable to the model. One approach is to encode the categories on a ordinal scale, this however imposes an order on the categories. In the case of weekdays, there exists a natural order among the categories, Monday comes before Sunday. The circular nature of the week leads to Sunday coming before Monday, which can be problematic. This can be even more troublesome in our particular case where many processes are executed with a period of 24 hours. The same is true for the months and days of the month. Another approach is to transform categorical features into binary values using *one-hot encoding*, creating a new column for each category in the original feature [35]. We implement and test both approaches to see if any one of them yielded desirable results.

The data sets contained features that have different ranges. Examples of such features are execution time of processes, and the amount of data inserted. We normalized these features by applying the min-max normalization technique [24, 26], transforming all values to the range [0, 1].

5.3.2 Feature Selection

Deciding what features to use in our model was a complex task. We were not only considering one process, meaning that features had a varying importance depending on the process being observed. For each example process provided to us by Sigma, we plotted the time series of features with continuous numerical values, such as execution time in seconds and sum of inserted rows. By observing the plots side by side we could observe if there was any correlation between features, for example a growing trend in both execution time and number of rows inserted. We produced correlation matrices for our features utilizing the Pearson correlation coefficient [36]. We then visualized the matrices by generating *heatmaps* using the seaborn data visualization library for Python. A heatmap is a matrix containing correlation values between features, where the values 1 and -1 represent perfect positive and negative correlation respectively, and 0 represents absence of correlation. The results varied between processes, which was expected since process behaviour varies. Data was also analyzed manually for the four example processes where we tried to find seasonal behavior or long term dependencies. This was made difficult by the somewhat unstable nature of the processes examined. No seasonality could be found by manually examining the data. Upon initial testing we found that the addition of categorical variables did not lead to better results. In fact, for a majority of cases the results deteriorated after the addition of categorical variables during initial manual testing. This observation lead us to the decision to exclude the categorical features from our experiments and only use the execution time and number of inserted rows features.

5.3.3 Missing Data

The processes used in our experiments had a period of 24 hours. All processes had missing observation with a varying degree. The proportion of missing observations never surpassed 10% for any process, and the length of missing sequences rarely exceeded three consecutive observations. To obtain an equidistant time series, the missing rows were imputed using the imputation technique *last observation carried forward* (LOCF). This technique is specific to time series and imputes the missing values by copying the values of the last observation. The advantage of using LOCF is the simplicity of the method. For larger gaps of missing observation in the time series, LOCF will however produce periods of constant values in the time series.

5.4 Data Sets

The work carried out in this thesis is centered around the notion of using data portraying normal behaviour for anomaly detection. When analyzing the processes made available to us, we quickly realized that it would be challenging to perform anomaly detection for a large part of the process, as these were random by nature. An anomaly is something that deviates from that which is normal, but if nothing can be identified as normal behavior, then by definition there can be no anomalous behavior either. For other processes, normal behavior could be defined, although most often these processes were somewhat unstable. By looking at the plots in appendix A it is made clear that for this case study, data embodying fully stable behaviour for longer periods of time is a rare occurrence. Outlier observations are found in all data sets, sometimes in both the process execution time and in the sum of inserted rows. Together with domain experts, we searched for anomalous processes with little or no outliers when behaving normally, that still retained other desirable properties such as a low percentage of missing observations and having an adequate amount of data for training. Finding such processes was difficult. We then considered removing outliers from the normal section of our data sets, but after further consideration we decided that this was unfitting for two reasons. Firstly, by manipulating data in such a manner we could no longer claim that the resulting data set still embodied normal behavior. Manipulating data to such an extent would lead to models learning what we perceive to be normal behavior, rather than learning what actually constitutes normal behavior for the given process. Secondly, it would not be realistic to manipulate data in such a way in a production environment. In a production environment of possibly thousands of processes, continuously analyzing and manipulating data would be resource draining. We therefore decided to investigate if it was possible to utilize our initial approach with imperfect input data.

The four processes considered in this thesis are presented in appendix A. Two processes are drawn from each of the client data sets, where one process displays an abrupt behavior change and the other displays an increasing trend. This results in a total of four processes, where the processes can be paired based on the similarity in anomalous behavior. The inputs to our networks are the execution time and sum of inserted rows. These variables naturally correlate, and that correlation may also shift for anomalous instances. For each process, a plot of the execution time and sum of inserted rows over time is included, with anomalous regions highlighted in red.

Chapter 6
Model Selection

In this section we present the chosen anomaly detection models. We describe how the models were implemented and motivate parameter choices.

6.1 Overview

An objective for this thesis is to compare the generic performance and suitability of machine learning models for anomaly detection. To get an orientation of the results we are to expect, a simple baseline model was implemented. The values for the evaluation metrics delivered by the baseline are used to indicate the level of performace we can expect with simpler models. We then implemented more complex models and evaluated the performance on the same data sets. To limit model complexity, the models presented in this thesis will only predict one time step ahead. All models utilize the 30 most recent past observations when predicting or reconstructing the next time step. Through conversation with domain experts we learnt that some seasonality could occur on a monthly time scale within processes. The motivation for this parameter selection is to include information from approximately the entire past month when predicting the next time step. The neural networks in this thesis were developed using the Keras library, running on top of TensorFlow.

6.2 Baseline

For our baseline model we chose to implement a simple recurrent neural network for prediction. The recurrent neural network is a reappearing theme in the related work section when working with time series, leading us to choose a trivial RNN as our baseline model. The network has an input layer of dimension two, which is the same as the number of features. We use a single hidden layer with a hyperbolic tangent activation function, which is the default in the Keras implementation. A fully connected layer was chosen for the output, consisting of a single unit with a ReLU activation function. The loss function for the network was chosen as the mean squared error. The network utilized Adam as the optimization algorithm.

6.3 Stacked LSTM

The first model we propose is a stacked LSTM network, i.e. a deep neural network with several hidden layers consisting of LSTM cells. This approach was inspired by the works in section 4.1.

The proposed stacked model consists of an input layer with the number of units corresponding to the number of features. Malhotra et al., Chauhan and Vig, and Filanov et al. all use two hidden layers in their LSTM models. Nanduri and Sherry state that they vary the number of hidden layers between one and two. With the related work as our basis, we chose to deploy two hidden, fully connected hidden LSTM layers in our model. Each hidden layer uses the hyperbolic tangent activation function. This results in a total of four layers, comprised of the input and output layers, in addition to the two hidden layers. Dropout layers were added after each hidden layer, with a fixed dropout probability of 0.5 for each layer as proposed by Srivastava et al. [37]. The output layer consists of a single dense unit with a ReLU activation function. Mean squared error was used as the loss function.

6.4 Autoencoder

The second model we propose is an autoencoder network consisting of LSTM cells. This model is inspired by the works listed in 4.2.

The proposed model consists of an input layer with the same dimensionality as the number of features, two hidden layers with a dimensionality higher than the number of features, and a fully connected output layer with the same dimensionality as the the number of features. This architecture is similar to that of the stacked LSTM model, the autoencoder however consists of additional layers such as the RepeatVector and TimeDistributed layers [30]. The input and output layers must also be the same size since we are trying to reconstruct the input as output. The input layer and the first hidden layer can be seen as the encoder, while the second hidden layer and the output layer are viewed as the decoder. Our choice of having two hidden layers was ispired by the work of Malhotra et al., while Zhu and Laptev do not describe the number of layers or the number of neurons in each layer. The same number of neurons was applied to both hidden layers, as done by Malhotra et al. Dropout layers were applied after each hidden layer to simulate sparsity, where the number of nodes in the hidden layer N multiplied by the dropout probability p result in a number smaller than the number of features. This approach effectively simulated a sparse autoencoder with two input features, the execution time and sum of inserted rows for each process. The hidden layers utilized the hyperbolic tangent activation function while the output layer applied the ReLU activation function. Mean squared error was used to calculate both loss and reconstruction error. Adam was chosen as the optimization algorithm, as described in the work by Malhotra et al.

Chapter 7
Model Evaluation and Optimization

In this chapter we describe the evaluation and optimization approaches for our proposed models and chosen hyperparameters. We then discuss and present upper and lower bounds for the quality of our results.

7.1 Evaluation

We split each data set into a training set and a test set, where the training set makes up 70-80% of the total amount of data, depending on the process. The training set contains normal process behaviour and was used to train networks, while the unseen test set contains the anomalous region. All models were trained on the train set of each respective data set. We deploy *batch testing* during evaluation, meaning that the test set is divided into small batches. Models are evaluated on batches, then refitted with the observations of that batch. Model performance is then evaluated on the test set by calculating recall, precision, and F1-score as described by 2.3.2. Models are then ranked by F1-score.

7.1.1 Prediction models

The prediction models include the baseline model and the stacked LSTM model. Since there were no labels available, we needed to select one feature to use as the prediction variable. Through conversation with domain experts, we decided to use process execution time as the prediction variable. A functioning scheduling of processes depends on processes completing within their assigned time window, meaning that the execution time of one process can impact the entire schedule. The execution time feature was separated from the rest of the data set and used as a label for the sum of inserted rows, making the model predict process execution time. By using this approach we effectively transform the problem into a supervised learning task.

The prediction models are tested by using a *walk forward* approach, where the model predicts the execution time of the next time step. The actual value of that time step is then made available to the model before predicting the following time step. This is very similar to a real production setting where the model is updated as new data is made available. During testing, the model "walks" over the entire test set, making predictions and continuously being refitted with new data. The mean squared error of the predictions and actual values of execution time is returned and used as a loss function. An error vector was created by taking the mean squared error of the corresponding vector elements of the predictions and the actual values for execution time. If any such error value exceeded the threshold value τ, the time step was marked as anomalous. To determine τ the standard deviation σ of the error values was computed, then a series of 50 equidistant candidate values was created from the range $[\frac{\sigma}{5},\ 5 \cdot \sigma]$. Our intuition behind this approach is that the standard deviation will represent the natural model error. We then create the range of threshold values by dividing and multiplying the standard deviation by an arbitrary factor of five. We then search the the range of threshold values to find the threshold value that provides the best F1-score. The metrics recall, precision, and F1 score, described in 2.3.2 were computed for all candidate values of τ and then plotted as described by Filonov et al. [26], showing the different scores the model could achieve by varying τ. An example plot is illustrated by figure 8.2.

7.1.2 Autoencoder

The autoencoder does not need labels for training as it is an unsupervised machine learning model. Both the execution time and sum of inserted rows were used as input to the model. The model tries to reconstruct the input from a compressed feature representation. The reconstruction error is calculated as the mean squared error between the input and the reconstructed output. In the same manner as with the prediction models, the autoencoder traversed the test set, reconstructing input values then refitting the model based on the reconstruction error. The error vector in the case of the autoencoder consisted of the mean squared error between the input and the reconstructed output of each time step. The procedure for calculating evaluation metrics in the prediction models was then applied in the same way for the autoencoder.

7.2 Optimization

Model optimization was performed by utilizing a *grid search* where we test combinations of input parameters. To optimize model performance, several hyperparameters were tuned. All models used the Adam optimization algorithm with the default learning rate and each model was optimized for each data set. Some choices for parameter intervals are derived from the related work section, others are derived from our own manual testing.

This section details the optimization procedure for all three models. Hyperparameters were varied in the same way across models, with the exception of the number of neurons. For the prediction models the number of hidden units in each layer were chosen from the set {5, 10}. For the autoencoder the number of hidden units was varied with values {2, 4}. No information regarding the method for choosing the number of hidden units was provided by the related works, and there seems to be no scientific consensus on a correct approach [38].

Jeff Heaton [39] provides some rules of thumb for selecting the number of neurons in the hidden layers. The rules suggest a hidden layer size up to twice the size of the input layer. We expanded the range suggested by Heaton since we were also using dropout in our models. It was also necessary to have more neurons in the hidden layers than in the input layer for the autoencoder model to simulate sparsity. The number of epochs was varied with values {10, 15}. Batch size was varied with values from the set {2, 4, 6}. The number of online epochs was varied with the values {2, 4, 6} and the online batch size was varied with values {2, 4}. All ranges for number of epochs and batch size were chosen by manual testing, where we saw good performance in these ranges. We favoured lower values for the ranges of the number of epochs as this particular hyperparameter has a significant impact on the training time.

7.3 Result Evaluation

This thesis studies data sets specific to this case study. There have been no other tests performed on these data sets to be used for comparison, making it difficult to assess the quality of our results. Although no experiments, apart from our own, have been performed on these data sets, it is still valuable to compare our results to the results of the related work. Table 7.1 shows the ranges for the best evaluation metrics obtained for different models, configurations, and data sets in the related works, for those cases where meaningful comparisons can be made to our own results. We can see that four out of five ranges of F1-score have an upper bound above 0.8, and precision ranges with almost perfect upper bounds. Recall value ranges have a lower bound below 0.10 in two out of five cases, while upper bounds are above 0.8 in two of five cases. It is important to note that the best range values presented in table 7.1 are not necessarily obtained simultaneously. The related works represent state of the art results, and therefore give a reasonable indication to what results we should strive to achieve.

Table 7.1: Ranges of top evaluation metrics from related works across models, data sets, and configurations.

Related Work	Recall	Precision	F1-score
Malhotra et al. [23]	0.03 - 0.19	0.71 - 0.98	0.69 - 0.90
Nanduri & Sherry [24]	0.36 - 0.82	1.00	0.62 - 0.90
Chauhan & Vig [25]	0.46	0.98	0.63
Filonov et al. [26]	0.35 - 0.92	0.45 - 0.98	0.39 - 0.87
Malhotra et al. [19]	0.01 - 0.08	0.83 - 1.00	0.65 - 0.83

To define what constitutes a poor result, we implemented a random model that arbitrarily classifies observation as normal or anomalous. The mean values of evaluation metrics for 100 executions are shown in table 7.2. The number of executions was chosen arbitrarily to obtain fair mean values for the different metrics. These mean values hardly change when the number of executions is varied. The results from the random model show that recall values are approximately 0.5, which is to be expected. Precision values for the random model are dependent on the ratio between the number of normal and anomalous instances. For process A.1 the number of normal instances is far greater than the anomalous instances, leading to low precision values and therefore also a low F1-score. For process B.2 the situation is reversed and the number of anomalous instances outweigh the normal observations, resulting in a high precision value.

Table 7.2: Mean value of recall, precision, and F1-score for the random model when executed 100 times.

Process	Recall	Precision	F1-score
1	0.52	0.04	0.07
2	0.50	0.13	0.20
3	0.50	0.32	0.39
4	0.50	0.68	0.57

Chapter 8
Results

This chapter begins with a description of observations that were classified as anomalous. We then present the results obtained from performing our experiments. The section related to our own results is divided on a process basis, where results related to a specific process are grouped together. For each model, the highest and lowest F1-scores are presented along with their configuration.

8.1 Anomaly selection

In this section we describe which observations that we marked as anomalous, and the reasoning behind our decisions. This procedure was done manually in collaboration with domain experts.

The execution time and sum of inserted rows for process A.1 are shown in figure A.1 and figure A.2, respectively. process A.1 displays an abrupt behavior change, and the first 10 observations in the anomalous region were marked as anomalous. As the model is able to learn new behavior, it would not be meaningful to mark all observations as anomalous, instead we focus on detecting the initial change correctly. Using the 10 initial observations yielded acceptable performance. This number was however chosen in an arbitrary fashion without optimization. Process A.1 was also used to demonstrate our models ability to adapt to new normal behavior. A possible scenario is that a software update changes process behavior and therefore also changes what constitutes normal behavior. In such situations it is desirable that networks are able to model new normal behavior and continue to classify anomalies correctly. In addition to the first 10 observations in the anomalous region, the observations corresponding to the four outlier spikes in execution time visible in figure A.1 were also marked as anomalous due to their deviating behavior from the relative context. Figure 8.1 shows the actual and predicted values for the execution time of process A.1. The x-axis shows discrete time steps and the y-axis displays the normalized execution time measurements. The abrupt behavior change causes large predictions errors, which stabilize over time.

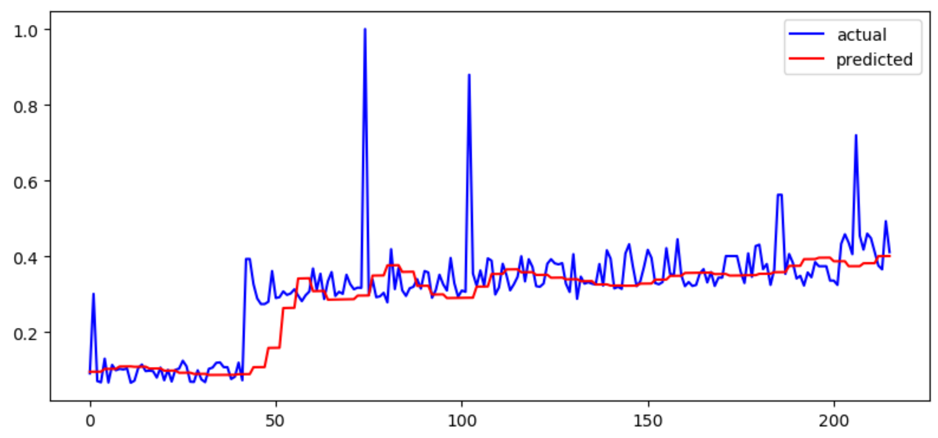

Figure 8.1: Actual and predicted values for the execution time of process A.1 over time for the best configuration of the stacked LSTM.

Process A.2 displays similar behavior of an abrupt change as process A.1, the large behavior change in process A.2 is however preceded by an increasing trend in execution time, as shown in figure A.3 and figure A.4. The three largest outlier peaks in execution time within the anomalous region preceding the abrupt change, and all the instances of the abrupt behavior change itself, were marked as anomalous. When allowing the model to learn more rapidly, for example by increasing the number of online epochs and decreasing the online batch size, models are able to follow the increasing trend relatively well. This is partly due to the fact that the increasing trend in execution time occurs simultaneously as the sum of inserted rows increases drastically, and then continues to increase with a subtle trend. We therefore chose to mark the three largest outliers preceding the abrupt change as anomalous, since these observations strongly deviated from the other observations in the same context. One could argue that the entire increasing trend of execution time should have been classified as anomalous, we chose however not to mark the entire trend as anomalous since we are testing model ability to detect abrupt behavior changes.

Process B.1 and B.2 both display an increasing trend, in both the normal and anomalous regions. The increasing trend is present in both the execution time and the sum of inserted rows. Process B.1 is described by figure A.5 and A.6, while process B.2 is described by figure A.7 and figure A.8. For both processes the start of the anomalous region marks a change point, after which future behavior past the anomalous region becomes highly unstable. process B.2 has more unstable normal behavior where the trend fluctuates over time, while the trend of process B.1 is more stable across time. Since both anomalous regions have a strictly increasing trend, all instances within the anomalous regions were classified as anomalous.

8.2 Abrupt Change Anomalies

In this section we present the results from running our test on the two processes that display an abrupt change in behavior. The processes originate from different data sets and correspond to the plots displayed in A.1. The values inside the brackets of the column parameter set represent number of nodes, number of epochs, batch size, number of online epochs, and online batch size in that order.

8.2.1 Process A.1

Table 8.1 shows the results from our tests on process A.1, described by figure A.1 and figure A.2 in A.1.

Table 8.1: F1-score and parameter set for top and bottom ranked configurations for each model.

	rank	F1-score	parameter set
Baseline	highest	0.82	{5, 10, 4, 4, 4}
	lowest	0.79	{5, 10, 4, 6, 4}
LSTM Prediction	highest	0.83	{10, 15, 4, 4, 4}
	lowest	0.78	{10, 15, 6, 6, 4}
Autoencoder	highest	0.81	{4, 10, 2, 4, 2}
	lowest	0.71	{4, 10, 6, 2, 4}

Table 8.1 shows that the stacked LSTM model performs best for process A.1, followed by the baseline model and the autoencoder. Both the stacked LSTM and baseline models have the same top parameter set. The autoencoder has the largest difference between the highest and lowest F1-score. All models achieve a top F1-score above 0.8.

8.2.2 Process A.2

Table 8.2 shows the results from our tests on process A.2, described by figure A.3 and figure A.4 in appendix A.1.

Table 8.2 shows the stacked LSTM model performs best for process A.2, followed by the baseline model. The best parameter set varies across models. The autoencoder has the lowest F1-score, while the baseline model has the largest difference between the highest and lowest F1-scores. All models achieve a top F1-score above 0.8.

8.3 Trend Anomalies

In this section we present the results from running our test on the two processes that display an anomalous trend. The processes originate from different data sets and correspond to the plots displayed in A.2.

Table 8.2: F1-score and parameter set for top and bottom ranked configurations for each model

	rank	F1-score	parameter set
Baseline	highest	0.94	{10, 10, 4, 2, 4}
	lowest	0.69	{10, 10, 2, 6, 2}
LSTM Prediction	highest	0.97	{5, 10, 2, 2, 2}
	lowest	0.82	{5, 15, 2, 6, 2}
Autoencoder	highest	0.87	{4, 10, 6, 6, 2}
	lowest	0.72	{4, 10, 4, 4, 4}

8.3.1 Process B.1

Table 8.3 shows the results from our tests on process B.1, described by figure A.5 and figure A.6.

Table 8.3: F1-score and parameter set for top and bottom ranked configurations for each model

	rank	F1-score	parameter set
Baseline	highest	0.76	{5, 10, 2, 2, 4}
	lowest	0.58	{10, 10, 2, 6, 2}
LSTM Prediction	highest	0.78	{5, 10, 2, 2, 4}
	lowest	0.49	{5, 10, 6, 6, 2}
Autoencoder	highest	0.82	{4, 15, 2, 2, 4}
	lowest	0.69	{2, 10, 6, 6, 4}

Table 8.3 shows that the autoencoder performs best for process B.1, followed by the stacked LSTM model. The best parameter set for the stacked LSTM model and the baseline model are identical, and very similar to the best parameter set for the autoencoder. The baseline model has the lowest F1-score, while the stacked LSTM has the largest difference between the highest and the lowest F1-score.

8.3.2 Process B.2

Table 8.4 shows the results from our tests on process B.2, described by figure A.7 and figure A.8.

Table 8.4: F1-score and parameter set for top and bottom ranked configurations for each model

	rank	F1-score	parameter set
Baseline	highest	0.84	{5, 10, 2, 2, 4}
	lowest	0.77	{5, 10, 6, 6, 4}
LSTM Prediction	highest	0.92	{5, 10, 4, 2, 4}
	lowest	0.78	{10, 10, 4, 4, 2}
Autoencoder	highest	0.98	{4, 10, 2, 4, 2}
	lowest	0.86	{4, 15, 6, 6, 2}

Table 8.4 shows that the autoencoder performs best for process B.2, followed by the stacked LSTM model. The best parameter set for the autoencoder model and the stacked LSTM model are similar. The baseline model has the lowest F1-score, while the stacked LSTM has the largest difference between the lowest and highest F1-score.

8.4 Threshold variation

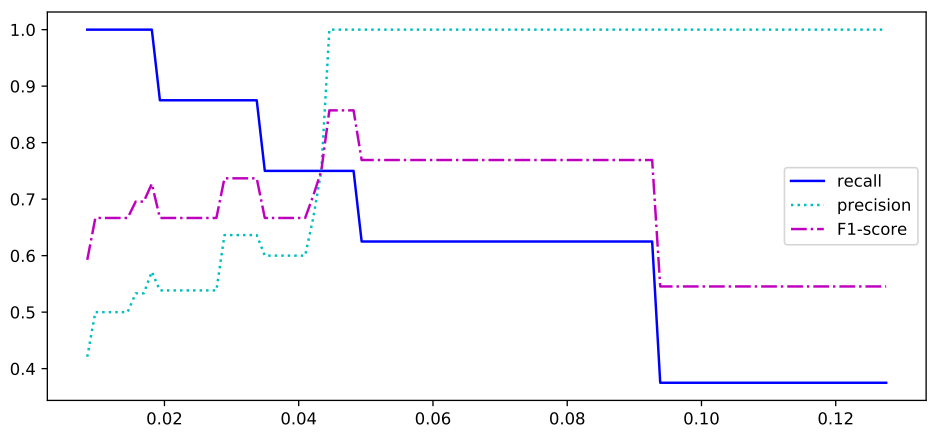

Figure 8.2: Evaluation metrics for process A.1 and the best stacked LSTM cofiguration when varying the threshold value. The x-axis shows the threshold value.

Depending on the domain in which anomaly detection is performed, operators may decide to favor either one of the parameters recall, precision, or F1-score. As discussed in 2.3.2, some situations might demand perfect recall or precision scores. Operators can tune models by varying the threshold value with which anomalies are flagged, an example of this is shown in figure 8.2. Generally, as the threshold becomes lower, recall rises while precision values drop. Conversely, as the threshold value increases the precision value rises while the number of true positives drops, resulting in a lower recall score. In the particular case of process A.1, the metrics remain fairly stable as the threshold value increases because of the large change in behavior, where the 10 first instances make up a majority of the anomalies in the set. If the threshold value continued to increase, it would reach a point where recall would start to decrease as fewer of these anomalies would be correctly flagged. Tuning models for higher recall will often lead to more false anomaly alarms that operators will need to address, while higher precision scores tend to miss more anomalies.

Chapter 9

Discussion

In this chapter we analyse and discuss the results obtained from our experiments, and the approach used to obtain the results.

9.1 Models

From our conducted tests we see that the baseline model never outperforms both the stacked LSTM model and the autoencoder model simultaneously. This indicates that the added complexity of the stacked LSTM and autoencoder models carries a performance gain. This performance gain is not without drawbacks, as the added complexity make the models more error prone and more time consuming to train and optimize. This merits further analysis of the advantages and disadvantages of adopting more complex models, as compared to using simpler networks for anomaly detection.

For the first two processes which display an abrupt behavior change, the stacked LSTM model outperforms the baseline model by 0.01-0.03 in F1-score. The autoencoder ranks the lowest for the same two processes. The performance gain offered by the LSTM compared to the baseline model for the first two processes is almost negligible. The confidence intervals presented in appendix B hint that the performance loss of the baseline model could be explained by the models inherent instability. We computed the 95% confidence intervals by training models 10 and 50 times on the same data set and with the same configuration. The best F1-score obtained from each new training cycle was then used to compute the confidence interval for mean F1-score. Looking at the confidence intervals, we see that the baseline model has broader intervals, indicating a more volatile nature compared to the stacked LSTM. It is unclear why the stacked LSTM achieves better performance and stability. A possible explanation could be the added depth of the stacked LSTM model. Another viable explanation would be the enhanced memory of the LSTM cells. Although the differences in performance between the baseline and stacked LSTM models may be moderate, the added stability of the stacked LSTM could make it a more attractive candidate. For settings where long term dependencies play a vital role in correctly modeling process behavior, the LSTM model would be the better alternative. The dramatic increase in training time and memory requirements could be an acceptable trade off for improved performance and stability.

The anomalies displayed by the first two processes are not subtle. They instead display large and rapid changes in process behavior. For such anomalies, the information from previous time steps is less important as models will struggle to follow this rapid change initially, although information from past time steps can not be dismissed. Process A.2 is an interesting example of anomalous behavior, where the increase in execution time and sum of inserted rows strongly correlate. Models which are not able to remember past time steps may only learn the relationship between execution time and the sum of inserted rows, predicting high execution time when the sum of inserted rows increases, effectively leaving the large behavior change undetected. Our results show that all three models are able to detect such anomalies, meaning that previous behavior, in addition to the relationship between execution time and sum of inserted rows, affects the prediction or reconstruction value. This indicates that recurrent neural networks are appropriate structures for time series data, which is also concluded in the related work.

In a production environment, an abrupt change does not indicate erroneous behavior by default. process A.1 was used to simulate a situation where the abrupt behavior change is not to be viewed as anomalous. Figure 8.1 illustrates the actual and predicted values of execution time when running the stacked LSTM model. The model struggles to learn the new behavior initially, and as more observations with the same high values of execution time are made available, the model incrementally increases the vertical step size to reach the new normal behavior. The network is then able to model the time series with a degree of accuracy that enables continued anomaly detection. Model inertia becomes an important factor for anomalies displaying abrupt behavior changes, as we want models to be able to learn new behavior while at the same time being resilient to rapid change. If the selected anomaly detection model has high inertia, it may take a considerable amount of time before the new behavior is modeled, the extreme case being when the model is unable to learn at all. Models with high inertia will learn new behavior slowly, meaning that all observations following an abrupt change will be classified as anomalous for a prolonged period of time. This is troublesome in situations when non-anomalous behavior changes occur, as simulated with process A.1, since operators will need to deal with constant error reports. Contrarily, low inertia may lead to over accelerated modeling and large oscillations in the prediction or reconstruction values, effectively creating a model too unstable to be used for anomaly detection. For the first two processes, the autoencoder seems to struggle more with the abrupt changes compared to the prediction models. This indicates that the autoencoder has moderately higher inertia than the prediction models, resulting in a slower ascend to new behavior. Model inertia can be manipulated by operators through varying the number of epochs or batch size during training. This provides operators with the possibility to select any model and tweak hyperparameters to achieve more desirable inertia characteristics.

For the two processes displaying an anomalous trend we see that the autoencoder model achieves the highest F1-score, followed by the stacked LSTM model. The baseline model achieves the worst F1-scores of the three models, for process B.1 it is however only marginally worse than the stacked LSTM model. The results of process B.1 show the largest average differences between the best and worst F1-score across processes. We believe this is caused by the fact that the rate with which models adapt to new behavior is highly responsive to the parameter optimization. The trend of process B.1 is very subtle, making it easy for models to adopt the new behavior quickly and therefore obtain a lower F1-score. The autoencoder model is not as receptive to the increasing trend in both execution time and sum of inserted rows as the two predictions models, giving the autoencoder a smoother error curve. This ties in with the behavior observed for the processes that exhibit an abrupt change, where the autoencoder adopts new behavior with a lesser rate than the two prediction models. This suggests that the autoencoder may be an appropriate model for trend detection. Current literature does not support the claim that the autoencoder structure is particularly good at detecting trends, which indicates that our observation may be limited to this specific domain.

Appendix B shows the mean values and 95% confidence intervals for 10 and 50 executions. By observing the confidence intervals across processes, we see that the autoencoder has the most narrow confidence interval. This indicates that the autoencoder has more stable performance as the results of executions vary less than with the prediction models. Our results show that both the stacked LSTM and the autoencoder are viable model options for anomaly detection in time series data. From our experiments and experiences, we find that the stacked LSTM is the most flexible model, allowing it to score well for both types of anomalies tested in this thesis while at the same time displaying relatively stable behavior. With more elaborated parameter tweaking, it is our belief that useful performance can be achieved for most processes where normal behavior can be defined, using the stacked LSTM model. Additional and more elaborate testing would however be needed to legitimize this claim. With expanded computing resources more intricate and extensive parameter tweaking could be performed to obtain better results. This would also allow for the use of more processes in our experiments, and thus provide more data for evaluating the generic nature of models.

The results show that some models perform better for certain types of anomalies, where the stacked LSTM shows the best performance for abrupt change anomalies, while the autoencoder model achieves the best performance for trend anomalies. The added complexity of the stacked LSTM and autoencoder models provide better performance. It is however important to note that the baseline model only performs considerably worse than both the stacked LSTM and autoencoder for process B.2. It is our experience from this thesis is that the added model complexity carries with it a considerably longer training time, as well as a higher strain on working memory. Depending on the context, it may be beneficial to trade the limited performance gain of more complex models for the considerably lower cost in computer resources offered by the baseline model. We do not however present any data pertaining to training times or working memory strain in this thesis, meaning no such conclusions can be drawn from our results.

A shortcoming in our work is that we have not been able to determine parameter sets and anomaly threshold values that will perform well for different types of anomalies. Ideally we would like a network that is able to model different kinds of normal behavior, while at the same time achieving high evaluation metrics when faced with different types of anomalies. Our work has only taken individual anomalies into consideration when determining top configurations and calculating evaluation metrics. A future extension would naturally be to investigate if useful performance can be achieved for different processes and anomalies with a general model configuration.

9.2 Approach

As we describe in section 5.4, our fundamental concept of using data embodying normal behavior for anomaly detection has proven difficult in this particular case. Many processes are completely random, while others have moderately polluted data. To our suprise, the outliers and gentle oscillations in regions classified as normal did not hinder us from obtaining useful performance, as show by our experiments. Similar studies, that are known to us, manage to separate normal and anomalous behavior during training. This allows networks to train on data that exclusively embodies normal behavior. The characteristic of being resilient to imperfect input data is highly desirable. This is particularly true in settings similar to that of this case study, seemingly suffering from a deficit of pure and stable normal data.

Deciding on how to mark observations as anomalous was non trivial since fluctuations and outliers were present within anomalous regions as well. For the processes used in this thesis, it has been challenging to clearly differentiate between normal and anomalous behavior. The length of the anomalous region and the number of observations classified as anomalous have a considerable impact on the evaluation metrics, especially when anomalous regions tend to be unstable. We found no literature that could support us in deciding how much of anomalous behavior that should be used for evaluation, which is why we have been forced to rely heavily on input from domain experts. In this thesis we have focused on the initial period of anomalous behavior, seeing as early detection is more useful since it prevents prolonged and possibly more severe anomalies. Anomalous regions can not be too long since the goal is not to model anomalous behavior. At the same time the region can not be too short since we want to assure that the model definitively identifies the anomaly by flagging several instances. A prolonged error signal draws more operator attention to more severe anomalies stretching across several time steps. A possible, and arguably necessary future extension, could be to assign an anomaly score to each observation, incorporating the importance of a prolonged anomaly signal, where the anomaly score increases with the number of past observations that continuously signal anomalous behavior. In addition, the magnitude of the prediction or reconstruction error should also influence the anomaly score. A drawback of this extension would be the added complexity of defining the rules for calculating the anomaly score.

Our tests show promising results, indicating that the concept of using data embodying normal behavior for anomaly detection can yield useful results, even when perfectly normal data is not available in abundance. However, one of the main challenges in this thesis has been the ill-defined normal process behavior. Our tests do not consider to what extent purity of training data impacts model performance, although it would be reasonable to assume that performance deteriorates with declining data purity. For the processes where some degree of normal behavior can be established, our work shows that useful results can be obtained. However, the approach of using normal data for anomaly detection would benefit if it was deployed in a context where normal behavior could be more clearly defined.

9.3 Network Optimization

Both time and computer resources were in a deficit during this case study, which required us to carefully ration our available resources. This meant reducing the number of configurations to test, and thereby narrowing the optimization search space. Possible hyperparameters such as lookback, learning rate, kernel initializers, dropout probability, and number of hidden layers were excluded from the grid search. It is however interesting to note that even with this meagre parameter optimization, most tests achieve a F1-score above 80%. It is possible that better results could have been obtained if some of the hyperparameters were substituted for one or more of the excluded parameters. It is also highly likely that better results would have been obtained if more parameters were included in the grid search, provided that project resources allowed for it.

9.4 Validity

On possible threat to the validity of our results and subsequent claims is that only four processes were considered. Incorporating more processes in our tests would add more validity to our findings. However, the limited project time frame did not allow us to expand our experiments with additinal processes. The fact that models seem to rank similarly for comparable anomalies originating from different client data sets adds validity to our claim that certain models perform better for specific anomalies. The F1-scores across data sets and models have similar ranges as those found in the related work, supporting the claim that all models can achieve useful results.

The low number of 10 executions used to obtain mean values for the highest and lowest F1-scores could also pose a threat to the validity of results. Ideally the number of executions would have been increased considerably, however project resources did not allow it. Repeating the training phase and test phase 10 times on the same data could take several hours for some tests. Seeing as we tested 72 parameter sets for three models across four data sets, time restrictions did not allow for expansion of the number of executions. As a remedy to increase the validity of our results, the mean value and the 95% confidence interval was calculated for the best parameter set, for each process and model. The tables in appendix B show the mean value and 95% confidence interval for 10 and 50 executions respectively. This shows that the highest difference in mean top F1-score is approximately 0.01 when increasing the number of executions by a factor of five. We also see that the confidence intervals change very slightly when the number of executions is increased. This indicates that increasing the number of executions would not have produced significantly different results for the mean F1-scores and confidence intervals.

Chapter 10

Conclusions

10.1 Summary

In this thesis we examine the possibilities of performing anomaly detection for time series data. We examine three different models on four data sets, and then evaluate and compare the results. To overcome the obstacle of unlabeled data sets, we utilize *a strategy where models are trained on normal behavior* and then assume that models will perform poorly when encountering anomalous input. We test two models consisting of LSTM cells, a stacked prediction net and an autoencoder network, and implement an additional simple recurrent neural network as a baseline model.

We show that *recurrent neural networks are appropriate structures for anomaly detection in time series data* by obtaining results in similar ranges as those found in related work. Our results show that *at least one of our proposed models always outperforms the baseline model*, indicating that the more complex models are better suited for the problem. Our test *do not show that any one model performs best in the general case*. The indication from our results is that the stacked LSTM model could achieve useful performance in the general case, but that further testing is required to validate this claim. Our results do however show that *the stacked LSTM model performs best for abrupt change anomalies, while the autoencoder model achieves the best performance for trend anomalies.*

We discuss the difficulty of defining normal process behavior, and how the lack of pure normal behavior might impact results. We show that it is *possible to obtain useful results even when training data defined as normal is contaminated with outliers and unstable process behavior*. We consider the implications of performing anomaly detection in a case where normal behavior can be difficult to separate from anomalous behavior, and conclude that *more well-defined normal and anomalous behavior could provide better results.*

10.2 Future Research

A fundamental part of this case study was the selection of suitable machine learning models. The literature study resulted in three candidate models, two of which were selected for this thesis. The third one, the convolutional approach described in 4.3, was dismissed because of the inherent model complexity due to the number of layers and long training times. However, the papers presented in 4.3 show promising results. It would therefore be interesting to conduct *a study that evaluates the performance of a convolutional approach* in a similar setting and compare the convolutional approach to the models in this thesis.

The work conducted in this case study shows promising results for the selected data sets. However, the results are restricted to this specific case where only two features were used. This limits the underlying complexity in anomalies, as an increase in the number of features also increases the way in which these can interplay. This also impacts the subtleness of anomalies, as having only two features makes it fairly easy to detect changes in the underlying behavior. Increasing the number of features can make the task of identifying anomalous behavior tremendously difficult. Our results show that certain models may perform better for specific anomaly types, which is not discussed in the related works. Our experience from this thesis is also that the stacked LSTM prediction network is the most flexible of our models, and best suited to achieve desirable performance in the general case. It would be interesting to conduct *a study that explores our approach in a more general setting*, for several data sets across multiple domains, to assess if our findings are valid outside of this case.

In this thesis we do not consider the errors of neighboring observations when deciding if the current observation should be flagged as anomalous. We only examine if the current error between the actual and predicted or reconstructed value exceeds the anomaly threshold value. Considering the amount of outliers and unstable behavior in the examined processes, only considering the current observation when flagging anomalies may be suboptimal. A more appropriate approach would be to consider the length of consecutive observations that exceed the anomaly threshold and their corresponding error magnitudes simultaneously. It would therefore be beneficial if *an anomaly scoring system could be developed* that assigns each observation with an anomly score, and that defines what scores that should be considered anomalous.

References

[1] Raghavendra Chalapathy and Sanjay Chawla. Deep learning for anomaly detection: A survey. *CoRR*, abs/1901.03407, 2019.

[2] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3):15:1–15:58, July 2009.

[3] Samaneh Aminikhanghahi and Diane Cook. A survey of methods for time series change point detection. *Knowledge and Information Systems*, 51, 09 2016.

[4] Shancang Li, Li Xu, and Shanshan Zhao. 5g internet of things: A survey. *Journal of Industrial Information Integration*, 10:1–9, 06 2018.

[5] Peter J. Brockwell and Richard A. Davis. *Introduction to Time Series and Forecasting*. Springer International Publishing, 2016.

[6] Jan G. De Gooijer and Rob Hyndman. 25 years of time series forecasting. *International Journal of Forecasting*, 22:443–473, 02 2006.

[7] Charu C. Aggarwal. *Neural Networks and Deep Learning: A Textbook*. Springer International Publishing, Cham, 2018.

[8] Li Deng and Dong Yu. Deep learning: Methods and applications. *Foundations and Trends in Signal Processing*, 7, 01 2013.

[9] Michael R. Anderson, Dolan Antenucci, Victor Bittorf, Matthew Burgess, Michael J. Cafarella, Arun Kumar, Feng Niu, Yongjoo Park, Christopher Ré, and Ce Zhang. Brainwash: A data system for feature engineering. In *CIDR*, 2013.

[10] Wlodzislaw Duch and Norbert Jankowski. Survey of neural transfer functions. *Neural Computing Surveys*, 2:163–213, 1999.

[11] Sebastian Ruder. An overview of gradient descent optimization algorithms. *ArXiv*, abs/1609.04747, 2016.

[12] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[13] Razvan Pascanu, Çaglar Gülçehre, Kyunghyun Cho, and Yoshua Bengio. How to construct deep recurrent neural networks. *CoRR*, abs/1312.6026, 2013.

[14] Zachary Chase Lipton, David C. Kale, and Randall C. Wetzel. Phenotyping of clinical time series with LSTM recurrent neural networks. *CoRR*, abs/1510.07641, 2015.

[15] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 06(02):107–116, 1998.

[16] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. Understanding the exploding gradient problem. *ArXiv*, abs/1211.5063, 2012.

[17] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[18] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *ArXiv*, abs/1406.1078, 2014.

[19] Pankaj Malhotra, Anusha Ramakrishnan, Gaurangi Anand, Lovekesh Vig, Puneet Agarwal, and Gautam Shroff. Lstm-based encoder-decoder for multi-sensor anomaly detection. *ArXiv*, abs/1607.00148, 2016.

[20] Nianyin Zeng, Hong Zhang, Baoye Song, Weibo Liu, Yurong Li, and Abdullah M. Dobaie. Facial expression recognition via learning deep sparse autoencoders. *Neurocomputing*, 273(C):643–649, 2018.

[21] Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Softw. Engg.*, 14(2):131–164, 2009.

[22] Justus Randolph. A guide to writing the dissertation literature review. *Practical Assessment, Research, and Evaluation*, 14, 2009.

[23] Pankaj Malhotra, Lovekesh Vig, Gautam Shroff, and Puneet Agarwal. Long short term memory networks for anomaly detection in time series. In *ESANN*, 2015.

[24] Anvardh Nanduri and Lance Sherry. Anomaly detection in aircraft data using recurrent neural networks (rnn). *2016 Integrated Communications Navigation and Surveillance (ICNS)*, pages 5C2–1–5C2–8, 2016.

[25] Sucheta Chauhan and Lovekesh Vig. Anomaly detection in ecg time signals via deep long short-term memory networks. In *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 1–7, Oct 2015.

[26] Pavel Filonov, Andrey Lavrentyev, and Artem Vorontsov. Multivariate industrial time series with cyber-attack simulation: Fault detection using an lstm-based predictive data model. *CoRR*, abs/1612.06676, 2016.

[27] Sven Erik Mattsson, Hilding Elmqvist, and Martin Otter. Physical system modeling with modelica. *Control Engineering Practice*, 6(4):501 – 510, 1998.

[28] Lingxue Zhu and Nikolay Laptev. Deep and confident prediction for time series at uber. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 103–110, Nov 2017.

[29] Tailai Wen and Roy Keyes. Time series anomaly detection using convolutional neural networks and transfer learning. *CoRR*, abs/1905.13628, 2019.

[30] François et al. Chollet. Keras. `https://keras.io`, 2015.

[31] Chuxu Zhang, Dongjin Song, Yuncong Chen, Xinyang Feng, Cristian Lumezanu, Wei Cheng, Jingchao Ni, Bo Zong, Haifeng Chen, and Nitesh V. Chawla. A deep neural network for unsupervised anomaly detection and diagnosis in multivariate time series data. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:1409–1416, Jul 2019.

[32] Nico Görnitz, Marius Kloft, Konrad Rieck, and Ulf Brefeld. Toward supervised anomaly detection. *J. Artif. Int. Res.*, 46(1):235–262, January 2013.

[33] Melissa Johnston. Secondary data analysis: A method of which the time has come. *Qualitative and Quantitative Methods in Libraries*, 3(3):619–626, 2017.

[34] Peter Romov and Evgeny Sokolov. Recsys challenge 2015: Ensemble learning with categorical features. In *Proceedings of the 2015 International ACM Recommender Systems Challenge*, RecSys '15 Challenge, New York, NY, USA, 2015. Association for Computing Machinery.

[35] Kedar Potdar, Taher S. Pardawala, and Chinmay D. Pai. A comparative study of categorical variable encoding techniques for neural network classifiers. *International Journal of Computer Applications*, 175(4):7–9, October 2017.

[36] Israel Cohen, Yiteng Huang, Jingdong Chen, and Jacob Benesty. *Noise Reduction in Speech Processing*. Springer Berlin Heidelberg, 2009.

[37] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[38] Sheela K. Gnana and Subramaniam N. Deepa. Review on methods to fix number of hidden neurons in neural networks. mathematical problems in engineering. *Hindawi Publish Corp*, 2013:1–11, 01 2013.

[39] Jeff Heaton. *Introduction to Neural Networks with Java, 2nd Edition*. Heaton Research, Incorporated, oct 2008.

Appendices

Appendix A

A.1 Abrupt behavior change

In this appendix we show plots of the execution time and sum of inserted rows over time for each process. Anomalous regions are highlighted in red.

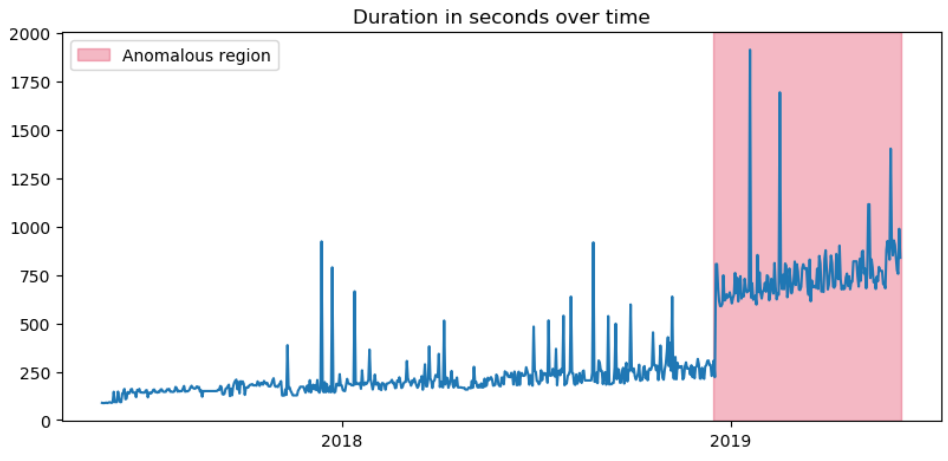

Figure A.1: Execution time of process A.1 across time, displaying an abrupt change in behavior.

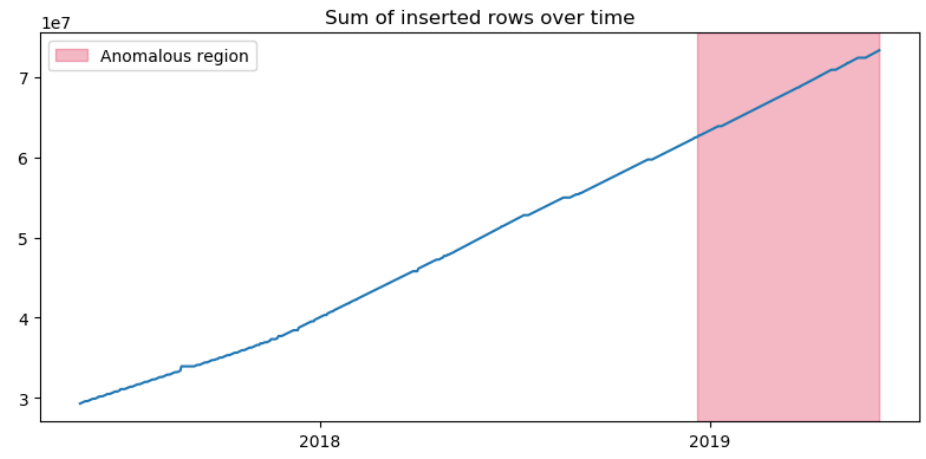

Figure A.2: Sum of inserted rows for process A.1 across time, displaying a stable increasing trend.

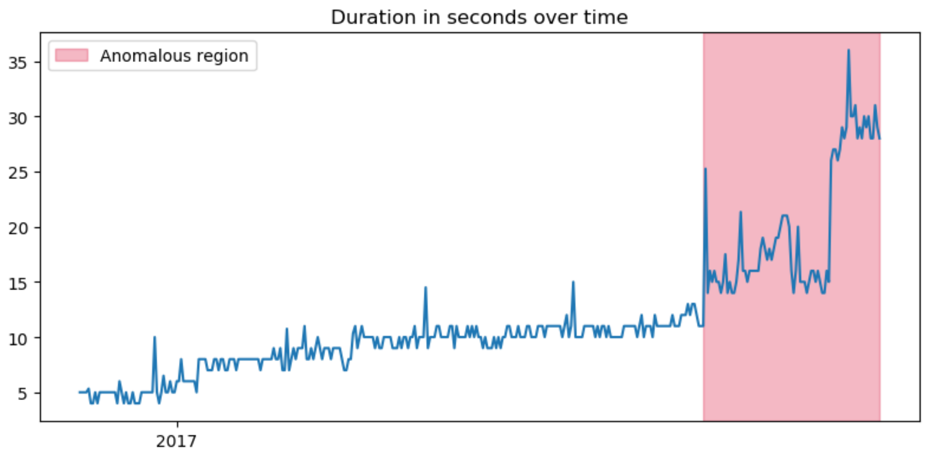

Figure A.3: Execution time of process A.2, displaying an unstable increasing trend followed by an abrupt change in behavior.

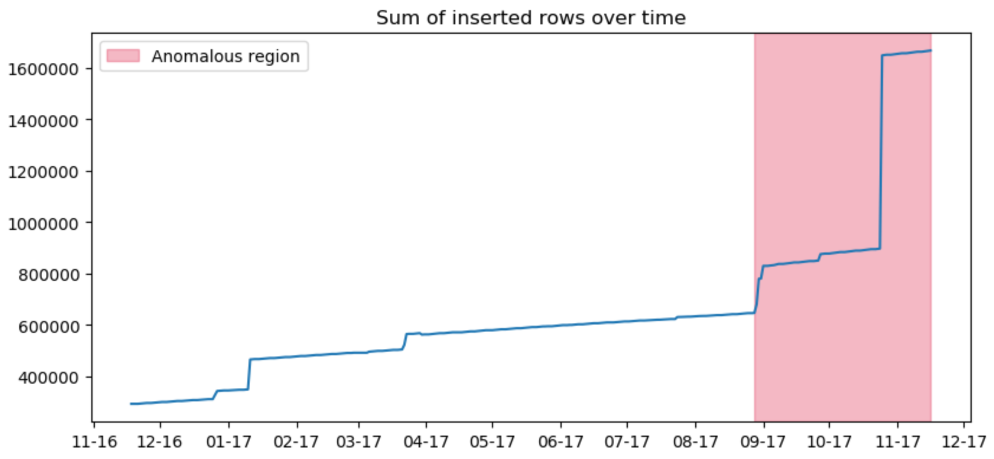

Figure A.4: Sum of inserted rows for process A.2, displaying two abrupt behavior changes, with a subtle increasing trend in between.

A.2 Anomalous trend

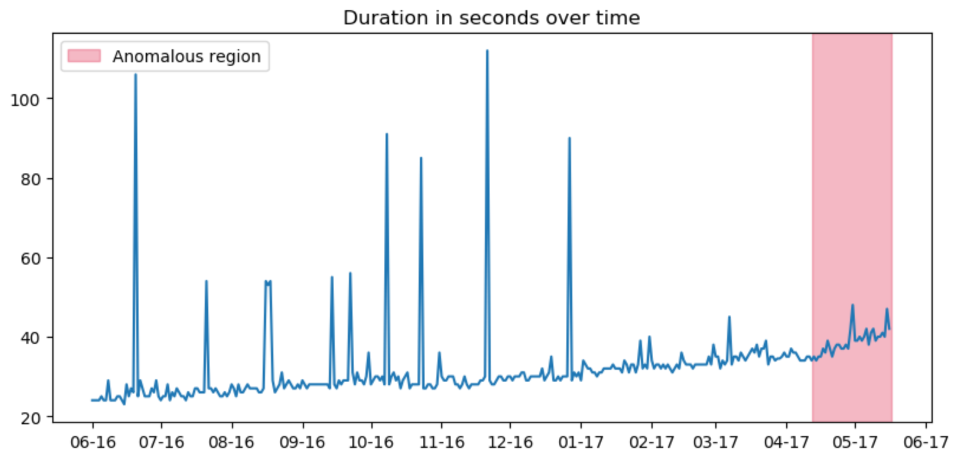

Figure A.5: Execution time of process B.1, displaying an increasing trend.

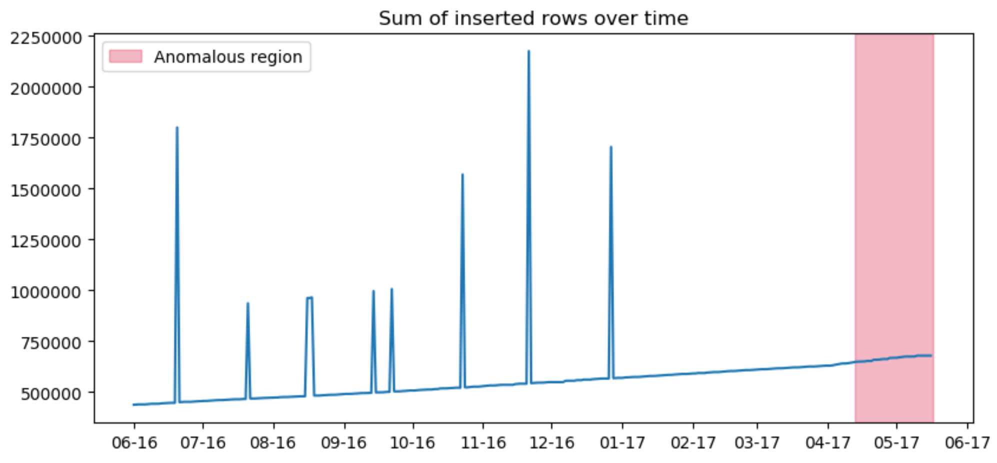

Figure A.6: Sum of inserted rows for process B.1, displaying an increasing trend.

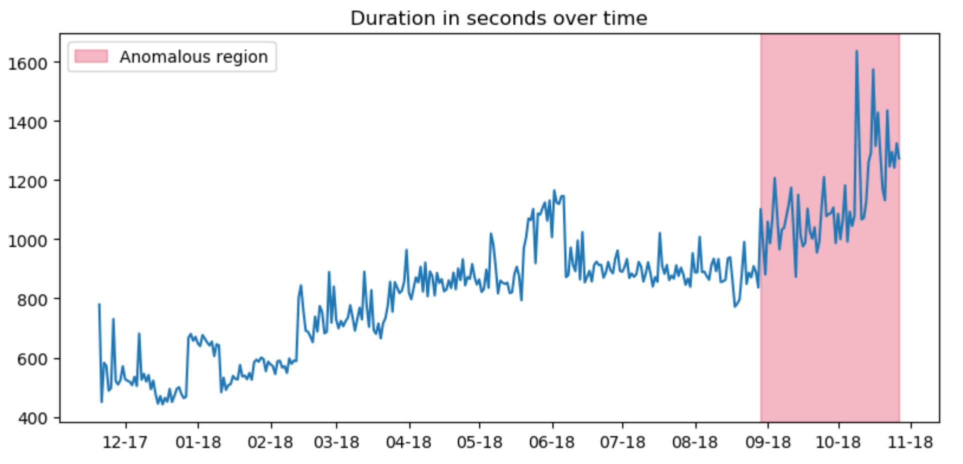

Figure A.7: Execution time of process B.2, displaying an unstable increasing trend.

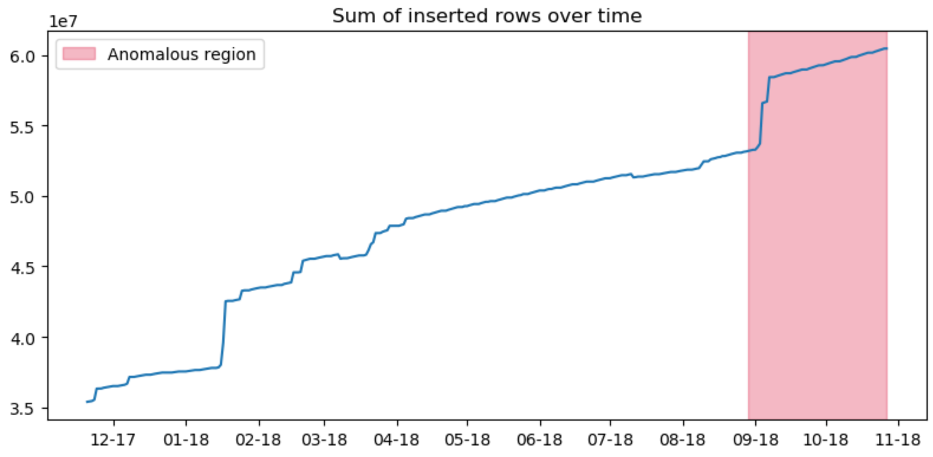

Figure A.8: Sum of inserted rows for process B.2, displaying an unstable increasing trend.

Appendix B

B.1 Mean F1-score and Confidence Interval:

In this appendix we present tables containing the mean F1-score for the top configuration across models and data sets, when varying the number of executions between the two values 10 and 50. The tables also contain the 95%confidence interval. This interval was computed with the values corresponding to the top F1-score of each execution.

B.1.1 process A.1:

Table B.1: Mean F1-score and confidence interval for two different runs for each model

model	number of runs	mean	confidence Interval	parameter set
Baseline	10	0.82	[0.76, 0.90]	{5, 10, 4, 4, 4}
	50	0.82	[0.73, 0.90]	
LSTM Prediction	10	0.83	[0.75, 0.90]	{10, 15, 4, 4, 4}
	50	0.83	[0.77, 0.89]	
Autoencoder	10	0.81	[0.76, 0.86]	{4, 10, 2, 4, 2}
	50	0.80	[0.75, 0.85]	

B.1.2 process A.2:

Table B.2: Mean F1-score and confidence interval for two different runs for each model

model	number of runs	mean	confidence Interval	parameter set
Baseline	10	0.94	[0.83, 1.00]	{10, 10, 4, 2, 4}
	50	0.93	[0.84, 1.00]	
LSTM Prediction	10	0.97	[0.94, 1.00]	{5, 10, 2, 2, 2}
	50	0.97	[0.94, 1.00]	
Autoencoder	10	0.87	[0.85, 0.89]	{4, 10, 6, 6, 2}
	50	0.87	[0.86, 0.88]	

B.1.3 process B.1:

Table B.3: Mean F1-score and confidence interval for two different runs for each model

model	number of runs	mean	confidence Interval	parameter set
Baseline	10	0.76	[0.71, 0.85]	{5, 10, 2, 2, 4}
	50	0.77	[0.71, 0.84]	
LSTM Prediction	10	0.78	[0.72, 0.82]	{5, 10, 2, 2, 4}
	50	0.78	[0.73, 0.82]	
Autoencoder	10	0.82	[0.79, 0.85]	{4, 15, 2, 2, 4}
	50	0.82	[0.8, 0.84]	

B.1.4 process B.2:

Table B.4: Mean F1-score and confidence interval for two different runs for each model

model	number of runs	mean	confidence Interval	parameter set
Baseline	10	0.84	[0.74, 0.96]	{5, 10, 2, 2, 4}
	50	0.84	[0.75, 0.93]	
LSTM Prediction	10	0.92	[0.87, 0.97]	{5, 10, 4, 2, 4}
	50	0.91	[0.87, 0.95]	
Autoencoder	10	0.98	[0.97, 0.99]	{4, 10, 2, 4, 2}
	50	0.98	[0.97, 0.99]	

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2020-03-12

EXAMENSARBETE Anomaly Detection From Metadata Through Deep Learning: A Case Study
STUDENTER Adnan Pasovic, Ahmad Ibrahim
HANDLEDARE Per Andersson (LTH)
EXAMINATOR Flavius Gruian (LTH)

Avvikelsedetektering genom djupinlärning

POPULÄRVETENSKAPLIG SAMMANFATTNING **Adnan Pasovic, Ahmad Ibrahim**

Maskininlärning blir ett allt mer populärt och kraftfullt verktyg för att lösa moderna problem. För att i större utsträckning förhindra kostsamma driftstörningar tillämpar vi avvikelsedetektering genom tekniker som efterliknar biologiska neurala nätverk.

Vårt arbete visar att användbara resultat kan uppnås i vår kontext då djupinlärningstekniker tillämpas för att detektera avvikelser. Vi ger även ett förslag på hur dessa tekniker kan användas i en produktionssättning. Våra resultat indikerar även att olika tekniker tycks vara bäst lämpade för att hitta specifika typer av avvikelser.

Många maskininlärningstekniker tar inte hänsyn till beroendet mellan mätvärden, utan behandlar varje mätvärde som en oberoende observation. Detta är problematiskt då man jobbar med data som naturligt bildar en serie av beroende mätvärden i tid. Exempel på sådana serier skulle kunna vara priset för en aktie på börsmarknaden, eller månadernas medeltemperaturer. Traditionella tekniker för avvikelsedetektering som bygger på statistisk analys visar sig ofta otillräckliga då datamängden och komplexiteten i data ökar. En typ av neurala nätverk som löser dessa problem är de så kallade återkopplade neurala nätverken. Dessa nätverk sparar information från tidigare mätvärden för att dra slutsatser om framtiden. Det finns olika typer av återkopplade nätverk, och i detta arbete jämför vi flera sådana nätverk och utvärderar deras prestanda i ett specifikt fall.

Mängden data som produceras i världen ökar ständigt, samtidigt som fler och fler verksamheter förstår värdet i att analysera sin data för beslutsstöd. Vårt arbete utfördes på Sigma IT AB, som bland annat erbjuder datalagerlösningar för verksamheter. Ett datalager kan ses som en databas där stora mängder data sammanförs från olika källor för att underlätta vidare analys. Data transporteras till datalagret och formas där med hjälp av specifika processer. Dessa processer kan vara väldigt många och upprepar sitt arbete periodiskt, exempelvis en gång om dagen. Det kan därför vara svårt att upptäcka när en sådan process börjar bete sig avvikande. Än värre är att dessa processer ofta upptäcks först när de orsakat allvarligare problem i systemet som kräver omedelbar översyn. Genom att tillämpa avvikelsedetektering kan man i större utsträckning fånga upp dessa processer i ett tidigt skede och undvika kostsamma verksamhetsavbrott.

I vårt arbete har vi utforskat möjligheterna för att tillämpa avvikelsedetektering i en specifik kontext. Arbetet innefattar även en jämförelse mellan olika tekniker som kan underlätta för framtida projekt på området. Vi ger även ett exempel på hur teknikerna skulle kunna tillämpas i en produktionsättning, där nätverken kontinuerligt fortsätter att lära sig. Vi styrker även slutsatser som dragits i relaterade arbeten, vilka legat till grund för vårt eget arbete.