Detecting semantic execution anomalies using deep learning techniques

Master Thesis

presented by
Andoneta Mile
Matriculation Number 1680486

submitted to the
Data and Web Science Group
Prof. Dr. Han van der Aa
University of Mannheim

July 2022

Abstract

Processes are everywhere and one of the biggest challenges for many companies and organizations is to assure anomaly-free process execution. An anomaly is identified as an outlying or unexpected behavior that does not conform to a standard process model or guidelines. Even though there are many widely used anomaly detection techniques, most of them have some major drawbacks. The majority identify anomalous behaviors based on frequency. If the behavior is infrequent, then it is anomalous. Another technique such as conformance checking requires a process model which in many real-life scenarios may be missing or incorrect. Some of the traditional machine learning techniques are also used for anomaly detection. However, most of them require a clean dataset and prior knowledge about anomalous behaviors. Therefore, the main goal of this research is to analyze a framework that overcomes these shortcomings where both the semantics and deep learning techniques are used to detect anomalous behaviors in event logs. The idea of the framework is to combine word2vec for feature extraction with LSTM-autoencoder for anomaly detection. In the end, an approach for classifying anomalies will be analyzed as well.

Keywords: *Anomaly Detection*, *Semantics*, *Deep Learning*, *NLP*

Contents

List of Figures

List of Tables

List of Abbreviations

AUC	Area Under ROC-Curve
BOW	Bag of Words
BPM	Business Process Management
CBOW	Continuous Bag of Words
KNN	K-nearest Neighbour
LOF	Local Outlier Factor
LSTM	Long-Short Term Memory
MAE	Mean Absolute Error
ML	Machine Learning
NLP	Natural Language Processing
NLTK	Natural Language Toolkit
OCC	One-Class Classification
OSVM	One-Class Support Vector Machine
PM	Process Mining
RNN	Recurrent Neural Network
ROC	Receiver Operating Characteristic
RPA	Robotic Process Automation
SVM	Support Vector Machine

Chapter 1

Introduction

Nowadays, anomaly detection is becoming a very important and crucial part of many companies and organizations. In Business Process Management (BPM), anomaly detection has traditionally focused on deviations from a reference model [35]. Businesses are naturally interested in detecting anomalies in their processes, because these can be indicators for inefficiencies in their process, badly trained employees, or even fraudulent behavior [18]. However, anomaly detection can also be applied to business process executions, for example to clean up datasets for more robust predictive analytics and robotic process automation (RPA) [21]. Especially in RPA, anomaly detection is an integral part because the robotic agents must recognize tasks they are unable to execute not to halt the process [19].

Anomalies can be found by analyzing the data that companies and organizations collect and store in their information systems. The structure, in which these data are stored, is called event logs. Event logs are considered to be the footprints of process execution. Event logs consist of traces which are a series of time-ordered executed events/activities with unique IDs. Process mining is one of the recent evolving fields that turns the data into insights through process discovery and conformance. Process discovery is used to discover a process model from the event logs, while conformance checks for deviations from standard guidelines/ process model. Conformance checking is one of many traditional anomaly detection techniques widely used by various authors.

This research proposes the analysis of a framework that combines the use of semantics and deep learning algorithms to detect anomalies in event logs. The semantics is captured by exploiting different NLP techniques. Through these techniques, the activities are converted into numeric representations recognizable by the computer. During this conversion, the semantics and the syntax between the activities are preserved. Afterward, deep learning algorithms suitable for sequen-

tial data are used for anomaly detection. This framework aims to overcome the drawbacks of traditional anomaly detection techniques and achieve the best possible results.

1.1 Problem Statement

Many of the existing anomaly detection techniques are designed to detect anomalies based on the assumption that anomalous behaviors are less frequent. Thus, if a given trace is infrequent in an event log, it is considered anomalous.

Another way to detect anomalies in event logs is through conformance checking. However, a known problem of PM is that in real scenarios the process model may not be available, be inadequate or incorrect, or significant domain knowledge may be required to elicit it [34].

More recently, Machine Learning (ML) has been proposed as an alternative to process-aware methods, as it can learn anomalies directly from the event log, without needing a reference model [18]. The training stage of ML requires, however, to hold enough ground truth labels for the output classes to be learned [14]. However, labeling the dataset requires a lot of manual work and can be very expensive.

Classic process anomaly detection algorithms require a dataset that is free of anomalies; thus, they are unable to process the noisy event logs [20]. It can be practically impossible to find datasets that are clean e.g., do not contain anomalies.

Recently, the autoencoder has been proposed as an unsupervised deep learning machine to detect anomalies and it can work perfectly with no labels, no reference model, no domain knowledge about the process, and no clean datasets. However, to the best of my knowledge, the authors that have used autoencoder or other deep learning techniques, have not taken into consideration the semantics.

1.2 Contribution

The main contribution of this research is to analyze evaluate the results of a framework where the semantics and deep learning machine will be used together to detect anomalies in event logs. This framework is a combination of the use of word2vec for feature extraction and LSTM-autoencoder for anomaly detection. Word2vec is an NLP technique used to create word embedding where words with similar semantic meanings will have similar vector representations. The vectors obtained from word2vec will be fed into the deep learning algorithm. LSTM-autoencoder is a combination of two different deep learning techniques. LSTM works well on sequential data while autoencoder is unsupervised learning that re-

produces its input. Therefore, this combination brings the advantages of both together.

Three other well-known traditional machine learning algorithms will be implemented using word2vec as feature extraction. The results obtained by using traditional machine learning will be then compared to the ones obtained by using deep learning, together with their pros and cons. The aim is to find the best-performing model for anomaly detection.

Finally, it is of interest to not only find if a trace is anomalous or not but to also detect the kind of anomaly. Therefore, throughout this research, an approach to classifying anomalies will be evaluated.

1.3 Related Work

Chandola et al. [8] give an overview of the existing techniques for anomaly detection for discrete sequences and they distinguish five different methods for anomaly detection: probabilistic, distance-based, reconstruction-based, domain-based, and information-theoretic novelty detection. However, this research has in its focus the use of semantics and deep learning to detect anomalies.

Processes cannot just be taken raw and used for anomaly detection. They need to be encoded in some kind of numeric form recognizable by the computer. The way that the processes are encoded determines if the semantics will be included or not. One of the most common ways chosen by authors until now is one hot and/or integer encoding, which ignores the semantics. Recently, many authors have proposed techniques for trace representations that incorporate the semantics between activities. Koninck et al. [10] proposed a different representation of learning for business processes at the activity, trace, log, and model levels. These representations can be used for various tasks including anomaly detection.

Tavares and Barbon [28] "argue that natural language encodings correctly model the behavior of business processes, supporting a proper distinction between common and anomalous behavior." They compared the word2vec encoding against token-based, replay, and alignment features in anomaly detection techniques. Results show the proposed encoding overcomes the representational capability of traditional conformance metrics for the anomaly detection task [28].

Luettgen et al. [17] extend Trace2vec to Case2vec where not only the events but their attributes are included in the representation. The authors used this representation for trace clustering, but it can surely be used for anomaly detections, providing that the context will be included as well.

Seelinger et al. [24] "presented a novel representation learning technique based on RNNs to obtain vector representations of cases in an event log automatically,

which can be used for different process mining techniques. Instead of forming an unsupervised learning problem, we train a neural network in a supervised fashion by using the sequence of activities including event attributes to predict the contextual factors of the corresponding case."

On the other side, many authors have used deep learning algorithms to detect anomalies. Nolle et al. [20] use a neural network approach that can learn a representation of the model and deal with the noise in the event log, unlike classic anomaly detection techniques. This neural network is represented by a denoising autoencoder that can deduce by itself normal traces from anomalous ones. This approach is especially interesting, as it shows that an autoencoder can capture the underlying process of an event log, without being provided extra knowledge [20]. Nolle et al. [18] is an extension of [20] where denoising autoencoders are again used for anomaly detections but with a refined approach and more sophisticated datasets which includes real-world event logs as well. Furthermore, the model can even distinguish which event in the traces and/or which characteristic of the event is anomalous.

Boehmer and Rinderle-Ma [4] incorporate other perspectives than control flow when detecting anomalies such as time, resources, etc. The performance and applicability of the overall approach are evaluated by means of a prototypical implementation and based on real-life process execution logs from multiple domains [4].

Nolle et al. [21] proposed BINet, a neural network architecture for multivariate business processes that can detect anomalies both on event and attribute levels. Due to its architecture, it can be used for real-time anomaly detection. BINet showed greater performance when compared to 6 other state-of-art anomaly detection techniques. The novelty of BINet lies in the tailored architecture for business processes, including the control and data flow, the scoring system to assign anomaly scores, and the automatic threshold heuristic[21]. Nolle et al. [19], which is an extension of [21], have "slightly simplified the architecture of BINet and present three different versions of BINet, each with different dependency modeling capabilities." In this paper, what makes it interesting and distinguishable from the others is that the model takes the output from BINet and based on simple rules can classify anomalies in different categories. Overall, the results presented in this paper suggest that BINet is a reliable and versatile method for detecting—and classifying—anomalies in business process logs [19].

Junior et al. [14] combined word2vec as feature extraction with OCC algorithms to detect anomalies. They tried to overcome the problem with a scarcity of labels by relying only on normal behaviors. It can be quite challenging to find event logs with labels and labeling them can be both exhausting and expensive. The authors compared the performance of Local Outlier Factor (LOF), One-Class

Support Vector Machine (OSVM), and Support Vector Machine (SVM) using both synthetic and real-world events logs with 6 different types of anomalies. The results from the paper showed that OOC algorithms outperformed traditional ML and presented better results in most cases. Thus, OCC is a suitable and straightforward solution to address scenarios with a scarcity of labels[14].

Van der Aa et al. [33] introduced a semantic-based anomaly detection technique by arguing that certain process behaviors are anomalous when they do not make sense and not when they are uncommon. This way of detecting anomalies is a completely novel way and the first of its kind to the best of my knowledge. It demonstrates "the capability of our approach to successfully detect semantic execution anomalies through an evaluation based on a set of real-world and synthetic event logs and show the complementary nature of semantics-based anomaly detection to existing frequency-based techniques."

Another way of detecting anomalies is by using graph autoencoders. Huo et al. [13] show in their paper "that the performance indeed can be substantially improved by enriching the autoencoder input data representation with activity relationships, i.e., edges between different events of a trace."

1.4 Outline

The remainder of this research includes the theoretical framework, the methodology, the experiments, and the conclusion.

Chapter 2 presents the theoretical framework of this research. First, it gives an overview of the event logs. Then, it presents the encoding strategy of transforming the processes into numeric representations recognizable by the computer. Afterward, it covers the anomaly detection techniques performed on these numeric representations. Lastly, it gives an overview of the evaluation metrics used to measure the performance of anomaly detection techniques.

Chapter 3 presents the methodology of the research. At first, it presents the datasets that will be used for the analysis. Then, it shows the pre-processing steps performed on these datasets and the feature extraction techniques. Afterward, it presents how to train a deep learning algorithm and how to detect anomalies. Lastly, it shows an approach to classifying anomalies.

Chapter 4 presents the experimental setups and experimental results obtained from different analyses.

Chapter 5 covers the conclusion, the limitations of this research, and future work.

Chapter 2

Theoretical Framework

This chapter will cover four main points of the theoretical framework as shown in figure 2.1.

Figure 2.1: Theoretical Framework

Section 2.1 covers an overview of event logs since the main point of the paper is to detect anomalies in event logs. What are event logs? How are they generated? Why it is important to study and analyze them? What kind of techniques are applied to event logs? What is the usefulness of the event logs?

Section 2.2 presents the encoding strategies of event logs that contain text, which is known as events/activities. The nature of NLP will be exploited to understand the processes from the semantic point of view. Since computers comprehend only numbers, for them to understand text, it needs to be encoded into some vectors of numbers. Some encoding techniques include TF-IDF, index, and Bag of Word (BOW), but they do not take into account the semantics. Therefore, word2vec and act2vec will be used as the encoding strategy to convert words/activities into a numeric vector representation.

Section 2.3 gives an overview of the anomaly detection techniques. After the numeric vector representations are obtained, anomaly detection techniques can be applied to them. These techniques include deep learning and machine learning

techniques. The machine learning techniques analyzed in this research include Local Outlier Factor (LOF), Support Vector Machine (SVM), and One-Class Support Vector Machine (OSVM). Regarding deep learning techniques, the combination of LSTM and autoencoder is chosen for the analysis.

Section 2.4 presents the metrics that will be used to evaluate the performance of the detection techniques such as accuracy, precision, recall, F-score, Receiver Operating Characteristic (ROC)- curve, Area Under the ROC curve (AUC), and the time of training an algorithm.

2.1 Event Logs

The amount of data that is being collected nowadays is enormously large. Al-most every organization is collecting data that is being generated and stored in their own information systems. In today's digital world, companies rely more and more on process-aware information systems (PAISs) to accelerate their processes [19]. The importance of information systems is not only reflected by the spectacular growth of data, but also by the role that these systems play in today's business processes as the digital universe and the physical universe are becoming more and more aligned [35].

Most of the stored data is unstructured, and raw which creates difficulties for organizations to deal with. The main challenge is to transform this large amount of raw data into valuable insights and information for the organization. This can be done with the help of PM techniques, one of the most recent and evolving areas in the field of data science and process management. Process mining is centered around the idea of human-readable representations of processes called process models [22]. Process mining techniques use event data to discover processes, check compliance, analyze bottlenecks, compare process variants, and suggest improvements [35]. The vast majority of existing process mining techniques either focus on process discovery, i.e., discovering a process model based on process execution data, or conformance checking, i.e., assessing whether a process model indeed is in correspondence with the observed behavior [31]. It is possible to record events such that (i) each event refers to an activity (i.e., a well-defined step in the process), (ii) each event refers to a case (i.e., a process instance), (iii) each event can have a performer also referred to as originator (the actor executing or initiating the activity), and (iv) events have a timestamp and are totally ordered [36].

An example of a simple event log with its main components is shown in table 2.1. An event log consists of cases, where each one of them has a unique case ID. Each case consists of timely ordered activities called a trace. Case ID, activities, and timestamps are the main attributes of the event log. However, there are

other attributes such as the user attribute that indicates the user that executed a certain activity, the cost attribute, department attribute, supplier attribute, etc. These attributes are useful to obtain further insights, an extended view of the case, and analysis from perspectives other than control flow. Anomalies occur when there are non-compliances with these standard guidelines or deviations from a proper process execution. Anomalies can be divided into work-flow anomalies and data-flow anomalies. Activities executed in the wrong order are an example of a work-flow anomaly, while the wrong user executing a certain activity is an example of a data-flow anomaly. The focus of this research will be work-flow anomalies and it will take into consideration only activities and timestamps attributes. Three types of anomalies are distinguished: order, exclusion, and co-occurrence. An order anomaly occurs when activities are executed in the wrong order. An exclusion anomaly is a case of an activity happening that in fact should not have. A co-occurrence anomaly occurs when an activity that should have been executed, in fact, has not.

Case ID	Activites	Timestamp
1155	Create Change Customer profile	1970-04-26 18:46:41+00:00
1155	Check Out	1970-04-26 18:46:43+00:00
1155	Choose Profile	1970-04-26 18:46:44+00:00
1155	Delivery Time	1970-04-26 18:46:45+00:00
1155	Credit Card Nbr	1970-04-26 18:46:47+00:00
...	...	...
2235	Credit Card Nbr	1970-04-26 18:46:50+00:00
2235	Complete Check Out	1970-04-26 18:46:51+00:00
2235	Confirmation Email	1970-04-26 18:46:52+00:00
2235	Cancel Order	1970-04-26 18:46:53+00:00
2235	Cancel Confirmed	1970-04-26 18:46:55+00:00

Table 2.1: Example of an event log

2.2 Encoding Strategy

Traditionally, ML and data mining techniques cannot be directly applied to PM event logs due to a mismatch at the representation level [29]. ML operates at the event level, where each event is an instance, and PM at the business case level, where a group of events represents an instance of the process [14]. This way, embedding techniques are necessary to overcome this gap [7].

Experts have designed various NLP text encoding techniques such as BOW, TF-IDF, word2vec, Glove, BERT, etc. Text encoding represents the process of converting raw text into numeric vector representation and preserving the relationship between words and sentences in a text. This idea will be used on the event logs where activities are seen as "words", traces are seen as "sentences" and the event log is seen as a "text".

Authors have chosen different encoding techniques for their anomaly detection techniques. In [20] and [21], the authors used one-encoding to transform each activity into an n-dimensional vector before feeding the autoencoder. The authors in [21], [19] used integer encoding and the vectors were fed into a neural network classifier called BINet that can detect anomalies on attribute levels as well. In [14], the authors compared the results of word2vec encoding with TF-IDF encoding on One-Class Classification Algorithms. Some other authors used more advanced and novel encodings. Based on the idea of word2vec, in [10], the authors proposed a learning representation on activity level, traces level, log level, and model level. In [17], the authors introduced case2vec encoding which includes besides activity encoding also attribute level encoding, trying to incorporate the context as well. The last two encodings were tested on trace clustering. However, the authors of both papers were confident that they will show great results in anomaly detection as well. Another type of encoding used for anomaly detection is graph encoding.

Some NLP text encoding techniques are explained below in more detail.

BOW is a type of encoding that represents the occurrence of a word in the text. The size of the vector equals the number of words in the text. If it is seen from the event logs perspective, the size of the vector representation would be equal to the number of words that appear in the event log. All these words would form the data corpus, have a specific position/index in the vector, and will be used as a baseline to create trace encoding. The encoding of a certain trace from the log would be obtained by placing 1 at the positions of the words that appear in the trace and 0 on the rest of the positions. This is called binary BOW. For non-binary BOW, instead of 1, there would be the number of times a certain word appears on a trace and the rest is 0. However, BOW comes with limitations because it does not take into consideration the semantics of the traces and the context of activities in a trace is overlooked.

TF-IDF is a type of encoding that represents the relevance of a word in a trace compared to the event logs. In other words, quantifies the relevance of a word in the event log. The importance of a word increases proportionally with the number of times it appears in one file, but it also decreases inversely with the frequency

it appears in the corpus [37]. TF stands for term frequency while IDF stands for inverse document frequency. TF-IDF is obtained by multiplying TF and IDF as shown in equation 2.1. Tf and IDF are calculated as it is shown in equations 2.2 and 2.3.

$$TF - IDF = TF * IDF \tag{2.1}$$

$$TF = \frac{\text{number of a particular word in a trace}}{\text{number of words in a trace}} \tag{2.2}$$

$$IDF = log\frac{\text{number of traces}}{\text{number of traces containing a particular word}} \tag{2.3}$$

Glove stands for global vector for word representation and is developed by Stanford. It is considered to be an extension of word to vector because captures information in a global context by aggregating the global word co-occurrence matrix. Similar to word2vec, Glove captures semantic information. GloVe, is a new global log-bilinear regression model for the unsupervised learning of word representations that outperforms other models on word analogy, word similarity, and named entity recognition tasks [23].

BERT stands for Bidirectional Encoder Representations from Transformers and is introduced by Google to understand search queries. The word embeddings generated, by being relied on an attention mechanism, are high-quality and capture contextual information. BERT is pre-trained on massive corpus such as Wikipedia datasets and its main usage is in translation tasks.

Word2vec , also known as word embeddings, turns words in natural language into dense vectors that computers can understand, and maps words that have similar meanings to nearby locations in the vector space [16]. Similar words would be closer to each other in the vector space, while dissimilar words would be apart from each other. These words embedding are obtained by training a neural network that has one hidden layer. Word2vec has two architectures that can be chosen depending on the given task: Continuous Bag of words (CBOW) and Skip-gram which are shown in figure 2.2a and 2.2b respectively. Skip-gram is trained to predict the context from the given words, while CBOW is trained to predict the word from the context. Which one to use? It depends on the task and datasets. Their main difference is that Skip-gram takes into account the distance between words while CBOW is indifferent to it. That means, the Skip-gram will give higher accuracy especially when the training corpus is large while CBOW will perform better.

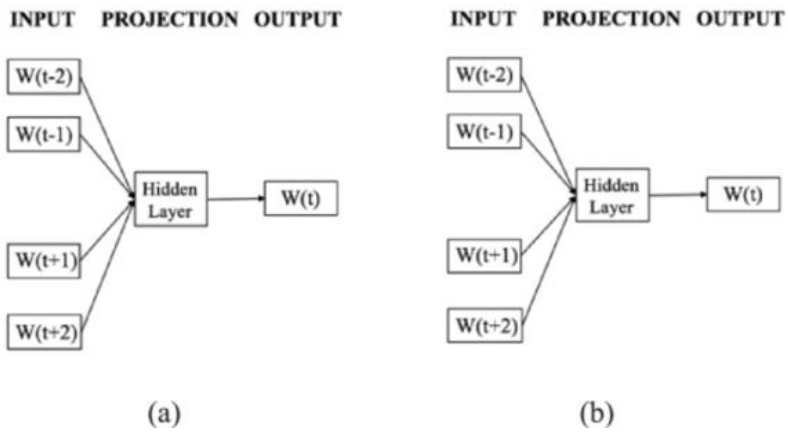

Figure 2.2: Architectures of Word2vec a) CBOW b) Skip-gram ([37], section 3A)

Act2vec does not differ a lot from word2vec. The main difference is that activities will be treated as a "single word" and instead of vector representation of words, vector representation of activities will be obtained. Let's take into consideration two different activities from the event log in table 2.1: **"create change customer profile"** and **"choose profile"**.

In the word2vec scenario, the activities would be tokenized into
"create"," change"," customer","profile","choose","profile"

In the act2vec scenario, there would be only two tokens
"create change customer profile" and "choose profile"

2.3 Anomaly Detection Techniques

Anomaly detection algorithms have gained a lot of popularity because of their ability to detect uncommon and unusual behaviors which are called outliers. The behavior of outliers deviates from the rest of the data points. The most common anomaly detection techniques are K-nearest neighbor (KNN), LOF, K-means, SVM, and Neural Network-based anomaly detection algorithms.

Learning techniques can be either supervised, unsupervised or semi-supervised. Supervised learning requires labeled trained data while in unsupervised learning such data are missing. In other words, in supervised learning, the correct answers

are known in contrary with unsupervised learning. Semi-supervised learning on the other hand combines the benefits of both.

Figure 2.3[1] gives a complete overview of all the anomaly detection algorithms and techniques divided into four main categories nearest-neighbor-based algorithms, clustering-based algorithms, classification-based algorithms, and statistical-based techniques.

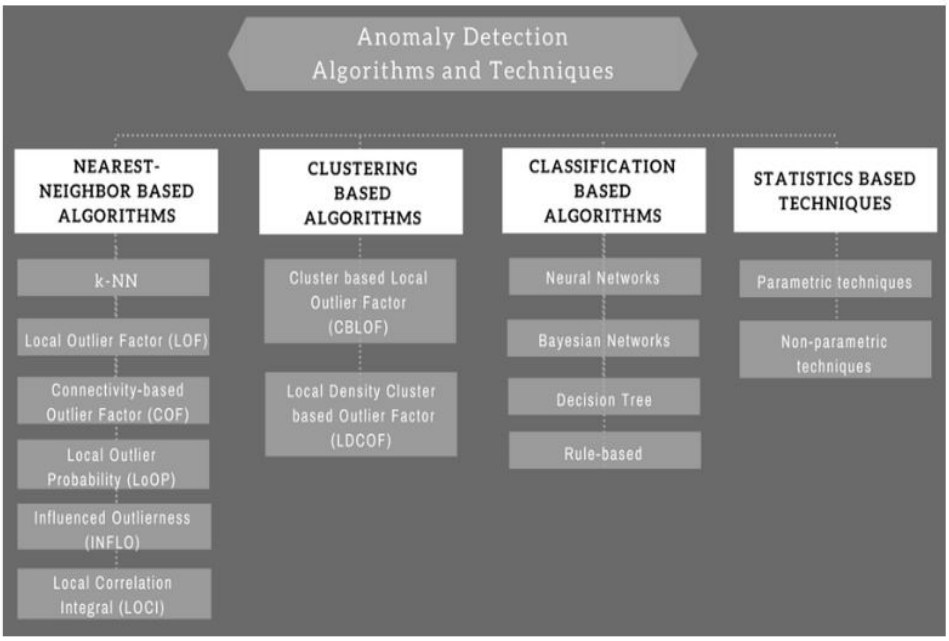

Figure 2.3: Anomaly Detection Techniques

Techniques analyzed throughout this research are divided into two groups deep learning and machine learning techniques, with the main focus on deep learning. Even though deep learning can sound like it is something different from machine learning, in fact, deep learning is machine learning but more advanced, complex, and ahead of machine learning.

[1] https://www.intellspot.com/anomaly-detection-algorithms/

2.3.1 Machine Learning Techniques

SVM, LOF, and OSVM belong to the class of traditional machine learning algorithms used for anomaly detection. SVM [9] belongs to the class of supervised learning and provides outstanding results in anomaly detection and not only. LOF [5] is another popular anomaly detection technique and together with OSVM [30], they belong to the class of One-Class Classification algorithms that take into consideration only the positive class of the dataset.

SVM is a supervised algorithm that performs classification, regression, and outlier detection tasks. The classification can be binary or multi-class. The main idea is to find a hyperplane, called the decision boundary, in an n-dimensional vector space that divides the data points into different groups. An example of SVM with hyperplanes is shown in figure 2.4. There might be many different hyperplanes, as in the left side of figure 2.4[2]. However, the desired hyperplane is the one with the maximal margin, as in the right side of figure 2.4. That means, that the data points of both classes will have the maximum distance between each other.

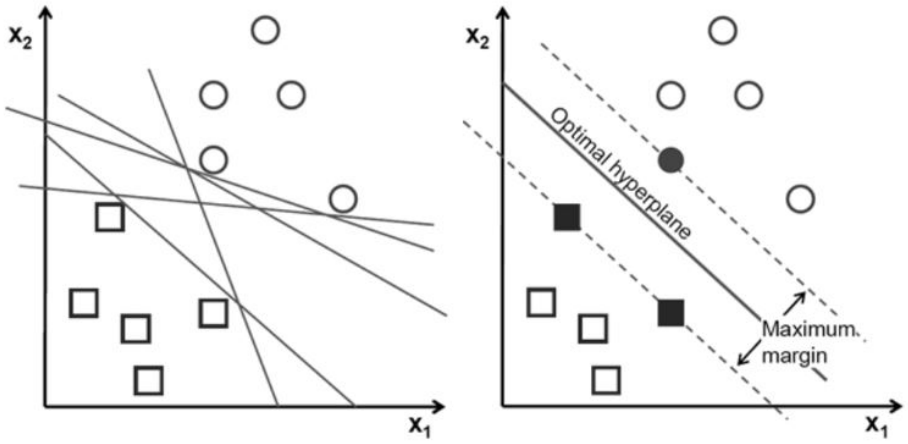

Figure 2.4: Hyperplanes in SVM

LOF is an unsupervised algorithm that identifies outliers by taking into consideration the density of its nearest neighbors and performs well on imbalanced datasets.

[2]https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

LOF detects outliers given a set of vectors, which are also treated as points in an n-dimensional space [14]. A given point is considered an outlier if its density is lower than the density of the k-nearest neighbors. An example of LOF is shown in figure 2.5[3]. LOF can be very sensitive to the chosen k parameter.

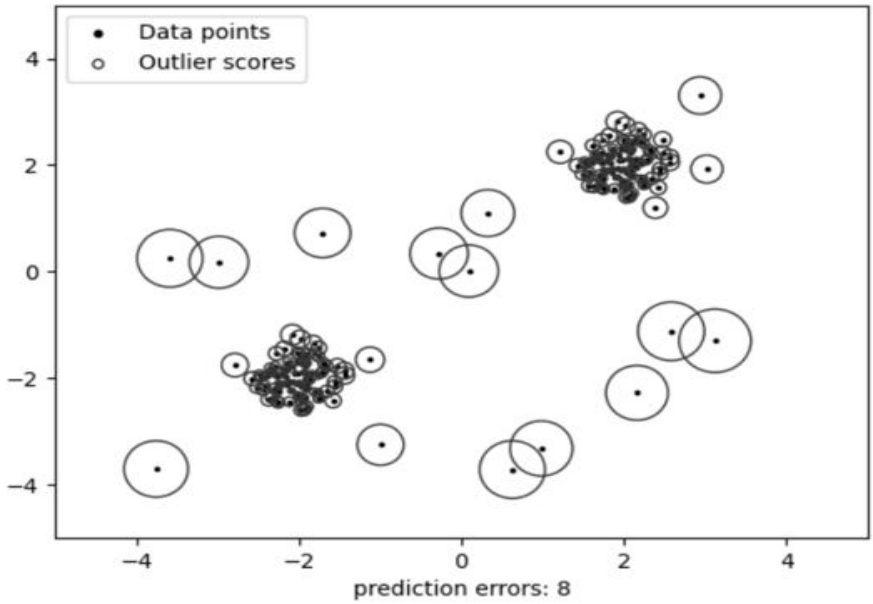

Figure 2.5: Example of LOF

OSVM is an unsupervised algorithm that is considered a version of SVM. Similarly, it considers the input vectors as points in an n-dimensional space but, instead of creating a hyperplane to separate the data into two classes, it creates an n-dimensional hypersphere that represents the input data such as the points that are inside the hypersphere are considered the normal behavior and the points that are outside are considered anomalies [14]. The main idea, as shown in figure 2.6[4], is to create a hypersphere that contains the maximum number of data points while keeping the radius as small as possible. The data points that lie outside this hypersphere are considered to be outliers.

[3] https://scikit-learn.org/stable/auto_examples/neighbors/plot_lof_outlier_detection.html

[4] https://www.analyticsvidhya.com/blog/2022/06/one-class-classification-using-support-vector-machines/

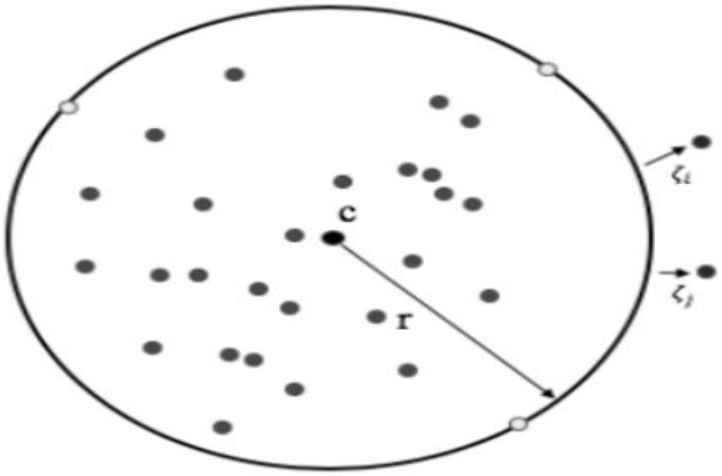

Figure 2.6: Hypersphere in OSVM

2.3.2 Deep Learning Techniques

Deep learning is a branch of machine learning that has been inspired by the human brain [15]. That is, deep learning methods try to replicate the way the human brain learns new concepts by connecting neurons with axons in the brain [20]. While reading a certain word in a text, the human brain remembers the previous words and that is how humans can understand the meaning or the context of what they are reading. This is something that traditional neural networks lack, they tend to forget the previous event(s). Figure 2.7[5] shows a simple neural network. It consists of the input layer, hidden layer, and output layer. These layers consist of nodes called neurons. The number of neurons in the input layer equals the number of features, while the number of neurons in the hidden layer is less. The information passes from the input layer to the hidden layer where random weights are assigned to the input nodes. Weights are multiplied with the inputs and some bias is added. The output is delivered by the output layer.

Luckily, the recurrent neural network (RNN) overcomes this major shortcoming of the traditional neural network. This is because their architecture consists of a loop that allows the information to persist. This is shown in figure 2.8[6]. The left

[5]https://levelup.gitconnected.com/a-tutorial-to-build-from-regression-to-deep-learning-b7354240d2d5

[6]https://ashwinijk19.medium.com/simple-multi-layer-recurrent-and-lstm-neural-network-with-

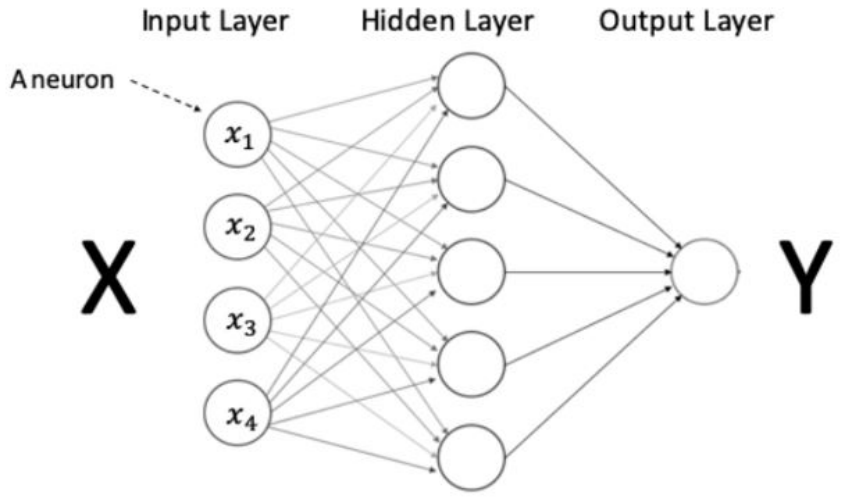

Figure 2.7: Simple Neural Network

side of the equation shows a neural network, which is given an input x, and gives an output o, while the loop let the information flow from one part of the neural network to the other. The right side of the equation shows what it will look like if the loop is unrolled. RNNs are successfully applied in various fields such as sentiment analysis, prediction problems, speech recognition, translation machines, language modeling, text generation, generating image descriptions, video tagging, anomaly detection, etc.

LSTM is a recurrent neural network (RNN) architecture published in 1997 [12]. Unlike traditional RNNs, an LSTM network is well-suited to learn from experience to classify, process when there are very long-time lags of unknown size between important events [37]. LSTM is built to solve the problem that RNN has with gradient vanishing. Gradient vanishing happens when the information vanishes in the recursive architecture of the RNN. LSTM overcomes this problem by adding additional input and forget gates in its structure and using the short-term memory to make the information stay longer. The structure of LSTM is presented in figure 2.9[7]. The problem of the information not vanishing too soon is fixed by another

tensorflow-9ee05d7e5aae

[7]https://medium.com/swlh/a-technical-guide-on-rnn-lstm-gru-for-stock-price-prediction-bce2f7f30346

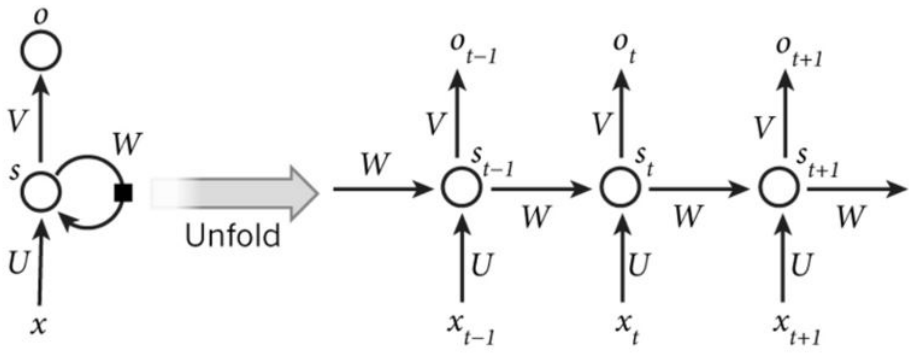

Figure 2.8: Recurrent Neural Network

layer that LSTM has in addition to the hidden layer which is called the cell state.

Autoencoder is an unsupervised classification technique that is based on an encoder-decoder architecture. The main idea is that the autoencoder takes an input, decompresses it into smaller dimensions, and tries to reconstruct it again in its original form as it is shown in figure 2.10[8]. By doing so, they can be a type of self-supervised learning as well. The autoencoder consists of three parts encoder, bottleneck (code), and decoder. The encoder takes the input data and transforms them into a lower-dimensional representation of the input. This representation is "kept" in the bottleneck layer in the middle. The decoder takes this output of the encoder from the bottleneck and reconstructs the input data. The input of the encoder is langer than its output, while the input of the decoder is smaller than its output. The goal is that the output from the decoder to be identical to the original output.

LSTM-autoencoder LSTM and autoencoder can be combined in one architecture where the advantages of both are exploited. This combination will be used for anomaly detection in this research. The idea is to implement an encoder-decoder LSTM architecture for the sequential data. In other words, LSTM is used to support sequential data while autoencoder is used to reconstruct its original input. Therefore, the autoencoder will be implemented to reconstruct the sequential data using the LSTM architecture. The performance is then evaluated by the ability to recreate the input in its original form.

[8]https://towardsdatascience.com/anomaly-detection-using-autoencoders-5b032178a1ea

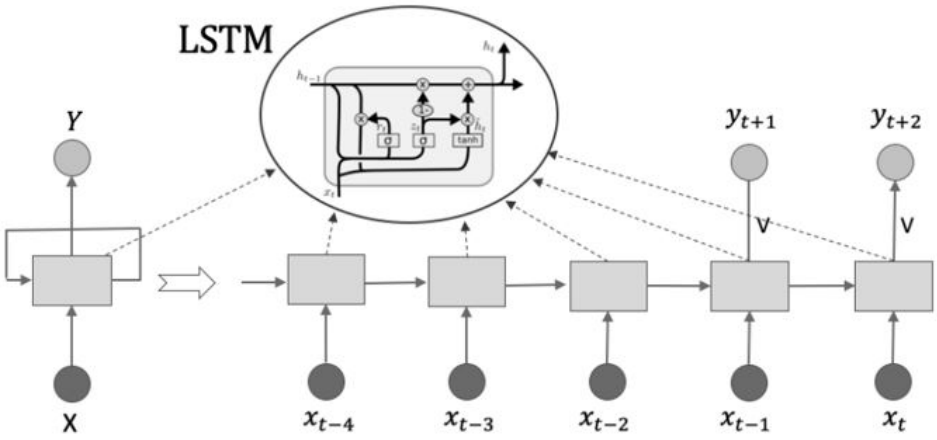

Figure 2.9: LSTM structure

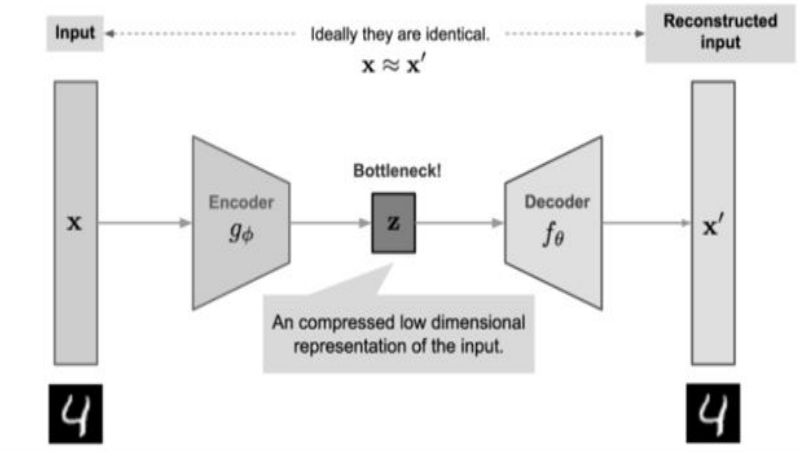

Figure 2.10: Example of an autoencoder

2.4 Evaluation Metrics

Evaluation metrics measure the quality and the performance of a given model. To quantify this performance different metrics such as accuracy, precision, recall, f -score, ROC/AUC, and time of training are used. A more thorough explanation of these metrics is given in [32]. The parameter used to calculate these metrics are True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN). These values can be obtained from a table called the confusion matrix, as shown in Figure 2.11 as well. TP represents the instances that belong to the positive class and are classified as positive. TN represents the instances that belong to the negative class and are classified as negative. FN represents the instances that belong to the positive class but are incorrectly classified as negative. FP represents the instances that belong to the negative class but are incorrectly classified as positive.

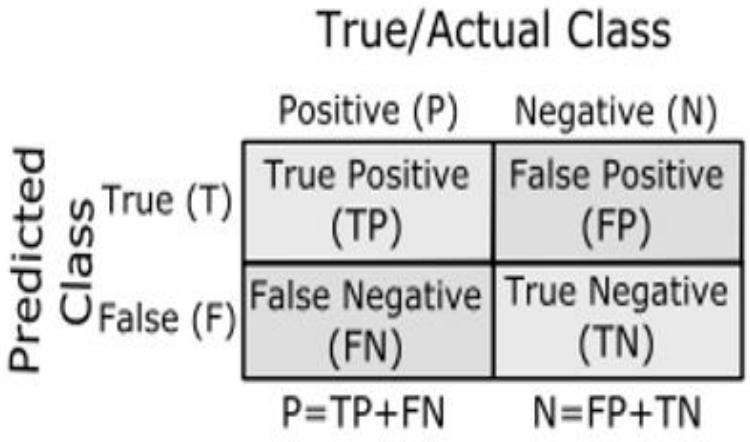

Figure 2.11: Confusion Matrix, ([32],Fig.1)

Accuracy is one of the most commonly used evaluation metrics. It represents the amount of correctly classified instances over the number of total instances, and it is calculated as presented in equation 2.4.

$$Accuracy = \frac{TP + TN}{TP + TN + FN + FP} \tag{2.4}$$

Precision represents the fraction of instances that are correctly classified as positive over the total amount of instances classified as positive and it is calculated as presented in equation 2.5.

$$Precision = \frac{TP}{TP + FP} \tag{2.5}$$

Recall represents the fraction of instances that are correctly classified as positive over the total amount of positive instances, and it is calculated as presented in equation 2.6.

$$Recall = \frac{TP}{TP + FN} \tag{2.6}$$

F-score represents the harmonic mean of both precision and recall and it is calculated as presented in equation 2.7.

$$Fscore = \frac{2 * precison * recall}{precision + recall} \tag{2.7}$$

ROC and AUC ROC is a probability curve that has False positive rate (FPR) as the x-axis and a True positive Rate (TPR) as the y-axis. FPR and TPR can be calculated respectively as it is shown in equations 2.8 and 2.9. AUC is a metric that represents the capability of the model to separate the positive class from the negative class.

$$FPR = \frac{FP}{\text{Negative Instances}} \tag{2.8}$$

$$TPR = \frac{TP}{\text{Positive Instances}} \tag{2.9}$$

Chapter 3

Methodology

This chapter covers the methodology of this research summarized in figure 3.1.

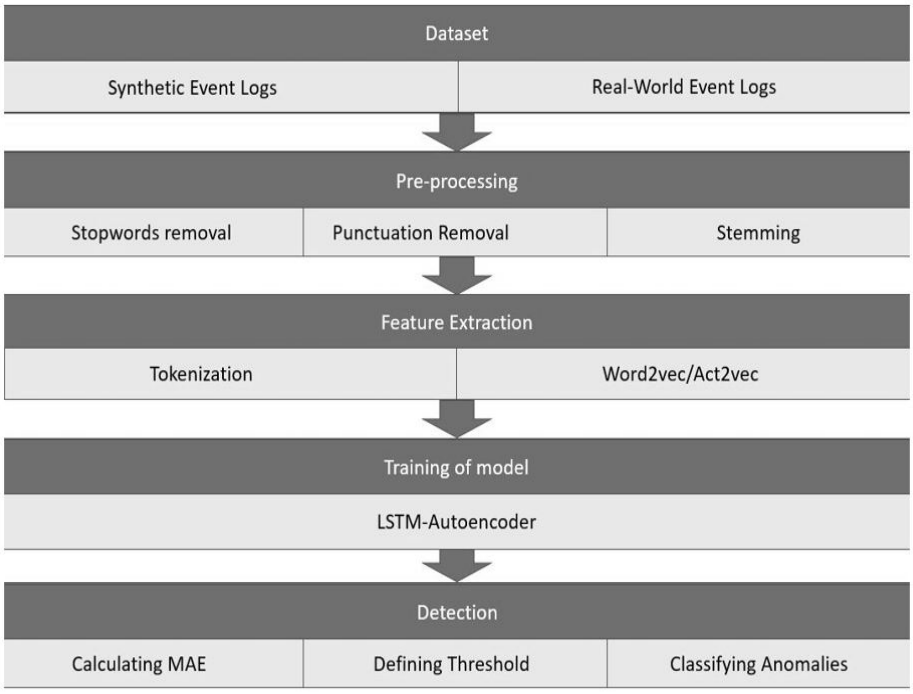

Figure 3.1: Methodolody

Section 3.1 gives an overview of the event logs. Both real-world and synthetic event logs will be used for the evaluation of the model performance.

Section 3.2 covers the pre-processing steps that will be performed on each event log. This includes the removal of stopwords, the removal of punctuation, stemming, etc.

Section 3.3 represents the feature extraction. First, the activities will be tokenization and afterward, they will be encoded into numeric vector representations using word2vec and act2vec.

Section 3.4 covers the training of a deep learning model. The vectors obtained from word2vec/act2vec will be fed into LSTM-autoencoder to train the model. As output, the model will reconstruct its original input. In other others, LSTM-autoencoder will recontract the representation of each trace that is fed into the training model. The model will be trained only on normal traces but evaluated on both normal and anomalous traces.

Section 3.5 gives presents the detection of anomalies. The recontraction error between the input and output of the model will be used as a detection metric. Firstly, the reconstruction error from the training datasets will be calculated, and based on that the threshold will be defined. Since the model was trained only on normal traces, it learned a representation of what the normal traces look like. Therefore, it will reconstruct normal traces with low reconstruction error and anomalous traces with higher reconstruction error. Therefore, a value will be defined as a threshold to divide normal traces from anomalous ones. If a trace is reproduced with a reconstruction error higher than the threshold, it will be considered anomalous. Otherwise, it will be considered normal. The reconstruction error will be calculated using the Mean absolute error (MAE) metric.

Section 3.6 presents an approach to classifying the anomalies.

3.1 Datasets

The “right” data is very crucial for the performance of the model. Sometimes finding the right datasets to evaluate the model is challenging. The performance of the model in this research will be evaluated using real-world and synthetic datasets. The real-world datasets are generated by the execution of real processes in the organizations, while the synthetic datasets are generated using different tools and algorithms. Synthetic datasets are very important for companies especially when they cannot find the data for their analyses or when they work with sensitive data and want to preserve anonymity. The datasets used in this paper are divided into three categories: real-world datasets from the Business Process Intelligence Challenge (BPIC), the first group of synthetic datasets, and the second group of syn-

thetic datasets.

Real-world event logs are taken from Business Process Intelligence Challenge. There are 10 of them in total, respectively one dataset from BPIC 2012 [1], three datasets from BPIC 2013 [2], five datasets from BPIC 2015 [3], and one dataset from BPIC 2017 [4]. Table 3.1 represents the characteristics of each of these datasets such as the number of unique tasks for each dataset, number of total traces, number of unique traces, number of normal and anomalous traces, and the percentage of anomalous traces that each of the datasets has. The number of tasks in an event log varies from 9 to 306, while the number of traces from 819 to 42,995. The percentage of anomalous traces is around 30% in all the event logs.

Event Logs	# unique Tasks	# traces	# unique Traces	# normal Traces	# anomalous Traces	% of anomalies
bpic12-0.3-1	71	13,087	5,694	9,115	3,972	30.35%
bpic13-0.3-1	13	1,487	417	1,098	389	26.16%
bpic13-0.3-2	25	7,554	2,911	5,237	2,317	30.67%
bpic13-0.3-3	9	819	224	658	161	19.66%
bpic15-0.3-1	290	1,199	NaN	886	313	26.11%
bpic15-0.3-2	306	832	NaN	579	253	30.41%
bpic15-0.3-3	279	1,409	NaN	995	414	29.38%
bpic15-0.3-4	274	1,053	NaN	745	308	29.25%
bpic15-0.3-5	287	1,156	NaN	809	347	30.02%
bpic17-0.3-2	17	42,995	967	30,032	12,963	30.15%

Table 3.1: Real-Worl Event Logs

First group of synthetic event logs was taken from [14]. The authors used the Process Randomization and Simulation (PLG)[6] tool to generate six random process models with different complexity, and for each process model, four different event logs were created. The average number of traces in an event log is 3,300, while the number of tasks per event log varies from 25 to 149. On top of all these event logs, four other event logs were generated based on the procurement-to-pay (P2P) process model. That makes a total of 28 generated event logs using the PLG tool shown in table 3.2. The percentage of anomalous traces in an event log is

[1] https://data.4tu.nl/articles/dataset/BPI_Challenge_2012/12689204/1
[2] https://www.win.tue.nl/bpi/doku.php?id=2013:challenge
[3] https://data.4tu.nl/collections/BPI_Challenge_2015/5065424
[4] https://data.4tu.nl/articles/dataset/BPI_Challenge_2017/12696884/1

around 30%, the same as in the real-world event logs group. One last thing to be mentioned is that, in gigantic, huge, large, medium, small, and wide event logs, the activities are labeled “Activity A”, “Activity B”, “Activity E”, “Activity K”, “Random activity 4”, “Random activity 19”, and so on.

Event Logs	# unique Tasks	# traces	# unique Traces	# normal Traces	# anomalous Traces	% of anomalies
gigantic-0.3-1	149	3,353	539	2,352	1,001	29.85%
gigantic-0.3-2	149	3,410	516	2,390	1,020	29.91%
gigantic-0.3-3	154	3,320	661	2,303	1,017	30.63%
gigantic-0.3-4	151	3,330	595	2,329	1,001	30.06%
huge-0.3-1	107	3,388	620	2,392	996	29.40%
huge-0.3-2	107	3,388	573	2,413	975	28.78%
huge-0.3-3	107	3,347	675	2,338	1,009	30.15%
huge-0.3-4	107	3,319	590	2,359	960	28.92%
large-0.3-1	83	3,623	857	2,541	1,082	29.86%
large-0.3-2	83	3,448	641	2,450	998	28.94%
large-0.3-3	83	3,310	687	2,343	967	29.21%
large-0.3-4	82	3,346	680	2,375	971	29.02%
medium-0.3-1	63	3,348	533	2,350	998	29.81%
medium-0.3-2	63	3,313	457	2,351	962	29.04%
medium-0.3-3	63	3,524	557	2,444	1,080	30.65%
medium-0.3-4	63	3,312	494	2,257	1,055	31.85%
p2p-0.3-1	25	3,387	442	2,442	945	27.90%
p2p-0.3-2	25	3,344	381	2,411	933	27.90%
p2p-0.3-3	25	3,369	492	2,318	1,051	31.20%
p2p-0.3-4	25	3,310	420	2,325	985	29.76%
small-0.3-1	39	3,638	523	2,558	1,080	29.69%
small-0.3-2	39	3,318	418	2,308	1,010	30.44%
small-0.3-3	39	3,351	424	2,319	1,032	30.80%
small-0.3-4	39	3,287	524	2,303	984	29.94%
wide-0.3-1	56	3,456	451	2,420	1,036	29.98%
wide-0.3-2	61	3,303	424	2,355	948	28.70%
wide-0.3-3	67	3,340	483	2,339	1,001	29.97%
wide-0.3-4	67	3,308	519	2,313	995	30.08%

Table 3.2: First Group of Synthetic Event Logs

Second group of synthetic event logs was taken from [33]. The authors generated events logs from 2,832 process models as part of the BPM Academic Initiative (BPMAI)[38]. The process models were first transformed into a workflow net, and then the event logs were generated using the functionalities of Process Mining for python (PM4PY)[3]. Apart from the generated event logs, there were created noisy

event logs with three types of anomalies: order, exclusion, and co-occurrence. That means, there were two types of event logs: one only with normal traces and the other one with injected anomalies. However, all these generated event logs were missing a trace ID and were not labeled, which means that the traces and their labels were unknown. And, to evaluate the performance of the framework presented in this research, it was important to have traces and labels ("anomalous" and "normal"). The files of generated event logs were in a .xes format with only two attributes: the "concept: name", and the "timestamp". If the file is open in a XML format, it is possible to see which activities belong to a trace. Therefore, to solve the mentioned issue, first, a trace ID was added to both groups of event logs. Afterward, traces from event logs with injected anomalies were compared with traces from the event logs with only normal traces. This was done to define which traces in the event logs with injected anomalies were normal and which were not. That is how the labeled traces were obtained. In total, 23 event logs were chosen for this paper and an overview of these event logs is shown in table 3.3. In this group, the percentage of anomalous traces in an event log is around 40% and goes up to 72% and 77% for two event logs. The number of tasks is relatively lower compared to the other groups. It varied from 3 to 20, while the number of traces varies from 1,000 to 83,462.

3.2 Pre-processing

Data pre-processing is a very important step because it ensures the performance enhancement of the model. It is necessary to ensure the good quality of the data before they are used. The pre-processing steps applied on the event logs include:

1. All null values in the event log are removed.

2. All punctuations and any other character that is not a letter are removed. This is done using a regular expression that finds and replaces certain characters in a given text.

3. Turning all the letters in lowercase.

4. Removing all the stopwords. Stopwords include the most common and used words in a language. In the English language, that will include "the", "that", "a", "is", etc. These words are considered to be unnecessary for NLP or text mining because do not give any meaning to the text. Therefore, it is better to remove them, so that the focus will be on the important words.

5. Applying the "stemming" technique to all the tokens. Stemming is an NLP technique that transforms each word into its own stem or root.

All the above mentions steps are applied in the example below that shows a trace before and after applying the pre-processing steps.

Event Logs	# unique Tasks	# traces	# unique Traces	# normal Traces	# anomalous Traces	% of anomalies
89550409_noisy	7	1,584	404	924	660	41.67%
203197147_noisy	16	2,076	2,061	1,202	874	42.10%
223194244_noisy	13	1,000	917	549	451	45.10%
233995501_noisy	7	9,840	9,679	5,503	4,337	44.08%
374732161_noisy	9	1,419	340	823	596	42.00%
390510117_noisy	10	1,000	876	230	770	77.00%
610177401_noisy	12	1,000	838	593	407	40.70%
629071047_noisy	8	3,322	821	1,899	1,423	42.84%
637371609_noisy	9	3,377	864	1,886	1,491	44.15%
751598303_noisy	7	1,000	909	582	418	41.80%
802538081_noisy	11	83,462	9,225	47,106	36,356	43.56%
810806251_noisy	4	13,860	2,339	7,716	6,144	44.33%
897179204_noisy	3	18,943	10,038	10,624	8,319	43.92%
1270159869_noisy	13	1,000	920	566	434	43.40%
1362303264_noisy	8	14,578	14,357	8,203	6,375	43.73%
1416807492_noisy	9	3,377	886	1,919	1,458	43.17%
1588460268_noisy	9	1,000	656	278	722	72.20%
1608879135_noisy	20	5,047	5,047	2,842	2,205	43.69%
1665514673_noisy	9	3,377	915	1,877	1,500	44.42%
1964047616_noisy	13	1,000	917	560	440	44.00%
974771973_noisy	11	1,000	321	553	447	44.70%
1167521718_noisy	6	1,000	242	570	430	43.00%

Table 3.3: Second Group of Synthetic Event Logs

Before Pre-Processing: ['cancel confirmed', 'create change customer profile', 'create change customer profile', 'create change customer profile', 'choose remove goods']

After Pre-Processing: [cancel confirm creat chang custom profil creat chang custom profil creat chang custom profil choos remov good]

3.3 Feature Extraction

Now that the event logs are clean and ready to be used, the feature extraction steps can be applied to them. Feature extraction represents the process of encoding into a numeric vector representation by using either word2vec or act2vec. It includes two steps: tokenization and encoding.

Tokenization is the process of dividing a text into smaller units called tokens. This tokenization process will be applied to the activities of the event logs, and the tokens received from this process will be used to train the encoding model. The tokens used for the word2vec model distinguish from the tokens used for the act2vec model.

To visualize this difference, let's consider the below example:

Trace after Pre-Processing: [cancel confirm creat chang custom profil creat chang custom profil creat chang custom profil choos remov good]

Tokens to train word2vec: "cancel" , "confirm","creat", "chang", custom" "profil", "creat", "chang" , "custom"," profil", " creat", " chang", "custom", " profil"," choos"," remov", "good"

Tokens to train act2vec: "cancel confirm "," creat chang custom profil "," creat chang custom profil "," creat chang custom profil " , "choos remov good "

Encoding is the next step after tokenization. After the tokens are fed into the model, the model is trained, and it will give as output the numerical vector representation of these tokens. The model used for word2vec and act2vec is the same. The only difference is the tokens that will be fed to it. Word2vec requires the setting of some parameters for its training which are defined below:

- **Vocabulary** represents all the words or tokens that will be fed to train the model.
- **Vector size** represents the size of the word embedding vector.
- **Window** represents the number of surrounding words that will be taken into consideration, or in other words the width of the context
- **Minimum count** tells the model if the infrequent words will be included in training or not. For example, if the minimum count is set to 3, it would mean that all the words with a frequency less than 3 will be excluded from the training.
- **Sg** represents the architecture of the model which can be either Skip-gram or CBOW. If sg is set to 1, Skip-gram architecture is used. If set to 0, CBOW architecture is used.
- **Alpha** represents the number at which the learning rate starts. Learning rate is the ability of hidden layers to adjust themselves to the new input.
- **Training epochs** represent the number of times the neural network of the model is trained with the given dataset.

Figure 3.2 represents a vector representation of size 100 of token “remov”.

```
array([-1.74982905e-01,  9.20739993e-02, -1.28595876e-02,  8.66724830e-03,
        8.28854740e-02, -7.54872784e-02,  1.03422645e-02,  1.99948087e-01,
       -1.39409274e-01, -1.33819893e-01,  3.56229045e-03, -3.01282690e-03,
       -7.03065544e-02, -4.35471386e-02,  1.04845479e-01, -4.10389900e-02,
        1.83388427e-01,  5.48702963e-02, -8.73628780e-02, -2.09621578e-01,
       -3.95553485e-02, -1.48364142e-01,  3.01923156e-01,  1.35844834e-02,
       -1.17197819e-01,  6.55232817e-02, -8.77059773e-02,  1.92511305e-01,
       -1.14487901e-01,  1.90879777e-01,  2.06692174e-01, -8.13862160e-02,
        2.03148506e-04, -1.14505187e-01,  6.73024580e-02,  1.33503884e-01,
        2.72504333e-02,  5.94959669e-02, -1.01614483e-01,  1.15113750e-01,
        1.96493626e-01, -1.21916577e-01, -1.47306591e-01,  2.22313732e-01,
        1.12920471e-01, -6.59496859e-02,  2.16886085e-02, -1.00694284e-01,
        8.07082653e-02, -1.26330666e-02,  7.60719180e-02, -1.23187430e-01,
        2.79229581e-02, -1.00742526e-01, -1.71730891e-01,  2.47250292e-02,
       -7.33430088e-02,  3.72006334e-02,  8.76973420e-02,  1.40293483e-02,
        4.66297232e-02,  4.37711999e-02,  2.11402014e-01,  3.69981769e-03,
       -5.61373532e-02,  1.12859279e-01,  1.23984337e-01,  2.36602634e-01,
       -6.34490103e-02, -5.74534945e-02,  8.27995986e-02,  9.50863808e-02,
        5.84445931e-02,  2.34947592e-01,  8.69747251e-02,  1.12906605e-01,
       -2.23048013e-02,  1.02239408e-01,  4.96059023e-02, -1.67455658e-01,
       -2.08057031e-01,  5.46956435e-03,  1.76308937e-02,  9.58919227e-02,
       -7.36190602e-02, -1.87451825e-01,  1.27558291e-01,  2.96789091e-02,
        8.80352631e-02, -1.23559766e-01,  4.54004481e-02,  4.23956141e-02,
        3.19163688e-02, -5.39659783e-02,  1.79196641e-01, -3.70616242e-02,
        2.16141298e-01, -1.18704572e-01,  4.28022444e-02, -5.41677661e-02],
      dtype=float32)
```

Figure 3.2: Vector representation of token ”remov”

Now that the numeric representations are known for each token, it is easy to obtain trace representations. Each trace consists of some tokens. Therefore, the trace vector representation is the average of all the token representations that the trace consists of. If the size of the token representation is 100, then the size of the trace representation would be 100. These trace vectors will then be fed into the anomaly detection models.

Figure 3.3 represents an example of the 10 most similar tokens with the token ”remov”, which was obtained by training the model.

```
[('choos', 0.559474527835846),
 ('custom', 0.5590000748634338),
 ('good', 0.5420181155204773),
 ('creat', 0.517149031162262),
 ('chang', 0.4848603308200836),
 ('profil', 0.4847297966480255),
 ('warm', 0.293454110622406),
 ('receiv', 0.27688419818878174),
 ('deliveri', 0.2712882459163666),
 ('prepar', 0.2653534710407257)]
```

Figure 3.3: Top 10 most similar words with token ”remov”

3.4 Training the model

The next step is to train the anomaly detection model. However, not all the traces will be used for training. Instead, the traces will be split into the training set and the testing set using the ratio of 8:2. That means, that 80% of the traces will be in the training set and only 20% in the testing set. Only the normal traces of the training set will be used to train the model, but both normal and anomalous traces of the testing set will be used to test the model.

Before training the model, it needs to be defined first which is shown in figure 3.4. The model is sequential and consists of two parts encoder and a decoder where each part has two hidden layers. The encoder consists of two hidden layers, drop out in each layer and the repeat vector, while the decoder consists of two hidden layers, drop out in each layer, and the time distributed. For each layer, some parameters need to be set.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 1, 128)            117248

 dropout (Dropout)           (None, 1, 128)            0

 lstm_1 (LSTM)               (None, 64)                49408

 dropout_1 (Dropout)         (None, 64)                0

 repeat_vector (RepeatVector  (None, 1, 64)            0
 )

 lstm_2 (LSTM)               (None, 1, 64)             33024

 dropout_2 (Dropout)         (None, 1, 64)             0

 lstm_3 (LSTM)               (None, 1, 128)            98816

 dropout_3 (Dropout)         (None, 1, 128)            0

 time_distributed (TimeDistr  (None, 1, 100)           12900
 ibuted)

=================================================================
Total params: 311,396
Trainable params: 311,396
Non-trainable params: 0
_________________________________________________________________
```

Figure 3.4: Defining LSTM-autoencoder

- **Number of neurons** defines the number of nodes that each of the hidden layers has. The deeper you go into the neural network, the smaller the number of neurons gets. The number of neurons in the hidden layers of the encoder corresponds to the number of neurons in the hidden layers of the decoder
- **Activation function** makes the model learn more complex features by adding non-linearity to it. The neural network without an activation function is said to be a simple linear regression model. This function "activates" a neuron, and decides if it should give output or not. Some common activation functions include sigmoid function, TanH function, ReLu function, Softplus function, etc.
- **Return Sequences** can be set either to "true" or "false". If set at "true, the model will output the hidden state at each time step, otherwise, it will output the hidden state at the final time step only.
- **Dropout** technique is used to neutralize or "ignore" some of the neurons during the training to add some noise to the model. This is done to regularize and avoid overfitting in the model.

- **Repeat Vector** acts as a bridge between the encoder and decoder and defines the number of times a feature vector is replicated.
- **Time Distributed** allows each input to have a layer and keeps the one-to-one relationship between input and output.

After the model is defined, it will be trained and again some parameters need to be set.

- **Training data** consists of normal traces of the training dataset.
- **Epochs** represent the number of times the model is trained with the given dataset
- **Batch size** corresponds to the number of samples sent to a model at once.
- **Validation data** represents the testing dataset.
- **Shuffle** has two states "true", and "false". If shuffle is set to "true", the sequence that is fed to the model is shuffled, otherwise, it is not.

After the model is trained, a graph is obtained to show how the accuracy, training loss, and validation loss change when the model is trained over different epochs. An example of this is shown in figure 3.5. In this example, it is observed that both training and validation loss start to stay almost constant after training with 20 epochs, while the accuracy keeps increasing until epoch 100.

3.5 Detection

Now that the LSTM-autoencoder is trained, the output can be used to detect anomalous traces. To do so, first, the reconstruction error between the output and the input of the training data is calculated. MAE is chosen as a metric to calculate the reconstruction error and the formula to calculate it is given in equation 3.1 where

$$MAE = \frac{1}{n}\sum_{j=1}^{n}|y(i) - y_{pred}(i)| \tag{3.1}$$

- n= number of traces in an event log
- y(i) = the label for a given trace i
- ypred(i) = the predicted label for a given trace i

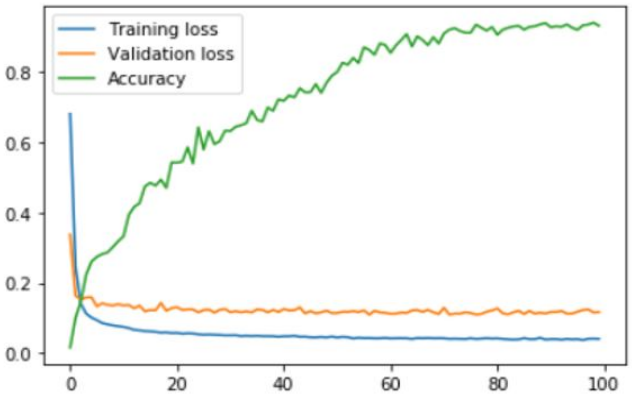

Figure 3.5: Training LSTM-autoencoder model over different epochs

Figure 3.6 shows an example of the reconstruction error graphs of both training and testing dataset. The x-axis represents the reconstruction error while the y-axis represents the number of traces having that reconstruction error. The reconstruction error in the testing dataset is higher than in the training dataset.

There are two ways to define the threshold for anomaly detection.

1. either manually from the graph of reconstruction error of the training dataset (e.g. figure 3.6a). The threshold is chosen at that point where most of the traces lie on the left of it but there are still some traces left on the right.

2. using the formula in equation 3.2 where trainMAE is the reconstruction error of the training dataset.

$$threshold = avg(trainMAE) + std(trainMAE) \tag{3.2}$$

Afterward, the recontraction error between the input and output of the testing dataset is calculated and compared to the defined threshold. If the reconstruction error is higher than the threshold the traces is labeled as anomalous, otherwise, it is labeled as normal. These labels are the predicted labels and will be compared to the real ones to get a model evaluation.

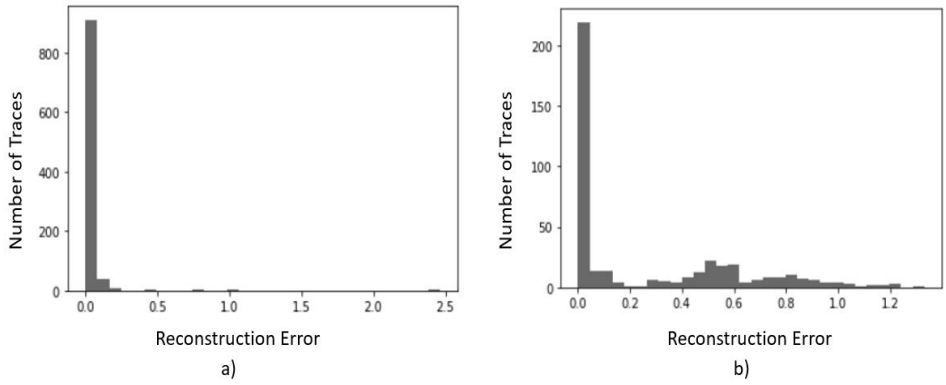

Figure 3.6: Reconstruction error a) training dataset b) testing dataset

3.6 Classifying Anomalies

Sometimes it is important to not only find the anomalous traces but also to be able to define what kind of anomaly a trace has and/or to classify the anomalies. The analyses were conducted only on the second group of synthetic event logs that contain only three types of anomalies order, exclusion, and co-occurrence.

The main idea of classifying anomalies is only slightly different from classifying traces into normal and anomalous. In this scenario, it will be calculated the reconstruction error of each activity in the trace instead of the recontraction error of the trace. In the trace classification scenario, the idea was to get vector representations of the words/activities in the trace, then average these vector representations to obtain a trace representation that will be fed into the anomaly detection model. In the anomaly classification scenario, the idea is to get vector representations of each activity in a trace, concatenate them together instead of averaging, and afterward fed them into the LSTM-autoencoder.

Let's assume a trace with two activities "create order", and "cancel" and a vector size of 100. The vector representations of the words "create", "order" and "cancel" will be of size 100 as well. If there is a trace classification scenario, it would be the average of vector representations of each word and the size of the trace being fed into the model is 100.

In the anomaly classification scenario, there are two activities and three words. The size of the first activity would be 2x100 which equals 200. The size of the second activity would be 1x100 which equals 100. Therefore, the size of the trace

would be 300, because it will not be averaged but concatenated instead. This would result in traces having different sizes because they consist of different activities.

This would cause a problem because the LSTM-autoencoder accepts traces of equal sizes. To overcome this issue, the padding solution [20] [21] is used. The idea is to find the longest trace in the event log and to extend the length of shorter traces to the length of the longest trace with vectors of 0. After doing that, the vectors can be fed into the LSTM-autoencoder.

The recontraction errors of each activity will be calculated instead of the recontraction errors of traces. Again, if the error exceeds the defined threshold, the activity is considered anomalous. The threshold is defined the same as before. If the activity is anomalous, that would make the trace anomalous. This would be another way to classify traces, not only anomalies.

After it is known which activity of the traces is anomalous, the use of human expertise can define the class of anomaly.

Chapter 4

Experimental Evaluation

This chapter covers the experimental results of the analyses. There are a total of 60 event logs on which the evaluation is made. The event logs are divided into three groups: the first group of synthetic event logs, the second group of synthetic event logs, and the real-world event logs. Two feature extraction are used as encoding strategies: word2vec and act2vec. The anomaly detection model is based on an LSTM-autoencoder architecture, and it will be compared to three machine learning techniques LOF, OSVM, and SVM. The metrics used for evaluation are accuracy, precision, recall, F-score, and AUC. Since training a deep learning model requires a lot of computational time, a time analysis for training an algorithm will be performed as well.

Section 4.1 presents the experimental setup such as the parameters, the environment, etc.

Section 4.2 presents the results and comparison between different models.

Section 4.3 concludes with discussions over the results.

4.1 Experimental Setup

All the experiments were conducted using a windows machine with 8GB memory and Intel Core i5 3,4 GHz CPU. The code was implemented using Python 3 in the Jupiter notebook launched by Anaconda Navigator.

After installing Anaconda, some other important packages/libraries were installed/downloaded including Nltk (Natural Language Toolkit), Gensim, Keras, and Tensor.

Nltk is one of the most important libraries used for different NLP analyses. It can be installed using the command **$ pip install nltk** and is imported as a python library using **import nltk** .

Gensim [1] is another open-source python library that is used for word2vec/act2vec. It can be installed using the command **$ pip install gensim** and is imported as a python library using **import gensim**.

Keras[2] is a deep learning API written in Python, running on top of the machine learning platform TensorFlow[3]. LSTM-autoencoder was built based on Keras. To use Keras, Tensorflow is to be installed using the command **$ pip install tensorflow**

For the training of word2vec and act2vec, the parameters are set such as vector size is 100, the window is 3, min count is 0, epochs is 10, start alpha is 0.025, and sg=1.

The LSTM-autoencoder is set to have two hidden layers for both the encoder and decoder where the number of neurons is 128 and 64 respectively with a dropout of 0.1 in each layer. As the optimizer was chosen the "adam", while activation was chosen "ReLu".

The LSTM-autoencoder was trained with epochs 100, batch size 32, and shuffle set to "True".

For LOF, OSVM, and SVM were used as parameters the ones that performed the best in [14]. In LOF, the contamination is 0.01, novelty is "True", and the number of jobs equals the number of CPU count. In OSVM, nu is 0.01, and the max number of iterations is 1.000.000 while, in SVM, the kernel is set to "poly", C is 1000, gamma is set to "scale", the maximum number of iterations is 1.000.000, number of estimators is 10, and number of jobs equals the number of CPU count.

4.2 Experimental Results

An overview of the performance of the LSTM-autoencoder is shown in the bar chart in figure 4.1. The graph represents the average accuracy, precision, recall, F-score, and AUC for all three groups of the event logs and for both types of feature extraction word2vec and act2vec. Overall, the model performed very well on all three groups of event logs. However, as is seen from the graph, the results achieved when using act2vec are only slightly higher than when word2vec was used.

The model achieved the best results for the second group of synthetics events logs for both word2vec and act2vec. The evaluation metrics for this group of event logs are shown in table 4.1. As it is seen, the average of all metrics for this group of event logs is around 90% and above. However, even though the performance is quite good on average, there is a big gap between the highest and the lowest

[1] https://radimrehurek.com/gensim/models/word2vec.html
[2] https://keras.io/guides/sequential_model/
[3] https://www.tensorflow.org/

scores measured for all the metrics. The highest that was achieved for accuracy, precision, recall, and F-score, was 100% and AUC was 1. The lowest was 62,5% for accuracy, 47,7% for precision, 42% for recall, 58% for F-score, and 0.697 for AUC.

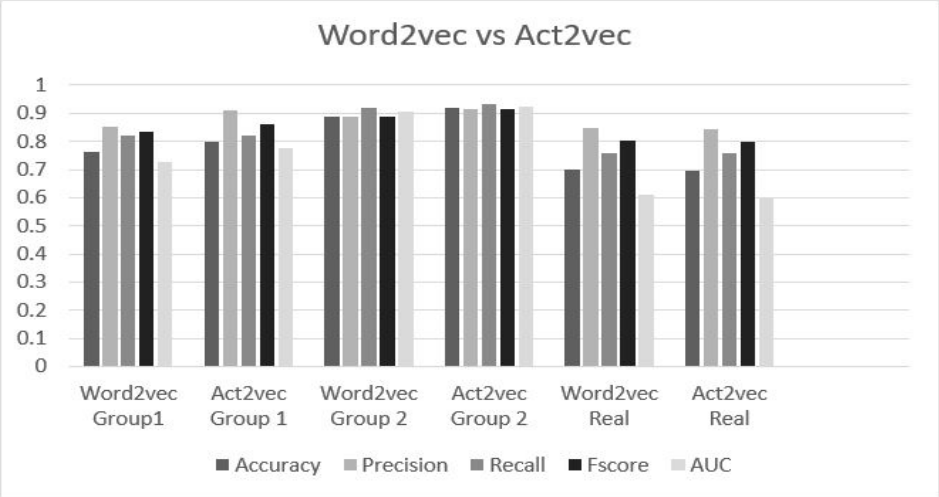

Figure 4.1: Performance of LSTM-autoencoder

	Accuracy	Precision	Recall	F-score	AUC
Word2vec Average	0.890488173	0.885957649	0.919252622	0.888477815	0.905227273
Word2vec Minimum	0.625	0.477477477	0.420634921	0.585635359	0.697
Word2vec Maximum	1	1	1	1	1
Act2vec Average	0.918320378	0.914779791	0.933793639	0.917567297	0.922227
Act2vec Minimum	0.67	0.775296506	0.452991453	0.61627907	0.714
Act2vec Maximum	1	1	1	1	1

Table 4.1: Evaluation Metrics for the second Group of Synthetic Event Logs

The first group of the synthetic event logs performed also quite well and the gap between the highest and the lowest scores for each of the metrics is not that big. The highest achieved for accuracy is 85.69%, precision is 98,8%, recall is 85,5%, F-score is 90% and AUC is 0.886. The lowest score is 68,2% for accuracy, 68% for precision, 78,3% for recall, 75,17% for F-score and 0.632 for AUC. These metrics are shown in table 4.2 as well.

Even though the results were quite satisfying for the groups of synthetic event

	Accuracy	Precision	Recall	F-score	AUC
Word2vec Average	0.765465152	0.853029246	0.819014052	0.83460581	0.726964286
Word2vec Minimum	0.682492582	0.680672269	0.787644788	0.751740139	0.632
Word2vec Maximum	0.841149773	0.961206897	0.849809886	0.896142433	0.834
Act2vec Average	0.798000041	0.910386228	0.82021917	0.862536703	0.7755
Act2vec Minimum	0.744011976	0.824053452	0.783001808	0.812294182	0.7
Act2vec Maximum	0.856932153	0.988071571	0.855218855	0.907706946	0.886

Table 4.2: Evaluation Metrics for the first Group of Synthetic Event Logs

logs, the model performed slightly worse on the group of real-world event logs. The highest accuracy is 80.7%, the highest precision is 96.7%, the highest recall is 86,82%, the highest F-score is 86.8% and the highest AUC is 0.792. The lowest accuracy achieved is 59.48%, the lowest precision is 76,39%, the lowest recall is 66,51%, the lowest F-score is 72,35% and the lowest AUC is 0.467. These metrics are shown in table 4.3 as well.

	Accuracy	Precision	Recall	F-score	AUC
Word2vec Average	0.69906565	0.850567867	0.759419303	0.801236107	0.6099
Word2vec Minimum	0.594827586	0.763975155	0.665158371	0.723529412	0.467
Word2vec Maximum	0.807303175	0.96744186	0.862903226	0.868835589	0.792
Act2vec Average	0.695906496	0.843398188	0.760312982	0.798621209	0.603889
Act2vec Minimum	0.61637931	0.787709497	0.680751174	0.739795918	0.469
Act2vec Maximum	0.780487805	0.958139535	0.868217054	0.861538462	0.752

Table 4.3: Evaluation Metrics for real-world Event Logs

Training a deep learning model can be computationally expensive and requires a lot of time. The graph in figure 4.2 shows the average time in seconds to train the LSTM autoencoder using both word2vec and act2vec on all three groups of event logs. This graph aims to show only the difference in training time between word2vec and act2vec for each of the groups individually. The training time in all three groups differs a lot because of the number of traces per event log in each of these groups. The number of traces in the event logs of the first group is around 3,400. The number of traces in event logs of the second group is from 1,000 up to 83,462, while the number of traces in real-world event logs of the is from 832 up to 42,995. In the first group of synthetic event logs, the time to train LSTM-autoencoder using act2vec is only slightly higher than when using word2vec. As it can be seen from table 4.4, the gap between the average, the highest, and the lowest training time is not that big. One of the main reasons is that the number of traces in all the event logs is almost the same.

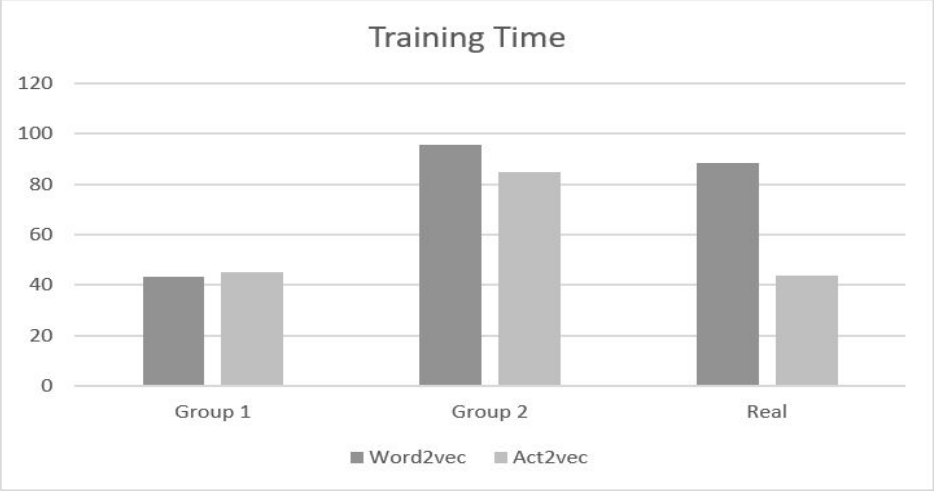

Figure 4.2: Training time of LSTM-autoencoder

In the second group of synthetic event logs, the time to train LSTM-autoencoder using act2vec is slightly lower than when using word2vec. However, as it can be seen from table 4.4, the gap between the average, the highest, and the lowest training time is quite big. The event log which recorded a time of 967.45sec with word2vec and 800.17sec with act2vec has a total of 83,462 traces. The event log which recorded the lowest time has a total of 1,000 traces.

In the group of real-world event logs, the difference in training time for word2vec and act2vec is more obvious. The gap between the average time, the highest and the lowest is quite big, as shown in figure 4.2. The event log for which the highest training time was recorded has a total of 42.995 traces, while the event log for which the lowest training time was recorded has a total of 832 traces.

	First group of synthetic event logs	Second group of synthetic event logs	Real-World Event logs
Word2vec Average	43.18038768	95.90405632	88.31027997
Word2vec Minimum	39.29778242	11.64218402	15.39281774
Word2vec Maximum	49.30590987	967.4580941	495.4050589
Act2vec Average	44.99018451	84.72285649	43.88392732
Act2vec Minimum	40.99531531	12.36899734	17.09656668
Act2vec Maximum	63.44573188	800.1739421	159.1960688

Table 4.4: Training time in sec of LSTM-autoencoder

The performance of the LSTM-autoencoder is compared to the performance of LOF, OSVM, and SVM using both word2vec and act2vec for each group of event logs.

Figure 4.3 represents a bar chart of the performance of LSTM-autoencoder, LOF, OSVM, and SVM on the first group of event logs using word2vec. At first glance, it can be noticed that the algorithm that performed the worse is OSVM, where all of the evaluation metrics, except AUC, are below 50%. Accuracy, recall, and F-score is the highest for the SVM algorithm, while precision is the highest for LSTM-autoencoder. The highest AUC is achieved for LOF.

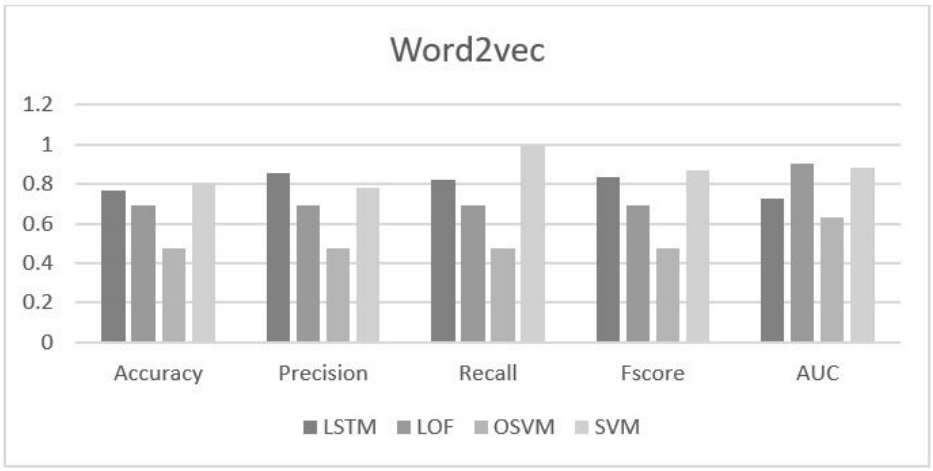

Figure 4.3: Comparision of algorithms using word2vec for the first group

Figure 4.4 represent a bar chart of the performance of LSTM-autoencoder, LOF, OSVM, and SVM on the first group of event logs using act2vec. Here, it can be noticed as well that, the algorithm with the worst performance is OSVM. All of the evaluation metrics for LOF, except AUC, are below 50%. Accuracy and precision are the highest for LSTM-autoencoder, recall is the highest for SVM, while F-score is almost the same for LSTM-autoencoder and SVM. Again, AUC is the highest for LOF.

Figures 4.5 and 4.6 represent the performance of LSTM-autoencoder, LOF, OSVM, and SVM on the second group of event logs using word2vec and act2vec respectively. Both figures show again that the algorithm that performed the worst is LOF. Except for AUC, this algorithm has very low accuracy, recall, precision, and F-score that reach 45% for wor2vec and around 20% for act2vec.

In figure 4.5, where word2vec is used, accuracy, recall, and F-score is the highest for SVM, precision is roughly the same for LSTM-autoencoder and SVM, while AUC is slightly higher for LOF compared to LSTM-autoencoder.

In figure 4.6, where act2vec is used, the accuracy and F-score are roughly the same for LSTM-autoencoder and SVM, precision is higher for LSTM-autoencoder while recall is higher for SVM. AUC again is only slightly higher compared to LSTM-autoencoder and SVM.

Figures 4.7 and 4.8 represent the performance of LSTM-autoencoder, LOF, OSVM, and SVM on real-world event logs using word2vec and act2vec respectively. While LOF performed the worst on the first 2 groups of event logs, its performance in real-world event logs is better and the evaluation metrics reach almost 60% for both word2vec and act2vec. The performance of all the algorithms is almost the same for both word2vec and act2vec. For both word2vec and act2vec, the accuracy, recall, and F-score are higher for SVM, while precision is higher for LSTM-autoencoder. In word2vec, the AUC is higher for LOF while in act2vec the AUC is higher for SVM.

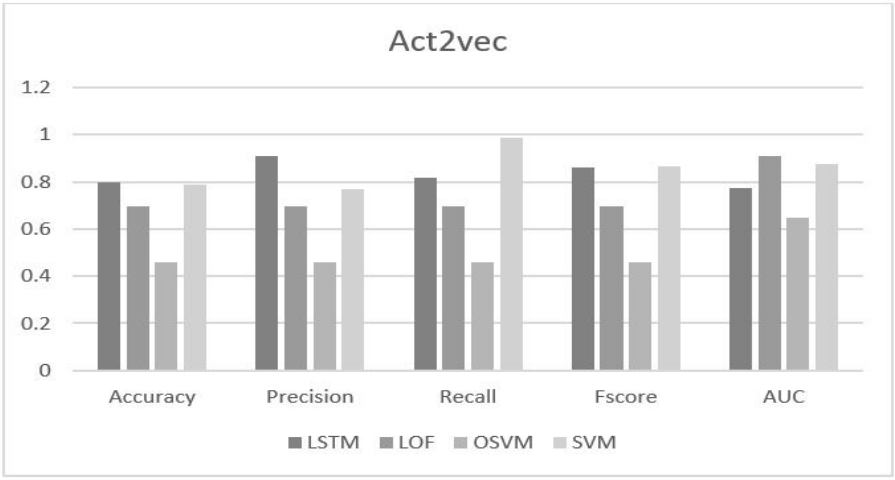

Figure 4.4: Comparision of algorithms using act2vec for the first group

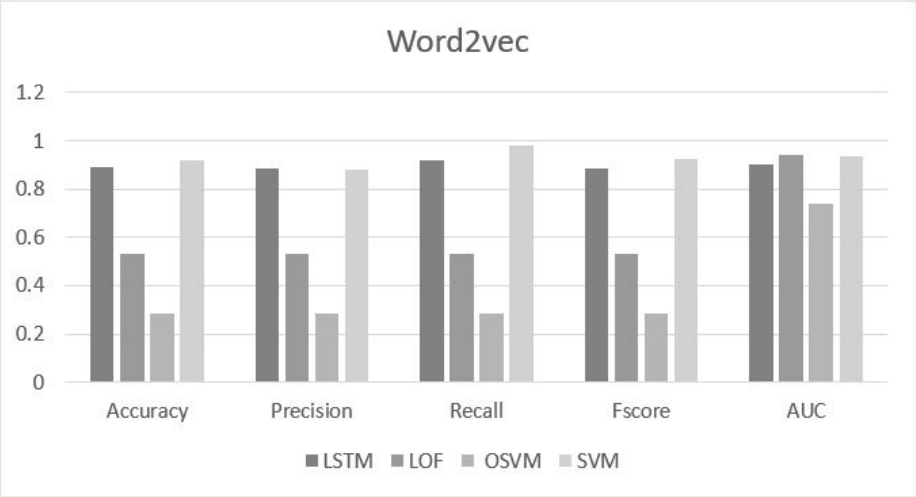

Figure 4.5: Comparision of algorithms using word2vec for the second group

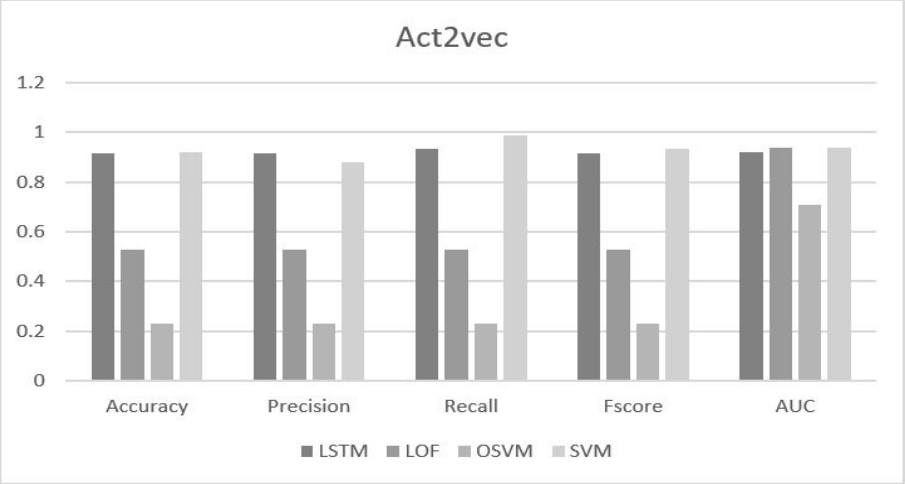

Figure 4.6: Comparision of algorithms using act2vec for the second group

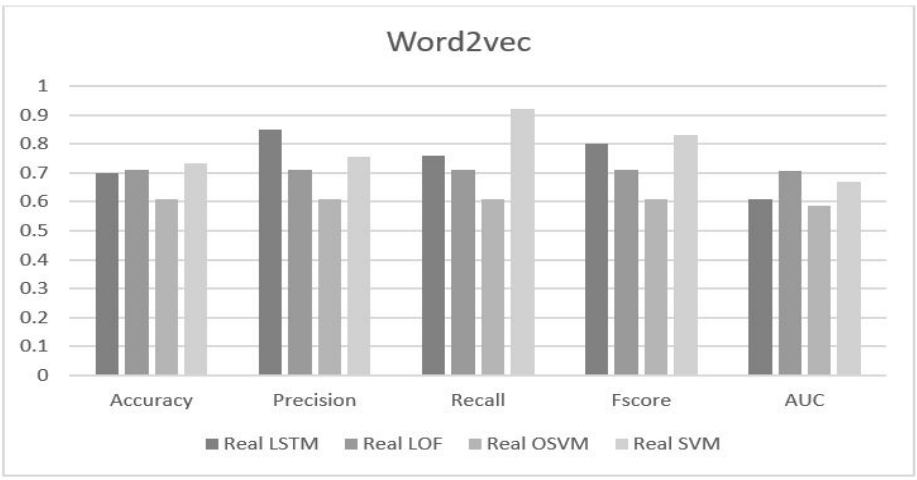

Figure 4.7: Comparision of algorithms using word2vec for real-world event logs

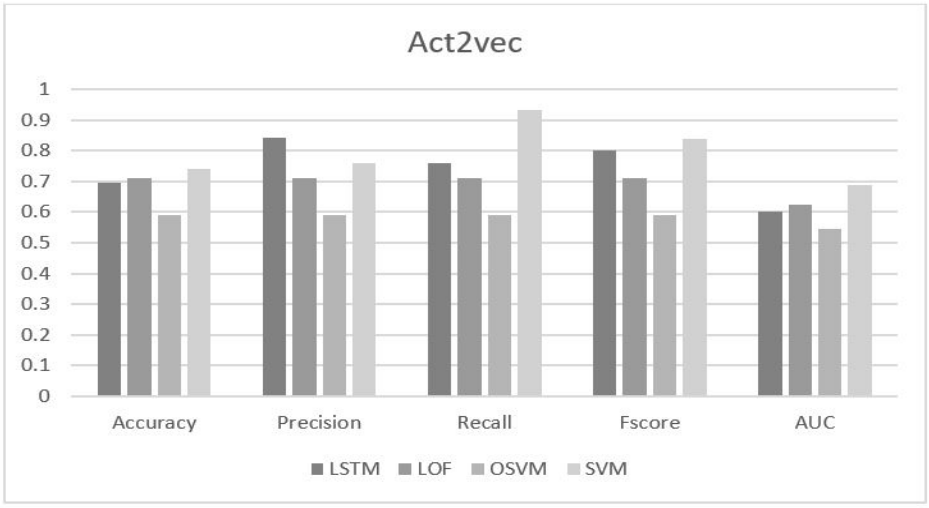

Figure 4.8: Comparision of algorithms using act2vec for real-world event logs

Training a deep learning algorithm takes a lot of time and this is perfectly represented in the bar chart in figure 4.9. The time shown in the graph represents the average time in sec for each group of event logs. Training a machine learning algorithm takes up to a few seconds while training a deep learning algorithm can go up to a couple of minutes. The algorithm with the lowest training time is OSVM, which is practically invisible in the graph. The highest training time is for LSTM-autoencoder when using word2vec.

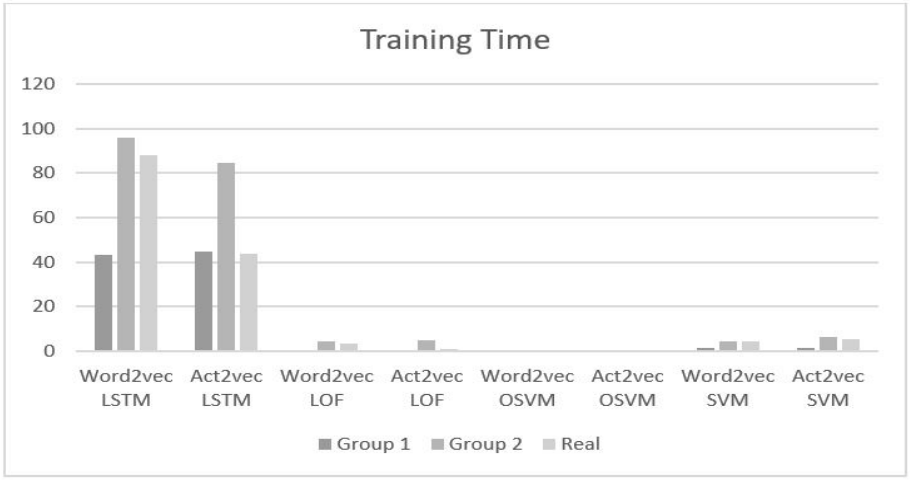

Figure 4.9: Comparision of training time between algorithms

Until now, the presented evaluations were to check the performance of algorithms in dividing traces into two groups normal and anomalous. However, this research evaluates also the approach to classifying anomalies. The evaluation was made only for the second group of synthetic event logs. The results seem to not be as satisfying as the above ones. The accuracy, precision, recall, and F-score are almost the same for both word2vec and act2vec. The average accuracy is around 62% which is not that satisfying. The highest score for accuracy is 94.88% while the lowest is 40%. The same results are for the precision, the recall, and the F-score, as it is shown in table 4.5.

The only difference between word2vec and act2vec was in the training time. The gap in the training time between word2vec and act2vec is quite big. The length of vectors fed into LSTM-autoencoder using word2vec is bigger than the length of vectors using act2vec. This will result of course in LSTM-autoencoder using word2vec being trained in a bigger amount of time than LSTM-autoencoder

	Accuracy	Precision	Recall	F-score	Time
Word2vec Average	0.615534957	0.615534957	0.615534957	0.61553496	232.1575349
Word2vec Minimum	0.4	0.4	0.4	0.4	35.27290392
Word2vec Maximum	0.939349112	0.939349112	0.939349112	0.93934911	991.7998357
Act2vec Average	0.625858909	0.625858909	0.62585891	0.625858909	198.84767
Act2vec Minimum	0.4	0.4	0.4	0.4	21.159796
Act2vec Maximum	0.94887218	0.94887218	0.94887218	0.94887218	510.17196

Table 4.5: Metrics for anomaly classification with word2vec and act2vec

using act2vec. This difference in training time is shown in the bar chart in figure 4.10.

Figure 4.10: Comparision of training time in classifying anomalies

4.3 Discussions

Based on the results presented in section 4.2, the performance of the LSTM-autoencoder is considered to be overall quite good in detecting anomalies. The second group of synthetic event logs achieved the best results in terms of accuracy, precision, recall, F-score, and AUC, followed by the first group of synthetic event logs. On the other side, even though, the group of real-world event logs didn't perform very well in

terms of accuracy, the precision for this group appeared to be quite satisfying.

In terms of encoding strategy, act2vec as feature extraction achieved slightly better accuracy, precision, recall, F-score, and AUC than word2vec and the training time of the LSTM-autoencoder was a bit lower with act2vec. The difference between word2vec and act2vec is not that noticeable when classifying traces into anomalous and normal, because the vectors were averaged which made the length of trace representations when using word2vec the same as when using act2vec. This difference becomes more noticeable in the training time when classifying anomalies. The length of traces depends on the number of words/activities of the trace. The lengths of traces with word2vec are longer than with act2vec, which in-creases the training time of the LSTM-autoencoder. Something interesting to be mentioned is that the event logs with 83,462 could not be trained with word2vec, only with act2vec. Its training time is 1963.0836, which is considered to be quite long, and the accuracy, precision, recall, and F-score are not that satisfying either, around 43,39%.

Another aspect of the analysis is comparing LSTM-autoencoder with the other ML algorithms LOF, OSVM, and SVM. Based on the results, LSTM-autoencoder outperformed the OCC algorithms LOF, and OSVM. However, the results obtained from SVM were quite similar to the results from LSTM-autoencoder, and in some situations, SVM outperformed the LSTM-autoencoder. The question then is, why not choose SVM, since it is also a widely used technique for anomaly detection.

SVM is a supervised machine learning technique. The performance of SVM depends a lot on the parameters chosen, which means it requires a lot of effort in parameter tuning. Furthermore, being a supervised machine, it requires labeled data, and it cannot be used in situations when the labels are missing, or they are simply not available. It requires a lot of human interactions as well.

On the other side, LSTM-autoencoder is an unsupervised (or a self-supervised to be more precise) deep learning technique. Having labeled data is not a requirement, because the model is trained to learn patterns from the data, "produce" its labels, or learn high-level features. LSTM-autoencoder is trained on normal data only to learn patterns of normal traces. That means, it can perform even in cases when data is highly unbalanced, and the anomalous cases are not that many. Training an LSTM-autoencoder does not require human interaction because everything is done by the neural network that is in its architecture. Deep learning algorithms have their disadvantages of course. They require a very large amount of data to be trained, they are computationally expensive and timely, and they are like a black box that no one knows what is happening inside. However, even though SVM achieved great results in anomaly detection, LSTM is the best choice, especially when the labels are missing, and the number of anomalous cases is very small.

The last aspect of the analysis is the classification of anomalies, and the results

were unfortunately not that satisfying. The results represent not only an evaluation of classifying anomalies but also classifying traces into anomalous or not. If the activity of a given trace has a reconstruction error higher than the threshold, that makes the activity anomalous, which makes the trace itself anomalous. The model would tell us indeed which activity is anomalous, but it would still require human interaction to decide which kind of anomaly it is. However, it would be interesting to dive a bit deeper into this aspect to discover a higher-performance deep learning architecture that will classify the anomalous cases, and do so with very little human interaction.

Chapter 5

Conclusion

In this chapter, at first, a summary of this research will be given, be further continued with limitations in section 5.1 and future work in section 5.2.

A very large amount of data is stored in the information systems of many organizations and companies. This data is stored in a structure called an event log. Therefore, in this thesis, it was presented a model to detect anomalies in event logs using semantics and a deep learning algorithm. Three different groups of event logs were used for the analysis. The event logs from the two groups were generated synthetically, while the ones from the third group were real-world event logs.

To ensure, that the semantics was taken into consideration, the word2vec was used as a word embedding technique to create numeric vector representations that will be fed into LSTM-autoencoder. Furthermore, using this technique, there were obtained not only word embeddings, but also embeddings of activities called act2vec. By comparing word2vec and act2vec, it was shown that act2vec performed slightly better than word2vec, and the time of training the LSTM-autoencoder with the embeddings from act2vec was less than with word2vec.

LSTM-autoencoder was used as a deep learning technique. LSTM performs very well on sequential data while the autoencoder is a self-supervised learning technique that can reconstruct its output and produce its high-level features or its label. Therefore, the combination LSTM-autoencoder was chosen because it brings the advantages of both together. LSTM-autoencoder can detect anomalies even when no labels are given, no process model is provided for comparisons, and anomalous traces are present only in a small amount. This can be done even without human interaction.

The performance of the LSTM-autoencoder was compared to the other machine learning algorithms LOF, OSVM, and SVM. OSVM showed the worst results in detecting anomalies. The results from SVM were comparable with the results

from LSTM-autoencoder, and in some cases outperformed the LSTM-autoencoder. However, SVM requires labels, a lot of human interaction, and depends a lot on parameters.

Furthermore, the LSTM-autoencoder is used to classify anomalies. Sometimes, it is interesting and helpful to know not only whether a trace is anomalous, but which activity is causing the anomaly in the trace. Unfortunately, the obtained results were not quite satisfying and desirable. The trace representations that were fed into the LSTM-autoencoder were very long because of the padding technique. This caused an increase in the training time of the LSTM-autoencoder.

The major disadvantage of deep learning algorithms is that they require a very large amount of data to be trained on, a lot of computational resources, and a lot of time to be trained. They are like a black box that no one knows what is happening inside.

To conclude, deep learning techniques with word2vec as word embedding are the "right" choice to successfully detect anomalies in event logs. Even though, the event logs used for this analysis were from business processes, the model proposed in this thesis can be applied to event logs from other areas such as healthcare, etc.

5.1 Limitations

This research was subject to some limitations, even though the results showed that the combination of word2vec and LSTM-autoencoder is very promising in detecting anomalies using the semantics. Finding and generating the appropriate event logs was the most important limitation in this research. Two ways to generate event logs are:

- **PLG tools** generate random event logs with different complexity from various process models and simulates noise or anomalies in these event logs as well. The process model could be either created using this tool or imported as BPMN (Business process model and notation) or PNML (Petri Net Markup Language) file. The activities of the process models generated by the tool were labeled as "activity A", "activity B", "random activity 12", and so on. However, it is more interesting to have proper names for the activities. Another limitation of this tool was when trying to import a BPMN/PNML file of a process model that had "proper" names for its activities, and the tool failed to import the file in the correct format. Therefore, it was practically impossible to generate event logs with "proper" activity names.

- **Prom plugin** for event log generation was another way to create synthetic event logs and simulate noises in the event logs. However, while trying

to generate event logs, it was either impossible to have the process model uploaded in the correct format, or the time to generate the event logs would be so long that the Prom application would crash.

Another important part of the analysis would be to have labels in the event logs so that an evaluation of the model could be made. Therefore, the limitation was not only to generate/ find the event logs, but these event logs to have labels as well. Labels are required for both synthetic and real-world event logs, to evaluate the model.

Lastly, training a deep learning algorithm requires a very large dataset to perform at its best. However, this would need more powerful and expensive machines which lack during this research. Apart from being computationally costly, deep learning requires a lot of time to be trained which limits the number of experiments that can be made.

5.2 Future Work

This research showed that using semantics and deep learning is a promising approach to detecting anomalies and it can be further extended to many other directions.

In future work, more advanced tools or algorithms can be used to generate synthetic event logs.

Encoding strategy is not limited to only word2vec and act2vec. It would be interesting to incorporate other techniques such as Glove or BERT.

Papers from various authors have shown that some other variations of LSMT such as bidirectional LSTM, attention-based LSTM, and stacked LSTM achieved great results, which indicates another direction for future work.

In this research, it was taken into consideration only the control-flow perspective. However, for an organization or company is important to detect anomalies from a different perspective for example to find out that a certain activity has been performed by the wrong user. Therefore, multi-perspective anomaly detection is another interesting point of detecting anomalies.

The threshold for detecting anomalies was decided manually based on MAE. Generating this threshold automatically could provide better results.

Since the results of classifying anomalies were not that promising, it would be interesting to work on developing a model that not only classifies traces as anomalous or not but also classifies anomalies in different categories with as little human interaction as possible.

Bibliography

[1] BPI Challenge 2012. April 2012. Publisher: 4TU.ResearchData Type: dataset.

[2] BPI Challenge 2017. February 2017. Publisher: 4TU.ResearchData Type: dataset.

[3] Alessandro Berti, Sebastiaan J. van Zelst, and Wil van der Aalst. Process Mining for Python (PM4Py): Bridging the Gap Between Process- and Data Science, May 2019. Number: arXiv:1905.06169 arXiv:1905.06169 [cs].

[4] Kristof Böhmer and Stefanie Rinderle-Ma. Multi-perspective Anomaly Detection in Business Process Execution Events. In Christophe Debruyne, Hervé Panetto, Robert Meersman, Tharam Dillon, eva Kühn, Declan O'Sullivan, and Claudio Agostino Ardagna, editors, *On the Move to Meaningful Internet Systems: OTM 2016 Conferences*, pages 80–98, Cham, 2016. Springer International Publishing.

[5] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. LOF: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, SIGMOD '00, pages 93–104, New York, NY, USA, May 2000. Association for Computing Machinery.

[6] Andrea Burattin. PLG2: Multiperspective Processes Randomization and Simulation for Online and Offline Settings, July 2016. Number: arXiv:1506.08415 arXiv:1506.08415 [cs].

[7] Paolo Ceravolo, Ernesto Damiani, Mohammadsadegh Torabi, and Sylvio Barbon. Toward a New Generation of Log Pre-processing Methods for Process Mining. In Josep Carmona, Gregor Engels, and Akhil Kumar, editors, *Business Process Management Forum*, Lecture Notes in Business Information Processing, pages 55–70, Cham, 2017. Springer International Publishing.

[8] V. Chandola, A. Banerjee, and V. Kumar. Anomaly Detection for Discrete Sequences: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 24(5):823–839, May 2012.

[9] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, September 1995.

[10] Pieter De Koninck, Seppe vanden Broucke, and Jochen De Weerdt. act2vec, trace2vec, log2vec, and model2vec: Representation Learning for Business Processes. In *Business Process Management*, pages 305–321. Springer, Cham, 2018.

[11] Van Dongen and B.F. (Boudewijn). BPI Challenge 2015. May 2015. Publisher: 4TU.ResearchData.

[12] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, November 1997. Conference Name: Neural Computation.

[13] Siyu Huo, Hagen Völzer, Prabhat Reddy, Prerna Agarwal, Vatche Isahagian, and Vinod Muthusamy. Graph Autoencoders for Business Process Anomaly Detection. In Artem Polyvyanyy, Moe Thandar Wynn, Amy Van Looy, and Manfred Reichert, editors, *Business Process Management*, pages 417–433, Cham, 2021. Springer International Publishing.

[14] Sylvio Barbon Junior, Paolo Ceravolo, Ernesto Damiani, Nicolas Jashchenko Omori, and Gabriel Marques Tavares. Anomaly Detection on Event Logs with a Scarcity of Labels. In *2020 2nd International Conference on Process Mining (ICPM)*, pages 161–168, Padua, Italy, October 2020. IEEE.

[15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015. Number: 7553 Publisher: Nature Publishing Group.

[16] Hao Lu, Kaize Shi, Yifan Zhu, Yisheng Lv, and Zhendong Niu. Sensing Urban Transportation Events from Multi-Channel Social Signals with the Word2vec Fusion Model. *Sensors*, 18(12):4093, November 2018.

[17] Stefan Luettgen, Alexander Seeliger, Timo Nolle, and Max Mühlhäuser. Case2vec: Advances in Representation Learning for Business Processes. In *Process Mining Workshops*, pages 162–174. Springer, Cham, 2021.

[18] Timo Nolle, Stefan Luettgen, Alexander Seeliger, and Max Mühlhäuser. Analyzing business process anomalies using autoencoders. *Machine Learning*, 107(11):1875–1893, November 2018. Company: Springer Distributor: Springer Institution: Springer Label: Springer Number: 11 Publisher: Springer US.

[19] Timo Nolle, Stefan Luettgen, Alexander Seeliger, and Max Mühlhäuser. BINet: Multi-perspective Business Process Anomaly Classification. 2019. Publisher: arXiv Version Number: 1.

[20] Timo Nolle, Alexander Seeliger, and Max Mühlhäuser. Unsupervised Anomaly Detection in Noisy Business Process Event Logs Using Denoising Autoencoders. In Toon Calders, Michelangelo Ceci, and Donato Malerba, editors, *Discovery Science*, pages 442–456, Cham, 2016. Springer International Publishing.

[21] Timo Nolle, Alexander Seeliger, and Max Mühlhäuser. BINet: Multivariate Business Process Anomaly Detection Using Deep Learning. In *Business Process Management*, pages 271–287. Springer, Cham, 2018.

[22] Timo Nolle, Alexander Seeliger, Nils Thoma, and Max Mühlhäuser. DeepAlign: Alignment-Based Process Anomaly Correction Using Recurrent Neural Networks. In Schahram Dustdar, Eric Yu, Camille Salinesi, Dominique Rieu, and Vik Pant, editors, *Advanced Information Systems Engineering*, pages 319–333, Cham, 2020. Springer International Publishing.

[23] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, 2014. Association for Computational Linguistics.

[24] Alexander Seeliger, Stefan Luettgen, Timo Nolle, and Max Mühlhäuser. Learning of Process Representations Using Recurrent Neural Networks. In Marcello La Rosa, Shazia Sadiq, and Ernest Teniente, editors, *Advanced Information Systems Engineering*, volume 12751, pages 109–124. Springer International Publishing, Cham, 2021. Series Title: Lecture Notes in Computer Science.

[25] Ward Steeman. BPI Challenge 2013, closed problems, April 2013. Medium: media types: application/x-gzip, text/xml Version Number: 1 Type: dataset.

[26] Ward Steeman. BPI Challenge 2013, incidents, April 2013. Medium: media types: application/x-gzip, text/xml Version Number: 1 Type: dataset.

[27] Ward Steeman. BPI Challenge 2013, open problems, April 2013. Medium: media types: application/x-gzip, text/xml Version Number: 1 Type: dataset.

[28] Gabriel Marques Tavares and Sylvio Barbon. Analysis of Language Inspired Trace Representation for Anomaly Detection. In *ADBIS, TPDL and EDA 2020 Common Workshops and Doctoral Consortium*, pages 296–308. Springer, Cham, 2020.

[29] Gabriel Marques Tavares, Paolo Ceravolo, Victor G. Turrisi Da Costa, Ernesto Damiani, and Sylvio Barbon Junior. Overlapping Analytic Stages in Online Process Mining. In *2019 IEEE International Conference on Services Computing (SCC)*, pages 167–175, July 2019. ISSN: 2474-2473.

[30] David M.J. Tax and Robert P.W. Duin. Support Vector Data Description. *Machine Learning*, 54(1):45–66, January 2004.

[31] Niek Tax, Sebastiaan J. van Zelst, and Irene Teinemaa. An Experimental Evaluation of the Generalizing Capabilities of Process Discovery Techniques and Black-Box Sequence Models. In Jens Gulden, Iris Reinhartz-Berger, Rainer Schmidt, Sérgio Guerreiro, Wided Guédria, and Palash Bera, editors, *Enterprise, Business-Process and Information Systems Modeling*, volume 318, pages 165–180. Springer International Publishing, Cham, 2018. Series Title: Lecture Notes in Business Information Processing.

[32] Alaa Tharwat. Classification assessment methods. *Applied Computing and Informatics*, 17(1):168–192, January 2020. Publisher: Emerald Publishing Limited.

[33] Han van der Aa, Adrian Rebmann, and Henrik Leopold. Natural language-based detection of semantic execution anomalies in event logs. *Information Systems*, 102:101824, December 2021.

[34] W. M. P. van der Aalst, V. Rubin, H. M. W. Verbeek, B. F. van Dongen, E. Kindler, and C. W. Günther. Process mining: a two-step approach to balance between underfitting and overfitting. *Software & Systems Modeling*, 9(1):87, November 2008.

[35] Wil van der Aalst. Data Science in Action. In Wil van der Aalst, editor, *Process Mining: Data Science in Action*, pages 3–23. Springer, Berlin, Heidelberg, 2016.

[36] W.M.P. van der Aalst and A.K.A. de Medeiros. Process Mining and Security: Detecting Anomalous Process Executions and Checking Process Conformance. *Electronic Notes in Theoretical Computer Science*, 121:3–21, February 2005.

[37] Mengying Wang, Lele Xu, and Lili Guo. Anomaly Detection of System Logs Based on Natural Language Processing and Deep Learning. In *2018 4th International Conference on Frontiers of Signal Processing (ICFSP)*, pages 140–144, September 2018.

[38] Mathias Weske, Gero Decker, Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo A. Reijers. Model Collection of the Business Process Management Academic Initiative, April 2020. Type: dataset.

Appendix A

Program Code / Resources

The code for pre-processing steps

```
#Cleaning the text
import re
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer

ps = PorterStemmer()
corpus = []
for i in range(len(activities)):
pre = re.sub('[^a-zA-Z]', ' ', activities[i])
pre = pre.lower()
pre = pre.split()
pre = [ps.stem(word) for word in pre if not word in
set(stopwords.words('english'))]
pre = ' '.join(pre)
corpus.append(pre)
```

The code to tokenize the activities in the event log

```
vocab_words = []
for i in range(len(corpus)):
all_words = nltk.word_tokenize(corpus[i])
vocab_words.append(all_words)
print(all_words)
```

The code to train the word2vec model

```
import gensim
from gensim.models import Word2Vec

model = gensim.models.Word2Vec(vocab_words, vector_size=100,
window=3, min_count=0, sg=1)
model.train(vocab_words,total_examples=model.corpus_count,
start_alpha=0.025, epochs=10)
model.alpha -= 0.002  # decrease the learning rate
model.min_alpha = model.alpha  # fix the learning rate, no decay
```

The code to define the LSTM-autoencoder

```
import keras

# define model
model = keras.Sequential()
#encoder
model.add(keras.layers.LSTM(128, activation='relu',
input_shape=(Normal_train_3d.shape[1],
Normal_train_3d.shape[2]),
return_sequences=True))
model.add(keras.layers.Dropout(rate=0.1))
model.add(keras.layers.LSTM(64, activation='relu',
return_sequences=False))
model.add(keras.layers.Dropout(rate=0.1))
model.add(keras.layers.RepeatVector(1))
#decoder
model.add(keras.layers.LSTM(64, activation='relu',
return_sequences=True))
model.add(keras.layers.Dropout(rate=0.1))
model.add(keras.layers.LSTM(128, activation='relu',
 return_sequences=True))
model.add(keras.layers.Dropout(rate=0.1))
model.add(keras.layers.TimeDistributed(Dense(Normal_train_3d.shape[2])))
model.compile(optimizer='adam', loss='mae',metrics=['accuracy'])
model.summary()
```

The code to train the LSTM-autoencoder

```
#Notice that the LSTM is trained using only the normal traces, but
 is evaluated using the full test set.
history = model.fit(Normal_train_3d, Normal_train_3d,
epochs=100,
batch_size=32,
validation_data=(test_data_3d, test_data_3d),
shuffle=True)
```

Text of Affidavit

I hereby guarantee that this thesis was written by my own and that I received no further aid from third parties. Furthermore, I declare that neither this work nor parts of it have been submitted by myself or others as proof of performance yet. All intellectual property of others is clearly cited as such. All secondary literature and other sources are certified and listed in the bibliography. The same applies for graphic illustrations, pictures and all internet sources. I agree that my work may be electronically saved and sent anonymized to be checked for plagiarism. I am aware that this thesis cannot be graded if this declaration is not signed.

Mannheim, 04.07.2022 Signiture