CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG

Detection of software incidents from large log material with the use of unsupervised machine learning

Master's thesis in Applied Data Science

DIMITRIOS ANASTASIADIS

JAKUB LENART

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2022

Master's thesis 2022

Detection of software incidents from large log material with the use of unsupervised machine learning

DIMITRIOS ANASTASIADIS
JAKUB LENART

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2022

DIMITRIOS ANASTASIADIS
JAKUB LENART

Supervisor: Niklas Zechner, Department of Swedish, multilingualism, language technology, University of Gothenburg
Advisor: Daniel Dalevi, Centiro Solutions AB
Examiner: Richard Johansson, Department of Computer Science and Engineering, Chalmers University of Technology and University of Gothenburg

Master's Thesis 2022
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in $\LaTeX$
Gothenburg, Sweden 2022

Detection of software incidents from large log material with the use of unsupervised machine learning
DIMITRIOS ANASTASIADIS
JAKUB LENART
Department of Computer Science and Engineering
University of Gothenburg, Chalmers University of Technology

Abstract

Computer systems generate log files, which contain information on the various operations performed by these systems. This information can support the process of error/failure detection and debugging. Therefore, anomalies can be spotted in the system through its produced log material. The task of anomaly detection can be treated as a binary classification of log files, with the two classes being anomalous and non anomalous. Due to the sheer volume of data and the complexity of the task, it is not possible for it to be performed manually by humans, thus creating the need for automation. Centiro, a Swedish software company, has decided to follow a machine learning approach for automating the task of software incident detection. In this thesis, we apply four machine learning models in order to detect anomalies. These are namely the Local Outlier Factor (LOF), the Isolation Forest (IF), the Principal Component Analysis (PCA) and the LSTM-Autoencoder. We make use of four publicly available datasets as well as a dataset gathered from the produced logs of the computer systems of the company. Preprocessing of the data and selection of the appropriate features are two tasks that needed to be carefully performed for the successful implementation of the models. Precision, Recall and F-Score were used as evaluation metrics to measure the performance of the models on the different datasets. The model with the best and most stable overall performance on the publicly available datasets is the LSTM-Autoencoder, therefore we decided to apply it on the data of the company in order to detect any possible software incidents.

Keywords: binary classification, log, anomaly detection, machine learning, Local Outlier Factor, Isolation Forest, PCA, LSTM-Autoencoder.

Acknowledgements

We would like to take this opportunity to express our gratitude to our academic supervisor Niklas Zechner at the Department of Swedish, Multilingualism, Language Technology for his guidance and valuable inputs throughout this thesis project. Also, we would like to thank Centiro for providing us with the opportunity as well as the resources to work on this thesis. Specifically, we want to thank our supervisor Daniel Dalevi at Centiro who was responsible for initiating this thesis project with us and Viktor Ingemarsson for his important contribution and support throughout our project. We would also like to express our appreciation to our examiner Richard Johansson at the Department of Computer Science and Engineering for his time, feedback and input on the thesis. Finally, special thanks to our families that supported us throughout our studies.

Dimitrios Anastasiadis & Jakub Lenart, Gothenburg, June 2022

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

AE	Autoencoder
CCFD	Credit Card Fraud Detection
DAGMM	Deep Autoencoding Gaussian Mixture Model
IIS	Internet Information Services
IF	Isolation Forest
LOF	Local Outlier Factor
LSTM	Long Short-Term Memory
LSTM-VAE	Long Short-Term Memory Variational Autoencoder
FN	False Negative
FP	False Positive
PCA	Principal Component Analysis
RNN	Recurrent Neural Network
SKAB	Skoltech Anomaly Benchmark
SMD	Server Machine Dataset
SVM	Support Vector Machine
SWaT	Secure Water Treatment
TP	True Positive

Contents

List of Acronyms **ix**

List of Figures **xiii**

List of Tables **xv**

1 Introduction **1**

1.1 Problem Statement 1

1.2 Research Questions & Novelty of the Project 2

1.3 Limitations 3

2 Background **5**

2.1 Machine Learning 5

2.1.1 Supervised Machine Learning 5

2.1.2 Unsupervised Machine Learning 6

2.1.3 Semi-supervised Machine Learning 6

2.2 Log files 6

2.3 Anomaly Detection 7

2.4 Classification Models 8

2.4.1 Local Outlier Factor 8

2.4.2 Isolation Forest 10

2.4.3 Principal Component Analysis 11

2.4.4 Autoencoder 12

2.4.5 Long Short-Term Memory 12

2.5 Performance Evaluation Metrics 14

2.5.1 Precision 15

2.5.2 Recall 15

2.5.3 F-Score 15

2.6 Related Work 15

2.6.1 Unsupervised cluster evolution approach 15

2.6.2 Real-time anomaly detection with unsupervised methods . . . 16

2.6.3 Anomaly Detection in Access Logs 16

2.6.4 System Log Analysis for Anomaly Detection 17

2.6.5 USAD: Unsupervised Anomaly Detection on Multivariate Time Series 17

3 Methods **19**

3.1 Dataset 19
3.1.1 Features of the dataset of the company 20
3.2 Preprocessing 22
3.2.1 Data scaling 22
3.2.2 Anomaly rate adjustment 23
3.2.3 One-hot encoding 23
3.2.4 Data Windows 23
3.3 Model Implementation 25
3.3.1 Train-Test Ratio 25
3.3.2 Local Outlier Factor 25
3.3.3 Isolation Forest 26
3.3.4 PCA 27
3.3.5 LSTM-Autoencoder 29
3.4 Evaluation Method 31
3.5 Used Software and Hardware 33

4 Results **35**
4.1 SWaT dataset 40
4.2 SKAB dataset 41
4.3 SMD dataset 42
4.4 CCFD dataset 43
4.5 Consistency check 43
4.6 Additional Precision-Recall Curve test 44
4.7 Dataset of Centiro 45

5 Discussion **49**
5.1 Approaches & Results 49
5.2 Future Work 51

6 Conclusion **53**

Bibliography **55**

List of Figures

2.1 Screenshot of Graylog's function. 7
2.2 Example of anomaly detection with one feature. 8
2.3 Anomaly Detection with the use of Local Outlier Factor. 10
2.4 Anomaly Detection with the use of Isolation Forest. 11
2.5 Dimensionality Reduction with the use of PCA. 12
2.6 Autoencoder architecture. 13
2.7 LSTM architecture. [29] 13

3.1 Example of One-Hot Encoding. 24
3.2 Example of choosing the number of principal components based on the percentage of cumulative variance. 28
3.3 Example of loss distribution plot for choosing the classification threshold. 30
3.4 Division of the test set into 10 subsets and iterative evaluation on each one of them. 32

4.1 Performance results on the SWaT dataset. 40
4.2 Performance results on the SKAB dataset. 41
4.3 Performance results on the SMD dataset. 42
4.4 Performance results on the CCFD dataset. 43
4.5 Performing consistency check on the AE model. 44
4.6 Precision-Recall Curve test on the standardized SWaT dataset, without the use of time windows. 45
4.7 Precision-Recall Curve test on the standardized SWaT dataset, with the use of time windows. 46
4.8 AE output on 5 seconds time windows. 47
4.9 AE output on 30 seconds time windows 48
4.10 AE output on 1 minute time windows. 48

List of Tables

3.1 *sc-status* codes and their corresponding messages 21

4.1 Performance results on SWaT dataset. 36
4.2 Performance results on SKAB dataset. 37
4.3 Performance results on SMD dataset. 38
4.4 Performance results on CCFD dataset. 39
4.5 Performance results on the dataset of the company. 47

1
Introduction

Anomaly detection is a task that has been researched and investigated thoroughly over the past few years, since both the industry and the academia have valued its increasingly high importance. This is because spotting anomalies on time can help in the mitigation or even prevention of major damage or dysfunction in the system. Therefore, several ways and methods have recently been applied by various companies and research teams in order to detect anomalies that occur in computer systems in real time [1]. The need for automation of this process is certain, as the sheer amount of data produced and managed every day by the computer systems of the companies is not amenable to manual manipulation performed by humans, due to time consumption and complexity of the task.

Centiro, a software company based in Borås, Sweden, provided us with log data in order for us to detect possible anomalies in their computer systems in the framework of this thesis project. Logs that are produced by the computer systems contain useful runtime information that can be analysed in order to extract relevant knowledge about any possible anomalies that might have occurred over a certain period of time [2]. For this reason, Centiro has taken a machine learning approach to gain insights and detect anomalies that can be spotted in the log material of the system.

Selecting and preprocessing the appropriate log files are two crucial tasks for the correct training and testing of the machine learning models that are used for anomaly detection. Logs from different applications might be able to provide us with different knowledge regarding the detection of software incidents. Furthermore, the selective use of different features of the logs might increase or decrease the performance of the machine learning models, as the features are not equally correlated to the fact of a log being anomalous or not. The selection of machine learning techniques is also of high importance, as some models yield in a better performance than others, depending on the dataset and the selected features that are used [1]. Therefore, in the framework of this thesis project, we investigate various alternatives in order to find an appropriate and efficient way of detecting anomalies from large log material.

1.1 Problem Statement

Anomalies may occur in several stages of an operation that is running in a computer system. The information that can be found in the log files of a computer system is able to provide us with insights about the root cause of the problem, thus helping us get one step closer to being able to detect a software incident on time or even prevent it from happening in the first place. When it comes to software companies like Centiro, being able to detect anomalies in real time is important for the correct function of the system and the successful operation of the company.

The main aim of this thesis project is the research and development of a model that will be able to spot the potential anomalies by making use of Centiro's produced log material. In order to achieve that, we initially use publicly available datasets by applying four different machine learning models on them, and then we apply the best performing one on the dataset of the company. This is a contribution in the field of Machine Learning with the use of four different approaches. The idea behind the development of such a system would be to provide an accurate detection of the anomalies of the system so that the company could have a clear insight and try to fix or avoid system disruption or any damage that could be caused in the future. This task requires effort in analysing the log files and extracting information that might be useful for the process of spotting anomalies. This effort is focused mainly on collecting the appropriate dataset, preprocessing the relevant log information, developing the models for the task, applying them and finally measuring and evaluating their performance.

1.2 Research Questions & Novelty of the Project

This thesis addresses the question of whether we can detect software incidents based on the content of Internet Information Services (IIS) log files using techniques of unsupervised machine learning. This question is answered by performing tests on the log data of Centiro.

Another research contribution of this project is the comparison among four algorithms on the task of anomaly detection on a combination of five datasets that have different features and have been preprocessed in four different ways. Four of these datasets are publicly available and the other one is the dataset of the company. The four models are the Local Outlier Factor (LOF), the Isolation Forest (IF), the Principal Component Analysis (PCA) and the LSTM-Autoencoder (LSTM-AE). From that comparison, we can have a clear overview of the model with the most universally stable performance that can work decently for a variety of datasets.

Additionally, by performing the mentioned tests, we also investigate the change in the performance of the models while the amount of anomalies included in the dataset changes.

Furthermore, another element of novelty of this thesis is the investigation of the usability of the feature *sc-status* in anomaly detection. This feature is present in all IIS logs and we investigate whether it could lead us to detect anomalies that are triggered by specific applications in the system. The *sc-status* is a protocol status code further explained later.

Lastly, a more general yet interesting question that this project deals with is how machine learning models are utilized in order to detect software incidents. This question is largely addressed based on a literature review and on our own implementation of machine learning models on publicly available datasets as well as the dataset of Centiro.

1.3 Limitations

This thesis work is limited as far as time and resources are concerned. The amount of log data produced every day by the software of the company is extremely large and therefore, it is computationally impossible to deal with all of it using only the available hardware resources. As a result, we deal with a subset of the data. This may lead to possibly lower performance results than if we took all the data into account. Also, specific features of the data were selected experimentally and based on a literature review. The logs of the company are not labeled and the process of labeling them would be labour intensive and would require a lot of time. For that reason, we were unable to obtain a sufficient labeled dataset from the company. Therefore, we used four publicly available labeled datasets that are suitable for anomaly detection tasks. We do this in order to observe and evaluate the performance of the models. Then, we apply the best performing model on the unlabeled dataset of the company. All four datasets come from different sources and differ in terms of dimensionality and anomaly rate, however, they do share some characteristics that allow us to compare them. Each of the four datasets is a multivariate time series dataset with only numerical values, containing outliers defined as anomalies. The diversity between these datasets can make the comparison more interesting as well as universal.

Furthermore, we do not know what anomaly rate can be expected in the data of the company. This depends, among others, on how we divide the data into time windows, which will be further explained in the following sections. The log dataset is highly unbalanced, in the sense that the number of anomalous logs is very small compared to non-anomalous ones. Nevertheless, since we are only considering a subset of the full data, the anomaly rate may be larger. We evaluate the performance of the four models on the publicly available datasets in order to select the model that will be used for detecting anomalies in the dataset of the company. The purpose of using four datasets is to demonstrate empirically that the chosen model works well on different types of datasets. Therefore, it is reasonable to apply this model to the final dataset of the company, which we are unable to evaluate without the assistance of company experts.

2
Background

The task of unsupervised anomaly detection can be performed in several ways. In this chapter, we present the theory behind it as well as the proposed methods. General concepts of machine learning and its categories are presented in Section 2.1. In Section 2.2, the task of anomaly detection as well as its basic features and the definition of anomaly are described. Next, in Section 2.3 lies the description of log files and what they are used for in general as well as more specifically in the case of the company of Centiro. A general description of the algorithms and models that are used can be found in Section 2.4 and in Section 2.5 their evaluation metrics and ways. Finally, in Section 2.6, some related work is described.

2.1 Machine Learning

Machine Learning is a subfield of Computer Science that is highly associated to Artificial Intelligence (AI) and it is increasingly used in various tasks over the past years. From social media services and language translation to Medicine and Life Sciences, machine learning is involved in numerous applications over a wide range of fields [3]. Consequently, it is applied in several methods and solutions in the field of anomaly detection, which is the main research object of the present paper.

Machine learning includes algorithms and statistical models that can be used in order to perform certain tasks without having to be explicitly programmed [4]. Prediction and early detection of incidents (e.g. health disorders) can be achieved through machine learning models that are trained on relevant available data. Machine learning can be divided into three main categories; Supervised, Unsupervised and Semi-Supervised. This division is based on the presence or absence of data that is tagged with one or more labels (labeled data). Unsupervised machine learning is used for the purposes of the present project.

2.1.1 Supervised Machine Learning

Supervised machine learning requires a training dataset that has been labeled [5]. This means that in order to perform supervised learning, a data sample with correct classification labels is needed [6]. An observation regarding the anomaly detection task with the use of supervised machine learning is that the anomaly class is usually much smaller than the non-anomaly one and therefore the dataset is highly unbalanced between the two classes. The general concept of supervised machine learning includes the application of past knowledge gained from previous labeled data to new data [5].

2.1.2 Unsupervised Machine Learning

In contrast with supervised machine learning, performing unsupervised machine learning does not require the use of a labeled dataset. When it comes to the task of anomaly detection, we can assume that normal events occur much more frequently than anomalous ones. In any other case we risk the occurrence of numerous false alarms (false positives) [7]. A common approach of unsupervised machine learning is clustering and there are many unsupervised models that follow this technique. An example case of clustering can be about dividing the data points into several clusters according to their distance from the centers of the clusters [5]. There are more methods and approaches of unsupervised machine learning and later on we analyze the ones used for anomaly detection in the framework of the present thesis.

2.1.3 Semi-supervised Machine Learning

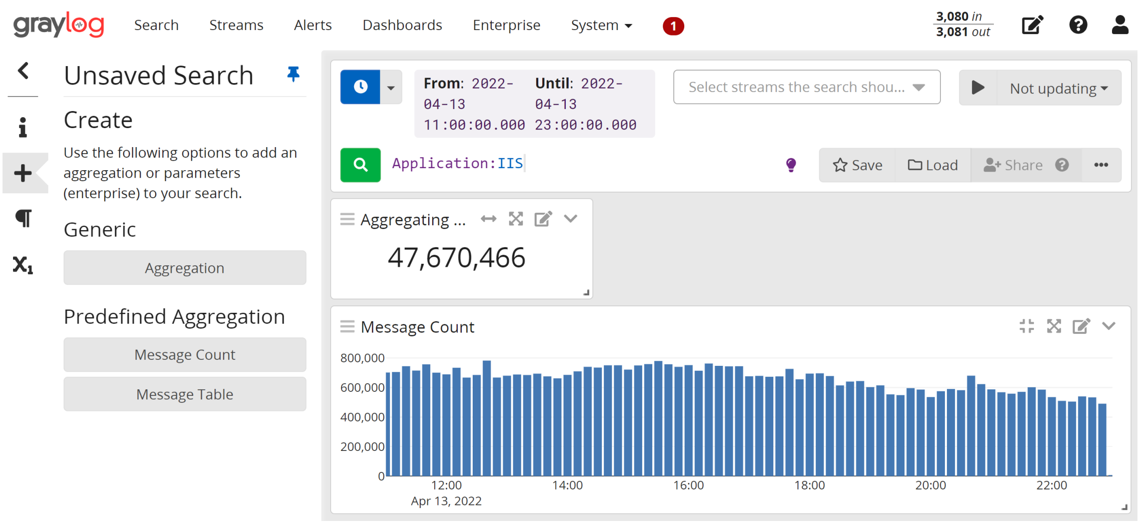
Semi-supervised machine learning combines both supervised and unsupervised learning. That is because small amounts of labeled data might be used in semi-supervised learning, but generally speaking, most of the data is unlabeled. Semi-supervised learning was introduced in order to overcome the disadvantages of both supervised and unsupervised machine learning. The drawback of supervised learning is that it requires large amounts of labeled training data in order to accurately classify the test data and the drawback of unsupervised learning is that clustering unknown data can be sometimes relatively inaccurate. This is why semi-supervised is used in some cases where it is reasonable for the models to learn based on a small amount of labeled training data [8].

2.2 Log files

Generally speaking, logs are generated records that derive from a number of sources and keep track of the history of the tasks that have been performed. Software developers often use the log files in the debugging/troubleshooting process of applications. In case of a system of small scale, it is relatively easy for a human to read the produced logs and understand if the behavior of the system is the expected one. However, the system of a company might be of a larger scale and therefore the size of the log material can make it impossible for one to read and process in order to figure out if the system behaves as it is supposed to. Apart from that, the complexity of the contents of the log files might be increased when dealing with a computer system of a big company. Thus, the need for automation of this process has grown bigger over the past few years, leading to innovative ways of log data manipulation that often include the use of machine learning. Some preprocessing and transformation of the data might be necessary in most cases in order for the computer to be able to process it [9].

The most typical form of logs is as timestamp based data that contain information about the various processes that took place on specific instances [5]. They usually include both messages and numerical values in the different fields of information that they contain.

As far as the specific logs of Centiro are concerned, their structure may vary from one application to another. The log files that are used in the framework of the present thesis project derive from the Internet Information Services (IIS). This is a modular network server application from Microsoft. The IIS web server runs on Windows systems and it provides a platform that hosts and manages web applications and serves the requests of HTML pages [10]. The log files that we use contain various fields of information, such as timestamp, source code, level, time-taken and IP addresses of the client and the server. The platform used for accessing and quering in Centiro's log file database is Graylog. This platform is used to store and perform real-time search and analysis in large amounts of log material produced every day by the company's computer systems [11]. Graylog makes use of a three-tier architecture and a scalable storage that has its basis on Elasticsearch and MongoDB [12].



From: 2022-04-13 11:00:00.000 Until: 2022-04-13 23:00:00.000

Application:IIS

47,670,466

Figure 2.1: Screenshot of Graylog's function.

2.3 Anomaly Detection

Generally speaking, anomaly is called an abnormal instance that indicates any behavior different than the expected one [5]. The importance of anomaly detection lies in the fact that by predicting or spotting anomalies, various errors on applications can be fixed in short periods of time, thus helping the mitigation of the total damage that could be dealt on the system's performance.

More specifically in the framework of the present thesis project, an anomaly in a computer system is an instance that does not follow the normally expected patterns. Also, we can assume that anomalies occur in a very small percentage of the total instances. They can be spotted on logs produced from different applications and they are usually connected to operations that have been executed in a way different than what they were expected. Anomalies might be of various levels of importance, according to which they can cause various degrees of damage in the system and affect its interaction with humans.

Anomaly detection can take place based on different machine learning techniques; supervised, unsupervised and semi-supervised learning. In this project, unsupervised learning techniques are being utilized and they are described in later sections. The following figure shows a simple example of anomaly detection based on the value of one feature plotted against time.

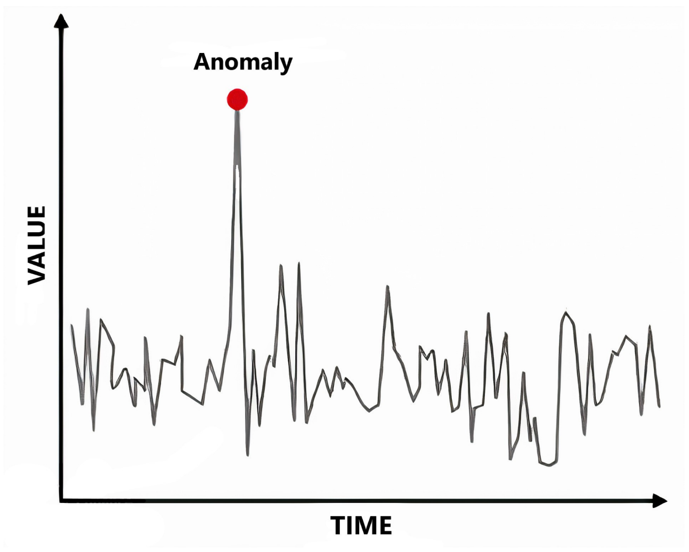


Figure 2.2: Example of anomaly detection with one feature.

2.4 Classification Models

Four unsupervised machine learning models were used in this thesis. These are the Local Outlier Factor(LOF), the Isolation Forest (IF), the Principal Component Analysis (PCA) and the Long Short-Term Memory Autoencoder (LSTM AE). These models follow different methods of classifying rows of the dataframes, which makes some of them more suitable for training and testing on specific datasets than others. The task is binary classification as one row (or a group of rows) of the dataset can be classified either as anomalous or non-anomalous.

2.4.1 Local Outlier Factor

Local Outlier Factor works as a density-based outlier detection algorithm which is able to spot outliers (anomalies) by calculating the local deviation of a given data point. This can be used among others, for outlier detection in unbalanced datasets. Determining whether a data point is an anomaly depends on the density between the data point and its neighbors. In particular, the lower the density, the higher the chance for the data point to be classified as anomaly [13]. The key-definitions of the LOF algorithm are namely the k-distance of a data point p, the k-nearest neighbors of p, the reachability distance of p with respect to o, the local reachability density of p and the LOF score of p [14].

In order to define the k-distance, we need to explain how to distance between two data points p and o is calculated. This distance is calculated by using a Euclidean n-dimensional space with the following formula [14]:

$$d(p, o) = \sqrt{\sum_{i=1}^{n} (p_i - o_i)^2} \tag{2.1}$$

Let the k-distance(p) be the distance of a data point p to the k-th nearest neighbor, where k is a positive integer. The k-nearest neighbors of p are all the data points up to this k-distance(p) [5].

As for the reachability distance of p with respect to o, it is defined with the following formula [14]:

$$reachdist_k(p, o) = \max\left(kdistance(o), d(p, o)\right) \tag{2.2}$$

Regarding the local rechability density (LRD) of a data point p, we define m as the minimum number of data points and we calculate the LRD of a data point as follows [14]:

$$LRD_m(p) = 1/\left(\frac{\sum_{o \in N_m(p)} reachdist_m(p, o)}{|m(p)|}\right) \tag{2.3}$$

In the above equation, we make use of the average reachability distance based on the m number of nearest neighbors of the data point p.
Taking all the above into account, the LOF score is calculated as follows [14]:

$$LOF_m(p) = \frac{\sum_{n \in N_m(p)} \frac{LRD_m(o)}{LRD_m(p)}}{|m(p)|} \tag{2.4}$$

If we get a LOF score that has a value of approximately 1, this is an indication that the data point is comparable to its neighbors and therefore it is not considered as an outlier. If the value is below 1, this is an indication that the data point is an inlier. On the other case where the values are larger than 1, we have an indication of outlier (anomaly) [15].

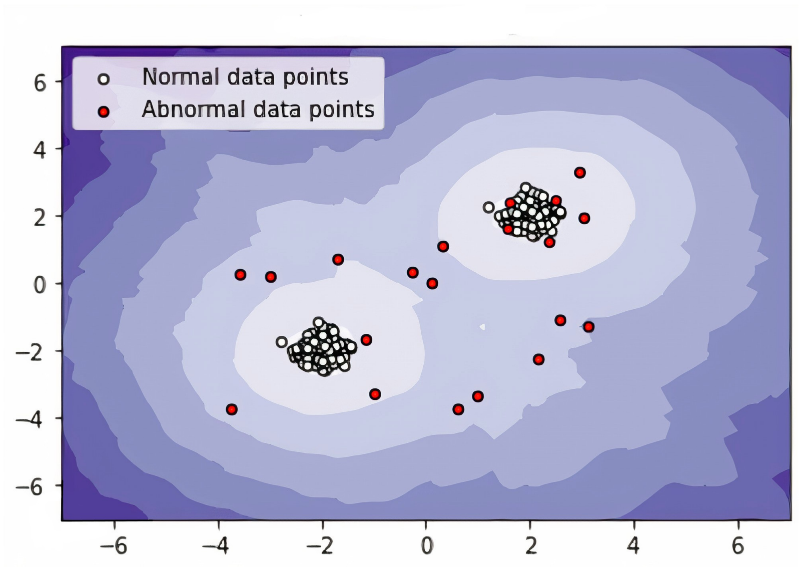


Figure 2.3: Anomaly Detection with the use of Local Outlier Factor.

2.4.2 Isolation Forest

We use the term "isolation" here to describe the process of separation of a (potentially anomalous) data point from the rest of the data points. Anomalies form generally a small percentage of the total samples of a dataset and they have some differences, therefore we are able to separate them from the rest of the normal data points [16].

When detecting anomalies with an Isolation Forest, the data is sub-sampled and processed within some tree structures (isolation trees). These trees perform random cuts in the values of the features of the data points. These features are selected randomly. The data points that go through the branches of the isolation trees and end up deeper into the trees have a smaller chance of being anomalous. Data points that travel in shorter branches of the trees are more likely to be anomalous [17]. This happens because anomalies will have shorter paths through this random partitioning as they have more distinguishable values of their features and therefore they are more likely to be isolated earlier than the rest of the data points [16]. Therefore, when an isolation forest produces short paths for isolating some specific data points, then these data points have high chances of being anomalies and thus, they are classified as such. We can see that the aggregated length of the tree branches gives us the so-called “anomaly score” of a given data point which serves as a measure of anomaly for this data point [17].

Concluding, in the process of anomaly detection with the use of Isolation Forest, at first isolation trees are being built and then the data points pass through the isolation trees that have been formed and an anomaly score for each data point is obtained [16]. This anomaly score is used to classify whether each data point is an anomaly or not.

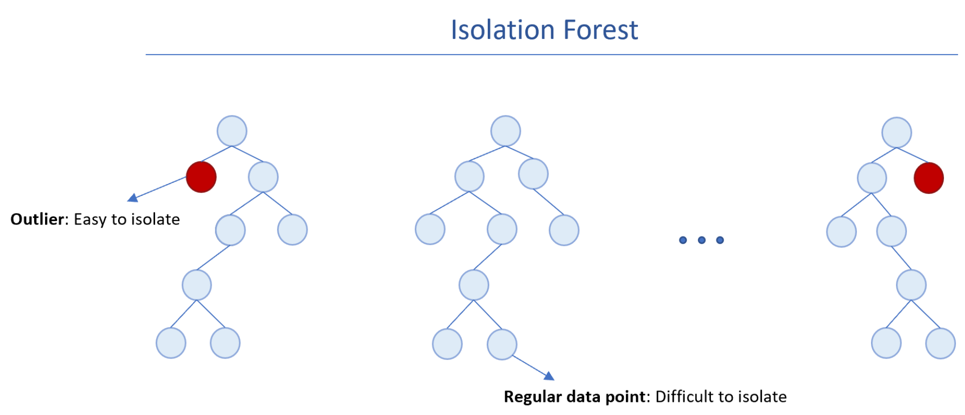


Figure 2.4: Anomaly Detection with the use of Isolation Forest.

2.4.3 Principal Component Analysis

Principal Component Analysis (PCA) is a method that can be used for dimensionality reduction. Given a dataset, PCA is based on determining the principal directions of the data distribution. The construction of the data covariance matrix as well as the calculation of the dominant eigenvectors are needed in order to determine the principal directions. The eigenvectors are considered as the principal directions, as they are the ones holding the most important information among all the vectors [18].

In other words, PCA is a process, through which the principal components of a collection of data points are computed. These principal components are a sequence of p unit vectors. The i-th vector is orthogonal to the first i-1 vectors and it is the direction of a line that is the best fit for the data. Considering the average squared distance from the data points to the line, we consider as the best fit the line that minimizes this distance. PCA uses the computed principal components in order to change the basis on the data [19]. It is able to reduce the big amount of correlated features to a much smaller amount of uncorrelated principal components [20]. Based on a literature review, there are certain cases where PCA takes into account just the first few principal components and ignores the rest of them [19].

However, in this thesis project, PCA is used for anomaly detection. Therefore, it is enhanced with a function that computes the anomaly scores of the data points of a given dataset. Based on [21], this algorithm reduces the dimensionality of the data and at the same time it tries to minimize the reconstruction error. Therefore, it attempts to capture the most valuable information of the original features in order to be able to reconstruct the original features from the reduced features as accurately as possible. However, PCA is not able to capture all the information contained in the original features as they move to a lower dimensional space. Thus, some error is observed in the reconstruction process and this error is called reconstruction error. The data points that occur the least often are very likely to yield the largest reconstruction error and to be considered as anomalies. The reconstruction error of each data point is computed by summing the squared differences between the original feature matrix and the reconstructed matrix. We also refer to the reconstruction error of a data point as its anomaly score.

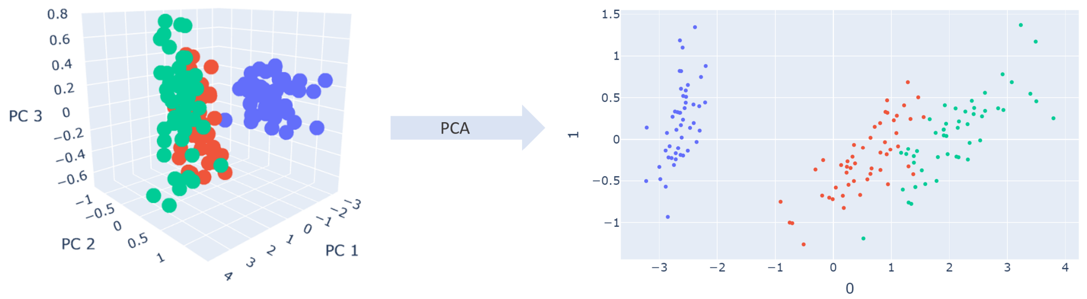


Figure 2.5: Dimensionality Reduction with the use of PCA.

2.4.4 Autoencoder

Autoencoder (AE) is a machine learning model working in an unsupervised manner which is a feed-forward neural network with an encoder-decoder structure [23]. The main objective of the AE is to train the model to replicate input vectors {x(1), x(2), ..., x(m)} as output vectors $\{\widehat{x}(1), \widehat{x}(2), \ldots, \widehat{x}(m)\}$ while minimizing the reconstruction error, which is determined by the difference between the input and output data [24]. The Autoencoder model consists mainly of two phases. The first is an encoding step. This phase is in charge of mapping the input data to the model's hidden layer. During this process, the model reduces the dimensionality of the input data, resulting in a latent representation of the data. The second phase of the model is decoding whose objective is to decode the mapping from the hidden layer to the output layer. Moreover, the model increases the dimensions of the transformed data to its original size during the decoding phase [25]. The following formula can be used to explain this procedure:

$$\widetilde{X} = D(E(X)) \tag{2.5}$$

Where X is the input data, E denotes an encoding step, D denotes the above-described decoding step, and $\widetilde{X}$ indicates the model's output. The model's overall goal is to train E and D to minimize the deviation between X and $\widetilde{X}$ [25]. An Autoencoder model, in particular, might be considered as a solution to the following optimisation problem:

$$\min_{DE} || \ X - D(E(X)) \ || \tag{2.6}$$

In Figure 2.1, the architecture of the AE model is presented. According to [25], an Autoencoder that consists of more than one hidden layer can be also called a deep Autoencoder. The AE that is implemented in this project consists of hidden layers that are built based on Long Short-Term Memory architecture, which is further explained in the next section.

2.4.5 Long Short-Term Memory

Long Short-Term Memory, or LSTM for short, is a type of recurrent neural network (RNN) that was developed in order to address the problem of long-term dependency

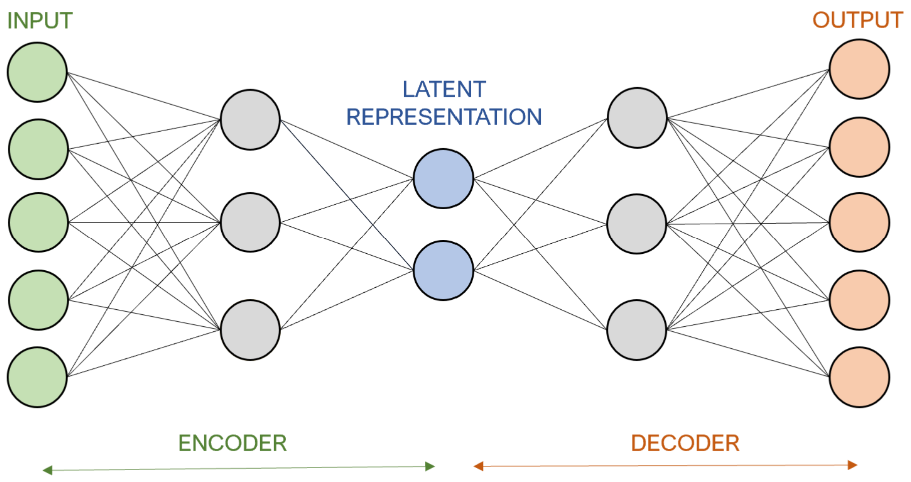


Figure 2.6: Autoencoder architecture.

caused by vanishing gradients. The above-mentioned issue was encountered during the process of training RNN [26]. While the model is being trained, the weights of the network are being updated. In the situation of a vanishing gradient, the gradient decreases gradually, until it eventually becomes vanishingly small, preventing the weights from changing their values and so, stopping the neural network from further training [27]. LSTM was a solution to this issue and its main objective was to control the information flow within the neurons of the network. Long Short-Term Memory introduces a gating mechanism that controls the process of adding, storing, and deleting information from the iteratively propagated cell state [28]. In the gating mechanism, three gates are being used. Forget gate, input gate, and output gate. In Figure 2.7, the architecture of the LSTM is presented.

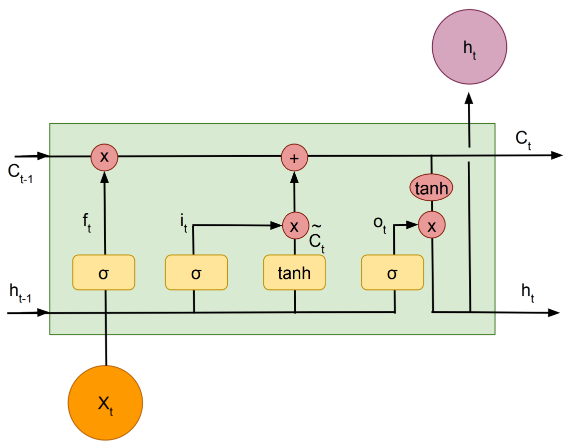


Figure 2.7: LSTM architecture. [29]

In Figure 2.7, C_t stands for cell state, h_t for hidden state, which is an LSTM output, and f_t, i_t, and o_t denoting forget, input, and output gates, respectively. The forget gate is in charge of determining how much information should be kept and how much should be forgotten from the network. The decision of how much of the information is kept is based on the sigmoid function which results in values between 0 and 1. LSTM unit remembers more from the past if the sigmoid result is near 1. Similarly, the closer the outcome is to 0, the less memory from the past is retained by the LSTM unit [30]. The following formula presents the operation of forget gate:

$$f_t = (W_f \cdot [h_{t-1}, x_t] + b_f) \tag{2.7}$$

The input gate, on the other hand, is in charge of determining how much information will be added to the cell state. The sigmoid function, like in the forget gate, is applied to the new input and previous state to make this decision. The result is multiplied by $\widetilde{C}t$, yielding a new vector of candidate values that can be added to the current cell state [31]. Both operations can be described by formulas as follow:

$$i_t = (W_i \cdot [h_{t-1}, x_t] + b_i) \tag{2.8}$$

$$\widetilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{2.9}$$

Before reaching the last gate old cell state has to be updated. This operation is simply done by multiplying C_{t-1} by f_t and then adding i_t multiplied by $\widetilde{C}t$ [29]. This step can be shown with the formula as follows:

$$C_t = f_t * C_t - 1 + i_t * \widetilde{C}t \tag{2.10}$$

The output gate is the LSTM unit's third and final gate. This gate affects the h_t value and determines the LSTM output. In that unit, the sigmoid function is also applied and works the same as in the previous gates. Firstly, the output gate is activated, and based on its result, h_t is computed. h_t is computed by multiplying the output gate's output by the previously updated cell state [29]. Both operations can be described with the following formulas:

$$o_t = (W_o \cdot [h_{t-1}, x_t] + b_o) \tag{2.11}$$

$$h_t = o_t * tanh(C_t) \tag{2.12}$$

2.5 Performance Evaluation Metrics

In order to measure how well the models perform in the task of detecting anomalies, we make use of some well-known evaluation metrics. In this project, we consider Precision (also called positive predictive value), Recall (also known as sensitivity) and F-Score (also called F1-Score or F-Measure). We avoid using Accuracy as an evaluation metric in this task of anomaly detection, since the datasets that are used are highly unbalanced. We already know that anomalies make up a very small percententage of the total dataset. Therefore, in an example of a dataset that consists of 99.5% non anomalies and only 0.5% anomalies, a dummy classifier that would classify every incident as non anomaly would end up yielding an accuracy of 99.5%. At first glance, that would seem like an outstanding performance, but it does not actually provide us with any useful knowledge about the anomalies that are to be detected.

2.5.1 Precision

Precision takes into account the True Positives (TP) of the predictions and the summation of TP with False Positives (FP). More specifically, when computing Precision, the number of successful predictions of anomalies (i.e., TP) is counted and divided with the total number of predictions (i.e., TP+FP). In other words, Precision penalizes FPs, which are the incidents that the model wrongly classified as anomalies (false alarms) [32]. Precision is computed with the following formula:

$$Precision = \frac{TP}{TP + FP} \tag{2.13}$$

2.5.2 Recall

Recall is computed by counting the number of successful predictions of anomalies (i.e., TP) in proportion to the total number of anomalies (i.e., TP+FN). Therefore, the key difference between Precision and Recall is that Recall penalizes FNs, which are the incidents that the model wrongly classified as non anomalies even though they are anomalies [32]. Recall is computed with the following formula:

$$Recall = \frac{TP}{TP + FN} \tag{2.14}$$

2.5.3 F-Score

F-Score is computed as the product of Precision with Recall divided by the sum of Precision and Recall and all this multiplied by two. F-Score is the harmonic mean of Precision and Recall and therefore provides a more combinatorial view of the performance's evaluation. F-Score has a highest possible value of 1.0 and a lowest possible value of 0.0. In case the F-Score is 1.0, Precision and Recall are both perfect (both have a value of 1.0) and in case the F-Score 0.0, either Precision or Recall is equal to 0.0 [33]. F-Score is computed with the following formula:

$$FScore = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{2.15}$$

2.6 Related Work

Based on the literature review conducted for the needs of this thesis, we observed that there has been extensive research that achieved valuable progress in the field of anomaly detection over the past few years. Here we describe the methods and results of some papers that are relevant to our thesis work.

2.6.1 Unsupervised cluster evolution approach

In the paper [34], M. Landauer et al present an online anomaly detection approach with the utilization of forecasting models that can spot instances that differ from normally expected behaviors. There is a clustering model that creates log line clusters based on several static cluster maps. In that way it detects the transitions among the different clusters. They evaluate the models by making use of metrics that are security related. In the approach that is described in this paper, there are some log lines that do not align with their expected periodicity, correlation and average frequency. These are called contextual anomalies. With this approach, the lines that were different and dissimilar could be detected and if they were occurring once, then they were treated as outlier. Apart from that, the changes in the system's behavior over time were treated as temporal anomalies. It is worth mentioning that any knowledge of the log data's structure or the percentage of anomalies is not needed beforehand as this approach is self-learning, similarly to the one we are applying in this thesis with the use of unsupervised learning models.

2.6.2 Real-time anomaly detection with unsupervised methods

Ahmad et al., in the paper [35], present an unsupervised approach of detecting anomalies in real time using streaming data. They compare various online unsupervised algorithms and models with the use of a dataset that is called Numenta Anomaly Benchmark. This dataset has been formed from data gathered from real-world data streams. 11 different methods were compared in this paper. Some of these methods were namely the Relative Entropy, the Bayesian Changepoint, the Etsy Skyline, the Hierarchial Temporal Memory, the Sliding Threshold and the Twitter ADVec. The highest evaluation metrics were achieved by the Hierarchial Temporal Memory model, which uses neural networks and estimates likelihood of the anomalies. The high latency requests' frequency, the temperature of the machine system and the utilization of the client's CPU were some numeric data that the models were tested and evaluated on. It can be concluded from this paper that the need for efficient anomaly detection algorithms is increasing, since the data stream numbers are also getting increased and an automation of the process is needed, as humans are not able of detecting anomalies manually in such sheer amounts of data.

2.6.3 Anomaly Detection in Access Logs

In the paper [36], M. Thrashini et al. reported a comparison of supervised learning models against unsupervised learning ones regarding the task of anomaly detection. The data that was used in this project derived from web access logs. They analyzed various web access log files in order to be able to spot attacks that might seem as anomalous incidents in the system. Detecting intrusions in real time is crucial for the effective protection of the system and therefore developing a model that would be able to do so is an important task. In this paper, access logs were considered to be containing some useful indications of attacks and that is the main reason they were chosen for this task. Naive Bayes Multinomial Text was the method that achieved the highest evaluation results. That was an supervised learning approach. Regarding the unsupervised one, a clustering model that made use of K-means clustering algorithm was utilized.

2.6.4 System Log Analysis for Anomaly Detection

H. Shilin et al., in the paper [37], which is an experience report, describe some methods of anomaly detection and their use in different cases and provide a comparison of six state-of-the-art log-based anomaly detection methods, with three of them being supervised and three unsupervised. The three supervised learning methods that were evaluated are Logistic Regression, Decision Trees and Support Vector Machines (SVM). Furthermore, the three unsupervised learning methods that were compared are Log Clustering, Principal Component Analysis (PCA) and Invariant Mining. Taking into account the F-Scores of the methods, we can say that supervised learning methods performed better in general. Invariant mining was reported to be the best performing method. Comparing the execution time, supervised learning methods needed much less as they were executed faster than the unsupervised ones. Of course, the downside of the supervised learning methods is that their use requires a labeled dataset. Regarding the unsupervised learning methods, PCA was the fastest executed one, and it is also used for the needs of this thesis project.

2.6.5 USAD: Unsupervised Anomaly Detection on Multivariate Time Series

In this article [38] Audibert J. et al. propose a new method for unsupervised anomaly detection on multivariate time series. This method is based on the AE architecture, and the way the model learns is inspired by the Generative Adversarial Network (GAN). The main objective of this method was to use the adversarial training which allowed the model to learn how to magnify the computation of the reconstruction error of inputs containing anomalies while also improving the model's stability, which was superior to methods based on GAN architecture. The architecture of the USAD model differs from that of a typical AE mainly due to the presence of two decoder networks. The final architecture consists of two autoencoders that share a common encoder but have separate decoders. The model was trained in two phases. Both AEs were trained in the first phase to reconstruct normal inputs (not containing anomalies), but in the second phase, AEs were trained in an adversarial manner, which means that one autoencoder attempted to fool the other autoencoder. The goal here was to train the second model to recognize whether the data was real, that is, whether it was input from raw data or reconstructed output from the first AE. The model was evaluated empirically, which means that it was tested on five public and one internal dataset (the internal dataset comes from Orange). It was then compared with the performance of other models using the same datasets. The state-of-the-art models that were used in the comparison process were: IF, AE, LSTM Variational Autoencoder (LSTM-VAE), Deep Autoencoding Gaussian Mixture Model (DAGMM), and OmniAnomaly, a stochastic recurrent neural network model. Precision, Recall, and F1-Score were the metrics used in the comparison section. The tests revealed that USAD performs very well, achieving the highest F1-scores in the majority of used datasets, but also that AE performs exceptionally well. Audibert et al. also investigate how different parameters such as down-sampling rate, window size, anomaly percentage rate, and latent space Z dimension affect model performance. The proposed model achieved the following scores on the final, internal dataset: Precision 74%, Recall 64%, and F1-score 69% and was able to detect all significant incidents that occurred in the dataset.

3

Methods

In this chapter, we present the methodology that was followed in order to complete the tasks needed for this thesis project. In Section 3.1, the dataset selection, processing and representation of the data are presented. In Section 3.2, the selection of the used features is described. In Section 3.3, we describe the implementation of the selected machine learning models that are used for anomaly detection. In Section 3.4, the methods used for the evaluation of the models' performance are described. Finally, in Section 3.5, we describe the software and hardware that we used in the framework of this thesis.

3.1 Dataset

As mentioned in the Introduction, the datasets that were used in this thesis project derived both from online sources as well as from the produced log material of the company.

The first publicly available dataset that we used is called Credit Card Fraud Detection (CCFD) dataset. We obtained this dataset from Kaggle and it serves the purpose of credit card companies being able to spot fraudulent transactions in order for their customers to be protected from being charged for transactions that themselves did not proceed with [39]. Transactions that are made by cardholders in Europe in September 2013 are contained in this dataset. The dataset is highly unbalanced, since only 492 of the total 284,807 transactions are fraudulent. This is translated in an anomaly rate of 0.172%. This dataset contains numerical input values coming after a transformation with the use of PCA. It contains, among others, "Time", "Amount" and "Class" features. Time is the number of seconds that elapsed between each transaction and the first one. Amount is the money amount of the transaction and Class contains the information of a transaction being fraudulent (value 1) or normal (value 0) [39].

Moving on, the second used dataset that derived from online sources is called SWaT (Secure Water Treatment) dataset. This dataset is a scaled down version of a real-world industrial water treatment plant producing filtered water. The SWaT dataset contains 11 days of continuous operation, of which 7 days are collected under normal operations and 4 days that include attack scenarios. The values of the dataset were obtained from 51 sensors and actuators. Data was labelled according to normal and abnormal behaviour. The anomaly rate of the SWaT dataset is 11.98% [40].

Next, we used a publicly available dataset called SMD (Server Machine Dataset). This dataset consists of data that has been gathered over the period of five weeks from the computer systems of a large Internet company. The data that has been collected for this dataset comes from 28 different machines, thus it consists of 28 different subsets of data. This means that the different subsets should be trained and tested separately. These subsets are all divided into training and testing parts, which are of equal length. The train set is the first half part of each subset of the dataset and the test set is the latter part of it. The testing parts include the labels of whether each point is an anomaly or not [41].

Finally, the last dataset that was retrieved from online sources is called SKAB (Skoltech Anomaly Benchmark). This dataset contains 35 individual csv-format data files. Each one of the files represents a single experiment and it includes a single anomaly. The data was collected as multivariate time series from sensors and gathered in the SKAB dataset. The dataset involves 11 columns (variables), the last one being the anomaly indication (0 for a non-anomalous and 1 for an anomalous data point) [42].

As far as the dataset of the company is concerned, it has been formed by collecting log data from internet-based applications that have been produced over the period of 12 hours of a weekday. We focused on applications that fall under the Internet Information Services (IIS) in order to create a dataset of logs that contain information about the source status of the operations. Also, each log contains information about the time of production (timestamp). All the logs that are collected are put in chronological order. The data that is used for training the model consists of about 24.6 million rows and the data that was used for testing the model consists of about 22 million rows, before applying the time windows division. Below, we provide an instance of the dataset and its structure, firstly as seen through Microsoft Excel and secondly as seen when loaded and managed as a dataframe in the Python development environment Jupyter Notebook.

3.1.1 Features of the dataset of the company

In order to detect anomalies in the dataset of the company, the proper features need to be selected. Forming a dataset that contains useful information and can be used for anomaly detection is of high importance, therefore particular attention was given to this task.

At first, a *timestamp* is included in the dataset, which holds the information of the time that each log was created. Also, the *timestamp* is treated as an index after dividing the data into time windows, which is explained later, in Subsection 3.2.4. Furthermore, the *id* of the logs is included in the dataset. This feature holds a unique value for each log and it is used to fasten the process of identifying particular logs in the dataset. Additionally, the dataset includes a feature called *cs-uri-stem*, which holds the information of the files that are requested by specific applications in the system. This feature is useful for identifying the applications that could be responsible for the detected anomalies.

Apart from the above, the last and most important feature of the dataset of the company is called *sc-status*. This is a protocol status code that is returned from a running application to the client. It contains the information of whether a request is successful. In some cases of unsuccessful requests, we are able to associate the returned *sc-status* code with a specific reason of failure. For instance, if the *sc-status* includes a code that is of the form 4XX, where $X \in \mathbb{N}$, this is an indication of an error that occurred on the client. The following table shows the messages indicated by the specific codes of the *sc-status* [43].

Table 3.1: *sc-status* codes and their corresponding messages

Status code	**Message**
1XX	Informational
100	Continue
101	Switching protocols
2XX	Success
200	OK-Succeeded
201	Created
202	Accepted
203	Nonauthoritative information
204	No content
205	Reset content
206	Partial content
3XX	Redirection
301	Moved permanently
302	Object moved
304	Not modified
305	Use proxy
307	Temporary redirect
4XX	Client Error
400	Bad request
401	Unauthorized
402	Payment Required
403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable
407	Proxy Authentication Required
408	Not Acceptable
409	Request Time-Out
410	Conflict
411	Gone

412	Length Required
413	Precondition Failed
414	Request-URL Too Large
415	Unsupported Media Type
5XX	Server Error
500	Internal server error
501	Not Implemented
502	Bad Gateway
503	Out of Resources
504	Gateway Time-Out
505	HTTP Version not supported

In some cases, the *sc-status* code might be 0, which indicates that a possible connection reset could have occurred and, as a result, the application did not send the actual *sc-status* code.

3.2 Preprocessing

In order to train the models and to detect anomalies, we needed to preprocess the datasets in a way that they can be fitted in the different models. In this project, data preprocessing is divided into three main tasks, namely data scaling, anomaly rate adjustment and one-hot encoding.

3.2.1 Data scaling

Data scaling is the first task, and it is applied to all five datasets. Many studies clearly showed that different ways to scale data for anomaly detection tasks can be used. Normalization of data was performed, for instance in [38], yielding good results. When the distribution of the features does not follow a gaussian distribution, normalization is recommended [44]. However, we can see in [45] that normalization is also very sensitive to outliers, which can be very impactful in anomaly detection tasks, as outliers might be considered anomalies. As a result, standardization can be also used to get satisfying results, as shown in [46]. Standardization is an especially suitable choice for multivariate time series datasets, according to Shanker et al. [47]. As a result, we decided to use both approaches in this research. This means that for each model, we first train it on normalized data and then compare the evaluation scores with the results achieved from the model trained on a standardized dataset. The Scikit-learn library was used in both scenarios, providing two scaling functions; $StandardScaler$ and $MinMaxScaler$.

The $MinMaxScaler$ scales the range of the data to a fixed value between 0 and 1, or -1 to 1 if we want to allow for negative values. The operation of normalization can be expressed by following formula:

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{3.1}$$

The *StandardScaler* transforms the data so that the mean μ is zero and standard deviation σ is one. In the standardization formula 3.2, Z represents the standardized data points and X the original data points.

$$Z = \frac{X - \mu}{\sigma} \tag{3.2}$$

3.2.2 Anomaly rate adjustment

The anomaly rate adjustment is only applied to one dataset, namely the SKAB dataset. This procedure was used to reduce the amount of anomalies in the SKAB dataset, which was originally roughly 53% when all subsets were combined.

This problem was handled by creating a function that takes each subset of SKAB dataset, computing the rate of the anomaly in that subset, and adjusting its value to our desired rate of 8%. In simple terms, the function was going through the rows of the subsets, identifying where the anomalies begin, and then cutting everything after that, leaving only the amount of anomalies that additionally makes up 8% of the subset.

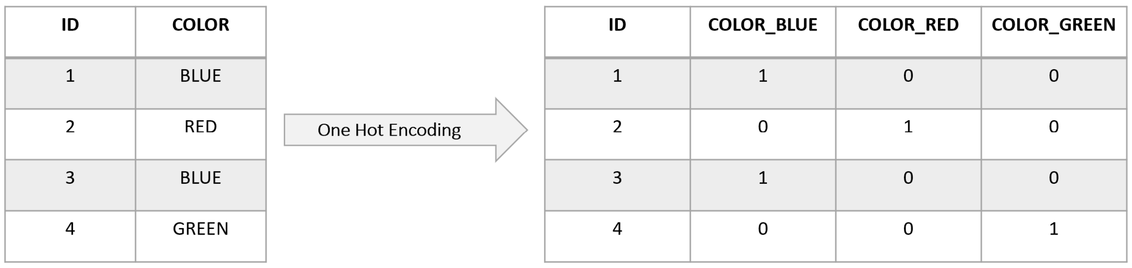
With the above process, we can ensure that, after the data is split into train and test, the train set will not contain any anomalies. However, a downside of it is that some non anomalous data was lost throughout this process. Nevertheless, as the purpose of anomaly detection is to detect abnormalities as soon as possible, removing a portion of the subsets after an anomaly happened will have a little impact on the performance of the models. We decided to reduce the anomaly rate of this dataset to 8% in order to have datasets with evenly increasing anomaly rates, i.e. 0.17, 4, 8, and 12 percent.

3.2.3 One-hot encoding

One-hot encoding is the final task of data preprocessing. This task is also performed on just one dataset. Specifically, it is carried out on the dataset of Centiro using the Pandas library in order to compute $sc - code$ occurrences during particular time periods (data windows). This operation results in a dataset with 21 new columns, where each one corresponded to a specific code. Figure 3.1 shows an example of a one-hot encoding operation.

3.2.4 Data Windows

As part of preprocessing, the data has been divided into windows and each window includes a number of rows of the dataset. In this way, we do not classify each row, but a group of rows as anomalous or non anomalous. When the division into data windows implemented based on the timestamps of the data points, we refer to these windows as time windows.


One Hot Encoding

ID	COLOR
1	BLUE
2	RED
3	BLUE
4	GREEN

ID	COLOR_BLUE	COLOR_RED	COLOR_GREEN
1	1	0	0
2	0	1	0
3	1	0	0
4	0	0	1

Figure 3.1: Example of One-Hot Encoding.

However, this division into data windows cannot be applied in the same way in the different datasets that were used in this thesis. That is because the datasets contain different information and they are used for different tasks. Below, we describe the way that data windows are created in each of the datasets.

As for the CCFD dataset, it is the only dataset where division into windows was not applied. The reason for this is that the dataset contains transactions from different cardholders that are not connected to each other in any way. Therefore, two transactions that follow one another in the dataset are completely irrelevant to each other and they it just happened that they took place at the same time. Thus, we did not find it reasonable to split the data into time windows, but we rather run the models on the dataset row by row.

Next, we have the SWaT dataset. In this dataset, we applied time windows division based on the timestamp feature. Each time window contains information of 5 seconds. We find it reasonable to split the data into time windows as the data points that are close to each other time-wise are also connected to each other in terms of indicating normal or abnormal behavior. Therefore, when we feed the models with the data, this treats each time window as a data point and assigns a label to it. This label occurs based on whether the model predicts that the time window is anomalous or non anomalous.

Moving on to the SMD, this dataset does not contain any timestamp feature, therefore division into time windows was not feasible. However, we divided each of the 28 subsets of the dataset into data windows by number of rows. More specifically, we included 20 rows in each window that were treated by the models as single data points and the models were evaluated based on their predictions of the windows labels.

As for the SKAB dataset, this included the timestamp feature, therefore we created time windows similarly to the SWaT dataset. The difference here is that the SKAB dataset consists of 35 subsets, so each of these subsets is divided into time windows. These time windows are treated also like data points and classified as anomalous or non anomalous by the models.

Lastly, regarding the dataset of the company, we divided it into time windows of 5 seconds, 30 seconds and 1 minute. Applying three different amounts of time for the creation of time windows helped in the better evaluation of the model.

3.3 Model Implementation

As it was described above, we have utilized four models in total for the needs of this thesis work. These are namely the Local Outlier Factor (LOF), the Isolation Forest, the Principal Component Analysis (PCA) and the LSTM Autoencoder.

3.3.1 Train-Test Ratio

In this paper, two methodologies were used to partition datasets into training and test sets. To begin, two datasets, namely SWaT and SMD, were already separated into training and validation sets. As a result, splitting was not required in this case. The Scikit-learn library was used to divide Credit Card Fraud Detection and SKAB datasets. The split ratio for both datasets was 80/20, indicating that the training set contained 80% of total data and the validation set 20%.

Even though we performed unsupervised learning, splitting the datasets into train and test set was useful, since our goal was to train the models on data that is clean from anomalies and evaluate them on data that is infected with anomalies. In that way, the model was trained on observing the normal behavior of the system where anomalous incidents do not take place, so that later it would be able to recognise any abnormal behavior and spot the anomalies in the test dataset.

3.3.2 Local Outlier Factor

As described in 2.4.1, the LOF algorithm detects anomalies by taking into account the density between the data points and their neighbors. Here we make use of the sklearn.neighbors package of the Scikit-learn library [48] and the most important parameters that we need to investigate are the *n_neighbors* and the *contamination* [49]. The *n_neighbors* parameter is the number of neighboring data points that the internal clustering algorithms of LOF use. The *contamination* parameter is the proportion of the most isolated data points (the points with the highest LOF score in the dataset) which will be considered by the model as anomalous. The *contamination* should be in the range (0, 0.5] [48]. Also, since we want to train the LOF model on the train dataset and predict the labels of the test dataset, we set the value of the parameter *novelty* to True.

Regarding the number of neighbors to be considered, this normally has to be greater than the minimum number of data points that a cluster contains and smaller than the maximum number of nearest data points that can potentially be considered as anomalies. However, generally in practice, we do not possess such information [49]. Therefore, we experimentally try different values for this parameter, values that seem to be logical and appropriate for the used datasets. Keeping the default *n_neighbors*=20 appeared to work decently well for the four publicly available datasets, thus we decided to keep this parameter equal to 20.

Since we are unable to know the proportion of anomalies in the dataset of the company beforehand, we cannot specify with high confidence the *contamination* parameter of the LOF model. We are only able to experimentally estimate this parameter. When working with the publicly available datasets, we already know the amount of anomalous data points in the whole dataset. Therefore, we are able to set the *contamination* parameter equal to the quotient of the division of this amount with the amount of total data points in the dataset. Indeed, this would generally lead to better results in the performance of the model, than if we would assign a random value to the *contamination* parameter or if we would attempt to empirically approach it. In our implementation, this parameter is empirically set equal to 0.01 when using the publicly available datasets. The reason behind this is that we intend to check the performance of the models under realistic conditions, where we do not have any information related to the labels or the contamination rates of the datasets. Since we are not able of having such information when it comes to the dataset of the company, then comparing the performance of the models under the same conditions seems fair and reasonable. After all, this comparison is done in order to decide for the best performing model that will be applied on the dataset of the company. As we describe later, the LOF model did not achieve the best results out of the four models, therefore we did not apply it on the dataset of the company.

3.3.3 Isolation Forest

The Isolation Forest model is an ensemble model that consists of a number of isolation trees, as mentioned in 2.4.2. Therefore, each tree needs to be fit with a number of samples from the train dataset. In our implementation, all samples of the train dataset are used to fit all trees. We make use of the sklearn.ensemble package of Scikit-learn library and the parameter that is responsible for the number of samples that each tree will use is *max_samples*. Thus, we set this parameter equal to the number of samples of the train dataset. This might lead to a slightly higher execution time than if we used a part of the train dataset for each tree. However, using the whole train dataset for each tree yields better performance results. Another parameter of the IF model that we need to consider is the *contamination*. It has the same role as in the Local Outlier Factor, which means that it represents the proportion of the anomalies in the dataset and it should be in the range (0, 0.5] [50]. The same logic that was described previously for the LOF model is also applied here. Therefore, we set the *contamination* parameter equal to 0.01 for this model as well. This is an empirical estimation of the proportion of anomalies that can be found on the different datasets. We can assume that anomalies make up a (very) small percentage of the whole dataset, even though this might vary from one dataset to another.

3.3.4 PCA

For the implementation of the PCA algorithm, the sklearn.decomposition package of Scikit-learn library was used. The PCA algorithm is used to reduce the number of components (dimensionality) as mentioned in 2.4.3, but here the model is enhanced with a function that computes the anomaly scores of the data points.

At first, we fit the PCA with the train data. Then, the model computes the anomaly scores of the test data based on the reconstruction error. Afterwards, a table (Pandas DataFrame) of data points along with their anomaly scores is created. This table gets sorted in a descending order, starting from the point with the highest anomaly score. Then, in order to compute the Precision of the model, the first 2% of the data points of the sorted table is kept and these data points are considered as the only anomalous ones, since they have the highest anomaly scores. We refer to these first data points that are kept as "cutoff". The decision to select the first 2% of the data as a cutoff was based on the assumption that was previously made regarding the *contamination* (i.e. proportion of anomalies in the dataset). We estimated that anomalies could make up about 1% of the total data points. Therefore, it seems reasonable to use a slightly larger percentage as a cutoff, since, experimentally, we can assume that the model will most likely not predict perfectly all the anomalies. This means, that it will probably not assign the highest anomaly scores only to actual anomalous data points, but the first 1% of the sorted anomaly scores table might also include some non anomalous data points. Therefore, our estimation is that by taking a cutoff of 2%, we investigate more data points that have relatively high anomaly scores and might be actual anomalies, thus increasing the total number of TPs and providing a clearer overview of the performance of the model. Furthermore, the larger the cutoff, the higher the amount of FPs, which translates into a decrease in the Precision of the model. Hence, finding a proper balance between increased TPs and FPs is important and our estimation is that a cutoff of 2% is a reasonable choice for that matter.

Regarding the evaluation of the performance of the model, apart from the Precision that was described above, we also compute the Recall and the F-Score. In order to compute the Recall, the same cutoff as previously is used and the amount of TPs is measured. However, all of the data points are taken into account when calculating the sum of TPs and FNs. Recall is the quotient of the amount of TPs divided by this sum. Finally, F-Score gets computed as it was described in 2.5.3.

An important parameter of PCA that needs to be properly tuned is the number of components (*n_components*) that will be used by the model. However, there is not one universal tuning approach that works the best in every case (dataset). Therefore, we needed to experiment and decide on an approach that could work decently for the datasets that we work on. In fact, we ended up using two ways for estimating the number of components that PCA takes into account.

More specifically, for the SWaT and the CCFD datasets, we plot the diagram of the numbers of components against their cumulative variance. As the number of components increases, the corresponding cumulative variance increases, as expected. Based on [51] and [52], we can establish a threshold of about 60% to 99% of cumulative variance in order to find the optimal number of components or a number of components that is close to the optimal. In our case, we empirically used a threshold of 95% and it yielded decently good performance results. The decision for this threshold was based on visually approaching the point above which the cumulative variance is about to be stabilized near 1.0. An example of this approach applied to the normalized SWaT dataset is shown in Figure 3.2. A red line is drawn near the point of stabilization of cumulative variance. This line corresponds to 95% of the variance and the components needed to reach this percentage are 12. This method for choosing the number of principal components is closely related to Kaiser's stopping rule, according to which, only the number of features that have eigenvalues over 1.00 should be considered, as described in [52].

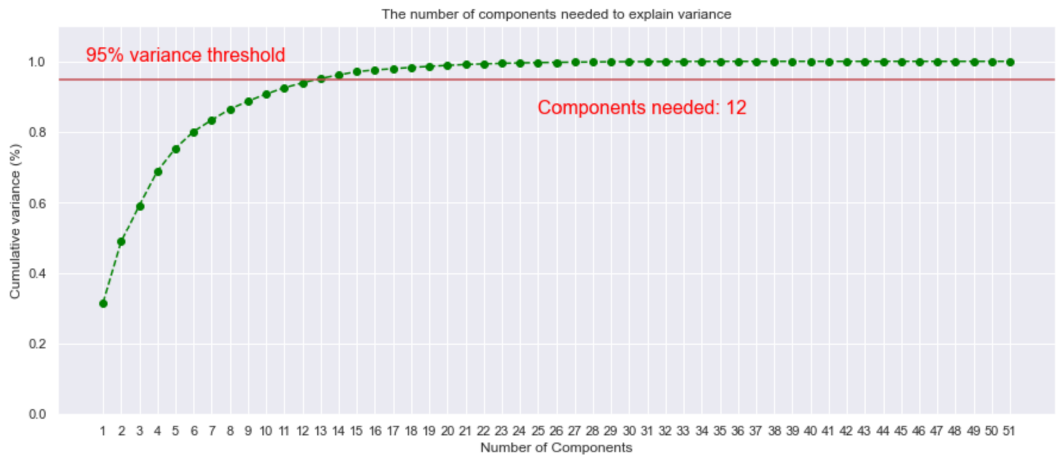


Figure 3.2: Example of choosing the number of principal components based on the percentage of cumulative variance.

As far as the datasets SKAB and SMD are concerned, a slightly different approach was followed in order to determine the number of components. It is based on a similar logic as the previous one. However these datasets consist of a number of subsets, thus the need for automation of the process appeared. We iterate through the subsets of these datasets, therefore it would not be efficient to visually approach the stabilization point of cumulative variance and empirically set it equal to an empirically reasonable percentage, as we did before. Thus, in this case, for every subset we iteratively find the point where the cumulative variance does not have an increase of more than 0.01 between a number of components and its following one. For instance, the algorithm goes through the cumulative variance matrix of a subset of SMD. In this matrix, it observes that the cumulative variance that corresponds to 6 components is equal to 0.9238 and the one that corresponds to 7 components is equal to 0.9289. Therefore, the increase of the cumulative variance between 6 components and 7 is less than 0.01. So, the number of components used in PCA is set equal to 6. We decided on this threshold of 0.01 increase in a similar logic that was applied in the previous method. We try to estimate the point where the cumulative variance stabilizes, which means that, above that point, adding more components will not have an important increase in the variance. Therefore, the rest of the components above that point can be ignored. We empirically found that 0.01 is a reasonable threshold and decided to proceed with using it in the datasets SKAB and SMD.

3.3.5 LSTM-Autoencoder

Implementing the LSTM-Autoencoder, the input data has to be reshaped, as the inputs to the layers of the LSTM architecture are expected to be 3-dimensional. The *reshape* function of Numpy library was utilized in order to transform the input data into a 3-dimensional array of the structure (*samples*, *timesteps*, $features$). The *samples* variable corresponds to the amount of data points and the *timesteps* variable defines how many past data points the model considers in addition to the current data point. Finally, the $features$ variable is the number of features that are taken into account at each time step.

The python libraries Tensorflow and Keras were used for building the layers of the LSTM-Autoencoder. The model's architecture is made up of seven layers, five of which are hidden layers. The data is compressed in the first two hidden layers, reducing the feature sizes to the defined ones. This part of the model is the encoder. The fourth and fifth hidden layers of the model decompress the data, restoring the original size of the features. These layers are the mirror image of the encoder layers, meaning that the layers of the decoder are stacked in reverse order of the layers of the decoder. The third hidden layer is a $RepeatVector$ layer, which works as a "bridge" between the encoder and decoder modules. The purpose of this layer is to replicate the encoded feature vector as many times as the *timesteps* variable that was defined. It is performed in order to pass these vectors to the next layer. This corresponds to the first out of two layers of the decoder module. The last output layer is the $TimeDistributed$ layer, which generates a vector with the same length as the number of features. This is a reconstruction of the input that was originally received from the model.

The activation function used by all the layers is the Rectified Linear Unit (ReLU). The optimizer used when compiling the model is the Adaptive Moment Estimation algorithm (Adam) and the loss function of the model is based on the mean absolute error (MAE).

The model gets trained for 100 epochs using the train dataset and early stopping is applied. The early stopping is responsible for terminating the training of the model, once it reaches a point, above which not any further essential improvement is being observed. This is achieved by monitoring the validation loss of the model over the training epochs. Once it is observed that this validation loss does not decrease more than 0.0001 for 15 continuous epochs, the model stops the training and it keeps the weights that yielded the lowest validation loss of the model. Finally, in order to reduce overfitting of the model, weight regularization is applied to the input layer.

The last step of deciding whether a data point is an anomaly is to define a classification threshold. For this, we followed two different approaches. The logic behind the two approaches is similar and it is based on the loss distribution computed on the train set.

The first approach consists of visualizing the loss distribution by plotting the loss against the density of the data. Afterwards, we manually choose the classification threshold by analyzing the loss distribution and picking a point where the density of the data becomes very close to zero. Thus, all data points above this threshold are classified as anomalies. By using this method, we can ensure that the threshold is set above the noise level of the data, so that we can avoid classifying normal data points as anomalies. This approach is followed on the datasets SWaT and CCFD. An example of this approach is shown in Figure 3.3, where the threshold is manually set equal to 0.225. When it comes to the datasets SKAB and SMD, we automate the process of deciding on the threshold. It is set at the point where 98% of the loss is reached. For finding this point, we use the *quantile* function provided by the Numpy library. The reason behind following a different approach for the datasets SKAB and SMD is that they consist of a number of subsets, unlike SWaT and CCFD. Therefore, we need to automate the process of finding thresholds in each of the subsets of the datasets SKAB and SMD. We iterate through the subsets and we compute the 0.98 quantile of the density for each one of them. This automation might yield slightly lower performance results than manually picking the threshold as it was done in SWaT and CCFD.

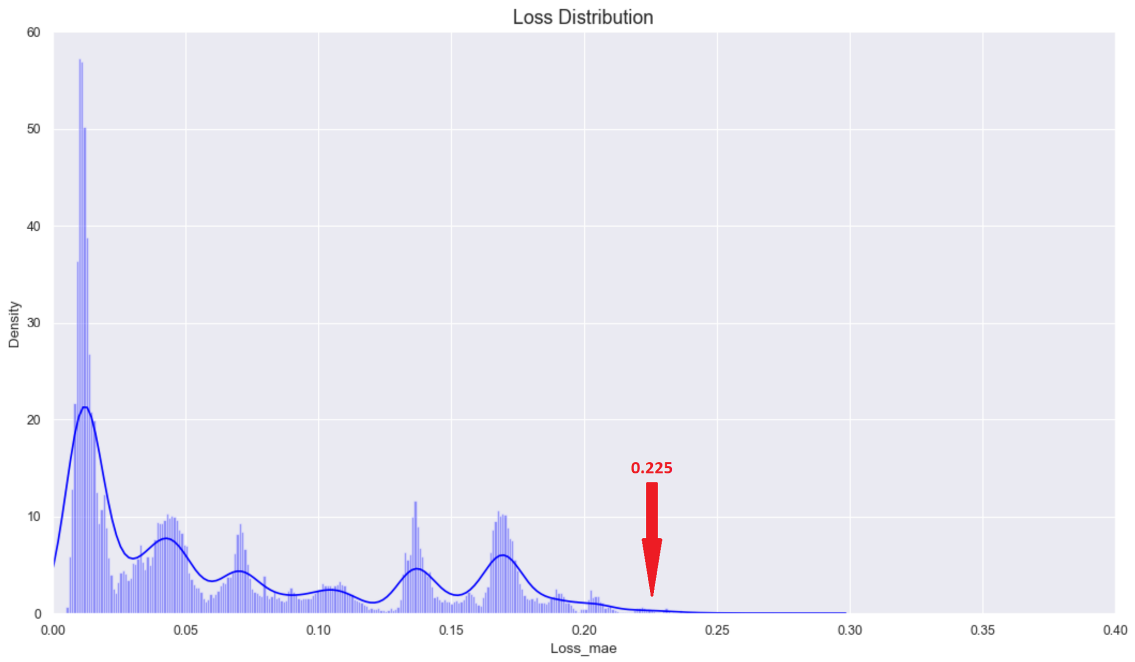


Figure 3.3: Example of loss distribution plot for choosing the classification threshold.

3.4 Evaluation Method

After preprocessing the data, selecting the appropriate features and training the selected models, we evaluate their performance based on the evaluation metrics Precision, Recall and F-Score. It was previously mentioned that we train on the train dataset that does not contain any anomalies and evaluate on the test dataset that contains some anomalous data points. Therefore, at first, the models learn the normal behavior of the system based on the clean from anomalies train dataset. Afterwards, when they encounter a data point of the test dataset that does not fit with the rest and it seems like abnormal, this data point gets classified as anomalous. In order to evaluate the performance of the models, we need to check how many of the data points were correctly classified and more importantly, how many anomalous data points were correctly classified (as anomalous). That is because the evaluation metrics that we use depend on the proportion of correctly classified anomalous data points (i.e., TP).

As we described previously, most of the datasets were divided into data windows. Therefore, when evaluating the models, we need to take that into account. The labels of the data windows are determined by the fact of whether they contain any anomalies or not. Thus, a window that contains at least one anomalous row is labeled as anomalous.

Furthermore, two of the publicly available datasets (SMD and SKAB) consist of a number of subsets, therefore the evaluation of the models on these datasets is slightly different than on the others. The models are applied iteratively on each of the subsets and their performance is counted with the use of the mentioned evaluation metrics. After this iteration is completed, we compute the average of the evaluation metrics of all subsets. In that way, we end up with the final evaluation metrics representing the performance of the models on the whole dataset.

As far as the dataset of the company is concerned, we do not have a fast and automatic way of measuring the performance of the models on it. The reason for this is that we do not have the labels of the dataset of the company in contrary to the publicly available datasets, where we were at least provided with the labels of the test set. In fact, this is a real-world scenario where thousands of log files are produced in a period of a few hours, therefore manually assigning labels for all of them would require lots of valuable time and effort from a number of people. So, the evaluation here is done in the following process. We select the model that had the most stably satisfactory performance on the publicly available datasets. By stably satisfactory, we mean the model that achieved decent evaluation metric results in all four datasets, without having big fluctuations in its performance. After selecting the mode,we fit it with data from the company that corresponds to a day, during which there were not any observed anomalies. Then, we apply the model on the dataset of the company that corresponds to another period of time, in which we want to detect anomalies. The model returns the labels that were assigned to the data points of this dataset. Afterwards, we provide the experts of Centiro with these labels, in order for them to go through the logs and manually check which of the model's detected anomalous data points are actually anomalies. In this way, we compute the evaluation metric results of the model's performance on the data of the company.

The model that had the most stably satisfactory performance in the publicly available datasets is the Autoencoder. Moreover, checking the consistency of the results is needed in order to reduce the factor of randomness in the models' performance results that were previously observed. Additionally, we perform this check in order to measure the performance of the models on sets of data with different anomaly rates and compare their results. The consistency check that we perform takes place on the standardized SWaT dataset both with and without the use of time windows division. The SWaT dataset is the largest of all and the only one where we can divide the test set into ten parts and still each one of them to be big enough for a valid evaluation. Apart from the above, the SWaT dataset is the only one where each of the ten parts of the divided test set can contain a decent amount of anomalies without the need of changing the order of the data. It is a time-series dataset, therefore preserving the order of the data is important for the proper functioning of the model. In the publicly available datasets, we can see that anomalous data points appear in bunches next to each other after periods of normal behavior. In the datasets SWaT, SKAB and SMD, the data was gathered continuously for some period of time. For instance, when it comes to SWaT, the data was gathered continuously for 11 days. Therefore, changing the order of data could possibly decrease the models' possibilities of finding abnormalities. This is also the case in the log dataset of the company, where the order of the data needs to be preserved as well.

After dividing the test set into ten parts, we train the models on the train set and we evaluate their performance iteratively on each of the ten sets. A visualization of this division of the test set and the iterative evaluation is shown in Figure 3.4.

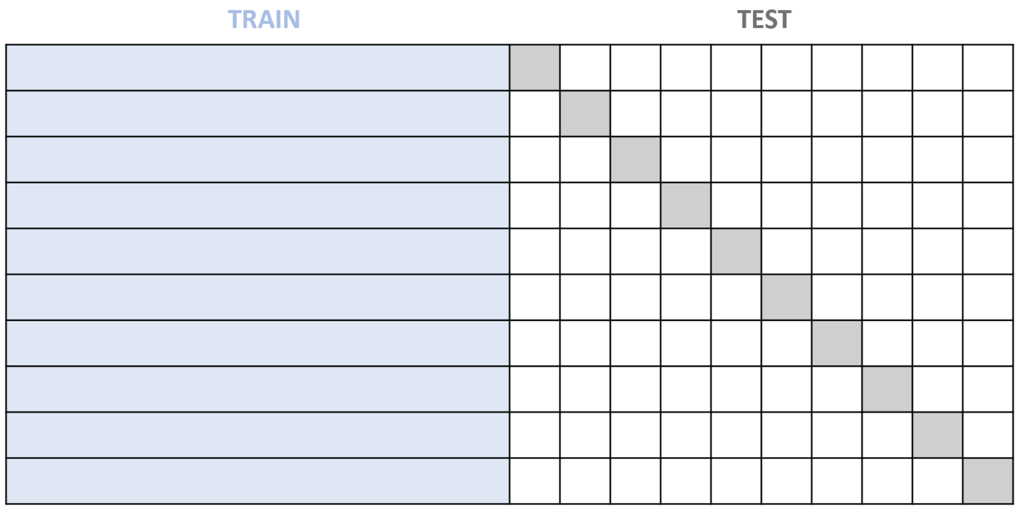


Figure 3.4: Division of the test set into 10 subsets and iterative evaluation on each one of them.

It is observed that, generally, the performance of the models increases as the amount of anomalies in the test set increases. However, this is not a universal rule, but it is rather an observation, which can also be seen later in the Results section and specifically in Figure 4.5. By performing this consistency check, we confirmed that the AE model has the most stably satisfactory performance and therefore it is utilized for detecting anomalies on the data of the company.

3.5 Used Software and Hardware

For the needs of this thesis, Python 3.7 was used and the main integrated development environment that was used is Jupyter Notebook. The python libraries that we used are NumPy, Pandas, Matplotlib, Seaborn, Scikit-learn, Tensorflow and Keras. Numpy is a library that includes high-level mathematical functions for the manipulation of multidimensional arrays and matrices [53]. Pandas offers support for numerical table and time series manipulation [54]. Matplotlib is a library used for data visualization in Python and it is well integrated with NumPy and Pandas. Seaborn uses some extra methods that support the creation of explanatory graphics as an extension of Matplotlib and it is well integrated for Pandas DataFrame manipulation [55]. Scikit-learn (or sklearn) is a machine learning library that includes various classification, regression and clustering algorithms [56]. TensorFlow is an open-source library that is used for machine learning and artificial intelligence related tasks. It is mainly used for creating and training deep neural networks [57]. Keras is a library that supports the building of deep neural networks and runs on top of TensorFlow [58].

The training and testing of models were performed on laptops provided by Centiro. Among their specifications is an Intel(R) Core(TM) i7-10850H CPU @ 2.70GHz, a 32GB DDR4 memory and an Nvidia Quadro T1000 GPU.

4
Results

In this section, we present the results of the four models on both the publicly available datasets as well as on the dataset of the company. The performance results vary depending on the model, dataset and preprocessing method (i.e. normalization, standardization, with/without time windows division). At first, the results of applying the models on the publicly available datasets are presented. Afterwards, the performance of the selected Autoencoder model on the dataset of the company is described.

The following tables present the performance results of the models on each publicly available dataset. The histograms that follow, visualize these results in order to ease the comparison of the performance of the models. As it was previously mentioned, Precision, Recall and F-Score are the evaluation metrics that are used in this project. The most important of the three metrics that we take into account in order to evaluate the stability in the performance of the models is the F-Score. The reason behind this is that the F-Score is a harmonic mean of Precision and Recall as described in 2.5.3 and therefore, we found it to be the most proper metric for performance comparison among the models.

As it was described in Section 3.4 regarding the evaluation method, we need to check the consistency of the performance results that we get in order to reduce the factor of randomness. In the subsection 4.5, we provide an example of this consistency check performed on the AE model on the standardized, divided in time windows, SWaT dataset.

Later, we describe the evaluation of the AE model on the dataset of the company after dividing it into time windows. The process of its evaluation differs from the one performed on the publicly available datasets, as we are not provided with any labels for the dataset of the company.

Table 4.1: Performance results on SWaT dataset.

LOF			
Case	Precision	Recall	F-Score
Standardized & No time windows	12.2%	99.5%	21.7%
Normalized & No time windows	12.2%	99.9%	21.8%
Standardized & time windows	12.4%	99.2%	**22.1%**
Normalized & time windows	12.4%	98.9%	**22.1**%
IF			
Case	Precision	Recall	F-Score
Standardized & No time windows	16.4%	88.3%	27.7%
Normalized & No time windows	23.6%	81%	36.5%
Standardized & time windows	29.2%	75.1%	**42%**
Normalized & time windows	23%	81.3%	35.6%
PCA			
Case	Precision	Recall	F-Score
Standardized & No time windows	98.7%	16.3%	**27.9%**
Normalized & No time windows	97.8%	16.1%	27.7%
Standardized & time windows	97.3%	16%	27.5%
Normalized & time windows	98.7%	16.2%	27.8%
LSTM AE			
Case	Precision	Recall	F-Score
Standardized & No time windows	81.4%	62.7%	71%
Normalized & No time windows	30.1%	72.2%	42.5%
Standardized & time windows	82.1%	65%	**72.6%**
Normalized & time windows	62.5%	63%	62.8%

Table 4.2: Performance results on SKAB dataset.

LOF			
Case	Precision	Recall	F-Score
Standardized & No time windows	48.4%	78.9%	51.7%
Normalized & No time windows	50.6%	79.2%	53.7%
Standardized & time windows	49.5%	75.2%	52.1%
Normalized & time windows	53.8%	78.4%	**56.6%**
IF			
Case	Precision	Recall	F-Score
Standardized & No time windows	25.7%	72.9%	32.2%
Normalized & No time windows	27.3%	74.3%	**34.4%**
Standardized & time windows	20.8%	46.2%	24.6%
Normalized & time windows	23.1%	49.4%	27.3%
PCA			
Case	Precision	Recall	F-Score
Standardized & No time windows	71.7%	16.6%	26.9%
Normalized & No time windows	69.2%	15.9%	25.8%
Standardized & time windows	77.5%	19.4%	**31%**
Normalized & time windows	76.5%	19.1%	30.6%
LSTM AE			
Case	Precision	Recall	F-Score
Standardized & No time windows	68.5%	41.5%	44.4%
Normalized & No time windows	66%	41.7%	43.4%
Standardized & time windows	65.5%	48.5%	**50.1%**
Normalized & time windows	66%	46.5%	48%

Table 4.3: Performance results on SMD dataset.

LOF			
Case	Precision	Recall	F-Score
Standardized & No time windows	69.3%	66.7%	10.6%
Normalized & No time windows	75.8%	65.1%	10.8%
Standardized & time windows	55.3%	22.5%	**20.7%**
Normalized & time windows	62.1%	19.8%	20%
IF			
Case	Precision	Recall	F-Score
Standardized & No time windows	54.4%	19.1%	21.3%
Normalized & No time windows	55.8%	20%	22.1%
Standardized & time windows	53.5%	39.8%	32.3%
Normalized & time windows	55.2%	39.8%	**32.9%**
PCA			
Case	Precision	Recall	F-Score
Standardized & No time windows	40.9%	29.3%	34.2%
Normalized & No time windows	40.8%	29%	33.9%
Standardized & time windows	52.5%	31.5%	39.4%
Normalized & time windows	56.4%	33.7%	**42.2%**
LSTM AE			
Case	Precision	Recall	F-Score
Standardized & No time windows	59%	42.5%	46.5%
Normalized & No time windows	59.9%	40.9%	41.3%
Standardized & time windows	61.2%	58%	**52.5%**
Normalized & time windows	57.5%	55.6%	45.6%

Table 4.4: Performance results on CCFD dataset.

LOF			
Case	Precision	Recall	F-Score
Standardized & No time windows	2.7%	0.3%	0.5%
Normalized & No time windows	5.3%	0.7%	**1.2%**
IF			
Case	Precision	Recall	F-Score
Standardized & No time windows	69.3%	9.5%	**16.8%**
Normalized & No time windows	64%	9%	15.5
PCA			
Case	Precision	Recall	F-Score
Standardized & No time windows	5%	77.3%	**9.6%**
Normalized & No time windows	5%	76%	9.4%
LSTM AE			
Case	Precision	Recall	F-Score
Standardized & No time windows	11.4%	45.3%	18.2%
Normalized & No time windows	22.8%	44%	**30%**

4.1 SWaT dataset

As it can be seen in Figure 4.1, the Autoencoder model yields the highest F-Score out of the four models and that is for all variations of preprocessing of the dataset. The best performance of the AE model is observed in the standardized dataset. More specifically, the standardized dataset that was divided into time windows led to slightly better results than the one that was not undergone such windows division. The second best overall performing model based on the F-Scores is the Isolation Forest. We can see that also this model performs the best on the standardized dataset that has been divided into time windows.

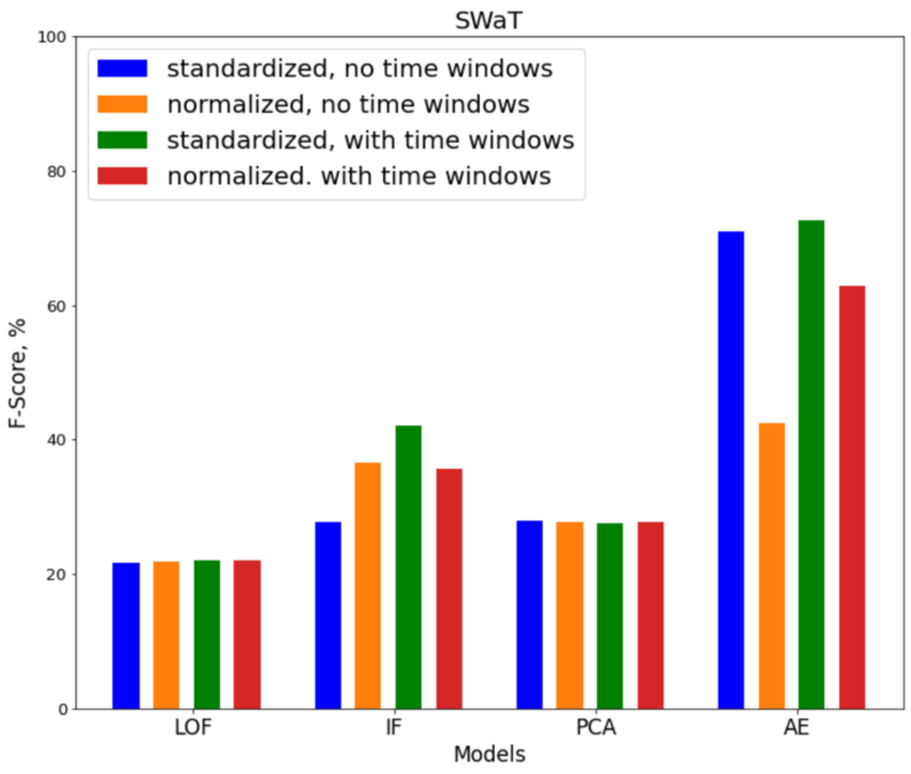


Figure 4.1: Performance results on the SWaT dataset.

4.2 SKAB dataset

From Figure 4.2, we can observe that the LOF model yields slightly better results than the AE model, which has the second best performance. More specifically, the performance of the LOF model was the best on the dataset that was normalized and divided into time windows. On the contrary, the AE model yielded better results on the standardized dataset. This is the only case where LOF performed better than the rest of the models. However, the results of the AE model on this dataset are still reasonably good.

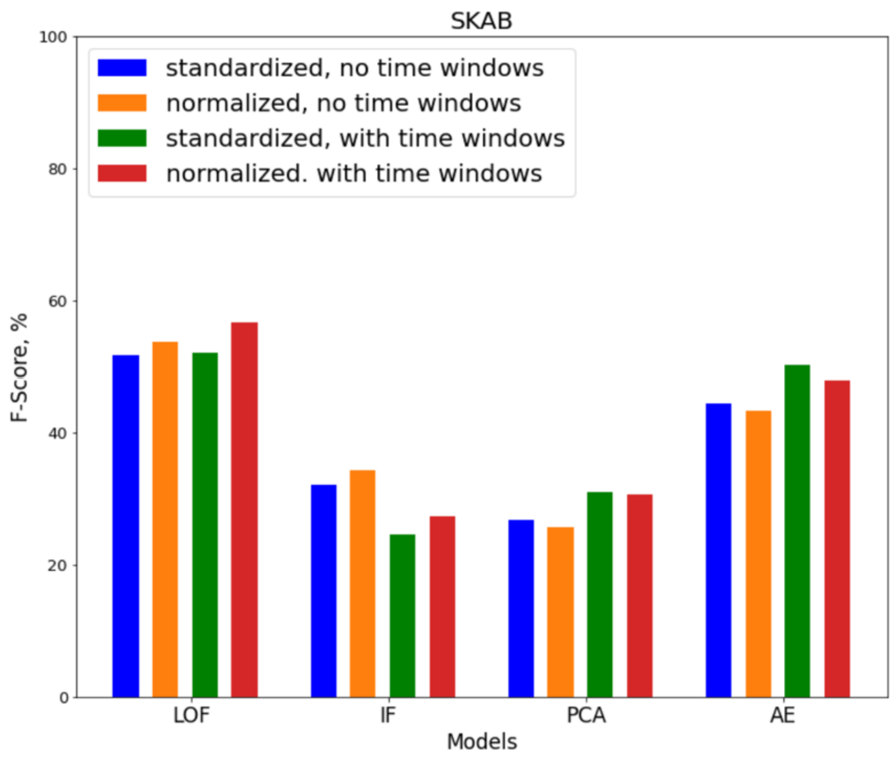


Figure 4.2: Performance results on the SKAB dataset.

4.3 SMD dataset

It can be seen in Figure 4.3, that the AE model performs the best out of the four models for every variation of preprocessing of the dataset. The best performance of the AE model is observed on the dataset that has been standardized and divided into time windows. The model with the second best overall performance on this dataset is PCA, which achieves its best F-Score on the dataset that is normalized and divided into time windows.

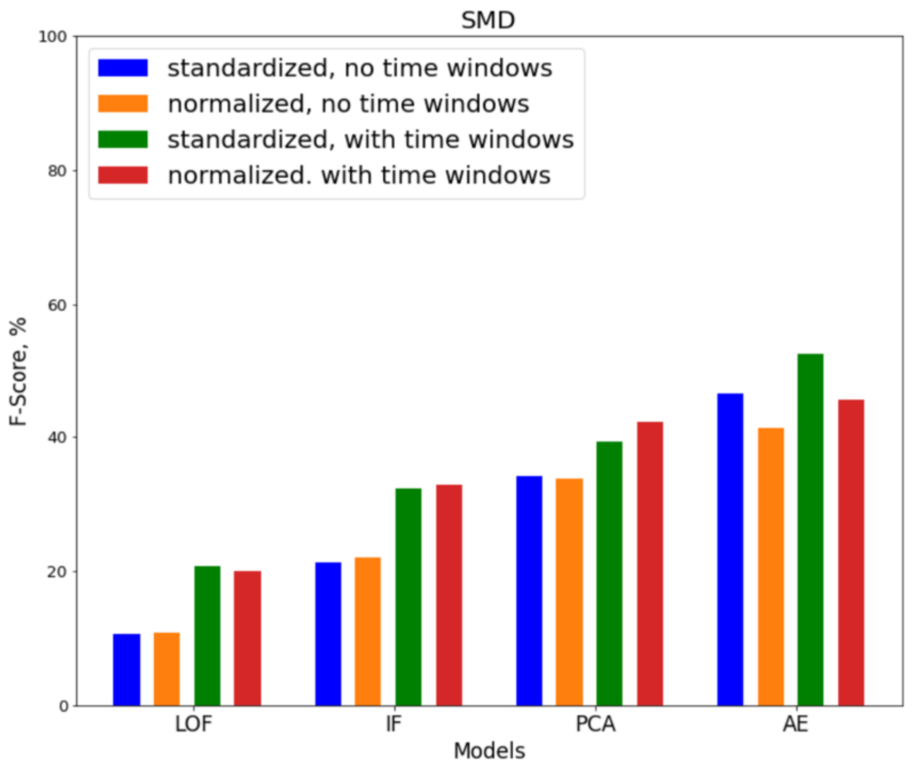


Figure 4.3: Performance results on the SMD dataset.

4.4 CCFD dataset

From Figure 4.4, we can see that the models did not achieve as high performance results as those that were previously observed on the other datasets. The peculiarity of the CCFD dataset is that it was previously undergone a transformation through a PCA algorithm. Furthermore, it has the lowest anomaly rate of all datasets, as the anomalies make up only 0.17 percent of all data points. The model that yielded the highest F-Score on this dataset is the AE model, which achieved its best results on the normalized dataset. The model with the second best performance on this dataset is the Isolation Forest.

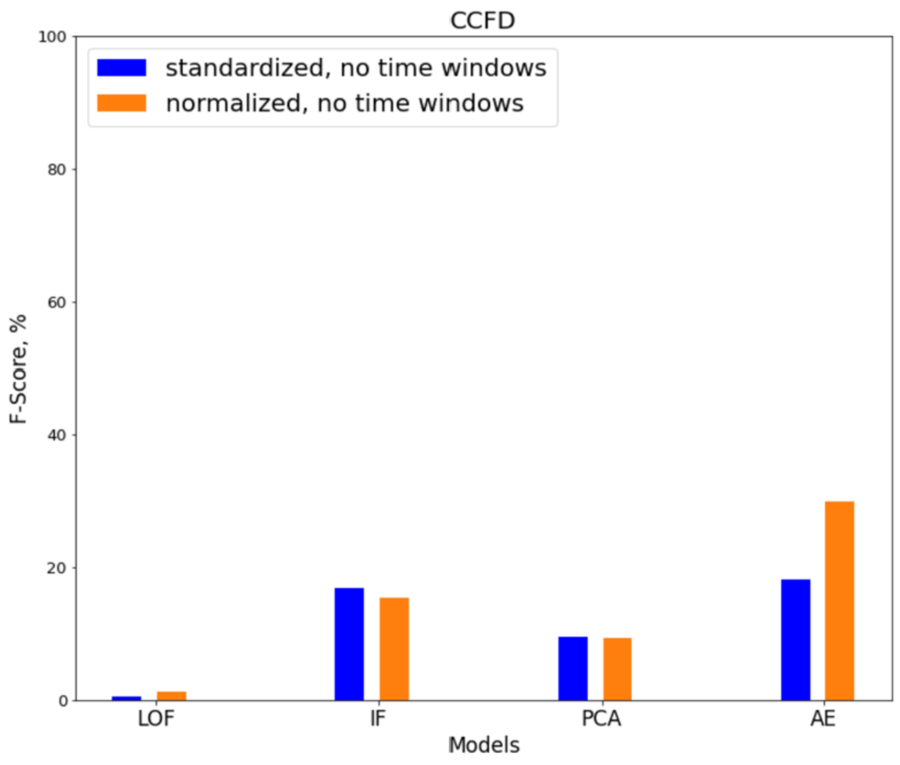


Figure 4.4: Performance results on the CCFD dataset.

4.5 Consistency check

As mentioned in the last part of Section 3.4 about the evaluation method of the model, we perform a consistency check in order to ensure that the results that were observed do not include much of randomness. In the example that is shown in Figure 4.5, we plot, in ascending order, the amount of anomalies that is included in each subset of the SWaT dataset against the corresponding F-Score that the AE model yielded for that subset. The dataset that was used for this consistency check is the standardized SWaT dataset. We can observe that, generally, the bigger the amount of anomalies in the subset, the higher the F-Score that is achieved by the model. In this example, this observation is valid for almost all subsets with a few exceptions, such as the one that appears pre-last in the left graph. It includes around

10% anomalies and its corresponding F-Score is lower than that of the previous subset, which includes 7.8% anomalies. This consistency check was also performed for the rest of the models. However, the yielded results of the other models were, in their majority, worse than those of the AE and, in some cases, their F-Score was 0.

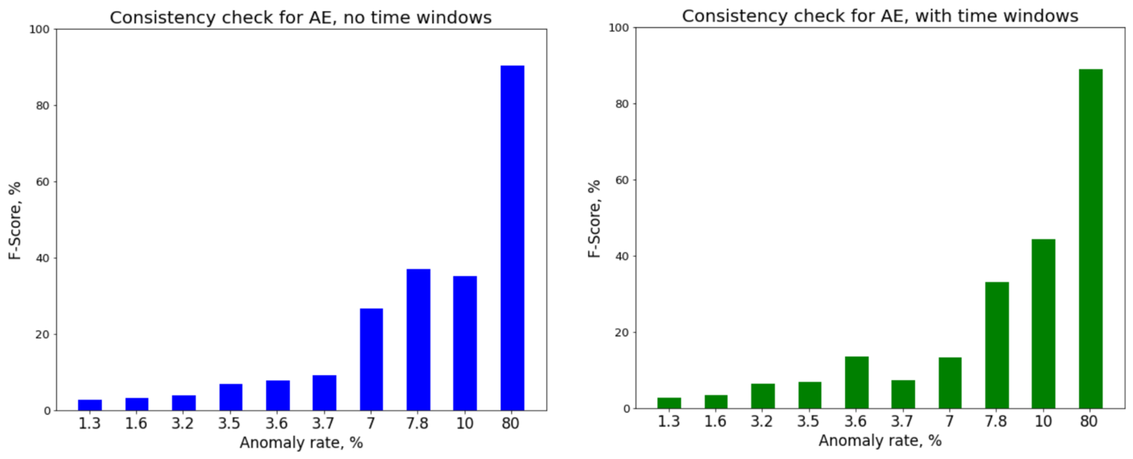


Figure 4.5: Performing consistency check on the AE model.

4.6 Additional Precision-Recall Curve test

In order to gain a better view of the performance of the models, we include here an additional test, which is based on the tradeoff between the yielded Precision and Recall. The area under the Precision-Recall curve represents the average Precision of the model. By doing so, we get a score that does not depend on a specific value of classification threshold.

We perform this test on the standardized SWaT dataset for the same reasons that were previously mentioned in Section 3.4 regarding the Consistency check. The test was performed once on the original standardized SWaT dataset and once on the same dataset that was divided into time windows.

From the definition of Precision that was described in Subsection 2.5.1, we can come to the conclusion that the Average Precision shows how accurate is a model when classifying actual anomalies as such without classifying a lot of normal data points as anomalies (FP). Therefore, the higher the Average Precision of the model, the better its ability of correctly classifying positives (anomalies).

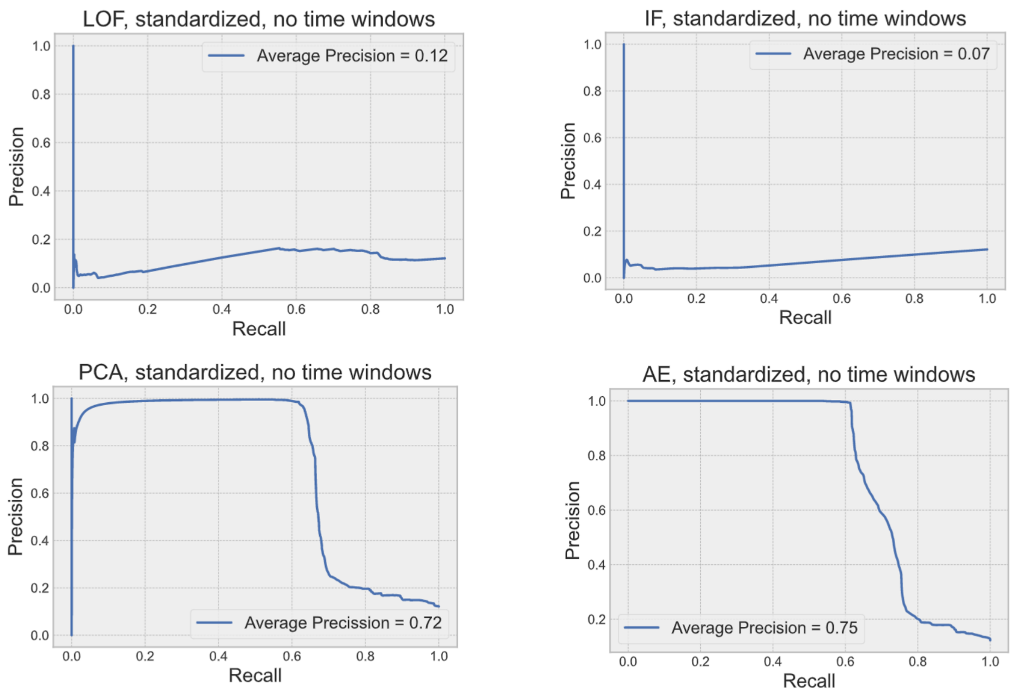


Figure 4.6: Precision-Recall Curve test on the standardized SWaT dataset, without the use of time windows.

From Figure 4.6, we can see that the model that yielded the best performance results on the standardized SWaT dataset that has not been divided into time windows is the AE model, with an Average Precision of about 0.75. The second best performance was achieved by the PCA model, which yielded a slightly lower Average Precision of about 0.72. Similarly to the previous test, the LOF and IF models yielded much lower performance results with an Average Precision of 0.08 and 0.07 respectively.

We can observe from Figure 4.7, that can be found in the following page, that the model that achieved the best performance results on the standardized SWaT dataset that has been divided into time windows is the PCA model, with an Average Precision of about 0.71. The AE model yielded the second best results with a slightly lower Average Precision of about 0.62. The other two models achieved relatively low performance scores, namely 0.12 and 0.07 for LOF and IF model respectively.

4.7 Dataset of Centiro

The model with the most stably satisfactory performance was the LSTM-Autoencoder, as it was mentioned in Section 3.4. Therefore, this model was applied on the dataset of the company and here we describe its yielded results.

From the evaluation that was performed by the experts of the company, we got some specific time periods, during which, it is valid to consider all the detected anomalies as actual anomalies. Therefore, based on that assumption, we were able to confirm

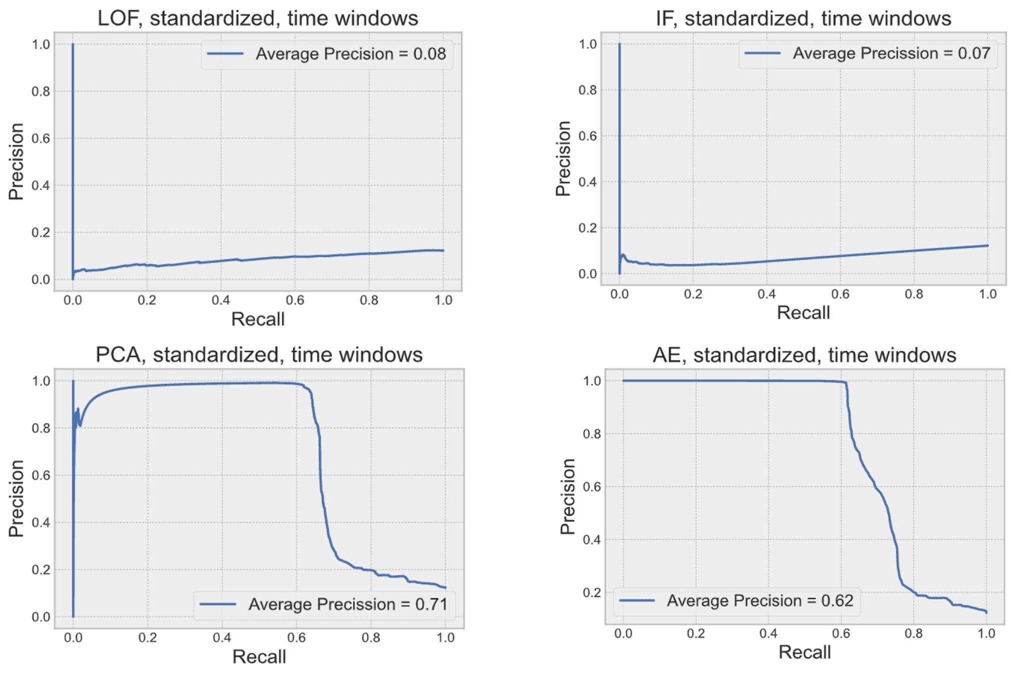


Figure 4.7: Precision-Recall Curve test on the standardized SWaT dataset, with the use of time windows.

the correct predictions (TPs) of the model. Additionally, we were able to spot the FPs, which are the logs that the model classified as anomalies, but they are not actual anomalies in the system. However, we were unable to obtain the amount of TNs and FNs. This is because of the sheer amount of data that has been used, and therefore, the detection of TNs and FNs would require a large amount of time and effort from the company, since most of the data points were classified as non anomalies. As a result, we are able to compute the Precision of the model, but not the Recall and the F-Score.

More specifically, we divided the standardized dataset of the company into time windows. The rationale for this choice is based on the fact that, in the majority of the cases, the models yielded better results on the standardized datasets. At first, we created 5-second time windows, then 30-second time windows and lastly, 1-minute time windows as mentioned in Subsection 3.2.4. Below, we refer to these time windows as data points, since they are treated as such. The results that we got are the following.

For 5-second time windows, the total amount of data points is 8641. Out of these, 63 were classified as anomalies, 39 of which were found to be actual anomalies (TPs). Next, regarding the 30-second time windows, the total amount of data points is 1441. 13 were classified as anomalies and out of these, 9 were found to be actual anomalies. Lastly, as for the 1-minute time windows, the total amount of data points is 721. There were 9 detected anomalies, 6 of which were TPs. These results are presented in the table below, in which the yielded Precision of each test can also be seen.

Table 4.5: Performance results on the dataset of the company.

LSTM AE			
time windows division	5 seconds	30 seconds	1 minute
Data points	8641	1441	721
Detected anomalies	63	13	9
True Positives	39	9	6
Precision	61.9%	69.2%	66.7%

In the figures below, we present visualized model outputs, based on computed loss distribution and set anomaly threshold in time between 9:00 and 21:00. From these graphs we can clearly see that we are dealing with four peaks, around 10, 12:30, 15 and 19, which repeat among the performed tests. The threshold for the dataset of the company was decided on both the computed value from the previously mentioned *quantile* function as well as the manual investigation of the loss distribution with respect to the density of the data.

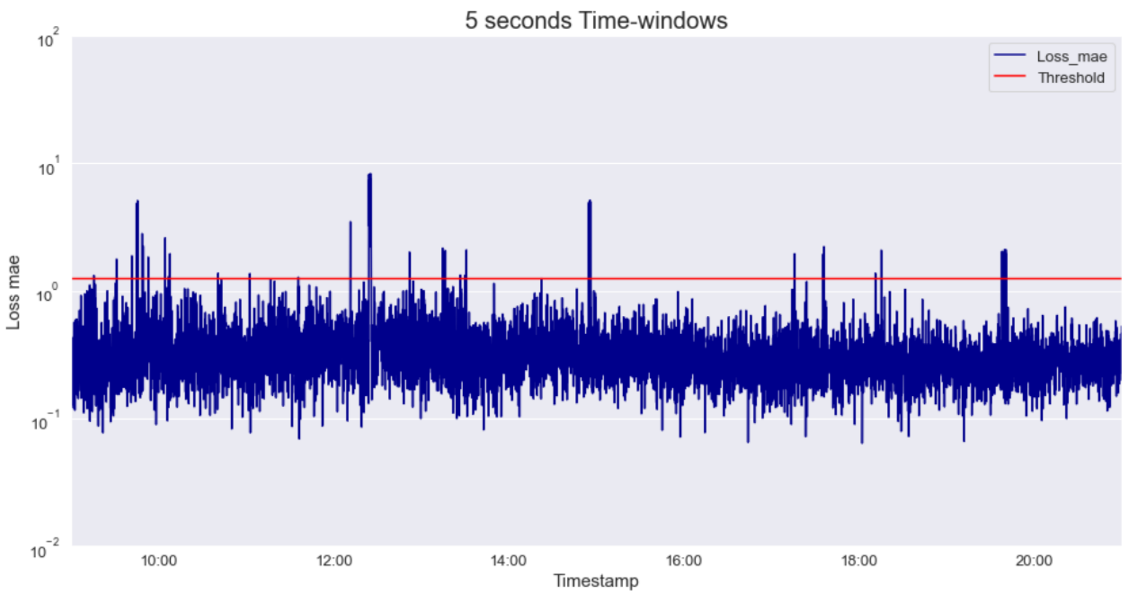


Figure 4.8: AE output on 5 seconds time windows.

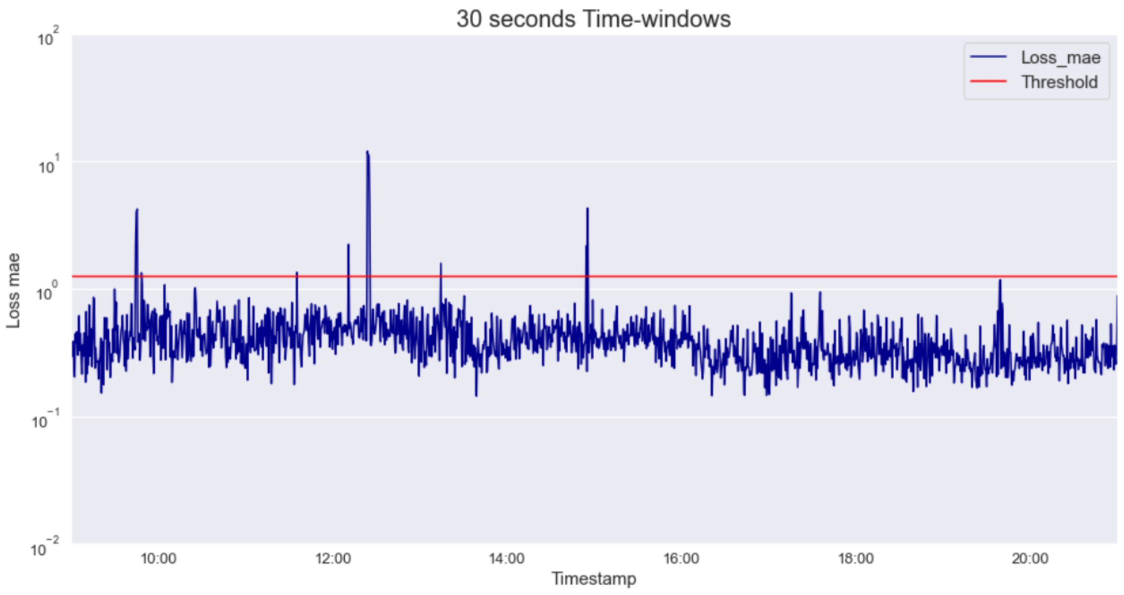


Figure 4.9: AE output on 30 seconds time windows

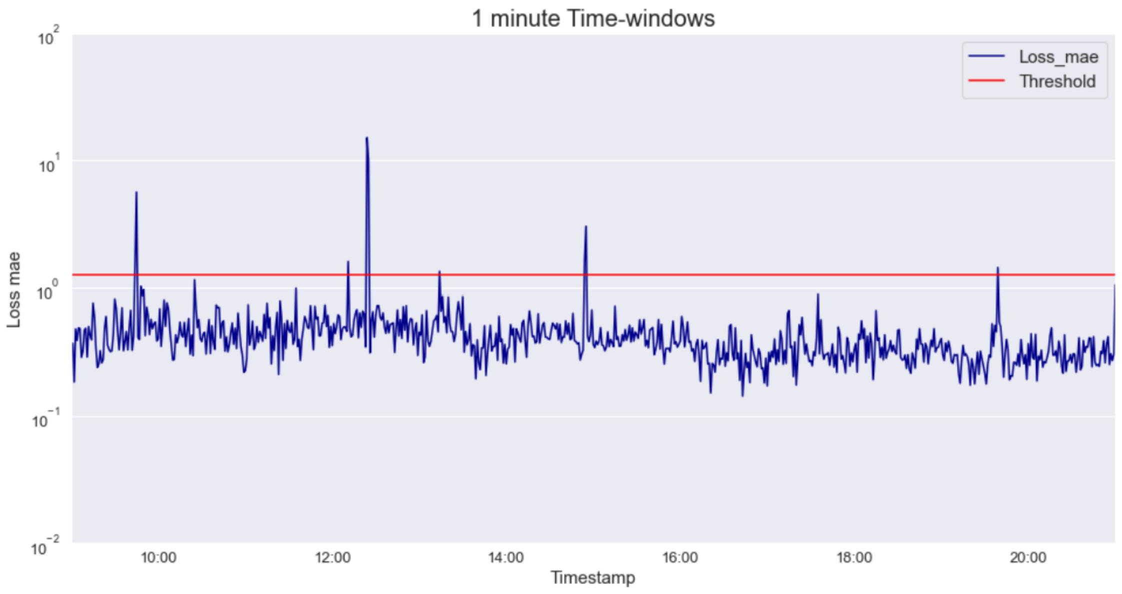


Figure 4.10: AE output on 1 minute time windows.

5
Discussion

In this section, we discuss the results that were previously presented among the different approaches that have been used. Furthermore, future work of this project is discussed.

5.1 Approaches & Results

Based on the results provided in the section above, we can see that the LSTM-AE, in most of the cases, outperformed the rest of the models. The only case where the Autoencoder did not achieve the highest scores was the SKAB dataset. LOF in yielded on average about 5% better results on the SKAB dataset. The highest difference could be seen on the case where data was normalized and not divided into time windows. We suspect that this phenomenon is caused by the transformation of the data, which is an adjustment in the anomaly rate. It can be seen that for all the other datasets, Local Outlier Factor is the most poorly performing model, therefore we believe that the structure of the data after such transformation could be possibly better for the nature of this density based algorithm. Because the performance of LOF on this particular case drew our attention, we also performed one additional test where we used original structure of SKAB dataset. From that, we could see that LOF performed much worse comparing to LSTM-AE. Based on that, we could ensure our reasoning on choosing LSTM-AE as our final model that was used for IIS log dataset.

In Section 3.3, we explained the reasoning behind the choice of values for the parameters and variables that are connected to the anomaly rate of the datasets. One such parameter is the *contamination* in LOF and IF models. When it comes to the PCA model, the cutoff variable is connected to the estimation of the anomaly rates of the datasets. For the AE model, the classification threshold that decides on whether a data point is anomalous is also connected to our estimation of the anomaly rate. In case we would like to measure the best performance of the models, using the information provided by the labels of the datasets, we would compute their anomaly rates and adjust the parameters of the models accordingly. This approach would lead to better performance results, however, it could not be applied in a real-life scenario of unsupervised learning, where the available data is unlabeled. Experimentally estimating the anomaly rate of the dataset of the company would require relevant information from the experts, which, in most cases, is hard to be found. When it comes to the publicly available datasets, we empirically approached the anomaly rate to 0.01. Therefore, the models are going to expect a lower number of anomalies than the actual number for three datasets and a higher number of anomalies for the other one. We could experimentally estimate the anomaly rates of the datasets by running the models for different values of anomaly rate and observing their performance results. However, this would require the use of the available labels in order to compute the performance results. Previously, we mentioned the reason why we do not want to make use of the labels, therefore, we excluded this option of experimentally approaching the anomaly rate individually for every dataset.

In the case of the CCFD dataset, we are dealing with a problem commonly known as extreme rare event classification. This task refers to highly unbalanced datasets, where the positive class makes up less than 1% of the total data points. This type of task is generally difficult to deal with. Deep learning has been extensively used in extreme rare event classification in the recent years [59]. Nevertheless, LSTM-AE still got some reasonably good results considering the above mentioned circumstances.

Having gathered all the results of the performed variations of tests, we can see that preprocessing of the data, which is standardization and normalization, has little impact on the model's performance. The only significant difference in the results between normalization and standardization can be observed on the performance of LSTM-AE applied on the CCFD dataset. Nonetheless, the majority of the tests performed better when the data was standardized.

Regarding the consistency check that was described in Section 4.5, we can observe that the selected AE model contains a degree of unreliability, since its performance depends on the anomaly rate of the dataset. However, this observation is valid for the rest of the models as well. Even though the AE model is inconsistent in that matter, it still yielded better performance results compared to the rest of the models. Therefore, after performing this consistency check, we became more confident of our choice of model, since the AE model had the highest performance results when tested on the ten subsets of the test set.

In terms of the results of the dataset of the company, we can conclude that LSTM-AE was successful in detecting some of the actual anomalies of the system. The developed model was able to properly describe the time periods during which the anomaly occurred, reducing the amount of time required to identify the exact causes of the issue. For the company, we were able to provide the file with all the applications that occurred in that precise time after establishing specified time windows of anomaly emerging. Although our model was unable to identify the particular application that caused the reported anomaly, we were able to reduce the number of applications that needed to be investigated, saving time from the experts of the company.

From the provided outputs of the model for 3 different time windows divisions, it can be seen that as the time window size increases, the number of peaks in the graph decreases. We can clearly observe 6 peaks that cross the defined threshold for the largest time window, that is 1 minute, but only 4 of them, between 12:00 and 16:00, are actual anomalies. We can conclude, based on our investigation of the results, that codes of levels 3, 4, and 5 have the biggest impact on the corresponding value of the computed loss, where codes 4 and 5 are the messages defined as error messages on the client and server sides, respectively.

5.2 Future Work

The work that can be done in order to improve and evolve this project includes the steps that are described below.

First of all, we find it reasonable, that the amount of data we fit in the model can affect its performance. Therefore, in the future we want to fit the model with data that corresponds to more than one day of continuously produced log material. This would require a bigger capacity of hardware resources. Apart from that, we would need to know when a period of normal behavior of the system occurred for some continuous days. This is because we want to train the model on a bigger amount of anomaly-free data and test it on data that might include anomalies.

Furthermore, another step of future work would be to modify the model, so that it would identify the exact applications that triggered the anomalies. This would ease the process of its evaluation and let the company know which applications require extra attention or updates. In order to do that, another approach of preprocessing the data would be needed. To specify the exact application that was called in certain log, the division of the dataset into time windows through the aggregation of the multiple data rows would not be applicable anymore. Therefore, the selection of new features would be necessary.

The investigation of different models and the modification of the current ones would be one more step that can be done in future work. More specifically, modifying the architecture of the AE model could possibly lead to a change in the performance of the model. When it comes to applying different models, our next step would be to try the use of a different architecture of the Autoencoder model, which would include convolutional layers.

Lastly, the development of a User Interface (UI) would allow the model to be used by more people that are not relevant to Python programming. A user-friendly application of anomaly detection could be beneficial for any organization that keeps track of the logs that are produced by its software, as its use would require little training from its technical engineers or other employees who have little knowledge about the subject.

6
Conclusion

This project investigated the use of four unsupervised machine learning models for the task of anomaly detection. Four models were used, namely Local Outlier Factor, Isolation Forest, Principal Component Analysis and LSTM-Autoencoder. For the purposes of this thesis, we made use of four publicly available datasets as well as a dataset provided by Centiro, the company that the project was in collaboration with. The datasets derived from various anomaly detection related backgrounds, sharing some common characteristics which made possible the comparison among the performances of the models on these datasets. The reason for using public datasets lies mainly in the fact that the dataset of the company was not annotated. Because of that, the models could not be tested in order to compare their performances. Therefore, the models were first tested on the four public datasets, namely SWaT, SKAB, SMD and CCFD. Then, the model with the best and most stable performance on these datasets was utilized in order to detect anomalies on the dataset of Centiro. Two variations of data scaling were used on the datasets and these are standardization and normalization. Also, most of the datasets were further divided into time windows.

Performing the above described tests, we were able to choose the most successful model, which was the LSTM-Autoencoder. This model outperformed the rest in the majority of tests and, therefore, it was selected as the model to be applied on the dataset of the company. The evaluation scores that the model achieved were not the highest for all datasets, however, the AE model showed some significant differences when compared to the rest, in terms of combining performance stability and good results. Thus, we found it reasonable to use it for the final experiment. In addition, based on the performed tests, the anomaly rate of the dataset is observed to have an impact on the performance of the models.

The utilization of the LSTM-AE model in the final tests on the dataset of the company provided us with the time periods of detected potential anomalies. After the investigation of these time periods from the experts of the company, we were able to evaluate the performance of the model. The average yielded Precision of the model was about 66%, a result that we find reasonably good.

Regarding the use of *sc-status* feature of the ISS logs, it was proved to be a reasonable choice for finding patterns that could lead the model to detect abnormalities in the system. However, the investigation of different features as well as preprocessing techniques would be considerable for future continuation of this project.

Bibliography

[1] Omar, S., Ngadi, A. and Jebur, H.H., 2013. Machine learning techniques for anomaly detection: an overview. International Journal of Computer Applications, 79(2). URL https://www.researchgate.net/profile/Salima-Benqdara/publication/325049804_Machine_Learning_Techniques_for_Anomaly_Detection_An_Overview/links/5af3569b4585157136c919d8/Machine-Learning-Techniques-for-Anomaly-Detection-An-Overview.pdf

[2] Zhu, J., He, P., Fu, Q., Zhang, H., Lyu, M.R. and Zhang, D., 2015, May. Learning to log: Helping developers make informed logging decisions. In 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (Vol. 1, pp. 415-425). IEEE. URL https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7194593

[3] Jiang, T., Gradus, J.L. and Rosellini, A.J., 2020. Supervised machine learning: a brief primer. Behavior Therapy, 51(5), pp.675-687. URL https://www.sciencedirect.com/science/article/pii/S0005789420300678#!

[4] Mahesh, B., 2020. Machine learning algorithms-a review. International Journal of Science and Research (IJSR).[Internet], 9, pp.381-386. URL https://www.researchgate.net/profile/Batta-Mahesh/publication/344717762_Machine_Learning_Algorithms_-A_Review/links/5f8b2365299bf1b53e2d243a/Machine-Learning-Algorithms-A-Review.pdf

[5] Mandagondi, L.G., 2021. Anomaly Detection in Log Files Using Machine Learning Techniques. URL https://www.diva-portal.org/smash/get/diva2:1534187/FULLTEXT02

[6] Sathya, R. and Abraham, A., 2013. Comparison of supervised and unsupervised learning algorithms for pattern classification. International Journal of Advanced Research in Artificial Intelligence, 2(2), pp.34-38. URL https://www.researchgate.net/publication/273246843_Comparison_of_Supervised_and_Unsupervised_Learning_Algorithms_for_Pattern_Classification

[7] Granlund, O., 2019. Unsupervised anomaly detection on log-based time series data. URL https://www.diva-portal.org/smash/get/diva2:1377830/FULLTEXT01.pdf

[8] Reddy, Y.C.A.P., Viswanath, P. and Reddy, B.E., 2018. Semi-supervised learning: A brief review. Int. J. Eng. Technol, 7(1.8), p.81. URL https://www.researchgate.net/profile/Eswara-B/publication/324050146_Semi-supervised_

learning_a_brief_review/links/5b5c02f8458515c4b24e2b15/Semi-supervised-learning-a-brief-review.pdf

[9] Wirehed, A. and Suhren Gustafsson, A., 2021. Log Classification using NLP Techniques Data-Driven Fault Categorization of Multimodal Logs using Natural Language Processing Techniques. URL https://odr.chalmers.se/bitstream/20.500.12380/302416/1/Masters_Thesis_Adam%20Wirehed%20och%20Adam%20Suhren%20Gustafsson%20210604.pdf

[10] Khalid, S., Abbas, H., Pasha, M. and Raza, A., 2012. Securing Internet Information Services (IIS) configuration files. International Conference for Internet Technology and Secured Transactions, Retrieved from: https://ieeexplore.ieee.org/abstract/document/6470913

[11] G2 Crowd. "What is Graylog?" Retrieved from: https://web.archive.org/web/20190326193941/https://www.g2crowd.com/products/graylog/details

[12] InfoWorld. "10 Splunk alternatives for log analysis". URL https://www.infoworld.com/article/3063614/10-splunk-alternatives-for-log-analysis.html

[13] Cheng, Z., Zou, C. and Dong, J., 2019, September. Outlier detection using isolation forest and local outlier factor. In Proceedings of the conference on research in adaptive and convergent systems (pp. 161-168). URL https://dl.acm.org/doi/pdf/10.1145/3338840.3355641

[14] Alghushairy, O., Alsini, R., Soule, T. and Ma, X., 2020. A Review of Local Outlier Factor Algorithms for Outlier Detection in Big Data Streams. Big Data Cogn. Comput. 2021, 5, 1. URL https://doi.org/10.3390/bdcc5010001

[15] Local outlier factor (2021) Wikipedia. URL https://en.wikipedia.org/wiki/Local_outlier_factor

[16] Liu, F.T., Ting, K.M. and Zhou, Z.H., 2008, December. Isolation forest. In 2008 eighth ieee international conference on data mining (pp. 413-422). IEEE. URL https://ieeexplore.ieee.org/document/4781136

[17] Hariri, S., Kind, M.C. and Brunner, R.J., 2021, April. Extended Isolation Forest. In IEEE Transactions on Knowledge and Data Engineering, IEEE. URL https://ieeexplore.ieee.org/abstract/document/8888179

[18] Lee, Y.J., Yeh, Y.R. and Wang, Y.C.F., 2012. Anomaly detection via online oversampling principal component analysis. IEEE transactions on knowledge and data engineering, 25(7), pp.1460-1470. URL https://ieeexplore.ieee.org/document/6200273

[19] Principal component analysis (February 2022) Wikipedia. URL https://en.wikipedia.org/wiki/Principal_component_analysis

[20] Imayakumar, A.A., Dubey, A. and Bose, A., 2020, February. Anomaly detection for primary distribution system measurements using principal component analysis. In 2020 IEEE Texas Power and Energy Conference (TPEC) (pp. 1-6). IEEE. URL https://ieeexplore.ieee.org/document/9042509

[21] Patel, A., Hands-On Unsupervised Learning Using Python, O'Reilly. URL https://www.oreilly.com/library/view/hands-on-unsupervised-learning/9781492035633/ch04.html

[22] Powell, V. and Lehe, L., 2015. Principal component analysis. URL https://setosa.io/ev/principal-component-analysis/

[23] Lee, D., 2017, December. Anomaly detection in multivariate non-stationary time series for automatic DBMS diagnosis. In 2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA) (pp. 412-419). IEEE. URL https://ieeexplore.ieee.org/abstract/document/8260666

[24] Sakurada, M. and Yairi, T., 2014, December. Anomaly detection using autoencoders with nonlinear dimensionality reduction. In Proceedings of the MLSDA 2014 2nd workshop on machine learning for sensory data analysis (pp. 4-11). URL https://dl.acm.org/doi/abs/10.1145/2689746.2689747

[25] Zhou, C. and Paffenroth, R.C., 2017, August. Anomaly detection with robust deep autoencoders. In Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining (pp. 665-674). URL https://dl.acm.org/doi/abs/10.1145/3097983.3098052

[26] Mohammadi, M., Al-Fuqaha, A., Sorour, S. and Guizani, M., 2018. Deep learning for IoT big data and streaming analytics: A survey. IEEE Communications Surveys Tutorials, 20(4), pp.2923-2960. URL https://ieeexplore.ieee.org/abstract/document/8373692

[27] Vanishing gradient problem (February 2022) Wikipedia. Retrieved from: https://en.wikipedia.org/wiki/Vanishing_gradient_problem

[28] Lindemann, B., Maschler, B., Sahlab, N. and Weyrich, M., 2021. A survey on anomaly detection for technical systems using LSTM networks. Computers in Industry, 131, p.103498. URL https://www.sciencedirect.com/science/article/pii/S0166361521001056

[29] Christopher Olah. Understanding LSTM Networks. 2015. URL: http://colah.github.io/posts/2015-08-UnderstandingLSTMs/.

[30] Berenji Ardestani, S., 2020. Time Series Anomaly Detection and Uncertainty Estimation using LSTM Autoencoders. URL https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1468763&dswid=997

[31] Greff, K., Srivastava, R.K., Koutník, J., Steunebrink, B.R. and Schmidhuber, J., 2016. LSTM: A search space odyssey. IEEE transactions on neural networks and learning systems, 28(10), pp.2222-2232. URL https://ieeexplore.ieee.org/abstract/document/7508408

[32] Lee, T.J., Gottschlich, J., Tatbul, N., Metcalf, E. and Zdonik, S., 2018. Precision and recall for range-based anomaly detection. arXiv preprint arXiv:1801.03175. URL https://arxiv.org/abs/1801.03175=

[33] F-score (March 2022) Wikipedia. Retrieved from: https://en.wikipedia.org/wiki/F-score

[34] Landauer, M., Wurzenberger, M., Skopik, F., Settanni, G. and Filzmoser, P., 2018. Dynamic log file analysis: An unsupervised cluster evolution approach for anomaly detection. computers security, 79, pp.94-116. URL http://www.sciencedirect.com/science/article/pii/S0167404818306333

[35] Ahmad, S., Lavin, A., Purdy, S. and Agha, Z., 2017. Unsupervised real-time anomaly detection for streaming data. Neurocomputing, 262, pp.134-147. URL https://www.sciencedirect.com/science/article/pii/S0925231217309864

[36] Tharshini, M., Ragavinodini, M. and Senthilkumar, R., 2017, December. Access log anomaly detection. In 2017 Ninth International Conference on Advanced Computing (ICoAC) (pp. 375-381). IEEE. URL https://ieeexplore.ieee.org/abstract/document/8441194

[37] He, S., Zhu, J., He, P. and Lyu, M.R., 2016, October. Experience report: System log analysis for anomaly detection. In 2016 IEEE 27th international symposium on software reliability engineering (ISSRE) (pp. 207-218). IEEE. URL https://ieeexplore.ieee.org/abstract/document/7774521

[38] Audibert, J., Michiardi, P., Guyard, F., Marti, S. and Zuluaga, M.A., 2020, August. USAD: unsupervised anomaly detection on multivariate time series. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery Data Mining (pp. 3395-3404). URL https://dl.acm.org/doi/abs/10.1145/3394486.3403392

[39] MACHINE LEARNING GROUP - ULB, Kaggle, Credit Card Fraud Detection. Retrieved from: https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud

[40] Singapure University of Technology, iTrust Centre for Research in Cyber Security, Secure Water Treatment (SWaT). URL https://itrust.sutd.edu.sg/itrust-labs_datasets/dataset_info/

[41] NetManAIOps, OmniAnomaly, GitHub. Retrieved from: https://github.com/NetManAIOps/OmniAnomaly

[42] Waico Inc., SKAB, GitHub. Retrieved from: https://github.com/waico/SKAB

[43] Suneetha, K., Krishnamoorthi, R. 2009. Identifying User Behavior by Analyzing Web Server Access Log File. IJCSNS International Journal of Computer Science and Network Security. URL https://www.researchgate.net/publication/255583124_Identifying_User_Behavior_by_Analyzing_Web_Server_Access_Log_File

[44] Lakshmanan, S. How, When, and Why Should You Normalize / Standardize / Rescale Your Data?. 2019. URL https://towardsai.net/p/data-science/how-when-and-why-should-you-normalize-standardize-rescale-your-data

[45] Kumpulainen, P., Kylväjä, M. and Hätönen, K., 2009, September. Importance of scaling in unsupervised distance-based anomaly detection. In Proceedings of IMEKO XIX World Congress. Fundamental and Applied Metrology (pp. 6-11).

[46] Su, Y., Zhao, Y., Niu, C., Liu, R., Sun, W. and Pei, D., 2019, July. Robust anomaly detection for multivariate time series through stochastic recurrent neural network. In Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery data mining (pp. 2828-2837).

[47] Shanker, M., Hu, M.Y. and Hung, M.S., 1996. Effect of data standardization on neural network training. Omega, 24(4), pp.385-397.

[48] Scikit-learn (2017) sklearn.neighbors.LocalOutlierFactor URL https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.LocalOutlierFactor.html

[49] Xu, Z., Kakde, D. and Chaudhuri, A., 2019. Automatic Hyperparameter Tuning Method for Local Outlier Factor, with Applications to Anomaly Detection, 2019 IEEE International Conference on Big Data

(Big Data), 2019, pp. 4201-4207, doi: 10.1109/BigData47090.2019.9006151. URL https://ieeexplore.ieee.org/abstract/document/9006151?casa_token=eUlbpDSojxEAAAAA:2aGAbgcd2hivYedzQ_D5ocyX8jfNmjRDx_Oc_HR2f9ZeUAKEWKXoyWt3qq7bS-dpA0yngAoZ

[50] Scikit-learn (2017) sklearn.ensemble.IsolationForest URL https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html

[51] Sharma, L. N., Danadapat, S., Mahanta, A. (2012). Multiscale PCA based quality controlled denoising of multichannel ECG signals. International Journal of Information and Electronics Engineering, 2(2), 107-111. URL http://www.ijiee.org/papers/63-I080.pdf

[52] Brown, J. D. (2009). Statistics Corner. Questions and answers about language testing statistics: Choosing the right number of components or factors in PCA and EFA. Shiken: JALT Testing Evaluation SIG Newsletter, 13(2), 19-23. URL https://hosted.jalt.org/test/PDF/Brown30.pdf

[53] Harris, C.R., Millman, K. J., van der Walt, S., J., et al., 2020, September. Array programming with NumPy. URL https://www.nature.com/articles/s41586-020-2649-2.pdf

[54] pandas (software) (March 2022), Wikipedia. URL https://en.wikipedia.org/w/index.php?title=Pandas_(software)&oldid=1002320935

[55] Bisong, E., 2019, September, Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners. URL https://link.springer.com/chapter/10.1007/978-1-4842-4470-8_12

[56] scikit-learn (January 2022) Wikipedia. URL https://en.wikipedia.org/wiki/Scikit-learn

[57] TensorFlow (May 2022) Wikipedia. URL https://en.wikipedia.org/wiki/TensorFlow

[58] Ketkar, N., 2017, Deep Learning with Python: A Hands-on Introduction URL https://link.springer.com/chapter/10.1007/978-1-4842-2766-4_7

[59] Ranjan, C. Extreme Rare Event Classification using Autoencoders in Keras. 2020 URL: https://processminer.com/autoencoders-in-keras/