DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING, SECOND CYCLE, 30 CREDITS

STOCKHOLM, SWEDEN 2021

Anomaly Detection in Computer Networks

POLYXENI IOANNIDOU

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Anomaly Detection in Computer Networks

POLYXENI IOANNIDOU

Master in Computer Science
Date: May 10, 2021
Supervisor: Borja Rodriguez Galvez
Examiner: Ragnar Thobaben
School of Electrical Engineering and Computer Science
Host company: CERN, the European Organization for Nuclear Research
Swedish title: Avvikelse upptäckt i datornätverk

Acknowledgements

I would like to thank my supervisor from KTH Mr. Galvez for all the discussions, and the insightful feedback to all of my problems. His support and experience were of great help during the whole time. Further thanks to my supervisor from CERN, Tommaso Colombo, for giving me the opportunity to conduct my degree project in CERN, where I did also my studentship in "Networks Engineering" next to him. I would also like to thank my colleague, Efstathios Tantalidis, from CERN for his valuable comments in code optimization and debugging.

Finally, I would like to specially thank my parents and my friends for supporting me and making me feel better in hard times.

Abstract

In this degree project, we study the anomaly detection problem in log files of computer networks. In particular, we try to find an efficient way to detect anomalies in our data, which consist of different logging messages from different systems in CERN's network for the LHC-b experiment.

The contributions of the thesis are double: 1) The thesis serves as a survey on how we can detect threats, and errors in systems that are logging a huge amount of messages in the databases of a computer network. 2) Scientists in the LHC-b experiment make use of the Elasticsearch, which is an open source search engine and logging platform with great reputation, providing log monitoring, as well as data stream processing. Moreover, the Elasticsearch provides a machine learning feature that automatically models the behavior of the data, learning trends, and periodicity to identify anomalies. Alternatively to the Elasticsearch machine learning feature, we build, test and evaluate some machine learning models that can be used for the same purpose from the scientists of the experiment. We further provide results that our models generalize well to unseen log messages in the database.

Sammanfattning

I detta examensarbete studerar vi problemet med att upptäcka avvikelser i loggfiler från ett datanätverk. Specifikt försöker vi hitta ett effektivt sätt att upptäcka avvikelser i datan, som består av olika loggningsmeddelanden från olika system i CERNs nätverk för LHC-b-experimentet.

Avhandlingens dubbla bidrag är: 1)Avhandlingen kan anses som en undersökning om hur vi kan upptäcka hot och fel i system som loggar en enorm mängd meddelanden i databaser från ett datanätverk. 2) Forskare i LHC-b-experimentet använder sig av Elasticsearch, som är en sökmotor och loggningsplattform med öppen källkod och ett avsevärt rykte, som tillhandahåller loggövervakning och automatisk datahantering. Dessutom är Elasticsearch försedd med en maskinlärningsfunktion som automatiskt modellerar beteenden med hjälp av data, trender och periodicitet för att identifiera avvikelser. Vi bygger, testar och utvärderar ett fåtal maskininlärningsmodeller som ett alternativt till Elasticsearch maskininlärningsfunktion. Forskarna i experimentet kan använda maskininlärningsmodellerna till samma ändamål som Elasticsearch maskininlärningsfunktion. Vi presenterar också resultat som visar att våra modeller generaliserar väl för osedda loggmeddelanden i databasen.

Contents

Chapter 1

Introduction

1.1 Problem Description

In recent years there is a huge demand of monitoring and controlling log messages in large-scale systems, which generate a massive flow of various information. One intended goal of collecting and monitoring these log messages is to detect instances that deviate from the majority of the data; these are called “anomalies“. Anomalous data usually arouse suspicion that systems have been attacked or are clearly malfunctioning. Consequently, the isolation of these instances can comprehensively enhance the stability and robustness of large-scale systems and computer networks. This problem is defined as “Log Anomaly Detection“, for which there is a proliferation of deep learning networks and classical machine learning models that can solve it.

1.2 Motivation

Over the years the “Anomaly Detection“ problem has been studied extensively by many researchers, and thus there is a variety of different approaches that perform successfully the detection and isolation of anomalous data. Therefore, we find our motivation to solve this problem for CERN’s computer network by studying similar surveys on “Log Anomaly Detection“ and trying to give our own approach working on a complete new dataset that has not been tested before. We used both deep learning and classical machine learning models, for which we provide an analysis of the results.

Deep learning techniques are a subset of machine learning algorithms that achieve improved performance by learning more complex patterns of large-

scaled data using neural networks with several hidden layers. Furthermore, deep learning techniques have become very popular while working on anomaly detection problems, as well as many surveys have proved that by using deep learning models we are able to solve several different kind of problems in science. The aim of this project is two-fold: $1)$ we describe and analyse important models that have been used to solve the problem of anomaly detection, and $2)$, we utilise and asses the performances of both machine learning and deep learning models. Moreover, we propose a hybrid model based on both machine learning models and deep neural networks.

1.3 Applications of Anomaly Detection

In this section, we refer to the diverse applications of anomaly detection, as explained by Chalapathy et. al. in [1]. As we mentioned above, it has been proved that deep anomaly detection techniques can solve this problem in many different applications, some of which are the followings:

- “Intrusion Detection", where system operators need to detect malicious activities in computer networks. In Intrusion Detection Systems (IDS) administrators are able to monitor and analyse network communication, as a consequence of monitoring, and thus detection of suspicious activities and intrusions [2].

- “Fraud Detection"where we need to detect illegal acts of deceit to access private resources. In particular, frauds are issues of high importance in banking and telecommunications, since they cause substantial privacy problems for governments and private businesses [1].

- “Log Anomaly Detection", where flows of various log messages can indicate the instability and deterioration of large-scale systems. In principle, in such systems, log messages have a structured format with domain-specific expressions, and hence we are able to detect system failures by pattern recognition. However, unstructured data impose suggestive challenges to log anomaly detection [1, 3].

- “Anomaly detection in Social Networks", where we detect behavioral patterns of users within a social network that indicate intentional actions of deception. Hence, such users might be proceed to sexual exploitation or solicitation, hate speech, bullying, harassment, intellectual property violation or urge for suicide, among others [1].

- “Medical Anomaly Detection", where the identification of anomalies assist the diagnosis of serious health problems, as well as the prevention of such conditions [1].

1.4 Research Question

Based on the above description of the “Log Anomaly Detection“ problem, as well as the significance of its applications, we aim to answer the following research question:

“Can we detect threats in CERN’s network using machine learning ?"

More specifically, by applying deep learning and machine learning models in the collected datasets from CERN’s network, we aim to investigate if we could be able to detect actual threats in its systems.

1.5 Outline

This thesis is organized in 6 chapters with the current chapter followed by Chapter 2, which provides a detailed description of the background theory related to “Anomaly Detection“, that will be used as a guidance for the reader to understand our approach. After that, Chapter 3 presents a description of our methodology with respect to the data collection and preprocessing, the implemented models, as well as the different evaluation metrics. Then, Chapter 4 includes all the results from our experiments followed by a qualitative and quantitative analysis. Finally, in Chapter 5 and Chapter 6 we provide, respectively, our interpretation of the results and our conclusions about the different models that we used.

1.6 Ethics and Sustainability

In recent years, with the advent of “Artificial Intelligence“ and “Machine Learning“ a raising concern about them has become widespread. What is remarkable is that eminent scientists and researchers have raised considerable concerns about AI and ML shedding humans, or in some extreme cases subjugating them. Apparently, the main reason is that an expanding amount of functions and tasks have already been ceded to machine learning models to the detriment

of humans. As regards our work in this project, we would like to state that the results do not prove that any scientist or system operator in CERN's network could be replaced by our proposed models. Even though we conclude in the end of this report that machine learning models can successfully learn structured data patterns so as to detect deviations in specific logs, we still need the on-site support of the scientists for two reasons. First of all, we already mentioned that the log messages have a structured format, which encompasses domain-specific regular expressions. These regular expressions have been defined by system administrators and operators, who have received an adequate training about handling and formulating the logging systems properly. Secondly, our neural networks are dependent on those system operators, because we make use of labeled data, which have been created in collaboration with system experts.

As concerns the need of sustainability, it is a crucial issue that should be taken into consideration with respect to all the aforementioned fears of eminent scientists and entrepreneurs about the increasing use of AI and ML models. A system or a model is sustainable if its input and output cannot cause any harassment in the environment. Considering how our approach in anomaly detection can preserve sustainability, we could confirm that our approach does not violate any system's integrity or private policy.

In overall, our implementation needs system experts in the loop, who will supervise the proposed models with respect to the labeled data that are already available in the logging system. Therefore, humans can control unconscious side-effects by monitoring the "True Positive Rate" and the "False Positive Rate" in the validation phase as regards the supervised and semi-supervised approaches. Hence, we can handle situations that false alarms are raised in the system by the models' predictions. All in all, we need to always take into consideration that AI systems should not be totally independent from humans, because humans can intuitively handle difficult situations and control unwanted results while working with AI systems, which is a condition that also applies in our case.

Chapter 2

Background

2.1 Anomaly Detection

2.1.1 What is Anomaly Detection?

Anomaly Detection is the process of identification of data points that significantly deviate from the majority of samples in a dataset. Such data points are known as *anomalies*, and the identification and isolation of these can comprehensively enhance the applications' stability and robustness. In recent years, there has been a proliferation of deep learning and machine learning models that can successfully solve the problem of anomaly detection. Overall, we will give a concise overview of the different aspects of anomaly detection models, as well as we will briefly refer to the different models that can solve this problem, as discussed in [1] and [4].

2.1.2 Anomalies vs Novelties

In the previous section, we referred to those points that are usually located far away from the bulk of the data samples, and hence are considered as anomalous points or anomalies. The existence of such instances may indicate malicious events, systems failure, and/or illegal intrusions, since we observe features and patterns in those samples that differ substantially from the majority of instances, which are considered as normal data. On the other hand, there are cases where isolated data points are not a result of a malicious action, or a suspicious event. In these cases, we detect novel or unobserved features in the dataset and we add these instances to the regular points and we call them *novelties*.

2.1.3 Different Aspects of Anomaly Detection Models

The selection of a machine learning algorithm, e.g. a deep neural network (DNN), that could solve the problem of anomaly detection is affected by many factors, and some of the most important are: (a) the nature of the input samples, (b) the availability of labeled data, (c) the type of anomalies, (d) the predictions of the model, and (e) the objective functions used in the training phase, as discussed in [1], [4] and [5].

Input Data

The choice of a machine learning algorithm is decisive with respect to the nature of the input data. There are two types of data, based on the inter-relationships of the data points: (a) sequential, where samples taken at a time t are dependent of previous samples, like time-series, speech, and text, and (b) non-sequential, where there is no dependency between the samples based on the time they were taken, like images. A simple example is that when we work with images we prefer CNN networks, whereas with text or time-series we prefer Transformers, LSTMs, or RNN models [6].

Type of Anomalies

Anomalies can be categorized into three groups: point anomalies, contextual anomalies, and group anomalies. The first type refers to a single event that signifies an abnormality, and thus is considered as fraudulent. In the second type we consider an event as an abnormality with respect to a specific context only, whereas in the third type of anomalies we refer to the collective ones regarding cases where anomalies are not in the presence of a single irregularity, but whenever they formulate groups.

Predictions of the Model

Another aspect in the process of designing the architecture of the anomaly detection model, is to define the form of the output. In particular, either we design a model that outputs an anomaly score, and thereby we classify anomalous points based on that value, or we output binary labels for clean and fraud predicted points.

Labeled Data

The availability of labels for both normal and abnormal instances, designates the learning procedure of a model [1]. In particular, if we have labels for every sample in the dataset, denoting if it is a normal instance or an anomaly, we follow a supervised learning procedure, and thus we have a classification problem that can be solved by using models like logistic regression, decision trees, or a multilayer perceptron. On the other hand, the absence of labeled data for all categories leads to an unsupervised learning procedure. In these cases, we should, for instance, apply clustering techniques, in order to identify the presence of common patterns among the data that could indicate the presence of clusters. Such techniques include isolation forests [7], k-Means [8], Boltzmann machines [9], autoencoders [10], or deep belief networks [9]. A third type of learning procedure is the semi-supervised, in which we obtain labels only for the bulk of normal data. A classical approach in this case is to employ autoencoders and their variations, since we can train them to learn what is a normal behavior by reconstructing only the normal samples, and thus anything that deviates from that would have a high reconstruction error indicating an anomalous behavior.

Training Objectives

The last aspect that we take into consideration is the training objective of the different techniques, as mentioned in [11] and [1]. The first category of models is used in cases where we work only with positive instances, or else one-class samples, due to the fact that in many cases our dataset includes only normal points, i.e we do not know all the different kind of anomalies that might occur in the future, or we have not collected any fraud instance. Hence, we extract the important features to generate rich vector representations, and then we use a one-class objective with the aim of finding an optimised hyperplane around normal points. In other words, vector representations in the latent space are driven by the one-class objective.

The other category includes hybrid deep learning models, in which a deep neural network (DNN) is combined with a traditional machine learning algorithm. In particular, approaches like these use a deep neural network in the feature extraction phase to generate rich vector representations in the hidden layers, which are then passed in traditional machine learning algorithms, like One-Class SVMs [12] or Decision Trees [13]. A common technique in deep learning is the utilisation of pre-trained models for feature extraction. Therefore,

in the first part of our architecture we could also use other pre-trained models from similar tasks. Furthermore, taking into account the cost of computational resources that we need to train DNNs, especially for large-scale datasets, these pre-trained transfer learning models can be adopted successfully and can also perform satisfactory [11].

2.2 Clustering Methods

Turning now to the different models that can solve the anomaly detection problem, we will give a clear description of both clustering and deep learning models. In this section though, we will focus on the first ones.

2.2.1 Isolation Forests

The Isolation Forest method, as proposed by Fei Tony Liu and others [7], can successfully isolate anomalies by using ensemble isolation trees in order to make partitions of the dataset, where each data point is isolated. The main criterion to decide whether a data point is anomalous or not, is that a normal sample can not be isolated easily, whereas anomalies can be isolated in only a few steps. To be more accurate, the actual definition of "steps" is the path length that we get from an isolation tree for the given sample. In other words, the path length, or else the number of partitions, indicates the detection of a fraud sample.

The basic algorithm includes two main steps, as described in [14] and [15]:

- Training phase: for every sample in the dataset, we build an isolation tree by randomly selecting a feature and randomly partitioning along the range.
- Testing phase: we use the previously trained isolation trees and we pass them all the new data points, for which we calculate an anomaly score as follows:

$$c(m) = \begin{cases} 2H(m-1) - \frac{2(m-1)}{n} & \text{if } m > 2 \\ 1 & \text{if } m = 2 \\ 0 & \text{else} \end{cases} \tag{2.1}$$

where n is the dataset size, m is the dimension of each sample, and H is a harmonic entity. The value of $c(m)$ helps us to find an approximate estimation

of the anomaly score for one sample vector X, as in (2.2), where $E(h(X))$ is the average path length across all trees. Hence, the score is calculated as

$$s(X, m) = 2^{-\frac{E(h(X))}{c(m)}}. \tag{2.2}$$

Based on this score $s(X, m)$, we have the following cases to classify each sample:

1. if $s(X, m) \simeq 1$, then X is an anomaly with high probability,
2. if $s(X, m) < 0.5$, then X is a normal point with high probability, and
3. for any other values close to 0.5, the sample X is unlikely to be an anomalous point.

The main characteristics of the Isolation Forest technique are that (a) it identifies anomalies as those samples with the shortest path lengths, or else the smaller number of partitions, (b) and that it builds multiple isolation trees for different anomalies by random sampling. In this technique, we do not have to isolate all the normal data points, but we can actually isolate the anomalies, which means that such a model can perform better with a small sample set instead of a bigger one; by using a bigger sample set, the normal points could intervene more with the fraudulent ones, which then could affect the ability of the model to isolate the anomalies, as described in [14].

The aforementioned advantages of this model make it ideal when it comes to deal with the effects of swamping and masking; two other issues that we have to consider in anomaly detection problems. The first one refers to those cases when we missclassify normal data as fraudulent ones, while the later one refers to those cases where anomalies are not a minority in our data, which increases again the missclassification rate. The characteristic of randomly sampling with a small sample size helps this model to randomly select a feature and randomly proceed to further partitions along that feature range, which increases the probability for each sub-sample to include different kind of anomalies.

To conclude, Isolation Forest has been extensively studied as a basic method to solve the problem of anomaly detection, as well as it has been shown to be able to succeed in isolating anomalies accurately and efficiently [7].

2.2.2 k-Means and Expectation-Maximisation

The k-Means and Expectation-Maximisation (E-M) are two popular algorithms that can be used for clustering, as explained in [16], [17], and [18]. This makes them similar in the sense that they both follow an iterative procedure to find a structure in a collection of unlabelled data. We should clarify that E-M is similar to k-Means only in the context of Gaussian Mixture Models (GMMs), and thus we compare these two approaches only for this particular case. Although the E-M strategy and k-Means can be compared in terms of clustering in the aforementioned case only, there is still a fundamental difference, which will be clarified in the end of this analysis between these two approaches. However, a more extensive treatment of these algorithms and their relationship can be found in [19].

Firstly, we will explain how we can learn a Gaussian mixture model that allows us to perform soft probabilistic clustering. This Gaussian mixture model is trained by the expectation-maximization procedure. The E-M is a category of learning and/or estimation algorithms, which in the context of clustering involves a model based approach in order to assign observed data into clusters. In particular, we represent each cluster with a Gaussian probability distribution, and then we assign all observations to one cluster based on their probabilities. In principle, each observation belongs to every cluster with different probabilities, as well as the number of clusters specifies the number of Gaussians in the mixture model.

Let us assume that we have a set of n vector observations $\mathcal{X} = \{X_1, ..., X_n\}$, a set of n hidden values $\mathcal{Z} = \{Z_1, Z_2, ..., Z_n\}$, a number of k clusters which also describes the k Gaussian distributions, and a vector of parameters Θ, e.g. the mean and covariance of the distributions. The main objective of the E-M approach is to find the optimal parameters Θ which maximise the likelihood of the data $\mathcal{X}$ in the presence of unknown hidden variables $\mathcal{Z}$, see equation 2.3. If we have completely observed all the data (i.e all variables in $\mathcal{Z}$ are known), we may simplify the problem by estimating the parameters Θ via maximum likelihood. On the other hand, a more challenging variant of the parameters estimation issue comes when the latent variables are unknown.

The log-likelihood that should be maximised is the following:

$$\log\big(P(\mathcal{X}|\Theta)\big) = \log\Big(\mathbb{E}_{\varkappa\sim P(\mathcal{Z})}\big[P(\mathcal{X}, \mathcal{Z} = \varkappa|\Theta)\big]\Big), \tag{2.3}$$

where we marginalised $\mathcal{Z}$ out of the joint probability distribution. Intuitively, this means that we need to find those parameters that maximize the probability of observing all the observations together by taking into consideration that each data point is a mixture of Gaussians.

Even though we usually do not know $\mathcal{Z}$, we can infer some information about it by the posterior $P(\mathcal{Z}|\mathcal{X}, \Theta)$. Therefore, in this point we use the two main stages of the E-M algorithm:

- the **E-step**: hidden variables are estimated by using the current parameters Θ. Thus, we estimate the aforementioned posterior distribution $P(\mathcal{Z}|\mathcal{X}, \Theta)$, which then is utilised to compute the expectation of the log-likelihood (equation 2.3).
- the **M-step**: The parameters Θ are maximised with the aim of maximizing the expectation of the log-likelihood.

These stages are alternated in iterations until the maximum likelihood does not get updated anymore based on a stopping criterion; or else a stopping tolerance.

In the end, a set of k clusters has been created with respect to the optimal parameters Θ that maximise the log probability of the observations [20].

But how is this related with the k-Means algorithm? In the E-M approach, as we explained, we initialise the models' parameters and iteratively we compute the maximum likelihood for our estimates. The key point is that the optimised maximum likelihood is achieved by dividing our problem in simpler sub-problems, where the corresponding estimations of Θ converge to a local maximum of log-likelihood. Similar to the E-M approach, k-Means is an iterative algorithm in the sense that it divides the initial problem to sub-problems to find the optimal solution.

In overall, k-Means is a basic clustering algorithm [8], that tries to partition the data points in non-overlapping clusters iteratively, as we just mentioned above. The main objective is to cluster the data in a way where the intraclass correlation is stronger than the outer class correlation of the clusters; in other words, samples in each one of the clusters strongly resemble each other, whereas different clusters should be kept as different and faraway as possible.

Regarding the work of Likas et. al. [8], and the work of Nazeer in [18], we summarize the basic steps of k-Means algorithm:

1. **Input:** k as the number of desired clusters, and a set of n vector observations $\mathcal{X} = \{X_1, X_2, ..., X_n\}$
2. Select a set of k arbitrary centroids $\mathcal{C} = \{C_1, C_2, ..., C_k\}$
3. $\forall X_i \in \mathcal{X}$ find its distance from each of the k centroids, $C_j \in \mathcal{C}$: $d(X_i, C_j)$, where $1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant k$, using equation 2.4.
4. Select the closest centroid C_j, $\forall X_i \in \mathcal{X}$, which is the $\min d(X_i, C_j)$, and assign all the data points in their closest cluster
5. Recalculate and update the centroids $\forall C_j \in \mathcal{C}$
6. Check termination criterion; exit, else go to 2.
7. **Output:** k clusters with assigned data

At this point, we should exemplify how the data points are being assigned into clusters during the whole iteration procedure. Regarding the work in [18], the k-Means algorithm uses the Euclidean distance to assign the samples of $\mathcal{X} = \{X_1, X_2, ..., X_n\}$ in the k desired clusters. In particular, we find the distance of X_i from C_j, taking into consideration that both X_i and C_j are vectors of m dimensions, by using equation 2.4:

$$d(X_i, C_j) = \sqrt{\sum_{s=1}^{m} (x_{i,s} - c_{j,s})^2}. \tag{2.4}$$

Another important part of the k-Means is that after each iteration we have k clusters with their k chosen centroids for each, and all the training samples have been assigned to the respective clusters. The purpose in this step is to evaluate how well the clusters have been formed after each loop, with the best k centroids in the end of this iterative procedure. Therefore, after a full iteration we minimise a cost function, which is the summation of the distance of all m-dimensional observations inside each j-cluster from their centroids, which is further summed over all k clusters:

$$\mathcal{J} = \sum_{i=1}^{n} \sum_{j=1}^{k} \sqrt{\sum_{s=1}^{m} a_{sj}(x_{i,s} - c_{j,s})^2} \tag{2.5}$$

where:

$$a_{sj} = \begin{cases} 1 & \text{, if } x_{i,s} \in j_{th} \text{ cluster} \\ 0 & \text{else} \end{cases}$$

In overall, the k-Means and E-M algorithms have some similarities, but based on the aforementioned explanations, we can see how they differ. Unlike the E-M algorithm, k-Means computes the actual assignment of the data into clusters, whereas in E-M we only compute the probabilities of each point to be assigned in a cluster. In particular, the k-Means algorithm is a hard distance-based method, whereas the E-M uses statistical methods, in order to find clusters that obtain the maximum likelihood of their parameters.

2.2.3 k-Medians

The strategy of k-Medians does not differ a lot from k-Means. In this section, we will explain their main differences, since the general idea is the same with the one we explained above for k-Means. The first difference is that k-Medians uses the Manhattan distance to assign each point into a cluster:

$$d(X_i, C_j) = \sum_{s=1}^{m} |x_{i,s} - c_{j,s}| \tag{2.6}$$

where X_i is a vector observation from the dataset $\mathcal{X} = \{X_1, X_2, ..., X_n\}$, C_j is an item of the k-centroids $\mathcal{C} = \{C_1, C_2, ..., C_k\}$, and both X_i and C_j are m-dimensional vectors.

A second difference between these two algorithms is that k-Means switches the centroids to the mean value of all of the instances assigned to the corresponding cluster. On the other hand, the k-Medians algorithm switches the centroids to the vector whose elements are the median value of each dimension of all of the instances assigned to the corresponding cluster.

With regards the last difference, k-Medians optimises another cost function, as defined in [21]:

$$\mathcal{J} = \sum_{i=1}^{n} \sum_{j=1}^{k} \sum_{s=1}^{m} a_{js} |x_{i,s} - c_{j,s}|, \tag{2.7}$$

where $a_{js} = 1$ if $x_{i,s}$ is in cluster j and $a_{js} = 0$ otherwise.

2.2.4 One-Class SVM

The SVM algorithm is basically a two-class algorithm [22], where we separate our samples with a hyperplane, for which the perpendicular distance of the closest points from both sides is maximised. However, in the work of Schölkopf

et. al. [12], the One-Class SVM methodology is introduced. In a nutshell, we use this algorithm in cases where we have only positive instances, or else samples from one class, and thus anything else is considered as an outlier and is classified as a negative sample. In other words, the purpose is to separate the data by estimating a decision boundary, which will isolate the outliers in one side, while the larger fraction of the samples will remain on the other. Taking into consideration that we have no labeled negative points, we separate the instances of the one class from the origin. In a sense, the origin is considered as the second class in the One-Class SVM algorithm.

The main difference of the One-Class SVM and the SVM is the formula for the soft and hard-margin separation. The hard margin separation ensures that we avoid all the errors in the training phase, while soft-margin separation tolerates a few errors close to the decision boundaries during that phase. We present the formulas in Table 2.1 and 2.2, as proposed by Goix Nicolas in paper [11].

Geometrically, consider a J-dimensional feature space $\mathcal{F}$, where $\mathcal{F} \subseteq \mathbb{R}^J$ and $J \in \mathbb{N} \cup \{\infty\}$. We are looking to find a hyperplane in $\mathcal{F}$, which can be described by a weight matrix W and a bias vector B. This hyperplane is used to predict a label for each input data:

$$\hat{y} = \text{sign}(W^T X + B) \tag{2.8}$$

Particularly, we want to find the “maximum-margin hyperplane" that separates the data points with respect to the offset of the hyperplane from the origin, which is determined by the parameter $\frac{1}{||W||}$. Concerning the hard-margin separation case (see Table 2.1), there are two different hyperplanes with respect to the matrix W:

$$\text{sign}(X_i W) \tag{2.9}$$

one for One Class-SVM that crosses the origin, and

$$\text{sign}(X_i W + B), \tag{2.10}$$

which does not cross the origin due to offset parameter B.

In the soft-margin separation case (see Table 2.2), ξ is utilised to relax the hard constrain of the separating hyperplane:

$$\text{sign}(X_i W + B) \geqslant 1. \tag{2.11}$$

As a result, every input data point is allowed to slack the aforementioned constraint by a distance $\xi \geqslant 0$. Finally, we build a maximum-margin hyperplane with slacks:

$$\hat{y_i}(W^T X_i + B) \geqslant 1 - \xi_i \tag{2.12}$$

Last but not least, ν is an upper bound on the fraction of outliers and a lower bound on the fraction of support vectors which is also considered in the calculation of the tolerance of this model with respect to the outliers; and ρ describes the margin from the separating hyperplane $\frac{\rho}{||W||}$.

This strategy has two strong advantages; one is that this algorithm works well for detecting the outliers when we have very few or zero "bad" observations, and the other one is that it can capture the non-linearities by using kernel functions. On the other hand, this algorithm fails when the outliers are not the minority in our dataset and can be assigned in dense clusters. In this case, they will be classified as normal points. To avoid this case, we must choose this technique when we are sure that our dataset includes only clean samples.

SVM	One-Class SVM
$\min\limits_{W,B} \frac{1}{2}\|W\|^2$	$\min\limits_{W} \frac{1}{2}\|W\|^2$
$\forall i, y_i(X_i W + B) \geqslant 1$	$\forall i, X_i W \geqslant 1$

Table 2.1: SVM vs. OCSVM (hard-margin separation)

SVM	One-Class SVM
$\min\limits_{W,B,\xi,\rho} \frac{1}{2}\|W\|^2 + \frac{1}{n}\sum_{i=1}^{n}(\xi_i - \nu\rho)$	$\min\limits_{W,\xi,\rho} \frac{1}{2}\|W\|^2 + \frac{1}{n}\sum_{i=1}^{n}(\xi_i - \nu\rho)$
$\forall i, y_i(X_i W + B) \geqslant \rho - \xi_i$, $\xi_i \geqslant 0$	$\forall i, X_i W \geqslant \rho - \xi_i$, $\xi_i \geqslant 0$

Table 2.2: SVM vs. OCSVM (soft-margin separation)

2.2.5 Local Outlier Factor (LOF)

As proposed by Breunig in [23], in this approach we compute an anomaly score, which indicates the deviation of one sample from its neighbors. In other words, this factor is the ratio of the average density of the sample's neighbors, and

its local density. As a result, we identify and separate outliers as those points with low scores or else low local density. Similarly, normal data have the same average density with all their neighbors.

The main characteristic of the LOF algorithm is that a sample is examined in order to be isolated or not with respect to its neighbors [15]. Furthermore, it performs well with large datasets because the more observations we have, the more accurate the deviation values are. In cases with fewer observations, this approach does not detect the deviation observation precisely.

2.2.6 Robust Covariance

The Robust Covariance method uses mathematical models to solve the problem of anomaly detection [24]. First of all, we assume an n-sized set $\mathcal{X} = \{X_1, ..., X_n\}$ of m-dimensional vector observations $X_i = \{x_{i1}, x_{i2}, ..., x_{im}\}$ based on the aforementioned notations. Furthermore, classical metrics of scatter and location are described by the mean μ and covariance matrix Σ. This model utilises the Mahalanobis distance (see equation 2.13) to assess the deviation of each sample from the centre of the distribution.

$$d(X_i, \mu, \Sigma) = \sqrt{(X_i - \mu)^\tau \Sigma^{-1} (X_i - \mu)} \tag{2.13}$$

In essence, we assume that an anomalous observation has a large distance from the mean of the distribution. This model is an appropriate approach in predictive maintenance, as it detects abnormal behavior of a system and fraudulent samples can be isolated in order to reduce the extent of the damage.

On the other hand, the problem with the Mahalanobis formula is that the existence of some anomalies can affect the mean and covariance matrix computations, and thus the real mean and covariance values can be misleading. In particular, an increased value of covariance could indicate a larger diffusion of the data in the space. However, large mean and covariance values could also be affected by extreme observations with large distance from the center of the distribution. Consequently, by eliminating the presence of anomalies, we decrease the covariance. Therefore, we need a robust method to increase the tolerance in the presence of fraudulent points, in the context of mean and covariance matrix estimations.

This is exactly the goal of this algorithm: the utilisation of a robust estimator

of covariance that guarantees that the estimation is resilient to "extreme" observations in the dataset. Thus, the final Mahalanobis distances intend to reflect the real distances of the observations from the center of the distribution.

2.3 Deep Learning: RNNs and LSTM

The long short-term memory (LSTM) was introduced to handle the problem of vanishing gradients in stochastic gradient descent [6], and is one of the most popular Recurrent Neural Networks (RNNs) architectures. Later on, the Gated Recurrent Units (GRUs) were proposed as a simpler and more efficient model, which also moderates the vanishing gradients problem.

2.3.1 Simple RNNs

The strategy of Recurrent Neural Networks is applied in processing sequences, while memorizing what has previously passed. Therefore, the term *'Recurrent'* means that the output of a RNN at a time step becomes part of the input in the next time step of the model, which introduces the memory in the network, as well as the data (e.g. words) dependencies. As an example, such a model is an appropriate solution for text generation, or sentiment analysis.

Regarding the aforementioned characteristic of the RNN, which is the ability to learn dependencies in a sequence, we can capture the similarity of such a model with the human way of processing sequential data. For instance, the words are not considered as isolated entities, but as a group which carries contextual information and transfers a meaning. Thus, RNNs can encode the information and process it through the model from one time step to another [25].

Let us consider a set of n observations, $\mathcal{X} = \{X_1, ..., X_n\}$, where each observation is an m-dimensional one-hot encoded vector. In particular, X_1 is the input of the network at time step $t = 1$, and it is one-hot encoded with length equal to m. Moreover, we use H_t as the hidden state vector of the network at time t, and it represents the "memory" of the network, which is calculated with respect to the current input and the previous time step's hidden state. In a simple feedforward network we output the log likelihood of the output vector Y given the input vector observation X (equation 2.14), as it has been explained in paper [6], where $\boldsymbol{W}_{hy}$ and $\boldsymbol{W}_{xh}$ are weight matrices, with subscripts describing from-to layer relationships: $\boldsymbol{W}_{xh}$ is a matrix that describes the input-hidden transformations, $\boldsymbol{W}_{hy}$ is a matrix for the hidden-output

transformations, and similarly a $\boldsymbol{W}_{hh}$ matrix describes hidden-to-hidden layers transformations. Moreover, B_h and B_y are bias vector representations and σ is the sigmoid activation function :

$$P(Y|X) = \sigma(\mathbf{W}_{hy}H + B_y) \tag{2.14}$$

$$H = \tanh\left(\mathbf{W}_{xh}X + B_h\right) \tag{2.15}$$

During training, we maximize the overall log-likelihood $\sum_{s=1}^{m} \log P(X_s|Y_s)$. Although this works when we map a single input observation to a single output representation, it is not an appropriate model to handle a situation where we want to map input sequences to output sequences of a different length.

In contrast, in RNNs we train and process all the data at the current time step by using the information of the hidden variables at previous time steps [26]. Let's assume that we start at $t = 0$ where $H_0 = 0$, since we have no knowledge from the past at that time. Consequently, from $t = 1$ till $t = T$:

$$H_1 = \tanh(\underbrace{\mathbf{W}_{hh}H_0}_{\text{at } t_0} + \underbrace{\mathbf{W}_{xh}X_1}_{\text{at } t_1} + B_h) \tag{2.16}$$

$$\vdots$$

$$H_T = \tanh(\underbrace{\mathbf{W}_{hh}H_{T-1}}_{\text{at } t_{T-1}} + \underbrace{\mathbf{W}_{xh}X_T}_{\text{at } t_T} + B_h) \tag{2.17}$$

In equations 2.16 and 2.17, the weights $\boldsymbol{W}_{hh}$ and $\boldsymbol{W}_{xh}$ are shared over timesteps, whereas the $\mathbf{W}_{hh}H_0$ and $\mathbf{W}_{xh}X_1$ are matrix multiplications. Based on the hidden variables from equations 2.16 to 2.17, equation 2.14 becomes as follows:

$$p(Y_T|X_1, X_2, ...X_T) = \sigma(\mathbf{W}_{hy}H_T + B_y) \tag{2.18}$$

An important drawback of simple RNNs is that they are unable to learn long time dependencies, even though they are capable to map successfully sequence-to-sequence representations between the input and output of the network.

2.3.2 LSTM

The Long Short-Term Memory (LSTM) was introduced not only to alleviate the problem of learning long-term representations in a reasonable amount of time in practice, but also to reduce the vanishing gradients problem in Vanilla RNNs. Furthermore, LSTM models are widely preferred to be used in Natural

Language Processing (NLP), as well as in problems like image captioning, and text generation.

Regarding the way that LSTM works, we should mention that this model learns only meaningful information in order to make predictions, and forget non important data. The core concepts of an LSTM model are the cell state, and three various gates: the input gate, the forget gate, and the output gate. In particular, a cell state is an important component that transfers the important information all the way down the network; in other words, it is the “main memory" of our model that carries relevant data throughout all timesteps. As a result, relevant data from the earlier time steps can be memorised and transferred to later time steps. As more data are getting processed, important information is getting added or rejected to this component through the three gates, which are also important component of the LSTM model. A brief reference for what each of these gates contributes to the whole network is that the forget gate rejects what is irrelevant from earlier time steps, the input gate keeps the relevant information from the current time step, and finally the output gate outputs the next hidden state.

In Figure 2.1, it has been illustrated that: $M_{t+1}, H_{t+1} = \text{LSTM}(X, H_t, M_t)$, where X is the input vector, H_t variables represent the hidden state vector, and M_t is for the memory cell state. It is obvious that the memory and hidden states are getting updated at each time step, because sequences are recognised as long-term dependencies with valuable context, which are needed to be memorised in order to encode the input data.

In the context of NLP, a suitable input for the LSTM are word embedding vectors, which are numerical vector representations; a word embedding can be, for instance, a one-hot encoded vector with a length equal to the number of words in the vocabulary. There are many ways to encapsulate meaningful information in the simple one-hot vectors, and that is why we use a word embedding matrix which includes word dependencies. Then, by simple vector to matrix multiplications, we capture the words' importance given the context of the sequence. The main objective of such models is that words with similar context are close to each other.

Last but not least, we must briefly refer to three important variants of the basic LSTM model; these are the GRUs, the Bidirectional LSTM, and the LSTM-attention model [6].

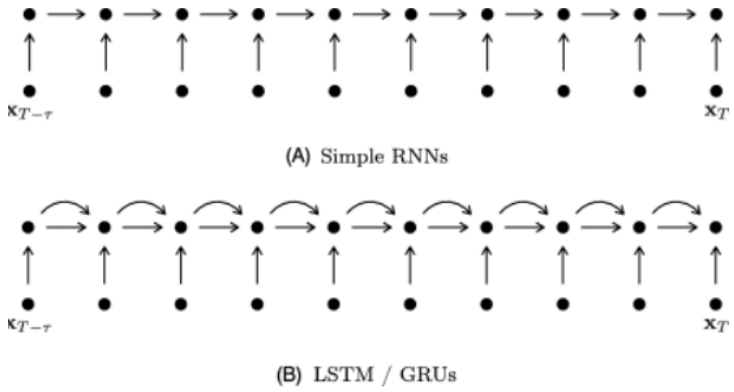


Figure 2.1: A visual representation of simple RNNs, LSTM, and GRUs models.(A) RNNs derive only one path between times t and $t-1$, with one path inhibited by linear transformations. (B) LSTM and GRUs give multiple paths between times t and $t-1$, with one path inhibited by no linear transformations. (Figure reprinted and adapted from [6])

2.4 Variational Autoencoders

General Idea

Variational Autoencoders belong to the class of deep generative models, which are models that learn normal patterns by learning the distribution of the training instances. Variational Autoencoders (VAE) are widely considered as an unsupervised anomaly detection technique that learns a probability distribution model by utilising only anomaly-free samples in the training phase. Thence, anomalies can be identified, because they diverge from the probability distribution of the model. Regarding the architecture of a Variational Autoencoder (VAE), it utilises an encoder-decoder mechanism, as it has been explained in paper [27].

Encoder's Model

The encoder model is a neural network that represents the visible input data X in the hidden space as a dense vector representation Z. The purpose is to maintain all the contextual information from input space to hidden layers, and to reject all the irrelevant parts at the same time. It does so by assuming that $Z|X$ is a random variable with a tractable density. For simplicity we will refer to the standard case where $Z|X$ is assumed to be a Gaussian random variable $\mathcal{N}(\mu, \sigma^2 I)$. Hence, in the latent space, the VAE learns from a distribution by calculating the mean and standard deviation of the variables; in other words,

the encoder does not output only a vector representation of size m, but two vectors of size m, which are the mean and the standard deviation. This is the most important feature of this architecture; the fact that the encoder does not directly represent the input sample's coordinates in the latent space, which is what a vanilla Autoencoder does, and instead it forces the vector representation Z to comply with a prior probability distribution $P(Z)$, as described in [28]. Nonetheless, for the rest of the section, we will assume that all the random variables are absolutely continuous and hence have probability density distributions (pdf), which we will denote with a lowercase p, e.g., the pdf of Z is $p(Z)$.

Decoder's Model

Turning now to the decoder model, the basic idea is that it samples from the density distribution of the latent space, which is assumed to be Gaussian $\mathcal{N}(\mu, \sigma^2 I)$, and then it generates reconstructions of the sampled encoded representations Z. As we explained for the encoder's model, the decoder (when a Gaussian distribution is assumed) also has an important feature: it learns that all nearby points of the sampled one refer to the same class, whereas in a vanilla Autoencoder the decoder learns that only the single sampled point belongs to that class.

Prior Distribution and Likelihood

The Variational Autoencoders are neural networks that are utilised to learn the density of the input data $p(X)$ by modeling a joint density distribution of the input data and the hidden variable $p_\theta(X, Z)$, parameterized by θ. In particular, they describe this $p(X)$ as a marginalization of this joint density distribution, as proposed in paper [27]:

$$p(X) = \int p_\theta(X, Z) dZ \tag{2.19}$$

In principle, we describe the joint density distribution $p_\theta(X, Z)$ as the product of the likelihood $p_\theta(X|Z)$ and the prior of the hidden variable $p(Z)$, so that:

$$p(X) = \int p_\theta(X|Z) p(Z) dZ \tag{2.20}$$

The reason that we take this approach is that by the marginalization over Z we use two simpler expressions of the likelihood $p_\theta(X|Z)$ and the prior densities of the hidden variable $p(Z)$ to describe a more complex expression for $p(X)$.

This mimics the way that a human brain works; given a specific topic, our brain precisely adds some limitations in our imagination so as to think and generate the relevant ideas and words regarding the specific context. Similarly, the hidden space of a VAE makes the latent variable Z very likely under our original data, and thereby the decoder generates reconstructions very similar to the input data, by using the likelihood $p_\theta(X|Z)$, as in equation (2.20).

As we mentioned above, the prior density distribution of the latent space is assumed to be the standard Gaussian distribution $\mathcal{N}(\mu, \sigma^2 I)$, where $\mu = 0$ and $\sigma^2 = 1$. On the other hand, the likelihood is also assumed to be described by another Gaussian distribution where the mean is described by a non-linear function of the hidden variable Z:

$$p(Z) = \mathcal{N}(0, I) \tag{2.21}$$

$$p_\theta(X|Z) = \mathcal{N}(\mu, \sigma^2 I), \mu = f[Z, \theta] \tag{2.22}$$

Posterior Distribution and Sampling

By calculating the posterior $p_\theta(Z|X)$, which is what describes the encoder's model, we understand the possible values of the hidden variable Z that might be responsible for a given observation X, see Figure 2.2. Using the Bayes' rule we have the following formula:

$$p_\theta(Z|X) = \frac{p_\theta(X|Z)p(Z)}{p(X)} \tag{2.23}$$

where

$$p(X) = \int p_\theta(X|Z)p(Z)d(Z) \tag{2.24}$$

In general, computing the posterior density distribution is intractable, since we cannot calculate the denominator $p(X)$, because the integral could not have an analytical solution, be high-dimensional, and difficult to estimate. Conceptually we could estimate it by first sampling a number of Z values $\{Z_1, .., Z_n\}$ and then compute it as follows:

$$p(X) \approx \frac{1}{n} \sum_{i=1}^{n} p_\theta(X|Z_i) \tag{2.25}$$

However, the problem here would be that for most Z_i values the likelihood would be almost zero, which imposes a difficulty on the estimation of $p(X)$.

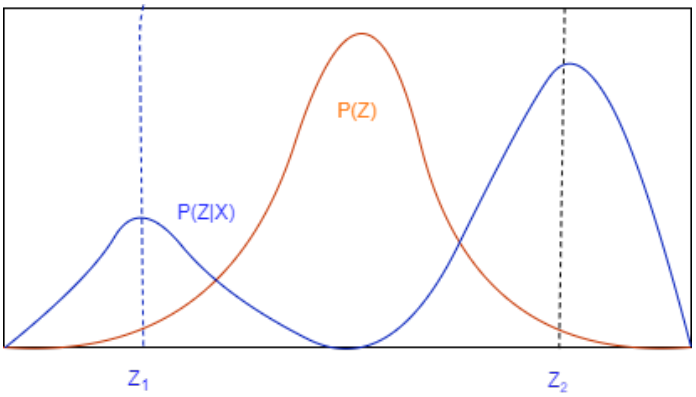


Figure 2.2: Here we can see that given an observation X it is more likely to have been created from Z_2 than Z_1, since $p(Z_2|X) > p(Z_1|X)$.

That is why we introduce sampling on those Z values that are more likely to produce X, and hence we can approximate $p(X)$ (in equations 2.23, 2.25) just from those. Therefore, we need another function $q_\phi(Z|X)$ which can describe those Z values that can produce the original X, as explained also in paper [28]. Most likely, the sets of Z values that are probable under the approximate $q_\phi(Z|X)$ will be less than all Z's that are probable under the prior $p(Z)$. As a result, we want to force this approximate density $q_\phi(Z|X)$ to stay close to the standard normal density of $p(Z)$. Hence, in the sampling phase, we could sample those Z values that are more likely to produce X.

Variational Inference (VI)

At this point, we use the Variational Inference (VI) method, which is a popular technique in Bayesian Inference, which has been described in paper [28]. In VI we can model an unknown distribution using a simpler one that is easier to be evaluated, such as the normal distribution. Therefore, our goal is to eliminate the difference between these two distributions using the KL divergence metric, as in the paper [27]. In short, the KL divergence tells us how much one distribution differs from another [29], as given in equation 2.26:

$$D_{KL}(q(X)|p(X)) = \int q(X) \underbrace{\log \frac{q(X)}{p(X)}}_{\Delta \mathrm{I}} dX \tag{2.26}$$

Similarly, in our case:

$$D_{KL}(q_\phi(Z|X)|p(Z)) = \int q_\phi(Z|X) \underbrace{\log \frac{q_\phi(Z|X)}{p(Z)}}_{\Delta \mathrm{I}} dZ \tag{2.27}$$

where ΔI is the information difference of two distributions. In overall, we use D_{KL} as a similarity measure between two distributions, since our goal is to ensure that the approximate density $q_\phi(Z|X)$ will stay close to the standard normal density of $p(Z)$, and thus we want to minimize the KL-Divergence.

Loss Function

Finally, the whole model is trained by maximizing the objective function, also known as "Variational Lower Bound" or the "Evidence Lower Bound" (ELBO), which further implies that we minimise the KL-divergence term:

$$\mathcal{L}(\theta, \phi, X) = \underbrace{\mathbb{E}_{q_\phi(Z|X)}[\log p_\theta(X|Z)]}_{\text{log-likelihood}} - \underbrace{D_{KL}(q_\phi(Z|X)|p(Z))}_{\text{KL-divergence}} \tag{2.28}$$

The first term is the expected log likelihood of the reconstructions X, and maximises the expectation that latent variable Z generates the observations X. As regards the later term, it forces the approximated posterior $q_\phi(Z|X)$ to be similar to the true prior $p(Z)$. In other words, the KL-divergence is used as a penalty in the way that the encoder places the encodings in the hidden space, i.e. whenever any point is misclassified the encoder is getting penalized [30].

Turning now to how these terms appear in the objective function using Jensen's inequality, we know that the density of X is given as:

$$\begin{aligned}
\log p(X) &= \log \int_Z p(X, Z) = \log \int_Z p(X, Z) \frac{q(Z|X)}{q(Z|X)} \\
&\geqslant \mathbb{E}_{q(Z|X)} \left[\log \frac{p(X|Z)p(Z)}{q(Z|X)} \right] \\
&= \mathbb{E}_{q(Z|X)} \left[\log p(X|Z) \right] + \mathbb{E}_{q(Z|X)} \left[\log \frac{p(Z)}{q(Z|X)} \right] \\
&= \mathbb{E}_{q(Z|X)}[\log p(X|Z)] + \int_Z q(Z|X) \log \frac{p(Z)}{q(Z|X)} \\
&= \underbrace{\mathbb{E}_{q(Z|X)}[\log p(X|Z)]}_{\text{log-likelihood}} - \underbrace{D_{KL}(q(Z|X)|p(Z))}_{\text{KL-divergence}}
\end{aligned} \tag{2.29}$$

Therefore, the VAE finds a lower bound of the expected log likelihood with the usage of Jensen's inequality. As we can see in Equation 2.29, the first part of ELBO maximizes the log likelihood, in order to make the latent variable Z more probable to generate the observations X, whereas the second part of ELBO minimizes the KL-divergence between the posterior and the prior distribution.

Conclusion

To sum up, VAE models are powerful generative models widely used in many applications, especially when we want to explore variations on data, and thus they have been used to solve many anomaly detection problems successfully.

2.5 Related Work

The following sections are aimed at describing the background about the remarkable research work in the same context as this degree project.

2.5.1 Log Anomaly Detection

Machine Learning Methods

In the work of Shilin et. al. [31], there is an extensive discussion about the "Log Anomaly Detection" problem, where he compares commonly-used machine learning algorithms in terms of a systematic validation. In particular, a set of supervised anomaly detection methods, i.e. "Logistic Regression", "Decision Tree" and "Support Vector Machine", is compared with a set of unsupervised learning algorithms, i.e. "Log clustering", "Invariant Mining", "Principal Components Analysis". According to the authors' conclusions, the "Decision Tree" would be preferred from all the supervised learning algorithms, whereas both "Log clustering" and "Invariant Mining" would be a better solution than PCA. Moreover, the supervised algorithms outperformed the unsupervised methods with respect to the F-score values. Furthermore, the authors released a tool with the implementation of the aforementioned anomaly detection algorithms, where they utilised the commonly-used scikit-learn package [32], as well as they implemented the unsupervised methods online due to memory constraints.

In another study performed in 2019 by Cheng Zhangyu [15], the authors proposed an ensemble method, which benefits from two widely-used algorithms in

order to accurately isolate anomalies; in more details, the "Isolation Forest" is combined with the "Local Outlier Factor" algorithm. According to the results presented in the survey, the proposed ensemble model has been proved that outperforms the traditional algorithms in terms of performance and speed. Moreover, in the work of Whelan et.al. [21], it has been explained how we can benefit by using "k-Medians" instead of "k-Means", although the first method is a variant of the later one. The mean value is a measurement that is thoroughly sensitive to anomalies; especially in big datasets, the mean value could be drastically changed by only one sample that deviates from all the others. On the other side, the median is to a great level resistant to anomalous data points, because in order to throw the median away from the real estimation, it requires a higher proportion of samples to be contaminated in the entire dataset rather than just one instance. In other words, the "k-Medians" algorithm is more robust due to the robustness of the median in statistics.

Deep Learning Methods

A significant survey has been conducted by Min Du et.al. [3], where they underline the significance of "Log Anomaly Detection" describing that the isolation of malicious log events prevents system failures and improves the performance of the networks. More specifically, they highlight the fact that the system logs are source of information for online monitoring, as well as they propose a deep neural network using the Long Short-Term Memory component, to model a system log as a natural language sequence. In particular, this model is trained as a multi-class classifier that learns patterns which correspond to normal events. Thenceforth, any deviation from what has been learnt as "normal" is identified as "anomaly". In the proposed "DeepLog" architecture, the anomaly detection stage is performed in an online streaming configuration, where each log file is parsed into a sequence of log entries using a window of size h. Then, the LSTM model outputs a probability distribution (equation 2.30) of having the pattern k_i as the next value for the incoming log message m_t, given the sequence of parsed log messages $w = \{m_{t-h}, ..., m_{t-1}\}$:

$$P(m_t = k_i | m_{t-h}, ..., m_{t-1}) \tag{2.30}$$

Another important survey has been introduced by Bontemps et.al. in [33], where an LSTM-RNN architecture has been proposed to isolate individual point anomalies at each time-step. However, since the goal is to detect collective anomalies, they also explain the concept behind a circular array. This array presents the "Average Relative Error" and the anomalous observations,

or else the "Danger Coefficients". By monitoring this array, we can identify a sub-sequence as a collective anomaly if both the aforementioned factors are higher than two predefined thresholds. Furthermore, according to the results, we observe that the proposed method can achieve a 86% true positive rate in case of the higher threshold, whereas a 100% true positive rate can be achieved if the threshold is decreased; apparently, in this case we observe a more false alarms, too.

Turning now in how the Variational Autoencoders have been proved very powerful in the recent years, we should refer to the work of Carl Doersch [28], which constitutes an introduction to the Variational Autoencoders algorithm. Although this paper has a bias towards computer vision applications, it can be used also as a guide regarding the theoretical background of this architecture. Another interesting approach has been introduced from An Jinwon and Cho Sungzoon in [34], where a VAE model is utilised, in order to propose a different probability measure as the anomaly score rather than the widely-used reconstruction error. In other words, VAE uses the reconstruction probability as the anomaly score instead of the reconstruction error, which has been utilised from the other models (a traditional autoencoder and a PCA). Looking at the experimental results presented in the paper, we can see that the proposed method outperforms autoencoder based and PCA based methods.

2.5.2 Deep Transfer Learning

In the work of Pan and Yang [35], the concept of transfer learning has been extensively described. In all machine learning algorithms and deep neural networks we build statistical models while working with a specific dataset. This implies that it is a necessity to rebuild the models when we need to sample from another feature space or another distribution. Apparently, this is extremely difficult with respect to: (a) the cost and limitations implied from the lack of computational resources in the training phase of the models, and (b) the inability to recollect new training and testing data. In these cases, transfer learning is a desirable technique that enhances the learning phase by avoiding expensive actions.

The concept of transfer learning is very beneficial in the hybrid deep learning models for anomaly detection applications. It can be used in the feature extraction phase, where we need to generate rich vector representations in order to train our traditional machine learning methods. The main characteristic of this

technique is that we can use a pre-trained model to transfer knowledge from some previous task or source distribution to another one. As a consequence, we can optimise the target training objective function by transferring knowledge from other sources.

In the paper of Zhuang et. al. [36], a new supervised learning method has been proposed, where variations of autoencoders can be used for feature extraction in order to produce vector representations that can be used to transfer knowledge from one domain to another. In this survey, the proposed model succeeds in eliminating the difference between the source and target distributions by minimising the KL divergence, which secures that the source and target distributions are similar, and therefore machine learning methods can learn the enriched vector representations from the latent space. Another advantage of this model is that it utilises encoded label representations, since the learning is supervised as stated above, and thus it successfully represents the discriminating information into the vector representations.

Chapter 3

Methodology

In this section, we give a review on the methodology that we used to solve the problem of log anomaly detection in our network. In this survey, the anomaly detection framework involves four phases: data collection, data pre-processing, and anomaly detection models.

3.1 Data

3.1.1 Data Collection

Large-scale systems consistently generate and store logs to monitor and control system states and important information, each including a severity level and a log message denoting what has happened. These messages are utilized for many purposes, like anomaly detection, and thereby log messages are collected for further usage. In the LHC-b experiment, the Elastic Stack platform [37] is used for all the monitoring needs of the collaboration. A significant number of applications and machines are producing a constant flow of various logs. Those logs are streamed through the network to a Kafka cluster used as a buffer area. Logstash pipelines then consume those logs, perform data transformation to them before storing them permanently to Elasticsearch [37]. System administrators are able to discover those logs and monitor the status of the systems in a user-friendly way using the web-interface of Kibana. For bulk operations and programmatic access the Elasticsearch API is available which we were able to query to receive the messages from all the switches in the network (i.e. logstash-switchlogs-*) with a time range of the last two years (2019-2020).

	Dataset 1	Dataset 2
info	16312	16312
warning	82980	82980
notice	464	464
error	**244**	**10244**
total	100000	110000

Table 3.1: Samples per category for CERN's datasets

Dataset	Dataset size	Logs count	Anomalies
dataset1	52 MB	100,000	244
dataset2	57 MB	110,000	10,244

Table 3.2: Datasets details

3.1.2 Description of the Datasets

Following the steps described in the previous section, we created two datasets: (a) dataset1 with 100K log messages, and (b) dataset2, with 110K messages. The messages are grouped in four different categories regarding their severity: info, warning, notice, and errors, from which only the last one indicates a serious problem with one or more systems in the network. The accurate samples of these four categories are presented in Table 3.1, where we can see that dataset2 has 10 thousands more messages in the error category. In this way, we managed to handle the high class imbalance that we have in the first dataset. In more details, in dataset1 we had only 0.25% error messages, whereas in dataset2 we managed to increase the instances of this category to 10.25%.

3.1.3 Data Pre-processing

As described above, we collected our data from the ElasticSearch API. The response was in a form of a long json file with a lot of information, from which we focused on extracting the log messages and their labels (severity level). As usual, we cleaned up our texts by removing unwanted characters, punctuation, and digits.

Another issue in data pre-processing is the removal of stopwords. As stopwords we define those words that lack a true meaning given a specific context, or else words with low contextual information. Although discarding words with low importance seems a necessity, it may be a bad option in many cases, like sentimental analysis, or text generation. Similarly, in our case, no words are

considered as stopwords, and thus we did not proceed in any deletion. The main reason is that our log messages are sentences from large-scale systems and thus they are difficult to be understood by humans, especially from those who have no technical knowledge about them. This means that removing stopwords could have a negative influence in the performance of our models, and that is why we did not delete any word from our data.

After log parsing and cleaning, we proceeded with the language modelling step. We must declare that we used two different techniques for this phase: one for the clustering methods, and one for the LSTM based network and the hybrid Variational Autoencoder.

Word Embedding in Clustering Methods

For the basic clustering algorithms we used a pretty straightforward preprocessing technique, as proposed in [38], where parsed log messages are converted to numerical feature vectors, whereby they can be applied in our machine learning models. In other words, the technique of Bag-of-Words (BoW) with Tf-Idf is utilised, since Bag-of-Words is an efficient, simple technique for classifying documents, or in this case log messages, as a whole.

In particular, we tokenized all the messages and we created a corpus of word tokens, for example in dataset1 (100K logs) we found 801 tokens. Then, we converted the logs set into vectors, where each term of the vector is indexed as the index corpus, for instance, the first term of the vector represents the first term of our corpus, and so on. This means that all feature vectors have the same dimensionality, which is equal to the size of our vocabulary. So, now each log message is considered as a 1-out-of-K vector representation, where $K = 801$, and each term in the feature vector indicates the frequency of this word in the given log message.

Due to the fact that BoW model discards grammar, word ordering and contextual information [39], we apply variants of tf-idf score to handle the contribution of words with low importance in the representations. The formula of tf-idf is the following:

$$\text{tf}_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}} \approx \mathcal{P}_j(i) \tag{3.1}$$

$$\text{idf}_i = \log \frac{D}{\text{df}_i} \tag{3.2}$$

$$\text{tf-idf} = \text{tf}_{i,j} \times \text{idf}_i \tag{3.3}$$

In equation 3.1, the entity $n_{i,j}$ describes the frequency of a word i in a log message j, divided by $\sum_k n_{k,j}$, which is the total number of occurrences of all words in a log message, or simply the total words in a log message. Therefore, $\text{tf}_{i,j}$ value is approximately the same with the prior probability of the word in the sentence. On the other hand, the idf_i, in equation 3.2, is the inverse document frequency for the specific word. In order to calculate this value, we just take the logarithm of the division of the total number of logs with the number of logs that the word i appears. In the end, the final score is the multiplication of those two values, as in equation 3.3. In Table 3.3, we can see the result of the tf-idf vector representations for some log messages.

To sum up, we preprocessed all the messages and we built a data matrix with all their feature vector representations, which can be applied to all machine learning models. In particular, for dataset1 we built a $[100000 \times 801]$ matrix with the 100,000 log messages. For each message, we got a 1-to-801 vector representation by using the tf-idf formula. Finally, we applied different dimensionality reduction techniques to compact the sparse vector representations into two important components. In this particular step, we utilised the PCA method from *sklearn* library, the t-SNE from *sklearn.manifold* package, and UMAP from *umap* python library. The comparisons among these techniques are presented in Chapter 4. In this point, our data are ready to be input in the clustering algorithms.

Dataset-1

	Vocabulary					
Samples	*'abort'*	*'accept'*	*'connection'*	...	*'down'*	*'url'*
D1	0.00	0.00	0.224	...	0.383	0.171
D2	0.00	0.105	0.114	...	0.383	0.00
D3	0.567	0.00	0.00	...	0.143	0.00
D4	0.00	0.00	0.214	...	0.313	0.671

Table 3.3: Feature vector representation with tf-idf scores for our log messages D1, D2, D3, D4.

Word2Vec Language Model

Although tf-idf BoW vector representations provide weights to measure relevance, instead of frequency, they still cannot capture the word importance given a context. For the implementation of the LSTM based neural networks, we had to switch to another language model, which includes word dependencies. Following the work of Lizhong Xiao and others in [40], we used the Word2Vec model to generate word embeddings with high semantic information.

The main objective of this model is that words with similar context have close positions, in other words, that the distance of similar words is minimised. Each word is again one-hot encoded, in the beginning, and thus it is represented by a vector that has the same length as the size of the vocabulary. On the other hand, one-hot encodings do not reflect the relevance of the words, which means that we still cannot capture the word dependencies; i.e. that even if the words "delete" and "remove" have a similar contextual information, they still have the same distance from a word like "server", which is totally different, and thereby they are considered similar in the same way, which is not true. Therefore, we need to generate distributed representations regarding the contextual similarity of the words. Word2Vec is basically obtained by using two methods: common BoW and Skip-Gram [41], where both enhances better representations in the context of semantic information, as we have already explained in Chapter 2.

We implemented two Word2Vec models, using the *gensim* package, for both datasets, see Table 3.4 for details. The difference in the vocabulary size between these two datasets is reasonable, since we had a bigger dataset to build the 2nd model. What is more, for the same dataset both language models , BoW and Word2Vec, gave us the same vocabulary size, i.e. for dataset1 we found 801 words with either tf-idf BoW method, and Word2Vec model. The difference is that we cleaned our data in a different way, because in this case we used the *gensim.utils.simple_preprocess* method, whereas in the tf-idf BOW, we manually removed the unwanted characters. However, since we built the same vocabulary, either way is accepted.

We confirm that our word embeddings capture the contextual information and the word dependencies, by checking that words that appear in the same sentence (e.g. license-cleared) have higher similarity than words that do not appear in the same message (e.g. server-class), as in Tables 3.5 and 3.6, where we used the following three log messages:

1. NTP Server is Unreachable
2. Alarm set: License color=YELLOW, class=CHASSIS, reason=BGP Routing Protocol usage requires a license
3. Alarm cleared: License color=YELLOW, class=CHASSIS, reason=VxLAN usage requires a license

Dataset	Corpus size	Time to build corpus	Training time	Train loss
dataset1	801	0.0073 mins	0.05 mins	526838.56
dataset2	831	0.0054 mins	0.05 mins	532169.75

Table 3.4: Word2Vec models' details

	set	cleared	unreachable	class
alarm	0.5102	0.5258	-0.195	0.536
license	0.6498	0.656	-0.203	0.595
chassis	0.463	0.476	-0.043	0.423
server	-0.146	-0.143	0.325	-0.128
class	0.448	0.459	-0.036	0.99999

Table 3.5: Test Word2Vec models' performance by measuring words' vectors similarity between the given word pairs.

Tested word	Top 1	Top 2	Top 3	Top 4	Top 5
alarm	license	requires	usage	chassis	reason
	0.786	0.755	0.676	0.6005	0.579
server	minpoll	maxpoll	bigger	configured	isolated
	0.648	0.621	0.611	0.610	0.581
license	alarm	usage	cleared	requires	set
	0.786	0.691	0.664	0.663	0.656
chassis	alarm	license	requires	usage	cleared
	0.6005	0.587	0.549	0.529	0.504
class	license	requires	alarm	usage	cleared
	0.625	0.59	0.545	0.529	0.459

Table 3.6: Test Word2Vec models' performance by finding the top 5 similar words for the ones given.

3.2 Models

3.2.1 Clustering Methods

Isolation Forests

We used the already implemented Isolation Forest model from the *sklearn-ensemble* package [32]. The number of recursive partitions, based on a randomly selected features of the dataset, is the final criterion to identify if a sample is an anomaly or not. We fine tuned our model using the 'Grid Search' strategy, and thus we used 100 base estimators and a maximum of 100 maximum samples. We tried to increase the number of maximum samples to 300 and 500, but as we described in chapter 2, with a larger sampling size, we had worst results and more missclassified anomalies. We also used a fixed *contamination* value, which is the proportion of the outliers in the dataset, and should be in the range of $[0, 0.5]$. In dataset2 with the small class imbalance, we had 10,244 outliers in 110,000 samples, which gives a proportion of 10%, while in dataset1, we only had 0.25% outliers. Therefore, in dataset1 we used a contamination value equal to 0.03, while in dataset2 we used a 0.1 .

k-Medians

We used the already implemented k-Medians model from the *sklearn-ensemble* package [32]. In addition, we fine tuned the model with respect to different number of clusters, which also are the number of centroids to generate. In the end, we evaluated our model's performance for each of the different values of the given number of clusters.

3.2.2 LSTM-based Model

We implemented an LSTM based neural network in Tensorflow 2.3.1, since we are working with sequential data and we need to learn long-term dependencies, as explained by Ian Goodfellow et. al. in [9]. As regards to the model's architecture, our first layer is an 'Embedding layer'. In the preprocessing phase, we already described that we used a Word2Vec model, and thus we mapped each word into a unique integer value. In other words, one log message with N words becomes a vector with N unique integers. However, the tf.keras 'Embedding layer' requires all individual samples to be of the exact same length. Hence, we padded the shorter samples with zeros with respect to the embedding size, which is equal to 64. As a result, for the 'Embedding layer', the input dimen-

sion is equal to the vocabulary size (801 for dataset1 and 831 for dataset2), the output dimension is equal to the embedding size 64, and we also load the pretrained weights of our Word2Vec model.

After the Embedding layer, we used the basic component of our architecture, which is the LSTM layer with 64 units, same as our embedding size. We should clarify also that the number of units represents the dimensionality of the output. Afterwards, we simply used a fully connected layer with a 'ReLU' activation layer, to reduce the dimensions from 64 to 32, and then, we added another dense layer with a 'softmax' activation function, in order to output two values, since our problem is a binary classification. Last but not least, for both fully connected layers we used a dropout of 0.3, see Figure 3.1.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, None, 64)          46464
_________________________________________________________________
lstm (LSTM)                  (None, 64)                33024
_________________________________________________________________
dropout (Dropout)            (None, 64)                0
_________________________________________________________________
dense (Dense)                (None, 32)                2080
_________________________________________________________________
activation (Activation)      (None, 32)                0
_________________________________________________________________
dropout_1 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 66
_________________________________________________________________
activation_1 (Activation)    (None, 2)                 0
=================================================================
Total params: 81,634
Trainable params: 81,634
Non-trainable params: 0
_________________________________________________________________
```

Figure 3.1: Architecture of the LSTM based network.

Regarding the fine tuning of the neural network, we used the Keras Tuner library. The main steps of the tuning process were to utilise the HyperModel subclass, where we provided specific values in *hp.Choice* for the learning rate and the optimizer. Afterwards, we tried to instantiate the tuner concept with all three strategies: RandomSearch, Hyperband, and Bayesian Optimization. The first two concepts gave us the same values, whereas with Bayesian Optimisation we got a different combination, for which our model had a bad performance on the testing set. As a consequence, we used the Hyperband tuner concept, and we trained our network with the *Adam* optimiser and a learning rate equal to

10^{-4} for dataset1, whereas for dataset2 we used *SGD* with learning rate equal to 10^{-4}.

3.2.3 A Hybrid Variational Autoencoder Model

The Variational Autoencoder architecture consists of 3 components, as explained in [34]: the encoder network, the latent space, and the decoder network. The main idea is that we represent the input data as vector representations in the hidden space, which comply with a prior probability distribution. Afterwards, we sample from that distribution, where the sampled data are decoded and the reconstruction error can be calculated. In the training phase, we backpropagate the reconstruction error and we try to minimise the loss function.

In practice, the encodings are chosen to be normally distributed, and thus the encoder model returns the mean and the covariance that describe the Gaussians; the distributions that are formulated by the encoder are usually imposed to be close to a normal distribution. In other words, an input is encoded as a distribution instead of a single point. Therefore, in the encoder network we use firstly an 'Embedding layer', for which the input dimension is equal to the vocabulary size (801 for dataset1 and 831 for dataset2), the output dimension is equal to the embedding size 64, and we also load the pretrained weights of our Word2Vec model [40]. Then, we use an LSTM with 96 units. The output of the LSTM generates our mean and variance from two fully connected layers with 32 output units each. In the end, we add a custom layer for the sampling phase, where we use the mean and variance to generate samples from the learned probability distribution.

Turning now to the decoder's network, we build a generator that can sample sentences from the learned distribution. Taking into consideration that we need to pass again all the samples from the LSTM layer, we need to repeat each sample 64 times, which means that we should reshape each point from (None, 32) to (None,64, 32), as the size of the embedding. This is why we use a RepeatVector layer just before the LSTM layer, which should have 96 units, as in the encoder network. The decoder outputs only the mean value using a Dense layer with 64 as the output dimension and ReLU as the activation function. The output now is in the shape of (None, 64, 64), but our goal is to reconstruct the samples in the original shape of (None, 64). Hence, we firstly reshape into (None, 4096), where $4096 = 64 \times 64$, and secondly, we simply use a Dense layer with 64 output dimensions.

```
Model: "encoder"
__________________________________________________________________________________
Layer (type)                 Output Shape              Param #   Connected to
==================================================================================
input_1 (InputLayer)         [(None, 64)]              0
__________________________________________________________________________________
embedding (Embedding)        (None, 64, 64)            46464     input_1[0][0]
__________________________________________________________________________________
lstm (LSTM)                  (None, 96)                61824     embedding[0][0]
__________________________________________________________________________________
z_mean (Dense)               (None, 32)                3104      lstm[0][0]
__________________________________________________________________________________
z_log_var (Dense)            (None, 32)                3104      lstm[0][0]
__________________________________________________________________________________
sampling (Sampling)          (None, 32)                0         z_mean[0][0]
                                                                 z_log_var[0][0]
==================================================================================
Total params: 114,496
Trainable params: 68,032
Non-trainable params: 46,464
```

Figure 3.2: Architecture of the encoder's network.

```
Model: "decoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 32)]              0
_________________________________________________________________
repeat_vector (RepeatVector) (None, 64, 32)            0
_________________________________________________________________
lstm_1 (LSTM)                (None, 64, 96)            49536
_________________________________________________________________
decoder_mean (Dense)         (None, 64, 64)            6208
_________________________________________________________________
relu (Activation)            (None, 64, 64)            0
_________________________________________________________________
reshape (Reshape)            (None, 4096)              0
_________________________________________________________________
decoder_out (Dense)          (None, 64)                262208
=================================================================
Total params: 317,952
Trainable params: 317,952
Non-trainable params: 0
```

Figure 3.3: Architecture of the decoder's network.

We should also mention that the loss function that we minimise is the sum of the normal distribution likelihood and the KL divergence. Minimising the KL divergence term means that we optimise the normal distribution parameters to

accurately resemble the real distribution we try to approximate, as it has been explained in Chapter 2. In addition, the training of VAE is unsupervised, since we only train the model with clean data, but no labels have been utilised.

The final step of this strategy with the hybrid model, is that we compute the reconstruction errors using the MSE formula, and we select a threshold that indicates when our VAE model is not able to reconstruct the messages properly. This also means that these instances with a reconstruction error above this threshold correspond to indications of real errors in our network. Finally, we perform a supervised learning using the target collected labels, and thus our problem is simply a binary classification.

Taking into consideration the way that our data are spread out in space, we chose a classifier that can find a non-linear hyperplane to separate the two classes. Support Vector Machines with 'rbf' or 'polynomial' kernel are useful for finding non-linear hyperplanes, as discussed in [42]. In particular, we used the Grid search concept [43] to fine tune the gamma and the C hyper-parameters of the SVM classifier with RBF kernel, and we found that the best parameters are $\{C : 1.0, gamma : 0.1\}$ with a score of 0.99. The gamma parameter defines how far the influence of a single sample extends, with small values meaning far and high values meaning close; in other words, the higher gamma value, the closer other data points should be to be affected. The parameter C affects the smoothness of the decision surfaces; the smaller the C value, the smoother the decision surfaces.

In the end, considering both Variational Autoencoder network, which is an unsupervised machine learning algorithm, and SVM [42], which is a supervised machine learning method, we end up with a semi-supervised learning model.

3.3 Evaluation Metrics

We evaluate the clustering methods using three different scores, as they have been presented in the work of Baarsch et. al. [44]:

- the Silhouette coefficient,
- the Davies-Bouldin,
- and the Calinski and Harabasz score

On the other hand, for the LSTM based model and the part of supervised learning in the hybrid model with the Variational Autoencoder, we simply used the AUC-ROC metric, as proposed in the work of Narkhede and Sarang [45]. Regarding the evaluation of the reconstructions in the Variational Autoencoder network, we used the Mean Squared Error between the inputs and outputs of the model as our evaluation metric. In particular, we select specific instances with high MSE values and we classify them again based on a threshold.

3.3.1 Silhouette Coefficient

The Silhouette coefficient method relates compactness to separation of clusters. In more details, it measures the distance between each data point and every other point in this cluster and the minimum average distance between that point and the other points in each other cluster. In other words, it measures the coherence within a cluster and the separation from all the other clusters. The higher the silhouette coefficients, the better the clustering results. The silhouette coefficient for each i-th data point is given as follows:

$$s = \frac{b_i - a_i}{\max(b_i, a_i)}$$

where, "b_i" stands for the mean distance between one data point and all the points in the next nearest cluster, and "a_i" represents the mean distance between this data point and all the other points within the cluster it belongs to, as explained in paper [46].

3.3.2 Davies-Bouldin Score

This metric describes the average similarity between the closer clusters. What is important in this technique, is that it measures the within similarity in each cluster with respect to the distance between all of them; in other words, it evaluates intra-cluster similarity and inter-cluster differences. Therefore, a better Davies-Bouldin score is defined by clusters which are far away from each other, but are less diffused in the space, as explained in the work of Davies and Bouldin [47]. Therefore, the difference with the Silhouette score is that the Davies-Bouldin is measured by comparing the distance between clusters with the size of the clusters, as opposed to the Silhouette coefficient which measures the average similarity of the data within a cluster and their distance to the other points in the nearest clusters. Moreover, we should mention that lower values correspond to better clustering, because a smaller Davies-Bouldin

value means less dispersion within clusters, and bigger distance between them. Mathematically, similarity in Davies-Bouldin score between clusters i and j is given as:

$$DB = \frac{1}{k}\sum_{i=1}^{k}\max_{i\neq j} R_{ij} \tag{3.4}$$

$$R_{ij} = \frac{s_i + s_j}{d_{ij}} \tag{3.5}$$

where, s_i is the diameter of cluster i and d_{ij} is the distance between the centroids of i and j clusters.

3.3.3 Calinski-Harabasz Score

As Baarsch et. al. explained in [44], the Calinski-Harabasz score is one of the most successful metrics for many surveys. In particular, this metric measures the relationship between an "inter-cluster scatter matrix" and an "intra-cluster scatter matrix". The definition of a scatter matrix is similar to the one of the covariance matrix, but should not be confused [48]. The inter-cluster scatter matrix is the sum of squares of the distances between the centroids of each cluster and the centroid of the data set, whereas the intra-cluster scatter matrix is the sum of the squares of the distances between the center of each cluster and every point in the cluster. In other words, this metric, which is also known as "Variance Ratio Criterion", is measured as the sum of the ratio of the internal dispersion of clusters and the dispersion between clusters, which makes it also a good choice while working with non convex clusters. On the other hand, the Silhouette score is calculated using the mean intra-cluster distance and the mean closest-cluster distance for each data point, which implies that it measures central tendency and consistency within clustered data.

3.4 Experimental Setup

We ran our experiments on a Linux machine with Intel Core i7-6700 CPU and 32GB RAM, on which 64-bit Ubuntu 20.04 with kernel x86_64 Linux 5.4.0-58-generic was running. Our implementation is written in Python 3.8 utilizing TensorFlow 2.3.1 and the Keras API version 2.4, which is meant to be a high-level API for TensorFlow. Other important libraries that we used are the followings: numpy 1.18.5, pandas 1.1.3, matplotlib 3.3.2, keras-tuner 1.0.1,scikit-learn 0.23.2, and umap-learn 0.4.6.

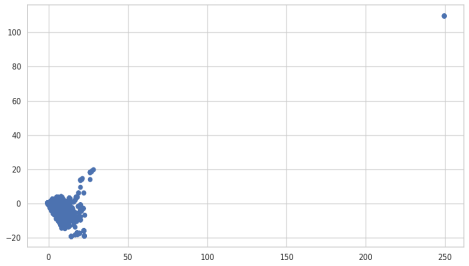
Chapter 4

Results and Analysis

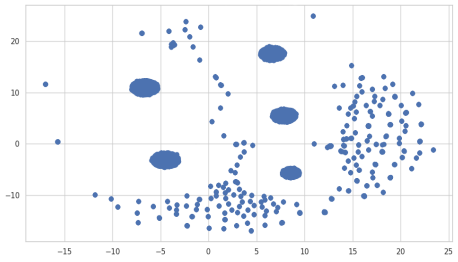
In this chapter, we present and analyse the results of our experiments, where we ran 10 independent tests for each experiment and we present the maximum and the minimum scores, as well as the variance of all of them in separate tables. However, we only display the models' best performance, i.e. clustered data and decision surfaces that correspond to the maximum scores. The main goal is to make predictions and generalizations based on our models, as well as to find some relationships between the log messages and the actual indication of an error in CERN's computer network.

4.1 Data Visualizations

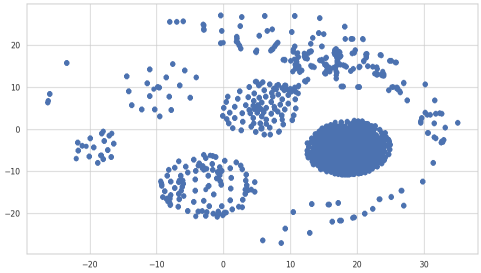
Visualizations capture local and global structure of our data in a way that the presence of clusters is revealed. Moreover, we can get an insight into the presence of anomalies in our dataset. Therefore, we plotted some 2D representations of dataset1 and dataset2, in Figure 4.1 and 4.2. Moreover, we plotted the clean and fraud instances, see Figures 4.3, 4.4, and 4.5. In Figures 4.1 and 4.2, we can see that we get three different 2D representations using three D-R techniques, i.e.PCA, t-SNE, UMAP. We note that by using UMAP and t-SNE, we find the same number of clusters in our data. The only difference between these two methods is that by using UMAP we manage to decrease more the within cluster variance and to increase the between cluster distance. On the other hand, by using PCA, the visualisation is extremely different. It is also worth mentioning that by reading carefully the log messages a user is able to detect 4-5 different groups of messages, while there are a lot of messages that are not related with the others, which is also captured in the Figures 4.1 and 4.2.



(a) PCA embedding

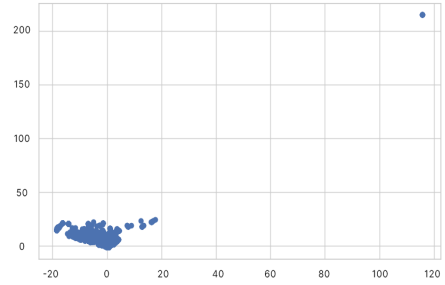


(b) UMAP embedding

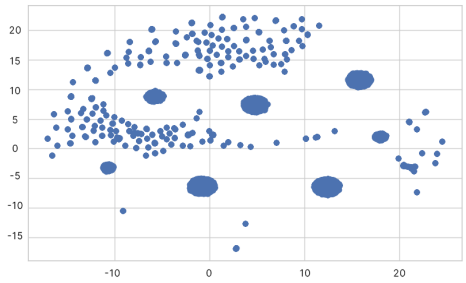


(c) t-SNE embedding

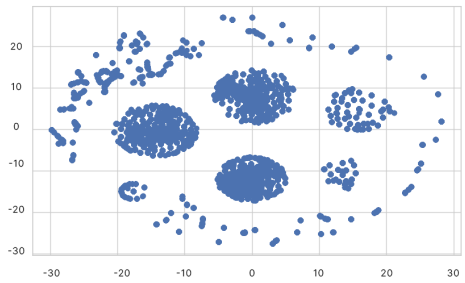
Figure 4.1: A 2D visual representation of dataset1 with different D-R techniques.



(a) PCA embedding



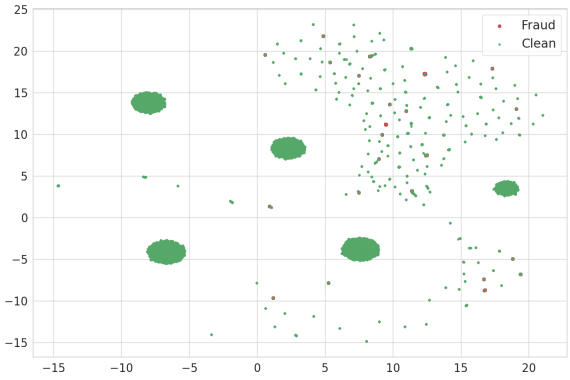
(b) UMAP embedding



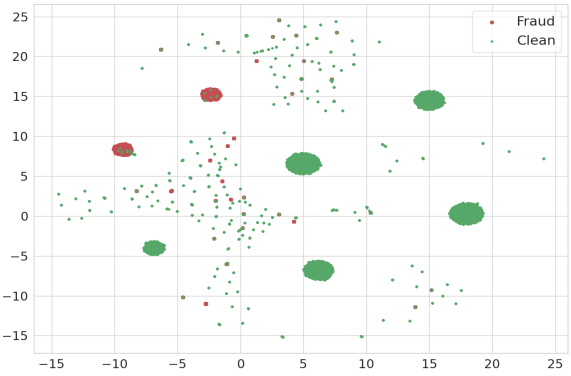
(c) t-SNE embedding

Figure 4.2: A 2D visual representation of dataset2 with different D-R techniques.

As regards the distinction between clean and fraud data, we should have a look at Figures 4.3, 4.4, and 4.5. Apparently, since the dataset2 has smaller class imbalance, we observe more anomalies. We should also underline the fact that in case of the UMAP embedding (Figure 4.3b) the anomalies are grouped in a compact cluster, which introduces a great challenge for the anomaly detection techniques: the "Masking" problem [11], where an anomalous dense cluster can be considered wrongfully as normal. The explanation is simple: a small number of splittings required to isolate a sample indicates anomalies, which are usually far away from clean data. However, when anomalies compose a cluster, the number of partitions is increased, due to higher cluster densities [14], and thus those samples can be wrongly considered as normal.



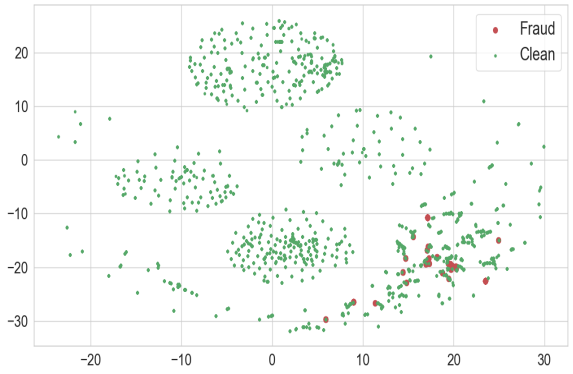
(a) dataset1



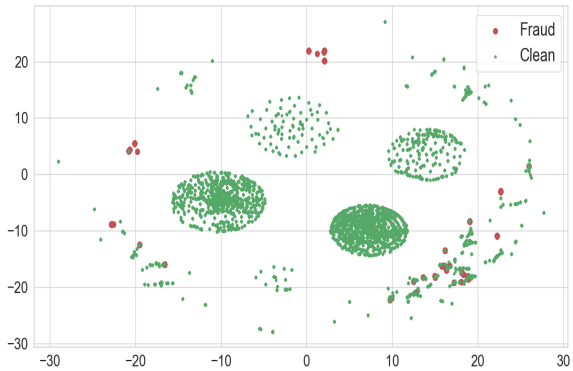
(b) dataset2

Figure 4.3: Visual representations with UMAP of clean and fraud data points.

On the other hand, in case of the PCA and t-SNE embeddings, we do not observe any anomalous clusters, instead all the anomalies are scattered in the space. Nevertheless, the fact that many clean and fraud instances are close together imposes another great challenge for the anomaly detection techniques: the “Swamping“ problem [11], where normal instances can be wrongfully labelled as anomalies. Both “Swamping“ and “Masking“ are the reason that many algorithms require a specific number of anomalies to be declared from the beginning, e.g. in “Isolation Forest“ we specify the “contamination“ attribute, which indicates the proportion of outliers in the dataset.

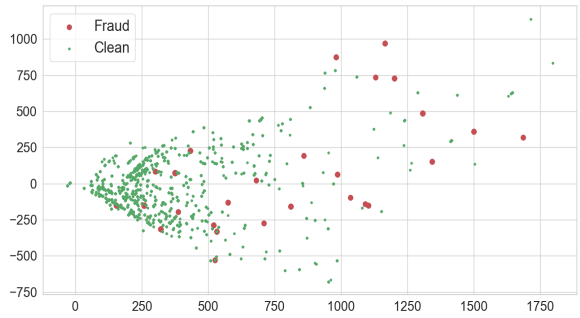


(a) dataset1

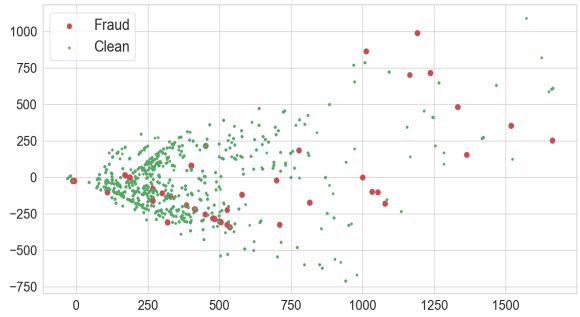


(b) dataset2

Figure 4.4: Visual representations with t-SNE of clean and fraud data points.



(a) dataset1



(b) dataset2

Figure 4.5: Visual representations with PCA of clean and fraud data points.

4.2 Clustering Methods

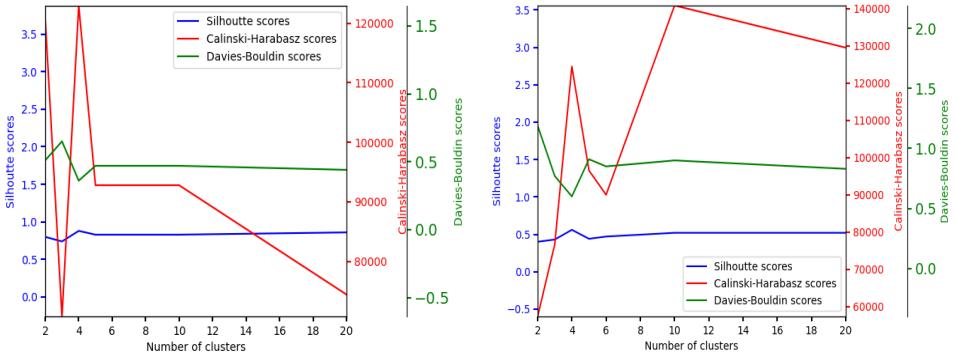
As mentioned in Chapter 3, in this part of the project, we utilised two classical anomaly detection algorithms implemented on the open-source scikit-learn library [32]: 1) *k-Medians*, and 2) *Isolation Forest*.

4.2.1 k-Medians

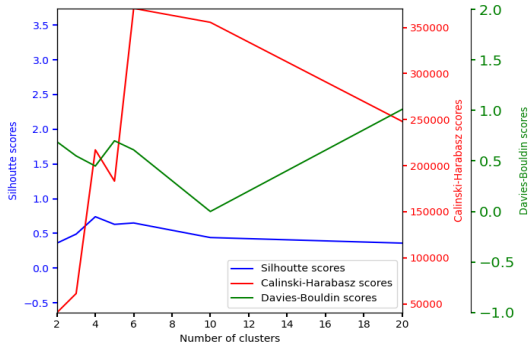
We run "k-Medians" for different number of desired clusters "k" for all three dimensionality reduction techniques, and we evaluate our model with respect to three different metrics, see Figure 4.6 and 4.7: (a) the Silhouette coefficient, (b) the Calinski and Harabasz score, and (c) the Davies–Bouldin score. Although the centroids were re-initialised differently in each of the 10 independent runs, we observed the same scores in the UMAP and t-SNE embeddings, but in PCA we observed two different values in the evaluation phase. The reason could be the fact that we got well-separated clusters by using UMAP and t-SNE;

thus slightly different centroids initialization did not affect the final clustering. Hence, in Tables 4.1- 4.6 we provide the maximum and minimum scores only in case of the PCA embedding (no variance has been estimated).

Choosing the ideal number of clusters is a key component for multiple clustering algorithms, like “k-Medians". Unless our dataset has 2 or 3 dimensions, it is not feasible to capture the best way to group our data visually. In our case, we build our event count matrix, as it has been explained in Section 3, where each of the log messages gets converted into a high dimensional vector representation. Hence, we firstly apply one of the available dimensionality reduction techniques on the high-dimensional samples, and then we evaluate the clustering results based on the aforementioned metrics.

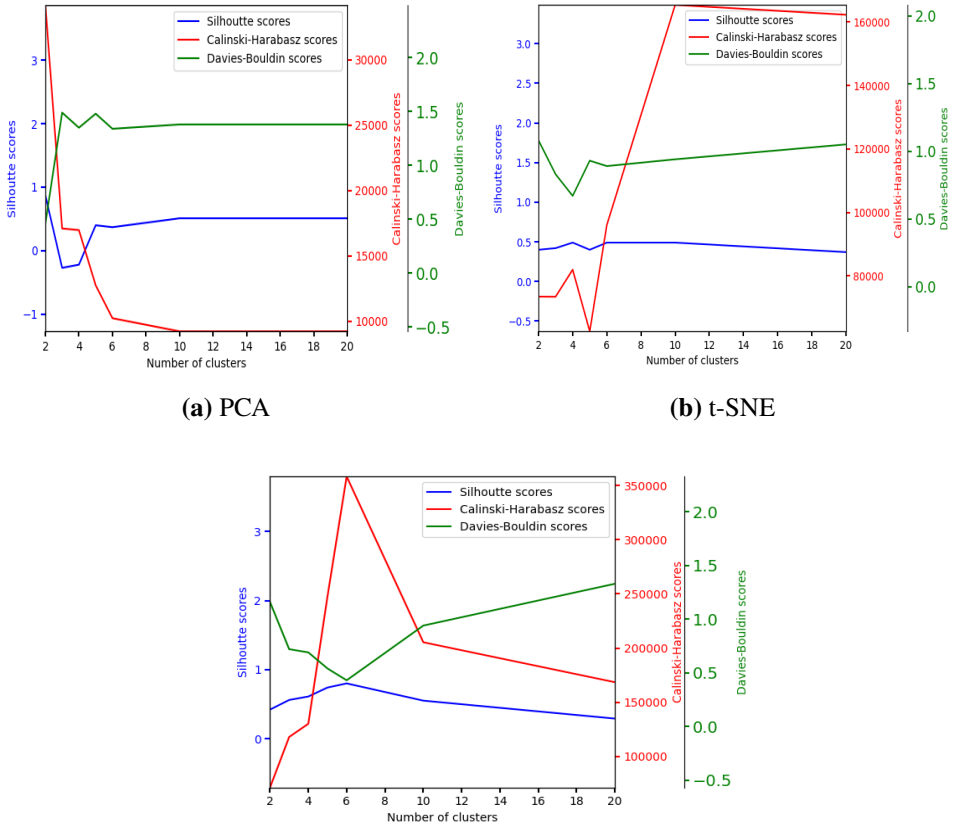


(a) PCA **(b)** t-SNE



(c) UMAP

Figure 4.6: Cluster validation for each dimensionality reduction technique and different number of desired clusters in dataset1.



(a) PCA

(b) t-SNE

(c) UMAP

Figure 4.7: Cluster validation for each dimensionality reduction technique and different number of desired clusters in dataset2.

k-Medians evaluated on Silhouette coefficients			
# clusters	PCA	t-SNE	UMAP
2	[0.80, -0.03]	0.4	0.36
3	[0.74, -0.09]	0.43	0.49
4	[0.88, -0.09]	0.56	0.74
5	[0.83, 0.41]	0.44	0.63
6	[0.83, 0.41]	0.47	0.65
10	[0.83, 0.45]	0.52	0.44
20	[0.86, 0.47]	0.52	0.36

Table 4.1: Evaluation of the “k-Medians“ algorithm with respect to Silhouette coefficients for all dimensionality reduction techniques in dataset1.

k-Medians evaluated on Calinski and Harabasz score			
# clusters	PCA	t-SNE	UMAP
2	[120677.1, 3656.7]	57286.69	40570.11
3	[70724.01, 4614.24]	76672.2	61244.4
4	[122977.7, 4614.24]	124424.55	217206.36
5	[92780.2, 11891.04]	96509.73	183252.32
6	[92780.2, 11891.04]	89984.12	370768.74
10	[92780.2, 12423.8]	140818.15	355655.13
20	[74423.24, 8238.94]	129499.15	248209.59

Table 4.2: Evaluation of the "k-Medians" algorithm with respect to Calinski-Harabasz scores for all dimensionality reduction techniques in dataset1.

k-Medians evaluated on Davies–Bouldin score			
# clusters	PCA	t-SNE	UMAP
2	[0.51, 1.53]	1.19	0.69
3	[0.65, 1.54]	0.77	0.55
4	[0.36, 1.54]	0.6	0.45
5	[0.47, 1.4]	0.91	0.7
6	[0.47, 1.4]	0.85	0.61
10	[0.47, 0.27]	0.9	1.01
20	[0.44, 1.12]	0.83	0.98

Table 4.3: Evaluation of the "k-Medians" algorithm with respect to Davies–Bouldin scores for all dimensionality reduction techniques in dataset1.

k-Medians evaluated on Silhouette coefficients			
# clusters	PCA	t-SNE	UMAP
2	[0.87, 0.14]	0.4	0.42
3	[-0.27, 0.14]	0.42	0.56
4	[-0.22, 0.29]	0.49	0.61
5	[0.4, 0.46]	0.4	0.74
6	[0.37, 0.46]	0.47	0.8
10	[0.51, 0.51]	0.47	0.55
20	[0.51, 0.51]	0.37	0.29

Table 4.4: Evaluation of the "k-Medians" algorithm with respect to Silhouette coefficients for all dimensionality reduction techniques in dataset2.

k-Medians evaluated on Calinski and Harabasz score			
# clusters	PCA	t-SNE	UMAP
2	[234197.4, 6669.36]	73492.17	71243.63
3	[17102.76, 6669.36]	73452.38	117915.44
4	[16981.15, 11448.03]	81991.83	130100.23
5	[12757.02, 8585.95]	62545.29	247667.47
6	[10242.69, 8585.95]	96047.48	358306.2
10	[9238.13, 10208.15]	165419.91	205317.51
20	[9238.13, 7836.99]	162311.69	168445.59

Table 4.5: Evaluation of the "k-Medians" algorithm with respect to Calinski-Harabasz scores for all dimensionality reduction techniques in dataset2.

k-Medians evaluated on Davies–Bouldin score			
# clusters	PCA	t-SNE	UMAP
2	[0.46, 1.25]	1.08	1.16
3	[1.49, 1.25]	0.83	0.72
4	[1.35, 1.63]	0.67	0.69
5	[1.48, 1.7]	0.93	0.54
6	[1.34, 1.7]	0.89	0.43
10	[1.38, 1.56]	0.94	0.94
20	[1.38, 1.29]	1.05	1.33

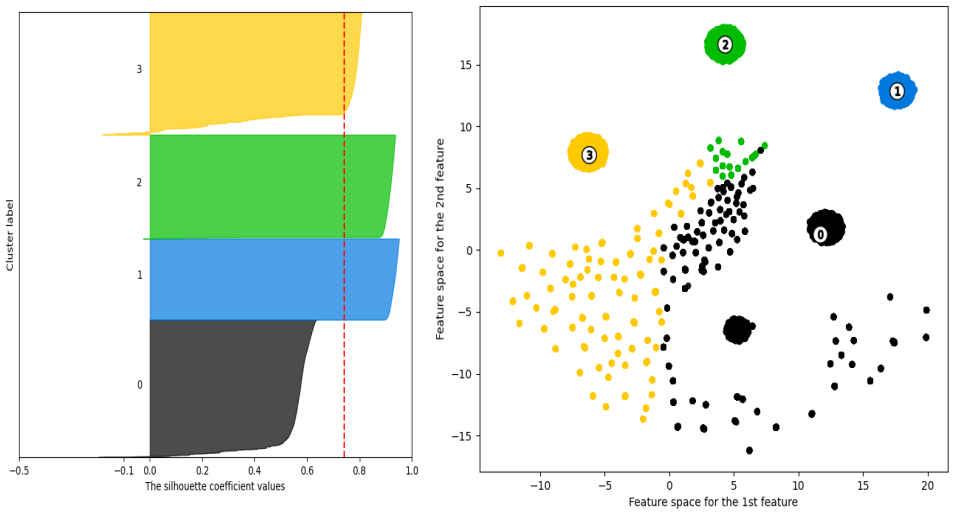
Table 4.6: Evaluation of the "k-Medians" algorithm with respect to Davies–Bouldin scores for all dimensionality reduction techniques in dataset2.

Silhouette Coefficient Analysis

In Figure 4.6, and Tables 4.1, 4.2, and 4.3, we present how all dimensionality reduction methods affect the clustering results in dataset1. First of all, we should underline the fact that $k = 4$ gives the best clustering results in terms of Silhouette score with respect to all dimensionality reduction techniques. As regards the UMAP embedding, we can see in Figure 4.8 the best case scenario with 4 desired clusters. Nevertheless, we should point out that when the clustering number is getting increased, the silhouette scores are getting decreased, like in cases $k = 5$ till $k = 20$, see Table 4.1. The explanation about that is the following: in Figures 4.1 and 4.3 , we confirm that our data can be grouped in 5 compact clusters using UMAP. On the other hand, there are many instances that are far away from these compact groups; thus, the higher the k values, the more clusters are created and each new point gets assigned to

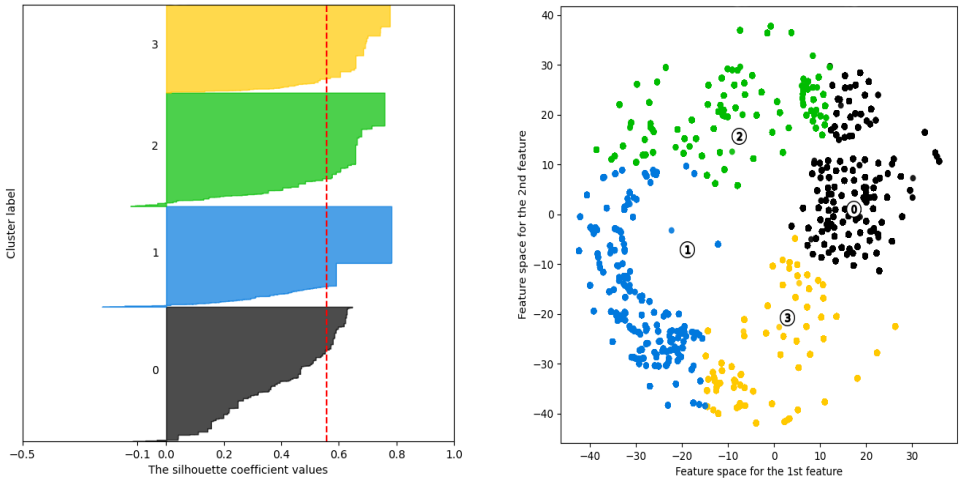
a new cluster. What is worse, is that while the number of clusters is getting increased, the compact clusters are getting divided, which is an undesired result.

Regarding the Silhouette analysis in the t-SNE embedding, we also found that the best clustering result corresponds to 4 desired clusters, see Table 4.1. For the same reason as we explained above, a higher number of desired clusters in this case gives an undesired result. As a consequence, the Silhouette scores is smaller for higher number of k, see Table 4.1. As regards the PCA technique, even though we observe a very different way that our data are scattered in the space, we found that the best clustering result corresponds to 4 desired clusters, see Table 4.1. Last but not least, we should highlight the fact that there are some outliers far away from the majority of the samples, which do not get assigned to a different cluster for all different numbers of the desired clusters that we explored. The algorithm does that, because it does not "care about" cluster density; hence, it splits large radius clusters and merges small radius ones, as explained in [8]. So, in Figure 4.10, "k-Medians" divides the data of the large dense cluster into unequally-populated clusters and assigns the outliers (blue points) to one of the clusters.



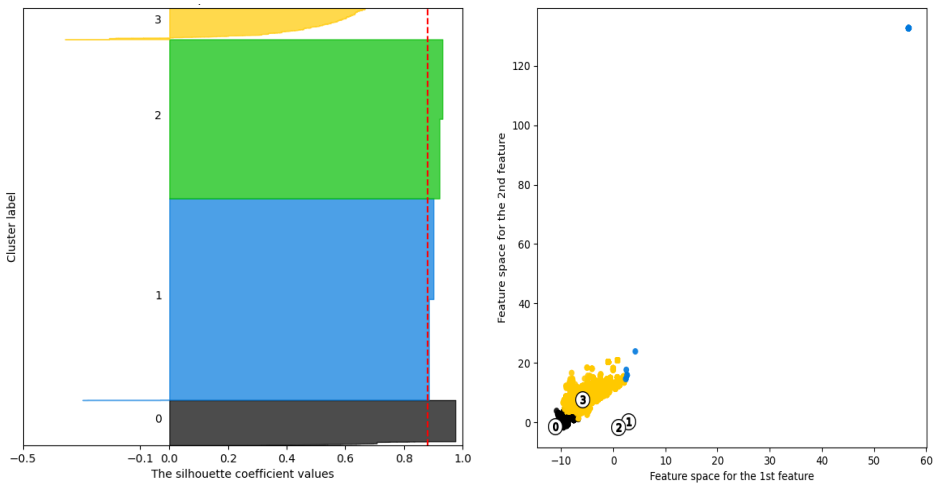
(a) Silhouette coefficients **(b)** Clustered data

Figure 4.8: Clustered data using UMAP embedding for the best case scenario in the k-Medians algorithm ($k = 4$) in dataset1: (a)Silhouette coefficient values for all cluster labels, (b)Visualization of all clustered data.



(a) Silhouette coefficients **(b)** Clustered data

Figure 4.9: Clustered data using t-SNE embedding for the best case scenario in the k-Medians algorithm ($k = 4$) in dataset1: (a) Silhouette coefficient values for all cluster labels, (b) Visualization of all clustered data.

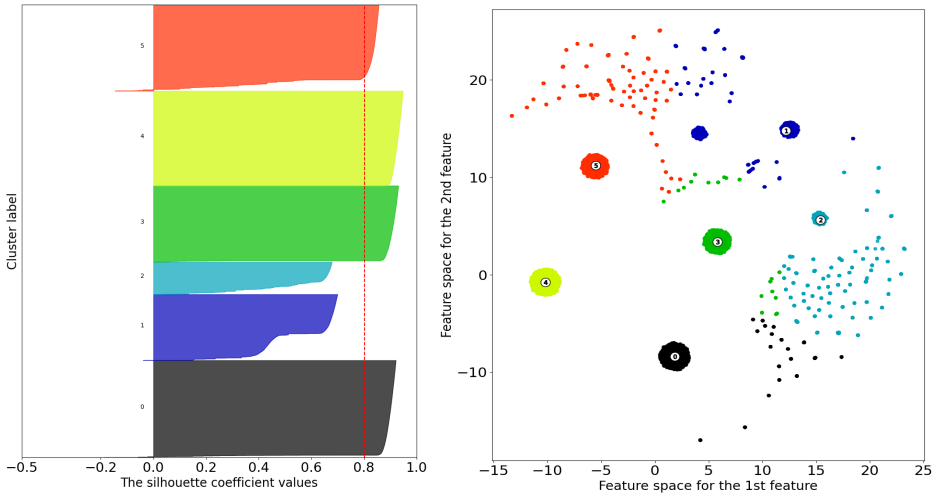


(a) Silhouette coefficients **(b)** Clustered data

Figure 4.10: Clustered data using PCA embedding for the best case scenario in the k-Medians algorithm ($k = 4$) in dataset1: (a) Silhouette coefficient values for all cluster labels, (b) Visualization of all clustered data.

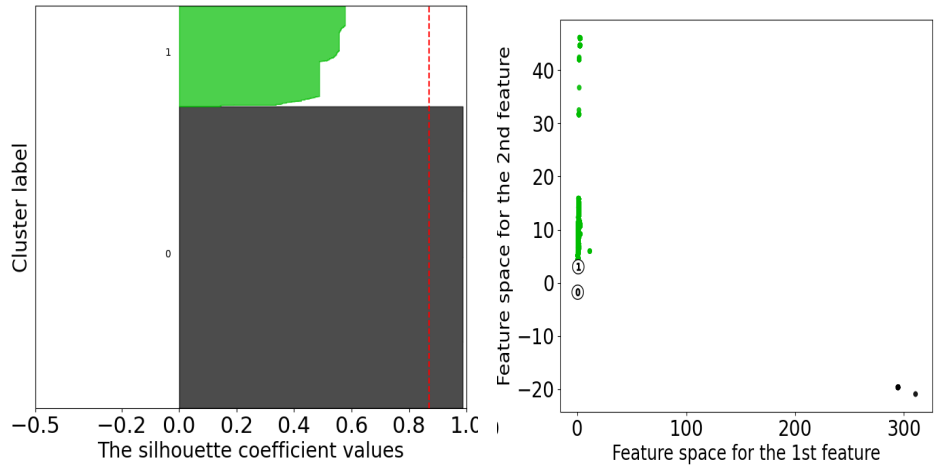
In Figure 4.7, and Tables 4.4, 4.5, and 4.6, we present the evaluation on the clustering results in dataset2. In respect to the PCA embedding in Figure 4.12, we observed that $k = 2$ is the best desired cluster value, whereas in the UMAP

case, the best score corresponds to $k = 6$, see Figure 4.11. As for the t-SNE embedding, the best result is $k = 4$, see Figure 4.13.



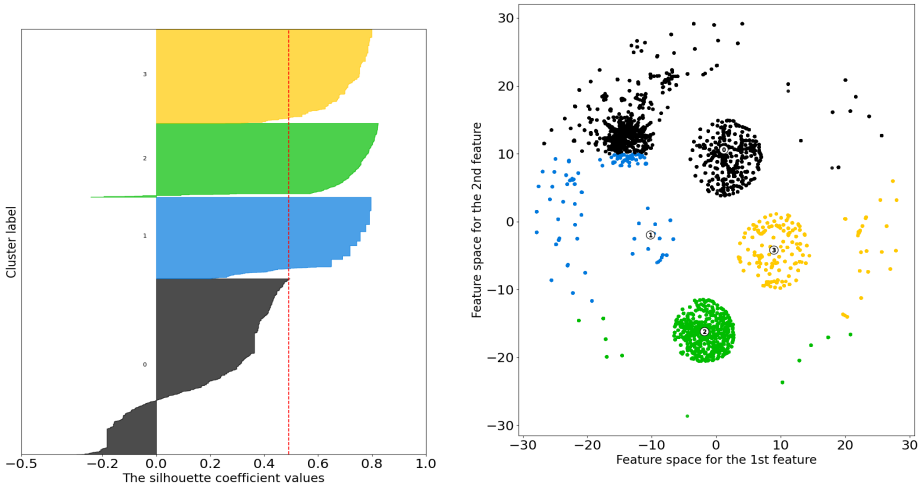
(a) Silhouette coefficients **(b)** Clustered data

Figure 4.11: Clustered data using UMAP embedding for the best case scenario in the k-Medians algorithm ($k = 6$) for dataset2: (a)Silhouette coefficient values for all cluster labels, (b)Visualization of all clustered data.



(a) Silhouette coefficients **(b)** Clustered data

Figure 4.12: Clustered data using PCA embedding for the best case scenario in the k-Medians algorithm ($k = 2$) for dataset2: (a)Silhouette coefficient values for all cluster labels, (b)Visualization of all clustered data.



(a) Silhouette coefficients **(b)** Clustered data

Figure 4.13: Clustered data using t-SNE embedding for the best case scenario in the k-Medians algorithm ($k = 4$) for dataset2: (a)Silhouette coefficient values for all cluster labels, (b)Visualization of all clustered data.

Calinski and Harabasz Score Analysis

In Table 4.2, we present the clustering results based on the Calinski-Harabasz scores about dataset1. Taking into consideration that the optimal clustering result is the one with the highest Calinski-Harabasz score, we select $k = 6$ in the UMAP embedding, but $k = 10$ in the t-SNE, which are both different from the ones we found in the Silhouette coefficient analysis. As regards the PCA embedding, we still got the best score for $k = 4$ desired clusters. With respect to dataset2, all the evaluation results are presented in Table 4.5, where we can see that $k = 2$ is the best value in the PCA embedding, $k = 10$ in the t-SNE embedding, and $k = 6$ in the UMAP case in terms of Calinski-Harabasz score.

Davies–Bouldin Score Analysis

In Table 4.3, we can see the results based on Davies–Bouldin scores, where the optimal clustering is the one with the smallest Davies–Bouldin score. A remarkable observation is that the best clustering result corresponds to 4 desired clusters ($k = 4$) for all three dimensionality reduction techniques. Hence, Davies–Bouldin score gives us the same results as Silhouette coefficients. With respect to dataset2, all the evaluation results are presented in Table 4.6, where $k = 2$ is the best value in the PCA embedding, $k = 4$ in the t-SNE embedding, and $k = 6$ in the UMAP case in terms of Davies–Bouldin index.

4.2.2 Isolation Forest

The Isolation Forest strategy has been tested and evaluated on our datasets returning the results in Tables 4.7 - 4.10 and Figures 4.14 to 4.16. The PCA embedding seems the best technique with respect to the highest Silhouette and Calinski-Harabasz, and the smallest Davies-Bouldin score that we got. Moreover, we should mention that we run this model for 10 independent runs, and we present both the best and worst scores for all dimensionality reduction techniques, as well as the variance for all of them. However, we display the prediction results only for the best case scenarios for PCA, t-SNE and UMAP embedding.

In general, we know that we should handle two challenges: the "Masking" and the "Swamping" issue. In Figure 4.14b, it seems that we can isolate many scattered anomalous data, as well as the anomalous clusters, which implies that we can handle the 'Masking' issue, where an anomalous cluster is considered as normal. As explained in Chapter 2, it becomes hard to detect those points as anomalous, because we have a big number of partitions for those samples, due to the high cluster's inner covariance, see Figures 4.3(b) and 4.14(b). However, the problem in Figure 4.14b is that many clean scattered samples are classified as anomalous, which makes UMAP a non optimal solution with respect to F1-score and AUC. This is known as the "Swamping" problem. Nevertheless, PCA provides more promising clustering results in terms of the evaluation metrics, whereas t-SNE is also a non optimal technique, similar to UMAP.

Isolation Forest evaluations (best case)			
Method	Silhouette	Calinski-Harabasz	Davies–Bouldin
PCA	**0.861**	**29367.209**	**0.512**
t-SNE	0.471	5753.966	1.519
UMAP	0.326	5059.06	1.596

Table 4.7: Evaluation metrics for all dimensionality reduction techniques in dataset1.

Isolation Forest evaluations (worst case)			
Method	Silhouette	Calinski-Harabasz	Davies–Bouldin
PCA	0.8604	29266.63	0.513
t-SNE	0.319	2656.818	2.406
UMAP	0.318	3677.39	1.965

Table 4.8: Evaluation metrics for all dimensionality reduction techniques in dataset1.

Isolation Forest evaluations (variance)			
Method	Silhouette	Calinski-Harabasz	Davies–Bouldin
PCA	2.84e-08	1185.772	1.557e-07
t-SNE	0.00329	1557087.87	0.137
UMAP	8.58e-06	343767.23	0.024

Table 4.9: Variance of metrics for all dimensionality reduction techniques in the "Isolation Forest" model in dataset1.

Isolation Forest evaluations (best case)			
Method	Silhouette	Calinski-Harabasz	Davies–Bouldin
PCA	**0.859**	**29835.32**	**0.565**
t-SNE	0.352	1933.17	3.08
UMAP	0.315	3983.45	1.886

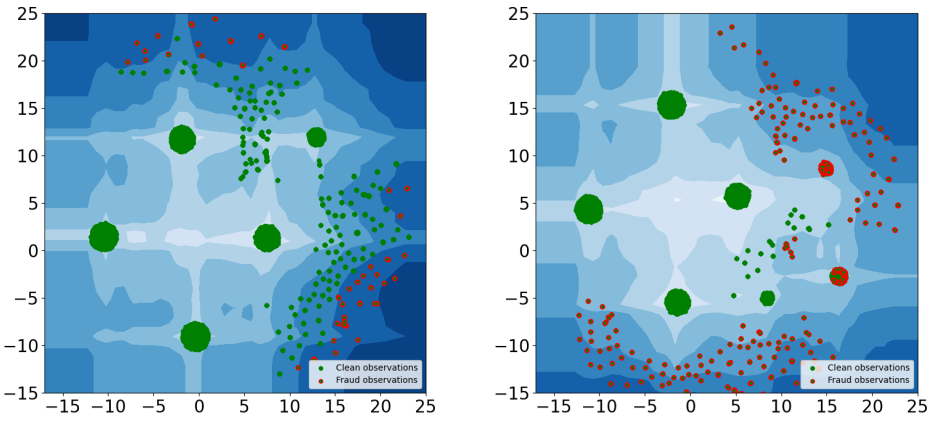
Table 4.10: Evaluation metrics for all dimensionality reduction techniques in the "Isolation Forest" model in dataset2.

Isolation Forest evaluations (worst case)			
Method	Silhouette	Calinski-Harabasz	Davies–Bouldin
PCA	0.841	29277.66	0.722
t-SNE	0.339	69.13	17.97
UMAP	0.308	2871.82	2.447

Table 4.11: Evaluation metrics for all dimensionality reduction techniques in the "Isolation Forest" model in dataset2.

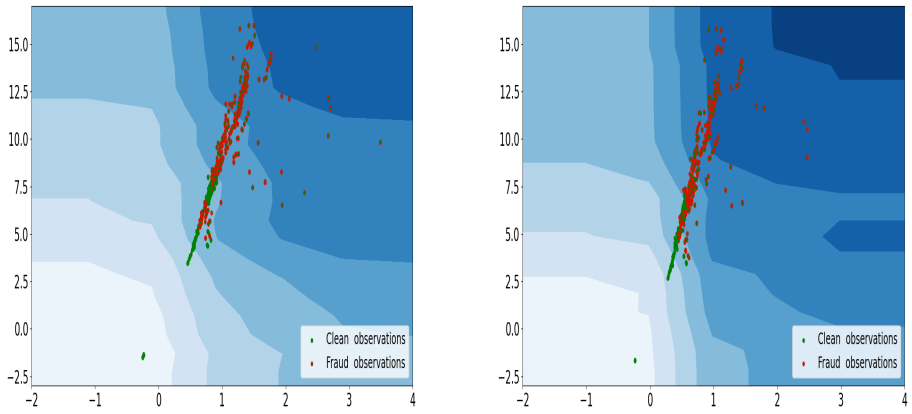
Isolation Forest evaluations (variance)			
Method	Silhouette	Calinski-Harabasz	Davies–Bouldin
PCA	8.05e-05	70203.0	0.006
t-SNE	2.26e-05	478035.2	34.467
UMAP	6.72e-06	167511.82	0.041

Table 4.12: Variance of metrics for all dimensionality reduction techniques in the "Isolation Forest" model in dataset2.



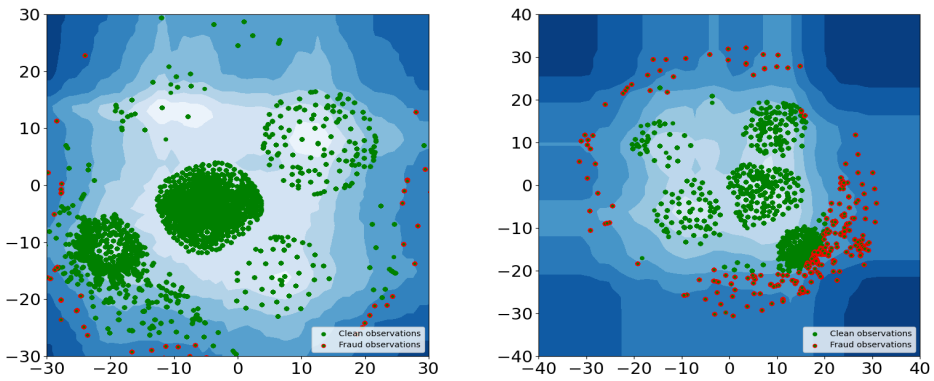
(a) dataset1 **(b)** dataset2

Figure 4.14: Anomaly score map and predicted outliers with UMAP (best case). Normal data are colored green and predicted outliers are colored red. Darker areas indicate higher anomaly scores.



(a) dataset1 **(b)** dataset2

Figure 4.15: Anomaly score map and predicted outliers with PCA (best case). Normal data are colored green and predicted outliers are colored red. Darker areas indicate higher anomaly scores.



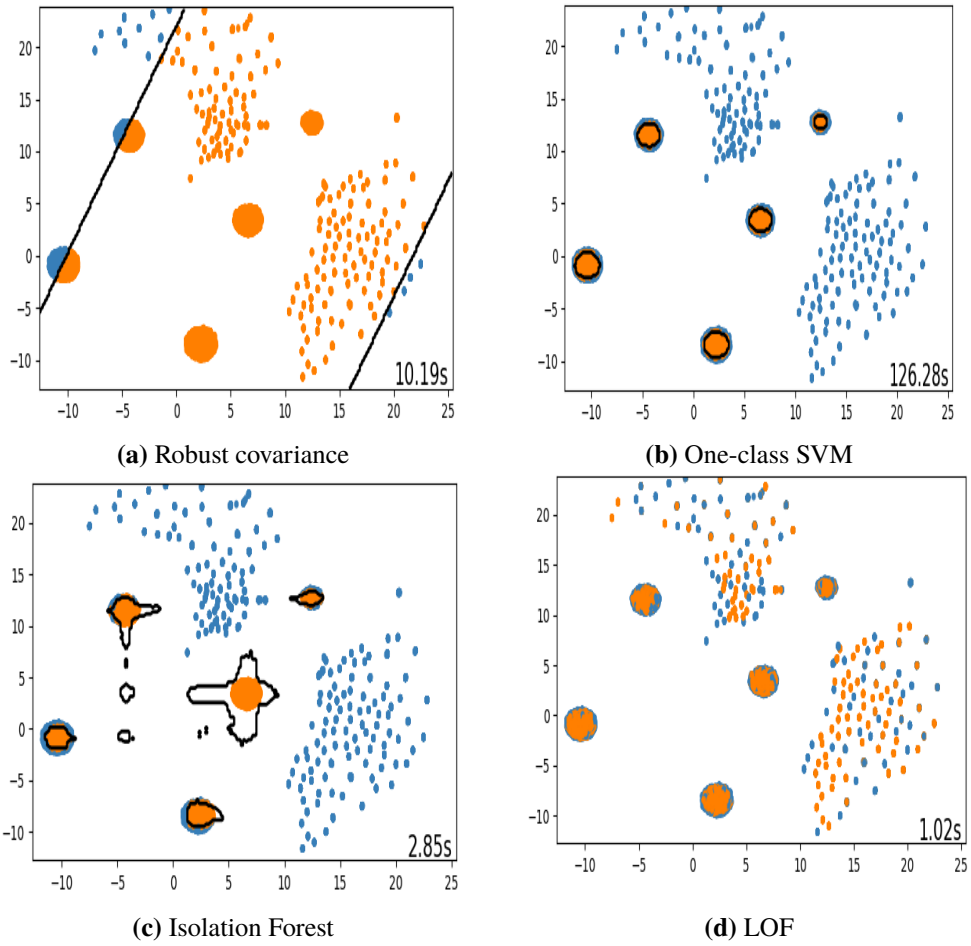
(a) dataset1 **(b)** dataset2

Figure 4.16: Anomaly score map and predicted outliers with t-SNE(best case). Normal data are colored green and predicted outliers are colored red. Darker areas indicate higher anomaly scores.

4.2.3 Other Methods: Robust Covariance, One-class SVM, and LOF

Figure 4.17 draws a comparison of four anomaly detection algorithms introduced in the Background. As regards the One-Class SVM algorithm, it performs very well when the data are two well-separated groups and non-Gaussian, considering the paper [11]. If we compare Figures 4.3 and 4.17(b), we observe that this method misclassifies many of the clean data that do not belong in a compact cluster. In addition, it detects outliers close to the compact clusters. Regarding the Isolation Forest, although it also misclassifies many of the clean data that do not belong in any compact cluster, it creates some decision surfaces around the compact clusters which make this model a better choice than the One-Class SVM.

As regards the Local Outlier Factor, it finds outliers inside the compact clusters where there are no anomalous instances, see Figures 4.17(d) and 4.3. This is the main reason that we still prefer the Isolation Forest model. Last but not least, Robust covariance performs pretty different from all the other methods. If we compare Figures 4.3 and 4.17(a), we observe that many clean data are misclassified as anomalous points. Therefore, we dropped out this model, since it does not isolate properly the outliers.



(a) Robust covariance

(b) One-class SVM

(c) Isolation Forest

(d) LOF

Figure 4.17: Predictions results using UMAP for different algorithms.

4.3 Deep Learning Models

This section presents our results obtained under the two experimental deep learning architectures implemented through the tensorflow-keras python library: tensorflow version 2.3.1 and tf.keras version 2.4 . We test each model for 10 independent runs similar with the clustering methods.

4.3.1 LSTM Model

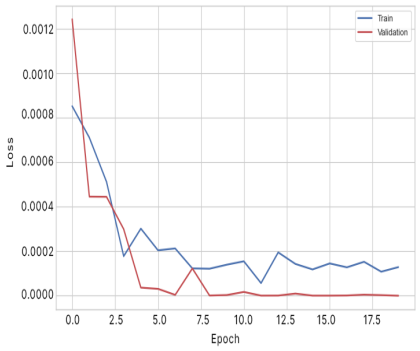
In the LSTM based neural network, we used two different datasets, which have been described in Section 3: (i) dataset1, with high class imbalance, and (ii) dataset2, with lower class imbalance. The evaluation metrics from the testing phase are presented in Tables 4.15, 4.16, 4.17 regarding the best scenarios.

In Figure 4.18a, we can see the training and validation loss of the dataset1 after 20 epochs, where we trained and tested our model using the *Adam* optimizer and a learning rate equal to 10^{-4}. In addition, we can see the ROC curve from our model's evaluation, which reached a test accuracy of 99.83 % on 30171 instances. However, we observe that the validation loss is lower than the training loss, which basically means that the data distribution in the validation set differs from the training set. As we already explained in Figure 4.1, there are a lot of data points that are scattered in the space and do not formulate a compact cluster, whereas other samples are only assigned in one of those compact groups. Therefore, if the validation set mostly consists of samples from the compact clusters, while the training set includes all the others, it is possible to have lower loss values for those instances. Moreover, another reason for this could be that during training we use the regularization technique of dropout, which is not used for the validation phase. In any case, having lower validation loss is not actually an indication that our model performs badly, regarding the nature of our data.

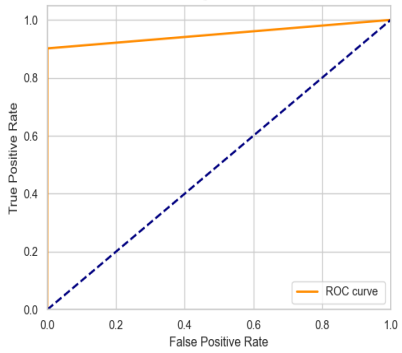
Furthermore, in Figure 4.18c, we can see the training and validation loss of the dataset2 after 20 epochs, as well as the ROC curve that we got from our network's evaluation, which reached a test accuracy of 99.51 % on 40171 instances. As for the hyperparameters' settings, we used *SGD* with a learning rate equals to 10^{-4} based on the fine tuning results for dataset2. Although, we trained our model with a different optimizer we observe that the validation loss is still lower than the training, but we have no big oscillations comparing to figure 4.18a, where we trained our neural network with *Adam*. The reason that we used *Adam* instead of *SGD* for dataset1 is that with *SGD* we had a bad AUC score equals to 0.5, which means that our predictions were totally random and our model could not learn anything at all. On the other hand, in dataset2 we used SGD, because our model performed better (compared to using Adam) in terms of the aforementioned evaluation metrics.

LSTM model evaluations (max scores)					
Data set	F1-score	Precision	Recall	Loss	AUC
dataset1	99.89 %	99.89%	99.89 %	0.897 %	0.94
dataset2	99.51%	99.51%	99.51%	2.45 %	0.991

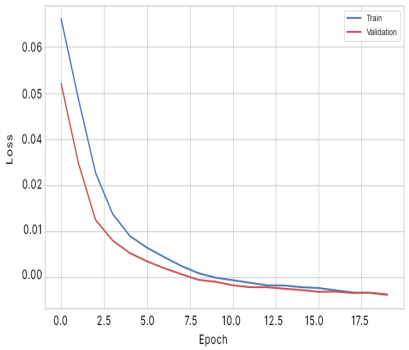
Table 4.13: Evaluation metrics of the LSTM model's performance



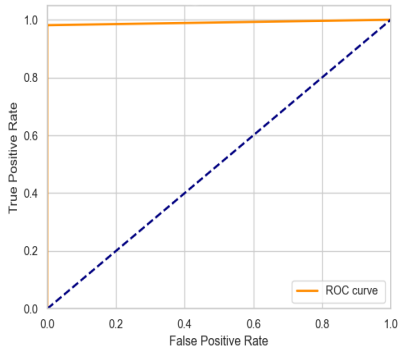
(a) Train-Validation loss



(b) ROC curve



(c) Train-Validation loss



(d) ROC curve

Figure 4.18: Validation results about the LSTM-based model (a) Dataset 1: Train and Validation loss; (b) Dataset 1: ROC curve; (c) Dataset 2: Train and Validation loss; (d) Dataset 2: ROC curve

LSTM model evaluations (min scores)					
Data set	F1-score	Precision	Recall	Loss	AUC
dataset1	99.82 %	99.82 %	99.82 %	1.39 %	0.928
dataset2	99.50%	99.50%	99.50%	2.44 %	0.9903

Table 4.14: Evaluation metrics of the LSTM model's performance

LSTM model evaluations (variance)					
Data set	F1-score	Precision	Recall	Loss	AUC
dataset1	1.58e-07	1.58e-07	1.58e-07	0.038	3.14e-05
dataset2	2.5e-09	2.5e-09	2.5e-09	3.03e-05	1.23e-07

Table 4.15: Variance of metrics of the LSTM model's performance

	Clean	Anomalies
precision	1.0	0.93
recall	1.0	0.86
f1-score	1.0	0.89
samples	29927	244

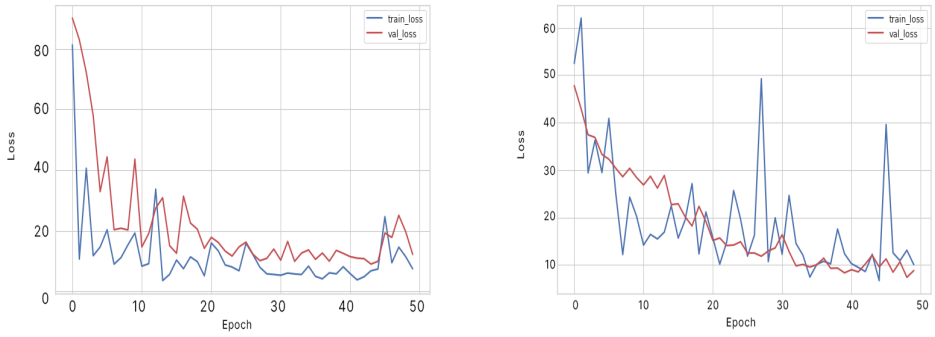
Table 4.16: Classification report on the testing set in Dataset 1

	Clean	Anomalies
precision	0.99	1.0
recall	1.0	0.98
f1-score	1.0	0.99
samples	29927	10244

Table 4.17: Classification report on the testing set in Dataset 2

4.3.2 Hybrid Variational Autoencoder Model

The VAE is an unsupervised learning algorithm that we utilised in order to approach the problem of anomaly detection. In Figure 4.19, we can see the training and validation losses for both datasets in 50 epochs. Although the training losses have big oscillations, the validation losses seem to be smoother, especially in dataset2. Moreover, we measure the performance of our model by computing the MSE between the input and output messages. Obviously, higher values of the reconstruction error represent an indication of system vulnerabilities, since we expect them to represent messages from the 'error' category. However, there are clean data that our model also finds difficult to reconstruct, which also need to be classified again, since they *'confuse'* it.



(a) dataset1 **(b)** dataset2

Figure 4.19: Training and validation loss (a) dataset1, and (b) dataset2.

In Figure 4.20, we visualised the MSE values for both clean and anomalous data. In the next step, we find a threshold in the MSE values, in order to classify again those instances that our model cannot reconstruct easily. In particular, we

use the Decision Tree strategy in order to find this threshold for both datasets. Therefore, all instances with a MSE value above this threshold are getting classified again using the target collected labels. Taking into consideration the partition of the data from the Decision Tree algorithm, we chose the value of the root node to be the threshold for each case. In particular, we choose 915.41 as threshold for dataset1 and 13.32 for dataset2, see Figure 4.21.

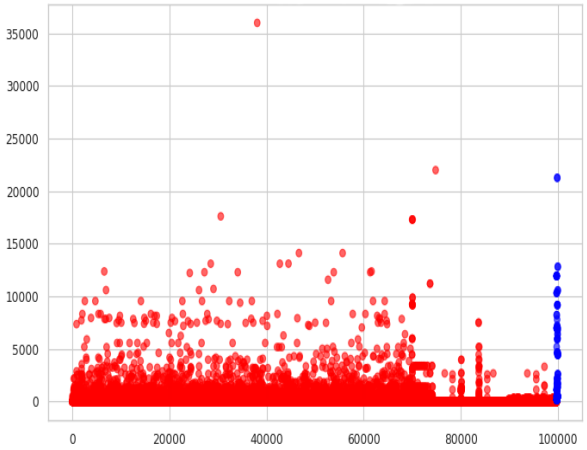
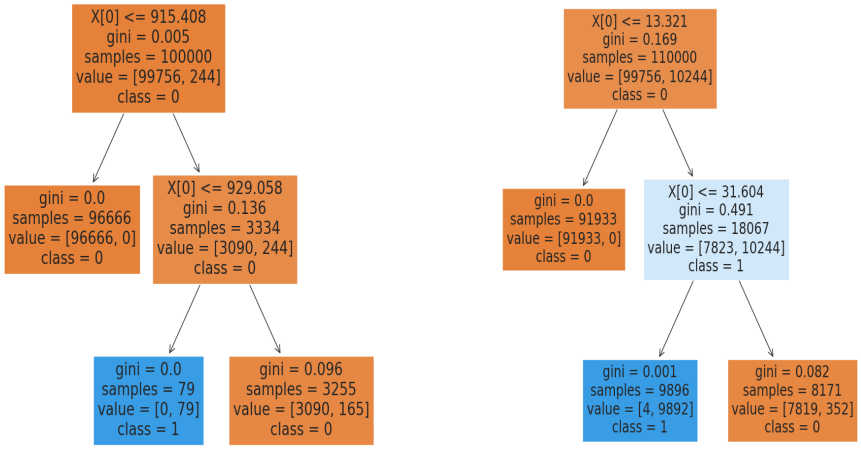


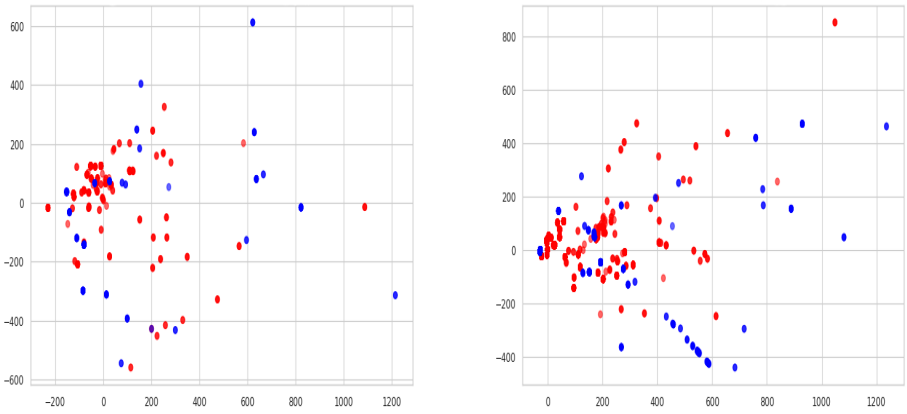
Figure 4.20: MSE values (y-axis) for clean (red) and anomalous (blue) data (x-axis).

The next step is the isolation of these instances that correspond to the reconstructions with high MSE values. In Figure 4.22, we visualise (with PCA embedding) those instances that Variational Autoencoder cannot reconstruct successfully. As we observe, they are not linearly separable, and thereby we used the SVM algorithm with a RBF kernel function as our classification method, to perform a supervised learning. We fine tuned the SVM model's parameters, γ and C, using grid search strategy, as presented in Figure 4.23. As a result, we chose $\gamma = 0.1$ and $C = 1.0$, for which we have the best model's performance. Finally, we evaluate the performance of SVM using the F1-score and AUC-ROC metrics, as presented in Tables 4.20, 4.21, and 4.22. Then we plot the decision surfaces of our classifiers for different C values, see Figure 4.25 and 4.26.



(a) dataset1 **(b)** dataset2

Figure 4.21: Illustration of Decision Tree algorithm using the MSE values.



(a) dataset1 **(b)** dataset2

Figure 4.22: Illustration of instances with MSE values above threshold (with PCA): (a)Dataset1: 683 instances out of the 30171 testing data should be classified again, and (b)Dataset2: 13511 instances out of the 40171 testing data should be classified again.

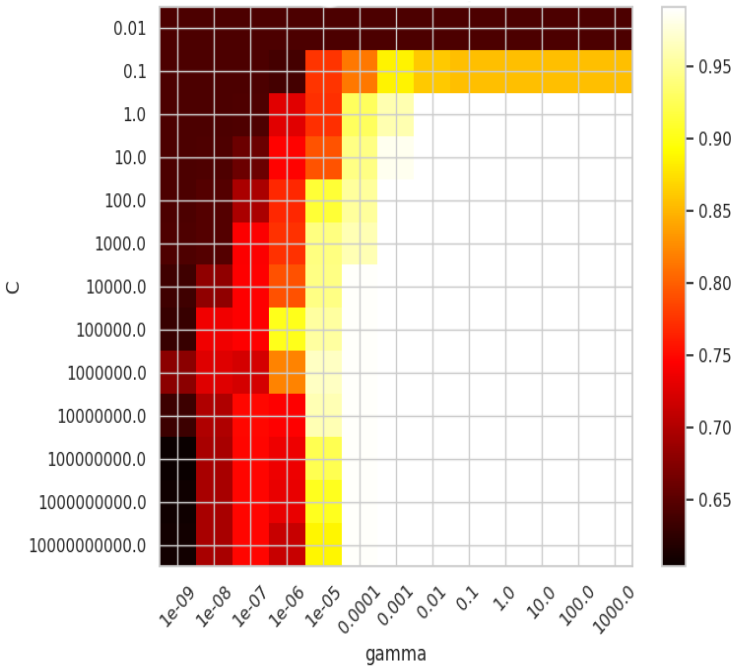
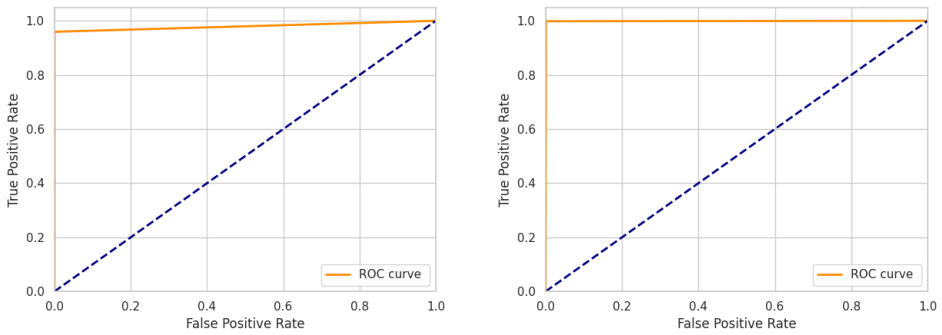


Figure 4.23: A 2D visualisation of the fine tuning results of the SVM model's parameters (gamma and C) using grid search. The Heat-map score list is on the right, where colors represent how the model's performance changes; the higher the score the lighter the color.



(a) dataset1 **(b)** dataset2

Figure 4.24: Evaluation results for the SVM with RBF kernel classifier with AUC-ROC curve for both datasets. The AUC metric is equal to 0.979 for dataset1 and 0.998 for dataset2.

Hybrid VAE model evaluations (max scores)					
Data set	F1-score	Precision	Recall	Accuracy	AUC
dataset1	97.93 %	100 %	95.95 %	98.54 %	0.979
dataset2	99.9 %	99.97 %	99.84 %	99.85 %	0.998

Table 4.18: Evaluation metrics of the Hybrid VAE model's performance in Dataset1 (big class imbalance) and Dataset2 (small class imbalance).

Hybrid VAE model evaluations (min scores)					
Data set	F1-score	Precision	Recall	Accuracy	AUC
dataset1	93.62 %	100.0 %	88.0 %	97.21 %	0.941
dataset2	95.84 %	99.97 %	95.71 %	97.99 %	0.956

Table 4.19: Evaluation metrics of the Hybrid VAE model's performance in Dataset1 (big class imbalance) and Dataset2 (small class imbalance).

Hybrid VAE model evaluations (variance)					
Data set	F1-score	Precision	Recall	Accuracy	AUC
dataset1	0.0004	0.0	0.0012	2.76e-05	0.0003
dataset2	0.00034	0.0	0.002	0.00013	0.00033

Table 4.20: Evaluation metrics of the Hybrid VAE model's performance in dataset1 (big class imbalance) and Dataset2 (small class imbalance).

	Clean	Anomalies
precision	0.98	1.00
recall	1.00	0.96
f1-score	0.99	0.98
samples	131	74

Table 4.21: Evaluation results of the SVM classifier for dataset1

	Clean	Anomalies
precision	1.0	1.0
recall	1.0	1.0
f1-score	1.0	1.0
samples	1008	3046

Table 4.22: Evaluation results of the SVM classifier for dataset2

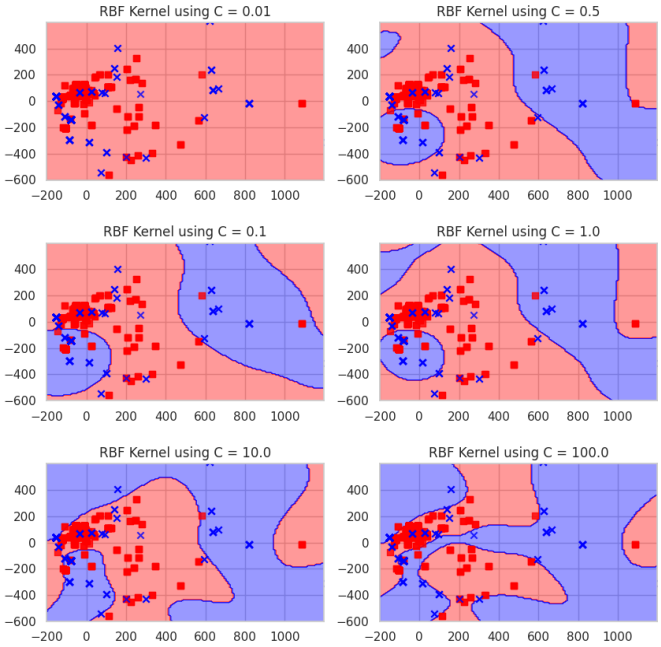


Figure 4.25: Decision surfaces of the SVM for different C values (dataset1).

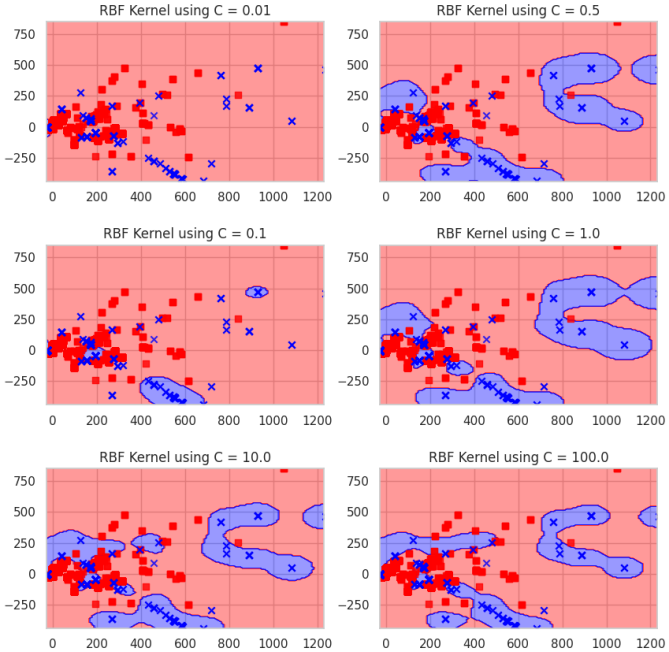


Figure 4.26: Decision surfaces of the SVM for different C values (dataset2).

Chapter 5

Discussion

In this chapter, we reflect over the results for both clustering methods and deep neural networks. In particular, we provide the reader with an interpretation of our results, in order to give them an insight over all the obtained models. To begin with, we highlight the fact that choosing the appropriate dimensionality reduction technique played a great role on the clustering methods' performance. Secondly, we need to point out that deep learning models outperformed the traditional clustering techniques with respect to the final predictions. However, if we had preprocessed the data in the same way for both the traditional clustering methods and the deep neural networks, our results might have been more comparable, but a deep learning approach might still be preferred, intuitively. The main reason is that in case of high-dimensional data deep learning models are more powerful tools, which learn features from very complex data.

The explanation about the different data pre-processing is related to the nature of the algorithms. In particular, in the beginning we followed the work of Shilin et.al. [31], where log messages are vectorized as event count vectors. This technique has been proved quite efficient for both the "k-Medians" model, and the "Isolation Forest" algorithm, since we observe that in the best scenarios we achieved a good model's performance varying from 85% to 90% in terms of Silhouette coefficients. On the other hand, in the LSTM-based model and in the Variational Autoencoder we used "Word Embeddings", because of the nature of the deep learning algorithms. Apparently, since we build an LSTM-based network and we use an Embedding layer, as it has been explained in Chapter 3, we need to create embedding representations. More specifically, the embedding and the Embedding layer allows the system to expand each token to a more massive vector; hence, the network represents a word in a meaningful way, which preserves the semantics in the log messages. Hence, we notice that the

deep learning models outperformed the clustering methods for both datasets, since we managed to attain pretty high scores (more than 96%) with respect to the F1-score, precision, recall, and AUC metrics.

5.1 Clustering Methods

The main goal of the “dimensionality reduction" phase in the clustering methods is to use important dimensionality reduction methods, e.g. PCA, t-SNE, UMAP, to combine or merge some features and keep only the significant characteristics of the original data points, given the fact that we work with high-dimensional data. We utilised the aforementioned techniques to check the impact in the performance of our clustering methods. From the results presented, it is clear that in terms of the Silhouette coefficient and the Davies–Bouldin score for dataset1, we can solve the anomaly detection problem using $k = 4$ desired clusters, whereas in terms of the Calinski-Harabasz index for dataset1 the best value is $k = 6$ in the UMAP embedding, $k = 10$ in the t-SNE case, and $k = 4$ in the PCA. On the other hand, using the PCA embedding in dataset2, the optimal cluster value is $k = 2$, whereas $k = 6$ in the UMAP technique in terms of all the aforementioned evaluation metrics. Last but not least, using the t-SNE technique either $k = 4$ would be preferred regarding the Davies-Bouldin and Silhouette index, or $k = 10$ considering the Calinski-Harabasz score.

However, it is extremely important to be able to understand what is the disadvantage of applying this algorithm to our data. Hence, we will revisit this method from a different perspective with regards to the nature of the results. Regarding the final shapes of the clusters, they are not what we would expect, even when the evaluation metrics were pretty high. In particular, we did not take a proper clustering result so as to isolate only the anomalous data in a separate cluster for either PCA, t-SNE, and UMAP. Nevertheless, for any different k-value we got some clusters that consisted either only of clean samples or that had been assigned with both clean and anomalous data points. It is worth to be mentioned that further partitioning in the data with higher k-values results in unwanted clustering, as explained already in Chapter 4. Even though we could isolate the anomalous points in a separate cluster for higher k-values, we would end up with an undesirable partitioning regarding the clean dense clusters.

Let us summarise now the main drawbacks of “k-Medians". First of all, we observed that some outliers, which are far away from the larger clusters, can

be assigned in those clusters instead of getting assigned in a new one, see Figure 4.10. First of all, this happens since the data is unevenly distributed across clusters. The “k-Medians“ algorithm fails to find a good solution, because it cannot adapt to different cluster densities, even when the clusters are well-separated [49]. Hence, it splits bigger clusters and merges small ones, which is an undesired clustering result. Secondly, this algorithm does not converge to the optimal solution, because we did not isolate the anomalous points in a separate cluster. This could be explained since “k-Medians" is a variant of the E-M algorithms[1] and the E-M algorithm may converge to local minimum.

Turning into the “Isolation Forest" model, it is evident that the dimensionality reduction technique is able to affect our model’s performance decisively. In particular, PCA gives significantly improved results comparing to t-SNE and UMAP; i.e. PCA gave us a Silhouette score equals to 0.86, while t-SNE and UMAP gave us only 0.47 and 0.33 respectively. This means that with “Isolation Forest" we cannot solve the anomaly detection problem if we use a dimensionality reduction technique other than PCA for both datasets. The reason that UMAP and t-SNE fail to produce good clustering results is the “Swamping“ issue, since many clean data are labelled as anomalies. As explained in the work of Liu et.al.[7], this happens when the clean instances become more scattered or when they are close to the anomalies, and thus a normal data point might look anomalous.

One proposed solution would be the sub-sampling technique [7], since both swamping and masking are a consequence of too many indistinguishable data. In particular, “Isolation Forest“ builds a tree by utilising many sub-samples to eliminate the results of swamping and masking. For each sub-sample, which has a smaller size than the entire dataset, the algorithm builds a better performing tree. This is reasonable, since sub-samples have fewer anomalies intervening with normal samples, and thus, making clean instances easier correctly recognised as normal, even though they are mostly closer to anomalies. Hence, we might be able solve the “Swamping“ issue for both UMAP and t-SNE, and eventually PCA would not be the only technique performing well.

Another proposed solution regarding the optimisation of “Isolation Forest“ would be to try different values of the “contamination“ attribute, which describes the proportion of outliers in the entire dataset. As mentioned in Chapter

[1]The relationship follows similarly to the relationship with k-means but with Laplace distributions instead of Gaussians.

3, this attribute should be in the range of $[0, 0.5]$. In dataset2 with the small class imbalance, we had 10,244 outliers in 110,000 samples, which gives a proportion of 10%, while in dataset1, we only had 0.25% outliers. Therefore, in dataset1 we used a contamination value equal to 0.03, while in dataset2 we used a 0.1 . However, we tried to increase the aforementioned initialization of this attribute and we observed worse results in terms of the F1-score and the AUC metric. As a consequence, we could run more independent tests for a range of different values of the "contamination" attribute.

5.2 Deep Neural Networks

Due to the fact that we followed a different preprocessing of the data for the LSTM-based model and the Variational Autoencoder, we compare these models separately, although both deep learning architectures are more powerful than traditional machine learning algorithms, as stated above. Regarding the LSTM-based network, we got improved results for dataset2, since we trained the model with more instances of the fraud data. This is a strong benefit of supervised learning; a model's performance gets improved when it can be trained with more data, especially when these extra instances can eliminate the issue of class imbalance [50]. Moreover, the right set of hyper-parameters affected the results, because in case of dataset2 we had to train the network with a different optimizer than in case of dataset1, otherwise our model's predictions were totally random.

Regarding the Variational Autoencoder, which is combined with an SVM classifier, it is considered as a semi-supervised approach, and it can identify the messages that indicate a threat or a failure for the system. In the case of the second dataset, for which the class imbalance issue has been eliminated, we found more reconstructions with higher error. First of all, this is reasonable because in principle we have more erroneous testing samples in the second dataset, see Figures 4.1, 4.2, and 4.3. Moreover, since we train the Variational Autoencoder only with clean samples, we expect that the anomalous instances would have higher reconstruction errors in the testing phase. Taking into consideration that these anomalous instances have been increased in dataset2 and their corresponding reconstruction error would be higher, we clarify that it is an expected result to find more instances with such high reconstruction error in dataset2 than in dataset1. On the other hand, in Figure 4.22 we can see that some of the instances with high reconstruction error are "clean" data, and thus we understand that it might also be difficult to learn features from clean data

when they are not so similar with other instances with which the VAE has been trained on. However, this is not a problem, since we managed to solve this as a simple binary classification problem using an SVM classifier using the subset of samples that were labeled.

Moreover, in Figure 4.22 we visualised the instances with higher reconstruction error using the PCA technique. Therefore, we should compare this representation with Figure 4.5, where we provide a 2D visualisation of how the clean and fraud samples in the entire dataset are spread in the space. We observe that only a small proportion of the data cannot be easily reconstruct from the Variational Autoencoder; 683 instances out of the 30171 testing data in dataset1, and 13511 instances out of the 40171 testing data in dataset2. Taking now into consideration the evaluation metrics of the SVM model in Figure 4.24, we conclude that the hybrid VAE model also performs very good. However, the disadvantage of this approach is that it requires more computational resources, and the training is much slower than in the LSTM-based network.

Chapter 6

Conclusions

This degree project investigated the capability of identifying erroneous messages that indicate instability and failures in large-scale systems by proposing two different LSTM-based networks, which are combined with Word Embeddings, and comparing them with standard anomaly detection techniques following a dimensionality reduction step.

The first proposed LSTM-based model is a supervised learning approach with respect to labeled data collected by the servers of the network. More specifically, it is a binary classification approach which has been proved that it can identify erroneous instances in the various flows of log messages. Turning into the second anomaly detection method, it is a semi-supervised learning strategy, which combines both a Variational Autoencoder (VAE) model and an SVM with an RBF kernel classifier. Specifically, we calculate the MSE of the reconstructions, which are generated by the VAE, and then with respect to a threshold we select those instances with high MSE values to be classified again by using their labels at this point.

Comparing all the proposed models, we conclude that the best approach would be the supervised learning with the LSTM-based model, although "Isolation Forest" and the hybrid "VAE" model gave also good results. First of all, in the "k-Medians" approach we did not manage to properly isolate all the erroneous messages neither using PCA or t-SNE, nor by the UMAP technique, since the anomalous data points are grouped together with the clean instances, even in cases of the best scenarios regarding the higher evaluation scores, see Figures 4.8, 4.9, and 4.10. Secondly, in case of the "Isolation Forest" method, we had an obvious missclassification of many clean samples. None of them gave us very promising results or better than "Isolation Forest", as explained in Chapter

4. Hence, “Isolation Forest“ outperformed the “k-Medians“, as we achieved a better clustering into only two separate clusters, but not the LSTM-based model or the hybrid Variational Autoencoder.

However, if we had time to optimise the “Isolation Forest“ method using the “sub-sampling“ technique or even trying a different data preprocessing method, it would be very interesting to provide further comparisons and conclusions regarding the “Swamping“ issue [7]. Nevertheless, we should point out that even though we did not handle this problem with UMAP and t-SNE, we got promising results with PCA. Last but not least, the semi-supervised learning approach could also detect anomalies in the network, but the only concern would be that the training will be harder in case of bigger datasets, where we would need more computational resources. All in all, regarding the current methodology the LSTM-based model would be a very promising solution.

The problem of “Log Anomaly Detection“ within computer networks’ logging systems is still only scratched on the surface. The number of domain-specific regular expressions is getting increased constantly, which in turn increases the attack surface and motivation for malicious actions. Hence, the research for this subject is still of great importance in order to maintain the integrity and safety of computer networks in all situations. In overall, we should underline the fact that the proposed models are able to capture the normal and abnormal patterns using two datasets that have never been used before, which implies that we can positively answer the research question. Hence, to the research question

“Can we detect threats in CERN’s network using machine learning ?"

we answer that, yes, we can detect erroneous messages in CERN’s network using ML techniques.

Bibliography

[1] Raghavendra Chalapathy and Sanjay Chawla. "Deep learning for anomaly detection: A survey". In: *arXiv preprint arXiv:1901.03407* (2019).

[2] Hanan Hindy et al. "A taxonomy and survey of intrusion detection system design techniques, network threats and datasets". In: *arXiv preprint arXiv:1806.03517* (2018).

[3] Min Du et al. "Deeplog: Anomaly detection and diagnosis from system logs through deep learning". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2017, pp. 1285–1298.

[4] Mohiuddin Ahmed, Abdun Naser Mahmood, and Jiankun Hu. "A survey of network anomaly detection techniques". In: *Journal of Network and Computer Applications* 60 (2016), pp. 19–31.

[5] Donghwoon Kwon et al. "A survey of deep learning-based network anomaly detection". In: *Cluster Computing* 22.1 (2019), pp. 949–961.

[6] Robert DiPietro and Gregory D. Hager. "Chapter 21 - Deep learning: RNNs and LSTM". In: *Handbook of Medical Image Computing and Computer Assisted Intervention*. Ed. by S. Kevin Zhou, Daniel Rueckert, and Gabor Fichtinger. Academic Press, 2020, pp. 503–519. ISBN: 978-0-12-816176-0. DOI: https://doi.org/10.1016/B978-0-12-816176-0.00026-0. URL: http://www.sciencedirect.com/science/article/pii/B9780128161760000260.

[7] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. "Isolation forest". In: *2008 Eighth IEEE International Conference on Data Mining*. IEEE. 2008, pp. 413–422.

[8] Aristidis Likas, Nikos Vlassis, and Jakob J Verbeek. "The global k-means clustering algorithm". In: *Pattern recognition* 36.2 (2003), pp. 451–461.

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[10] Jerone TA Andrews, Edward J Morton, and Lewis D Griffin. "Detecting anomalous data using auto-encoders". In: *International Journal of Machine Learning and Computing* 6.1 (2016), p. 21.

[11] Nicolas Goix. "Machine learning and extremes for anomaly detection". PhD thesis. Paris, ENST, 2016.

[12] Gunnar Ratsch et al. "Constructing boosting algorithms from SVMs: An application to one-class classification". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24.9 (2002), pp. 1184–1199.

[13] S Rasoul Safavian and David Landgrebe. "A survey of decision tree classifier methodology". In: *IEEE transactions on systems, man, and cybernetics* 21.3 (1991), pp. 660–674.

[14] Zhiguo Ding and Minrui Fei. "An anomaly detection approach based on isolation forest algorithm for streaming data using sliding window". In: *IFAC Proceedings Volumes* 46.20 (2013), pp. 12–17.

[15] Zhangyu Cheng, Chengming Zou, and Jianwei Dong. "Outlier detection using isolation forest and local outlier factor". In: *Proceedings of the conference on research in adaptive and convergent systems*. 2019, pp. 161–168.

[16] Sara Nasser, Rawan Alkhaldi, and Gregory Vert. "A modified fuzzy k-means clustering using expectation maximization". In: *2006 IEEE International Conference on Fuzzy Systems*. IEEE. 2006, pp. 231–235.

[17] Yong Gyu Jung, Min Soo Kang, and Jun Heo. "Clustering performance comparison using K-means and expectation maximization algorithms". In: *Biotechnology & Biotechnological Equipment* 28.sup1 (2014), S44–S48.

[18] KA Abdul Nazeer and MP Sebastian. "Improving the Accuracy and Efficiency of the k-means Clustering Algorithm". In: *Proceedings of the world congress on engineering*. Vol. 1. Association of Engineers London. 2009, pp. 1–3.

[19] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[20] Usama Fayyad, Paul S Bradley, and Cory Reina. *Scalable system for expectation maximization clustering of large databases*. US Patent 6,263,337. July 2001.

[21] Christopher Whelan, Greg Harrell, and Jin Wang. "Understanding the k-medians problem". In: *Proceedings of the International Conference on Scientific Computing (CSC)*. The Steering Committee of The World Congress in Computer Science, Computer ... 2015, p. 219.

[22] William S Noble. "What is a support vector machine?" In: *Nature biotechnology* 24.12 (2006), pp. 1565–1567.

[23] Markus M Breunig et al. "LOF: identifying density-based local outliers". In: *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*. 2000, pp. 93–104.

[24] Norm A Campbell. "Robust procedures in multivariate analysis I: Robust covariance estimation". In: *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 29.3 (1980), pp. 231–237.

[25] Jitendra Kumar, Rimsha Goomer, and Ashutosh Kumar Singh. "Long Short Term Memory Recurrent Neural Network (LSTM-RNN) Based Workload Forecasting Model For Cloud Datacenters". In: *Procedia Computer Science* 125 (2018). The 6th International Conference on Smart Computing and Communications, pp. 676–682. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2017.12.087. URL: http://www.sciencedirect.com/science/article/pii/S1877050917328557.

[26] Graham Williams et al. "A comparative study of RNN for outlier detection in data mining". In: *2002 IEEE International Conference on Data Mining, 2002. Proceedings.* IEEE. 2002, pp. 709–712.

[27] Stephen Odaibo. "Tutorial: Deriving the standard variational autoencoder (vae) loss function". In: *arXiv preprint arXiv:1907.08956* (2019).

[28] Carl Doersch. *Tutorial on Variational Autoencoders*. 2021. arXiv: 1606.05908 [stat.ML].

[29] Benyamin Ghojogh et al. "Factor Analysis, Probabilistic Principal Component Analysis, Variational Inference, and Variational Autoencoder: Tutorial and Survey". In: *arXiv preprint arXiv:2101.00734* (2021).

[30] Andrea Asperti and Matteo Trentin. "Balancing reconstruction error and Kullback-Leibler divergence in Variational Autoencoders". In: *IEEE Access* 8 (2020), pp. 199440–199448.

[31] Shilin He et al. "Experience report: System log analysis for anomaly detection". In: *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE. 2016, pp. 207–218.

[32] F Pedregosa et al. ““Scikit-learn: Machine learning in Python.” Journal of machine learning research 12 (Oct), 2825–2830”. In: *URL: http://jmlr.org/papers/v12/pedregosa11a. html* (2011).

[33] Loic Bontemps, James McDermott, Nhien-An Le-Khac, et al. “Collective anomaly detection based on long short-term memory recurrent neural networks”. In: *International Conference on Future Data and Security Engineering*. Springer. 2016, pp. 141–152.

[34] Jinwon An and Sungzoon Cho. “Variational autoencoder based anomaly detection using reconstruction probability”. In: *Special Lecture on IE* 2.1 (2015), pp. 1–18.

[35] Sinno Jialin Pan and Qiang Yang. “A survey on transfer learning”. In: *IEEE Transactions on knowledge and data engineering* 22.10 (2009), pp. 1345–1359.

[36] Fuzhen Zhuang et al. “Supervised representation learning: Transfer learning with deep autoencoders”. In: *Twenty-Fourth International Joint Conference on Artificial Intelligence*. 2015.

[37] Clinton Gormley and Zachary Tong. *Elasticsearch: The Definitive Guide*. 1st. O’Reilly Media, Inc., 2015. ISBN: 1449358543.

[38] AMIT PURUSHOTTAM Pimpalkar and R Jeberson Retna Raj. “Influence of Pre-Processing Strategies on the Performance of ML Classifiers Exploiting TF-IDF and BOW Features”. In: *ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal* 9.2 (2020), pp. 49–68.

[39] Minmin Chen, Kilian Q Weinberger, Fei Sha, et al. “An alternative text representation to tf-idf and bag-of-words”. In: *arXiv preprint arXiv:1301.6770* (2013).

[40] Lizhong Xiao, Guangzhong Wang, and Yang Zuo. “Research on patent text classification based on Word2Vec and LSTM”. In: *2018 11th International Symposium on Computational Intelligence and Design (ISCID)*. Vol. 1. IEEE. 2018, pp. 71–74.

[41] David Guthrie et al. “A closer look at skip-gram modelling.” In: *LREC*. Vol. 6. Citeseer. 2006, pp. 1222–1225.

[42] Shunjie Han, Cao Qubo, and Han Meng. “Parameter selection in SVM with RBF kernel function”. In: *World Automation Congress 2012*. IEEE. 2012, pp. 1–4.

[43] Katherine Bennett Ensor and Peter W Glynn. “Stochastic optimization via grid search”. In: *Lectures in Applied Mathematics-American Mathematical Society* 33 (1997), pp. 89–100.

[44] Jonathan Baarsch and M Emre Celebi. “Investigation of internal validity measures for K-means clustering”. In: *Proceedings of the international multiconference of engineers and computer scientists*. Vol. 1. sn. 2012, pp. 14–16.

[45] Sarang Narkhede. “Understanding auc-roc curve”. In: *Towards Data Science* 26 (2018), pp. 220–227.

[46] HANA ŘEZANKOVÁ. “Different approaches to the silhouette coefficient calculation in cluster evaluation”. In: *21st International Scientific Conference AMSE Applications of Mathematics and Statistics in Economics 2018*. 2018, pp. 1–10.

[47] David L Davies and Donald W Bouldin. “A cluster separation measure”. In: *IEEE transactions on pattern analysis and machine intelligence* 2 (1979), pp. 224–227.

[48] Hannu Oja, Seija Sirkiä, and Jan Eriksson. “Scatter matrices and independent component analysis”. In: *Austrian Journal of Statistics* 35.2&3 (2006), pp. 175–189.

[49] Yordan P Raykov et al. “What to do when K-means clustering fails: a simple yet principled alternative algorithm”. In: *PloS one* 11.9 (2016), e0162259.

[50] Nathalie Japkowicz and Shaju Stephen. “The class imbalance problem: A systematic study”. In: *Intelligent data analysis* 6.5 (2002), pp. 429–449.

[51] Haowen Xu et al. “Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications”. In: *Proceedings of the 2018 World Wide Web Conference*. 2018, pp. 187–196.