NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

ANDRÉ FILIPE NEVES BASTOS

Bachelor in Computer Science

DEGRADATION EVENTS DETECTION PREDICTIVE MAINTENANCE

MASTER IN ANALYSIS AND ENGINEERING OF BIG DATA

NOVA University Lisbon
November, 2021

NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

DEGRADATION EVENTS DETECTION PREDICTIVE MAINTENANCE

ANDRÉ FILIPE NEVES BASTOS

Bachelor in Computer Science

Adviser: Ludwig Krippahl
Assistant Professor, NOVA University Lisbon

Co-adviser: Luis Guimarais
Analytics project leader, NOS

MASTER IN ANALYSIS AND ENGINEERING OF BIG DATA

NOVA University Lisbon
November, 2021

Degradation Events Detection Predictive Maintenance

Acknowledgements

I would first like to thank my thesis adviser Professor Ludwig Krippahl of the School of Science & Technology at NOVA University. Krippahl always attended to my questions and troubles, and steered me in the right direction.

A very special thanks to my co-adviser Luis Guimarais of the NOS company, who always managed to make the time for helping me with every encountered obstacle, being a great source of support and motivation.

Finally, I must express my profound gratitude to my parents and to my close group of friends for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis.

This accomplishment would not have been possible without any of the above. Thank you.

"We gaze continually at the world and it grows dull in our perceptions. Yet seen from another's vantage point, as if new, it may still take the breath away. " (Alan Moore)

Abstract

NOS is a Portuguese telecommunications company that offers a wide variety of services, like mobile plans, Broadband internet, HDTV, and more. To maintain an impeccable quality of service, NOS extracts performance indicators from its equipments to analyse in a GDPR compliant fashion. Ideally, the data would be perfectly labeled, and all anomalies reported. However, it is up to the clients to report issues, therefore the problem of label uncertainty arises. This is due to lack of communication or different levels of tolerance and patterns coming from the customer-side. Also, several services have different types and frequency of anomalies.

To overcome the anomaly detection problem, we propose an unsupervised approach that uses a set of autoencoders. The autoencoder is trained to capture the characteristics of the data without the need for absolute truth. Instead of fine-tuning a different model for every service, the autoencoder should be able to adapt to each service, overcoming the problem of scalability if the number of use-cases among the services is large.

The goal of this work is to investigate the performance of scalable approaches with minimal human interaction to the ages old problem of outlier and anomaly detection.

Keywords: Predictive Maintenance, Machine Learning, Deep Learning, Neural Networks, Recurrent Networks, Convolutional Networks, Autoencoders, Autoencoder Ensemble, Anomaly detection, Outlier Detection

Resumo

A NOS é uma empresa portuguesa de telecomunicações que oferece uma grande variedade de serviços, como planos móveis, internet de banda larga, HDTV, e muito mais. Para manter uma boa qualidade de serviço, esta extrai indicadores de desempenho dos seus equipamentos para analisar de forma compatível com o GDPR. Idealmente, os dados estariam totalmente rotulados, e todas as anomalias seriam reportadas. No entanto, cabe aos clientes relatar os problemas, pelo que surge o problema da incerteza nos rótulos. Isto deve-se à falta de comunicação ou a diferentes níveis de tolerância e padrões vindos do lado do cliente. Além disso, vários serviços têm diferentes tipos e frequência de anomalias.

Para ultrapassar o problema de detecção de anomalias, propomos uma abordagem não supervisionada que utiliza um conjunto de autoencoders. O autoencoder é treinado para captar as características dos dados sem a necessidade de verdade absoluta. Em vez de afinar um modelo diferente para cada serviço, o autoencoder deverá ser capaz de se adaptar a cada serviço, superando o problema da escalabilidade se o número de casos de utilização entre os serviços for grande.

O objectivo deste trabalho é investigar o desempenho de abordagens escaláveis com um mínimo de interacção humana com os antigos problemas de detecção de anomalias e anomalias.

Palavras-chave: Manutenção Preditiva, Machine Learning, Deep Learning, Redes Neuronais, Redes Recorrentes, Redes Convolucionais, Autoencoders, Conjunto de Autoencoders, Deteção de Anomalias, Deteção de Outliers

Contents

List of Figures

List of Tables

Glossary

Codeword In communication, a codeword is an element of a standardized code or protocol. Each code word is assembled in accordance with the specific rules of the code and assigned a unique meaning. Code words are typically used for reasons of reliability, clarity, brevity, or secrecy. Each have redundancy capable of reconstructing itself if corrupted. 40

dBm dBm or dBmW (decibel-milliwatts) is a unit of level used to indicate that a power level is expressed in decibels (dB) with reference to one milliwatt (mW). 27

dBv dBv – voltage relative to 1 volt, regardless of impedance. This is used to measure microphone sensitivity, and also to specify the consumer line-level of -10 dBV, in order to reduce manufacturing costs relative to equipment using a +4 dBu line-level signal. 27

Hyperparameters Hyperparameters are parameters whose predefined values control the learning process and determine the values of model parameters that a learning algorithm ends up learning. 4, 70

Overfitting Overfitting is a modeling error that occurs when a function is too closely fit to a limited set of data points. Thus, performing poorly when handling a slightly different, but valid, dataset. 18, 19

Panel data Panel data, also known as longitudinal data or cross-sectional time series data in some special cases, is data that is derived from a (usually small) number of observations over time on a (usually large) number of cross-sectional units like individuals, households, firms, or governments. 24, 27

Time series data A time series is a series of data points indexed (or listed or graphed) in time order. Most commonly, a time series is a sequence taken at successive equally spaced points in time. Thus it is a sequence of discrete-time data. Examples of time series are heights of ocean tides, counts of sunspots, and the daily closing value of the Dow Jones Industrial Average. 2

Acronyms

1D-CNN One Dimensional Convolutional Neural Networks 14

ARIMA Autoregressive Integrated Moving Average 6

BAE Boosting-based Autoencoder Ensemble 18, 19, 64, 68, 69

CMTS Cable Modem Termination System 24, 25, 40
CNN Convolutional Neural Networks 8, 10, 13
CNN-AD Convolutional Neural Network - Anomaly Detection 8

DBSCAN Density-based spatial clustering of applications with noise 6
DNN Dense Neural Networks 10, 11
DOCSIS Data Over Cable Service Interface Specification 24

GDPR General Data Protection Regulation 3, 36

HFC Hybrid Fibre Coaxial 24

kNN k-Nearest Neighbors 6
kNN-TSAD k-Nearest Neighbors - Time Sensitive Anomaly Detection 6
KPI Key Performance Indicator 24, 27, 69

LOF Local Outlier Factor 6
LSTM Long Short-Term Memory 8, 10, 11, 12, 14, 17, 18, 39, 52, 57, 64, 67, 69, 70
LSTM-AD Long Short-Term Memory - Anomaly Detection 8

MLP Multi-layer Perceptron 10

MSE Mean Squared Error 8

OCSVM One Class Support Vector Machine 7, 12, 20, 39, 48, 67, 69
OHE One Hot Encoding 36

PCA Principal Component Analysis 6

RNN Recurrent Neural Networks 10, 11, 14, 16, 17

STB Set-top Box 25, 40
ST-BDBCAN Spatio-Temporal Behavioral Density-based Clustering of Applications with Noise 6
SVM Support Vector Machine 7, 48

TCN Temporal Convolutional Networks 8, 14, 15, 16, 17, 18, 57, 67, 69
TCN-AE Temporal Convolutional Networks Autoencoders 17

1

Introduction

In this chapter an historical contextualization is made, followed by the motivation and problem at hands.

This work was developed in the course of "Preparation of the Dissertation"of the Master in Analysis and Engineering of Big Data.

1.1 Predictive Maintenance

There is a big inconvenient truth about our reality, material objects deteriorate, especially production machines, equipment, devices, and many others. Apparatus can have a considerable degree of complexity and may require many simpler tools. Each of said individual tools may have different durabilities. By actively monitoring and repairing the issues with individual components, the lifetime of the whole apparatus can be increased.

This is commonly known as **maintenance**, and it dates way back into the history of Humankind, to the first Egyptian records that depict the stoppage of wood supply for sacred boat's repair. It has gained a lot of importance in the last centuries due to the Industrial Revolutions and the rise of machine work (Poor, 2019). But it was only through the third Industrial Revolution that statistical methods began to be considered as part of the maintenance process, and only in the fourth that these became widely used (Poor, 2019). Thus, outlier detection was born.

The first official definition of an **outlier** was published by Grubbs in 1969 (Grubbs, 1969, V. and T., 1994): "An outlying observation, or outlier, is one that appears to deviate markedly from other members of the sample in which it occurs". Since the 60's, this definition kept changing through the publication of author's works in the field. These are known by two important traits:

- They are different from the norm with respect to their features,
- They are rare in a dataset compared to normal instances.

Outlier Detection started as manual statistical methods, but soon grew alongside the need for an automated way of discover bank frauds, structural defects, medical problems, and hardware problems (as is the case of this thesis) (Hodge and Austin, 2004).

One question that might arise is: What is the difference between anomaly detection and outlier detection? First it is important to differentiate outliers from anomalies. Outliers arise due to mechanical faults, changes in system behaviour, fraudulent behaviour, human error, instrument error or simply through natural deviations in populations (Hodge and Austin, 2004). Anomalies only represent the data generated by a system with a faulty component. This means that outlier detection is a much broader term aimed at the discovery of the wider concept that is the outlier, whereas anomaly detection only tries to find mechanism failures among the healthy data points. An official and more suited definition for anomalies is proposed in Hawkins (1980): "An anomaly is an observation that deviates so much from other observations as to arouse suspicion that it was generated by a different mechanism.". In this field's earlier years, people only talked about outlier detection as the concept was still being defined. Therefore, it is common to refer to outlier methods, even though the real scope of this thesis is anomaly detection. Back then, the main reason for the detection was to remove the outliers from the training data, since pattern recognition algorithms were quite sensitive to these points. This procedure is also called data cleansing. But as stated earlier, around the beginning of the 21st century, the interest in anomalies grew due to suspicious data records (Goldstein and Uchida, 2016).

It is also worth to point out the diversity at which anomalies might appear, as is presented in the following list of items (notions taken from Lin et al., 2020). Because the dataset is considered Time series data (as will be further explained in the next chapters), is very unlikely that its anomalies are point ones.

- Point anomalies: The easiest to detect because data points can be treated independently during detection and no temporal relationships need to be considered.
- Contextual anomalies: These depend on the value of surrounding data points and thus local information is required for their detection.
- Collective anomalies: These last ones occur when a series of data points together exhibits abnormal behaviours.

1.2 Motivation

NOS is one of the many companies that invest in predictive maintenance in order to optimize their service, organization and profit. The services that NOS provides involve the use of equipment such as routers, TV boxes and phone sites. These devices generate data that concern their performance and usage, which can be used for predicting the occurrence of anomalies. There is, not only between different devices but also within

the same type of device, a complex hierarchy that categorizes each unit and its available features. Also, by having several suppliers, it is expected that each model has different possible failures and average lifetimes.

While fault fixes made through software updates do not imply the allocation of people, common reported failures require the physical displacement of resources, both human and material, as well as a nuisance caused to the client. Due to this fact, it is of great importance to understand if this displacement will really help people with the problems encountered. It is therefore important to understand what really constitutes a failure that potentiates a decline in the quality of the service provided, whether this is related to the network or equipment. This analysis translate into a more accurate and inexpensive maintenance process, therefore making NOS's services more robust.

1.3 Problem Statement

There is the need for a generalized way to predict an anomaly event from each of the different supplier's devices, as stated earlier. As well as an appropriate model for the time bounded characteristic of the data (time-series data).

Usually the data has a column that represents the ground truth or commonly known as the targets, in this case if the row is an anomaly or healthy behavior. Acquiring these labels is very expensive, therefore leading to a lack of targets in the dataset. For some approaches this is a deal breaker, but there are specialized models that can handle this type of problem.

As stated earlier, there is also the difference between outliers and anomalies, which might represent a big challenge. If a data point seems to be an anomaly but was, in fact, legit and generated by a sudden predictable change in behavior (e.g. an election, super bowl, big event), then it was in fact an outlier and should not be considered by an anomaly detection system.

Unfortunately, due to new General Data Protection Regulation (GDPR) rules, the company cannot keep this kind of detailed information for very long. Therefore, the datasets will assume a smaller size than previously thought, although it will vary according to the data. Consequently, it is not possible to make an analysis that span through the annual pattern.

On the other hand, there is a much smaller time scale which is way more important according to the technicians of NOS, the difference between the week and weekend patterns. This is due to its regular occurrence, while holidays can be occasionally treated separately. Therefore, all models will be tested with both week and weekend scenarios.

1.4 Objectives

The main goal of this study is to assess if the deep learning models can distinguish anomalies among healthy data, using a 24 hour information window regarding the target device (further explained in section 3.5). In order to accomplish this, the modeling, evaluation and comparison between a few widely used techniques for anomaly detection will be carried out (further detailed in chapter 5). One machine learning model will be applied to gather a baseline performance to which compare the following deep learning ones. Each model has its Hyperparameters that need to be predefined in the moment of the train. Thus, there will be the model optimization through the experimentation of multiple combinations of these Hyperparameters. In the end, this analysis will generate a recommendation for which model to use in a problem similar to the current one.

There is also the contribution of a pipeline prototype that can preprocess the inputs, model multiple hypothesis, create and apply adequate test cases and extract appropriate metrics for each one. This prototype can be used for further investigation in the subject, either using new methods or experimenting variations of those implemented.

1.5 Document's Structure

This report is organized in the following manner: In chapter 1 an introduction is made containing the context, motivation and problem statement. Followed by chapter 2 which describes appropriate and used models in the literature and those chosen to be implemented. A description of the dataset will be made in chapter 3, as well as its exploratory analysis and preprocessing. In chapter 4, the testing process is shown, while its obtained results can be observed in chapter 5. Finally, the conclusions are depicted in chapter 6.

2

State of The Art

This chapter presents several approaches adopted by other authors and those chosen for this thesis.

2.1 A look at production's example

Since most of the company's traditional methods for outlier detection rely on simple, yet effective, rules that may vary a lot between context, using an almost trial and error among what may work on that specific case. Such is the case for an outlier detection case, concerning a backup log dataset, in which was made a simple plot (shown in figure 2.1) of the two most important variables, the backup duration and size. One may observe that all entries lie inside the defined non-anomalous area, while there is one outlier outside. In that case, it was enough to make a reasonably effective prediction, but it is not guaranteed that the same will work for other problems in the same domain, or even in different ones. There is then, the need for a more generalised model that will understand the context (every aspect, feature of the dataset) in a certain kind of problem.

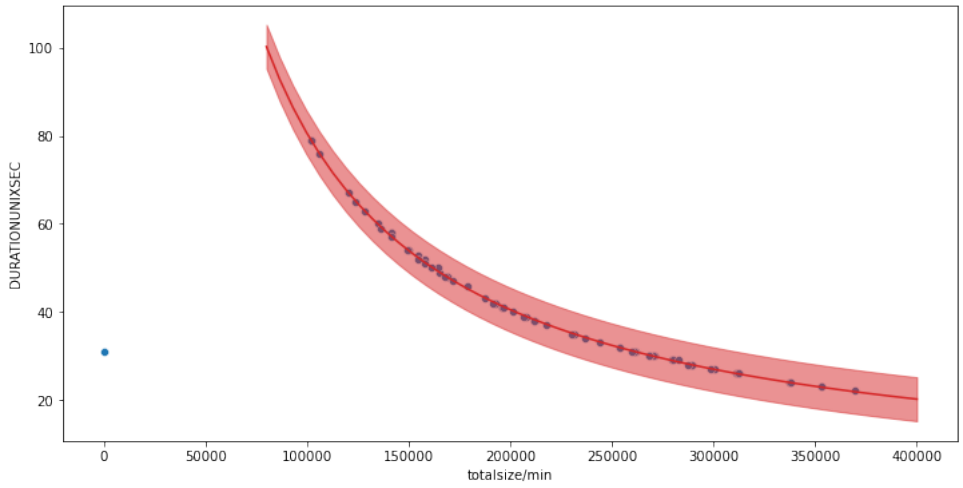


Figure 2.1: Backup's total size variable versus duration variable plot.

2.2 An Overview

Throughout time specialists naturally categorized outlier detection procedures that appeared, as these were many and diversified, thus handling the need for an higher organization and taxonomy. These categories are: statistical, distance, density, clustering, graph, ensemble and learning methods (H. Wang, Bah, and Hahhad, 2019).

Such techniques vary from the simple 3σ rule (V. and T., 1994 and Hodge and Austin, 2004) and Grubb's Z value method (Grubbs, 1969; Pang, Shen, Cao, and Hengel, 2020), to more recent and complex methods like the autoencoder neural network (Chalapathy and Chawla, 2019, Zhou and Paffenroth, 2017). Principal Component Analysis (PCA) also was very used in these kind of problems (Zhikun, Zhiwen, Weihua, and Bin, 2013), but soon the lack of non-linearity have put this technique behind many other emergent methods. Another prominent and ever present statistical method is the Autoregressive Integrated Moving Average (ARIMA) (Yaacob, Tan, Chien, and Tan, 2010; Yu, Jibin, and Jiang, 2016; Hodge and Austin, 2004) and its Seasonal variant, which has been used for a long time.

The anomaly detection model catalogue is already quite long but not every technique will work in our case, due to the problem's constraints: the dataset constitutes of time series and multidimensional data, and the lack of labels. These narrows our search to a few options.

Regarding distance approaches, an interesting technique is the k-Nearest Neighbors (kNN) variant called k-Nearest Neighbors - Time Sensitive Anomaly Detection (kNN-TSAD) which can achieve better results than its predecessor when facing sequential data without labels, as proposed in Wu et al. (2019).

Clustering based approaches are other notable algorithms for unsupervised outlier detection. In Nascimento, de L. Tavares, and Souza (2015), the authors tried the C-AMDATS algorith with the malahanobis distance to achive good results in a real case of carbon monoxide measurements. Another interesting method is an hybrid one in Chen, Li, Proietti, Zhu, and Yoo (2019), where a clustering unsupervised phase (Density-based spatial clustering of applications with noise (DBSCAN)) is applied and then the results, which are considered labels, are sent to a supervised approach (dense neural network). Its extension (Spatio-Temporal Behavioral Density-based Clustering of Applications with Noise (ST-BDBCAN)) also uses spatial and temporal attributes simultaneously to define the context (Duggimpudi, Abbady, Chen, and Raghavan, 2019). Another approach, and this time a density one, is the Local Outlier Factor (LOF) which determines the local outlier factor (LOF) of a row. It is local because the anomaly score depends on how isolated the object is with respect to the surrounding neighborhood (Breunig, Kriegel, Ng, and Sander, 2000).

In multivariate time series data, strong temporal dependency exists between time steps. Due to this reason, distance, clustering and density based methods, may not perform well since they cannot capture temporal dependencies properly across different

time steps.

An interesting ensemble approach is the Isolation Forest, as was applied in Yao, Ma, Chen, Zhao, and Bai (2019). Can be used in semi-supervised, unsupervised mode and multiple variable data. It has been used for its effectiveness, as it utilizes isolation via recursive partitioning by randomly selecting a feature and then randomly selecting a split value for the selected feature. The output is the anomaly score for each data point. Despite the effectiveness of Isolation Forest technique, it isolates data points based on one random feature at a time which may not represent spatial and temporal dependencies between data points properly in the spatio-temporal dataset.

2.3 One Class SVM

In Rasheed and Tang (2020), the boundary method One Class Support Vector Machine (OCSVM) was used to detect anomalous brain activity. It is based on the very popular Support Vector Machine (SVM) classifier which can be used in a variety of applications: handwritten digit recognition, object recognition, speaker identification, text categorization and also anomaly detection. This Structure Risk Minimization model performs at least as good as other methods in terms of the generalization error, but also feature a convex optimization objective ensuring that the global optimum will be reached. Moreover, in comparison to other kernel-based approaches, it is really efficient due to its solution being sparse (Amer, Goldstein, and Abdennadher, 2013). Although, in our case it is confronted with the obstacle of class imbalance since anomalies are rare.

OCSVM overcomes this problem. It needs some hyperparameter tuning but showed good results while requiring less volume of data. This is because in contrast to the SVM, the variant in question will attempt to learn a decision boundary that accomplishes the higher separation between the points and the origin. These project the data into a higher dimensional space through a transformation function defined by the kernel. A hyperplane that separates the majority of data from the origin is learnt, and those points that lie outside of this boundary will be considered anomalies (Amer et al., 2013). In Bounsiar and Madden (2014), a study that undertakes many transformation functions on several standard datasets, found that the best performances are always obtained with RBF kernels.

Unlike most machine learning techniques, OCSVM overcomes the class imbalance problem but, like others, suffers from the lack of temporal dependencies capture. Thus, it will be used as a baseline for this thesis, in order to compare and evaluate further more advanced and appropriate techniques. Another reason to consider deep learning models instead is because these shallower ones, mentioned hitherto, might only detect point anomalies. While methods described in the next subsections deal with windowed inputs, this one treats inputs independently. This might reveal some challenge when it comes to evaluate the models.

2.4 Deep Learning Approaches

Regarding deep learning techniques, there are two possible approaches for our problem (of Sydney, Centre, Institute, and Hbku, 2019):

- A neural network model is used for data forecasting and then the prediction is compared to the actual value,
- A specialized model in reconstructing the input data that can generate a reconstruction error (Autoencoders).

Concerning the first approach, we can transform this prediction problem into a detection one by computing the error between the model's output and the actually new input, by using the Mean Squared Error (MSE) (or other loss functions). Then, these errors are compared with a defined threshold and if they surpass the established value it is considered an anomaly. The threshold value is arbitrary, some people just plot the errors distribution and define the value based on the given tails, although there are more sophisticated methods (e.g. fit the errors into a gaussian distribution He and Zhao, 2019). This is a common method named Long Short-Term Memory - Anomaly Detection (LSTM-AD) when using Long Short-Term Memory (LSTM), and Convolutional Neural Network - Anomaly Detection (CNN-AD) in the case of the Convolutional Neural Networks (CNN). Many paper authors used this method alongside the LSTM as, for example, in Bontemps, Cao, McDermott, and Le-Khac (2016) for network traffic dataset attack detection, Thill, Däubener, Konen, and Bäck (2019) to detect anomalies in electrocardiogram readings, and You, Wang, and Sun (2019) which used a bidirectional variant for another network traffic dataset. There is also an increase in the usage of Temporal Convolutional Networks (TCN) for time series data, as used in Cheng, Xu, Zhong, and Liu (2019), to detect IoT devices anomalies , He and Zhao (2019) for univariate real world datasets, a multivariate case (three different datasets) in Y. Wang, Liu, Hu, and Zhang (2019), and others (Baharani, Furgurson, Parkhideh, and Tabkhi, 2020; Lara-Benítez, Carranza-García, Luna-Romera, and Riquelme, 2020). However, this prediction-based approach is not able to predict time series affected by external changes (Pereira, 2018).

The second approach makes use of the Autoencoder network infrastructure which its objective is to compress the given data into a representative smaller latent space, and then reconstruct into the original input. This architecture has some subcategories like Denoising AE, Contractive AE and Variational AE, but the most appropriate one is the Undercomplete Autoencoder. By having smaller dimensions for hidden layers compared to the input/output one, it can obtain the most important features in the data. It minimizes the loss function by penalizing the reconstruction of the data for being different from the original input. It has two modules: the encoder (compresses data) and the decoder (reconstructs data). Between the two modules resides low dimension unit layers in order to create a non-linear representation of the data, as can be observed in

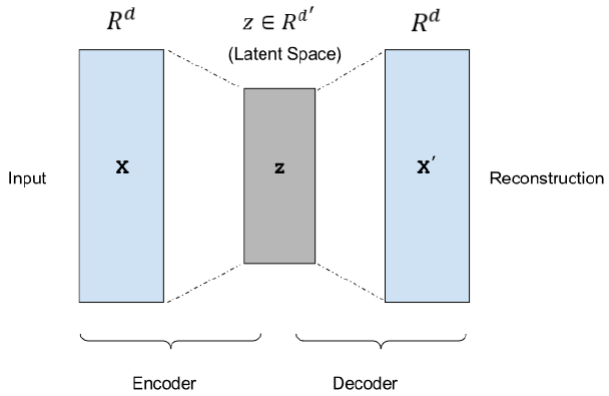


Figure 2.2: Representative illustration of an Autoencoder. Image taken from Karadayi, Aydin, and Öğrencí (2020).

figure 2.2. An autoencoder is trained in an unsupervised fashion to learn how to build its original input with a minimum reconstruction error (Karadayi, Aydin, and Öğrencí, 2020). It takes an input vector $x \in R^d$, and converts it into a latent representation $z \in R^{d'}$ through the mapping:

$$z = f_\theta(x) = Wx + b \tag{2.1}$$

Where the function f_θ represents the encoding step and is parameterized by $\theta = \{W, b\}$. W is the weight matrix $(d' \times d)$ and b is a bias vector. The latent representation can then be reconstructed into $x' \in R^d$ in the input space:

$$x' = g_{\theta'}(z) = W'z + b' \tag{2.2}$$

Where $g_{\theta'}$ represents decoding step and is parameterized by $\theta' = \{W', b'\}$. The training procedure consists of finding a set of parameters $\{W, b, W', b'\}$ that approximate the vector x' close as possible to the original x. The parameters are optimized by minimizing a loss function that measures the quality of the reconstructions. A common autoencoder loss function is the sum-of-squared differences between the input and the output. Therefore, an autoencoder tries to minimize the following loss function during the training (where φ is the training dataset):

$$L(x, x') = \sum_{x \in \varphi} \sum_{i=1}^{d} \left\| x^i - x'^i \right\|^2 \tag{2.3}$$

Such a structure can be visualized in figure 2.2.

It is more of a structure instead of a specific neural network since you could apply this concept to other networks by having them as layers in both the encoder and decoder (Autoencoders started with feed forward dense networks). These networks, when fed normal data, can learn to reconstruct normal instances of the data, thus performing poorly at reconstructing unusual data points. Often, other simpler unsupervised methods are applied in order to remove obvious outliers from the data (Gupta, Gao, Aggarwal, and Han,

2014). Therefore, we can use the **reconstruction error** as an indicator of how anomalous that given input is. The last step is to define a **threshold** that decides whether the data is anomalous given the errors. In the predictor's approach discussion we talked about a manual way of defining the distance threshold and the way through gaussian fitting. Although the errors might not fit such a curve, hence another method is considered. Another approach is to compute the mean and summing x times the standard deviation, which will be used. Several iterations using different x will be used in order to reach a more optimized borderline with respects to the metrics defined in section 2.9.

Another interesting aspect of these networks is that inputs compressed into the latent space can be grouped, using a clustering technique, into many different device functioning profiles. In Sainburg, Thielk, and Gentner (2020) the authors used the autoencoder configuration in order to retrieve many animal vocal sequence profiles.

The problem with reconstruction methods such as this last one is that, because errors are attributed to a window frame scale and anomalies occur in a small section of one, the reconstructed error for both healthy and anomaly cases might not be as gaped as those retrieved from the earlier prediction approach. However, this is a preferred issue to that of the first approach, which being sensitive to noise leads to even worse robustness (Tang et al., 2020). Therefore, the autoencoder approach is the chosen one, along with threshold finding method.

Both of the earlier approaches need a chosen base network. Dense Neural Networks (DNN) have many applications, but one domain on which these fail is in time series data problems because they don't consider its sequential aspect. Recurrent Neural Networks (RNN) are specialized networks that have connections between nodes to form a directed graph along a temporal sequence. LSTM are the most successful type of recurrent network and, therefore, the ones we will use (further explanation on these networks will be in the next section) (Radford, Apolonio, Trias, and Simpson, 2018; Nam, Jeong, and Park, 2020). Recently, CNN (which is the usual choice for image problems) also have been tried for time series when using a 1 dimensional kernel, yielding good results (Russo, Disch, Blumensaat, and Villez, 2020).

Another motivation for using RNN and CNN is their triumph over contextual and collective kinds of anomalies as described in Lin et al. (2020). Some papers, such as of Sydney et al. (2019), also argue that deep anomaly detection techniques work on all types. Multi-layer Perceptron (MLP) would have been enough if the dataset would not have any sequential relevancy, that is, if only point anomalies were present.

2.5 LSTM Autoencoders approach

RNN started as simple dense networks with additional connections to the next neuron in the layer in order to represent the flow of time for the past into the future. This simple process tend to run into two problems, known as exploding and vanishing gradients. Exploding gradients depict the accumulation of large error gradients, usually due to the model's high complexity. Therefore, it results in very large updates to the weights and further instability while training or, sometimes, inability to learn. Vanishing gradients account for a similar case, when the error gradient propagated keeps getting too small for any change to take place in the first layers of neurons (error propagation starts at the final layers). One solution to these issues is to reduce the number of hidden layers within the neural network, eliminating some of the complexity in the RNN model. There are more recent and complex types of recurrent networks that tackle this problem such as the LSTM.

LSTM differ from other types of RNN by their "smarter"units, which have three gates that decide how to react to an input. Such gates are: input gate, output gate and forget gate. Based on the information received, it decides to block it or to pass it further on. The information is also filtered with the set of weights associated with the cells when it is transferred through these cells as would happen in a regular DNN. At each time step, a unit updates a hidden state vector h_t and a cell state vector c_t responsible for controlling state updates and outputs using current input, previous cell state and hidden state values. By applying only linear transformations over the cell state, thus preventing the derivatives in the backpropagation process from disappearing. These gate functions are described with the following equations (Karadayi et al., 2020):

$$i_t = \sigma\left(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i\right) \tag{2.4}$$

$$f_t = \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f\right) \tag{2.5}$$

$$o_t = \sigma\left(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o\right) \tag{2.6}$$

$$c_t = f_t \circ c_{t-1} + i_t o \tanh\left(W_{xc}x_t + W_{hc}h_{t-1} + b_c\right) \tag{2.7}$$

$$h_t = o_t \circ \tan h\left(c_t\right) \tag{2.8}$$

Where x_t and h_t are the input and hidden state vectors, the notations i_t, f_t, o_t, c_t are the activation vectors of the input gate, forget gate, output gate, and memory cell, respectively. W_{ab} is the weight matrix between a and b, and b_a is the bias term of a, σ is the sigmoid function and $\circ$ depict the product-wise product.

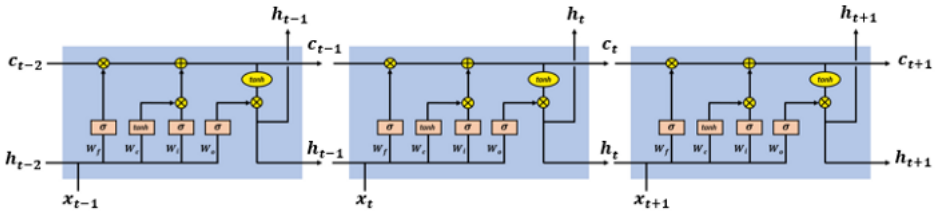


Figure 2.3: Representative illustration of an LSTM network. Image taken from "How LSTM networks solve the problem of vanishing gradients" (n.d.).

As mentioned before, Long Short-Term Memory has the power to incorporate a behaviour into a network by training it with such corresponding data. The model learns the data variations, making predictions focused on two aspects: the value of a sample and its position at specific time. This results in a system that may output different results for an input if the position is not the same as it is able to use Back Propagation through time to consider the past of a sample, thus understanding the context better (Bontemps et al., 2016). Therefore it can predict a coherent output in agreement with the context of the sample.

Using the LSTM technique with the Autoencoder architecture, anomalies can be detected using the reconstruction score, as said before. This is exactly what the authors of Malhotra et al. (2016) did with three different datasets by assuming a gaussian distribution from the output errors. In Saumya and Singh (2020), this technique was used to detect which consumer's goods reviews were spam, and in Nguyen, Tran, Thomassey, and Hamad (2020) it was used in a supply chain management case. These authors used a different method for determining the threshold, both paper's approach involves the clustering of the outputted values: a simple silhouette coefficient clustering step for the former, while the latter used an OCSVM. In all these papers, there are hyperparameters to be optimized, such as: learning rate, number of cells, dropout rate (when used). The quantity of units (cells) is an important choice because it will directly impact the width of the sliding window used for the inputs, as each input represents an instance in time and each unit handles one input. Therefore, if our objective is to capture the data behaviour for a long time it is best to also choose a bigger window. This was tried in You et al. (2019), a LSTM prediction approach paper, with a smaller problem time scope. Therefore, if our objective is to capture the data behaviour for a long time, it is best to also choose a bigger window as was experimented in You et al. (2019) (further explanation on this is present in chapter 3).

The LSTM is chosen as one autoencoder network option due to its appropriate nature regarding temporal dependencies and its surpassed problems present in earlier recurrent networks. Still, one needs to recognize its hard and lengthy training process, requiring a lot of memory-bandwidth-bound computation. Due to this unpredictable fact about these networks, another option will also be considered.

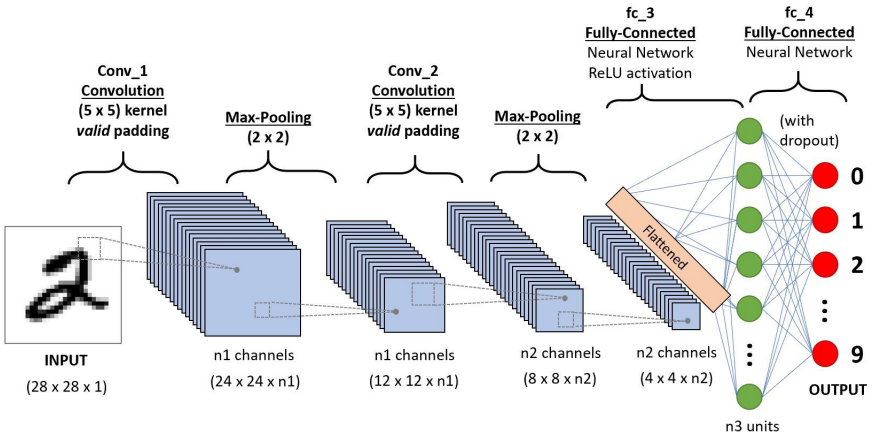


Figure 2.4: Representative illustration of an CNN network. Image taken from "A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way" (2018).

2.6 TCN Autoencoders approach

CNN, which are trained end to end have shown very good results regarding image classification and segmentation problems. The kernel, which is a set of values grouped in a square formation, slides through each input image, multiplying the contained values with the pixel ones and summing them to generate many feature maps. These feature maps identify certain characteristics and their size is reduced along the network. Their layers are arranged to represent spatial information within data in a grid structure. Their spatial relationships are inherited from one layer to the next creating representative features based on a small local spatial region in the previous layer (Karadayi et al., 2020). There are three different types of layers which usually constitute a convolutional neural network: convolution, pooling, and activation function layers. Each layer represents a three-dimensional grid structure of size $h \times w \times d$, where height (h) and width (w) are spatial dimensions, and depth (d) refers to the number of channels or the number of feature maps in the hidden layers. In addition to hidden layers, a final set of fully connected layers is often used in an application specific way to generate the desired output. Such a structure is represented in 2.4. Another extremely used operation is the pooling layer, which outputs one value without a kernel, requiring the choice of a function. The most used operations for pooling are the average pooling, which calculates the average value for each value, and the max pooling, thus retaining the maximum value.

These networks have also shown to work fairly well on time-series data for anomaly detection as was tried in (Wen and Keyes, 2019) and (Russo et al., 2020) using the one dimensional convolutional network architecture over univariate datasets (one dimensional operations ilustrated in 2.5). This can be extended to a multivariate scenario by turning the data into a three dimensional dataset. This is achieved by appending all the variables, thus getting a matrix of size T (which is the time window size) for the first dimension, and

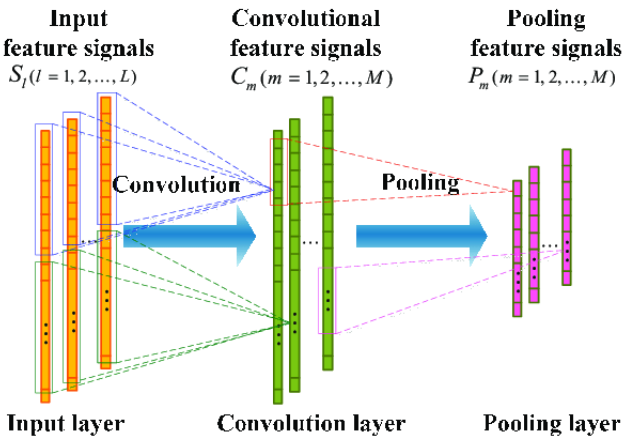


Figure 2.5: Representative illustration of one dimensional convolution operations. Image taken from Wen and Keyes (2019).

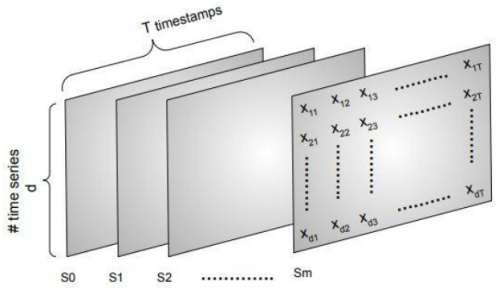


Figure 2.6: Representative illustration of spatio-temporal pre-processing. Image taken from Karadayi, Aydin, and Öğrencí (2020).

d for the second one (depicting the amount of features in the date). The last dimension is achieved by cutting the matrix into multiple ones, amounting to a quantity of S. This spatio-temporal pre-processing is described in (Karadayi et al., 2020) and ilustrated in 2.6.

The One Dimensional Convolutional Neural Networks (1D-CNN) based TCN architecture was presented in Bai, Kolter, and Koltun (2018), their authors proved many advantages over its predecessor and, especially, recurrent networks. In this paper both LSTM and TCN are equally tested and the latter turned out to perform better in less training time. Later, Berglind (2019) came out and confirmed, in many scenarios, the results and conclusions taken from the earlier mentioned paper. The paper Wan, Mei, Wang, Liu, and Yang (2019) is an example of a successful application of these networks for time-series prediction, and both Meng, Jiang, Wei, and Wei (2020) and Thill, Konen, and Bäck (2020) are anomaly detection cases.

The convolutions in the TCN architecture are causal, meaning that only past information is able to explain the value to be predicted. Its architecture, just like RNN, is able to map an input sequence of any length, therefore it needs a chosen time window size.

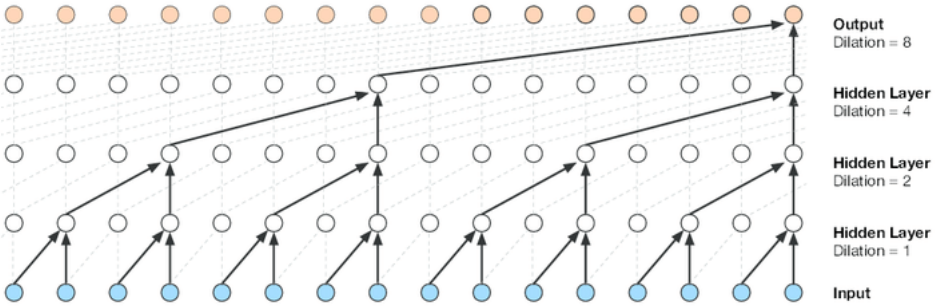


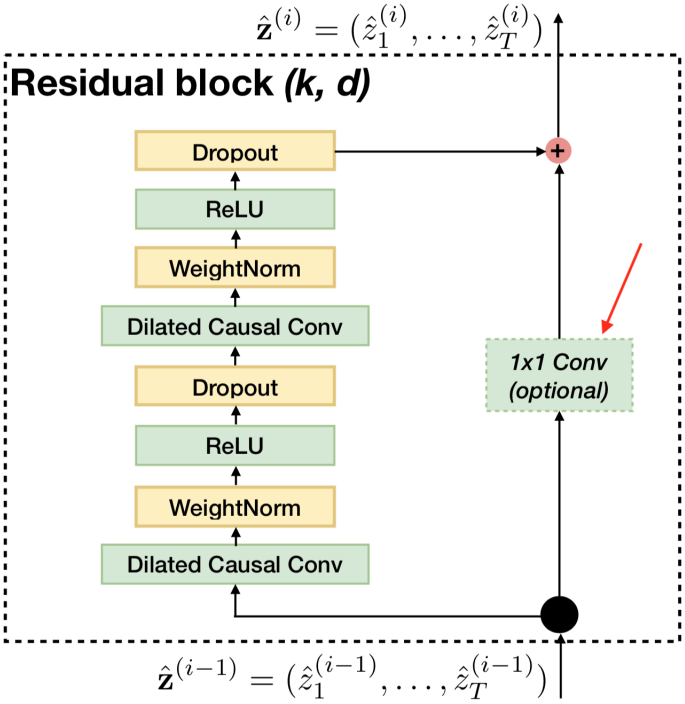
Figure 2.7: Representative illustration of stacked causal convolution layers with dilation. Image taken from Oord et al. (2016).

The receptive field of a sequence model denotes the number of sequential samples that are used to compute an output value. To increase the receptive field of a TCN model, convolution layers are stacked. Although stacking too many layers leads to an harder to train network, it can be overcome with the use of dilated convolutions. A dilated convolution skips inputs at regular intervals, by stacking these layers with an exponentially increasing dilation rate, it is possible to get an exponentially increasing receptive field while still hitting every input. In image 2.7 is depicted the case of three stacked layers of causal convolution with progressively higher dilation. For a 1D sequence, input $x \in R^n$ and filter $f : \{0,\dots,k-1\} \rightarrow R$, the dilated convolution F with respect to the sequence element s can be defined as:

$$F(s) = (x \times f_d)(s) = \sum_{i}^{k-1} f(i) \cdot x_{s-d \cdot i} \tag{2.9}$$

As shown in Xu and Duraisamy (2019), different kernels operate over each data feature in the TCN, represented by channels. While causal convolutions are the smaller units in the framework, TCN are usually made of residual blocks, which are made of convolution layers. In Meng et al. (2020), the authors applied residual blocks that consisted of two dilated causal convolution layers, two weight normalization layers and two rectified linear units (ReLU) nonlinear functions. In order to achieve regularization, two dropout layers are added after the activation functions. To account for differences between the input and output widths, the TCN uses an additional 1×1 convolution for element merging to ensure that the two tensors are the same shape. This block architecture can be observed in image 2.8.

In the same paper, Meng et al. (2020), instead of using the pre processed data directly into the TCN model, the authors retrieved enriched time series features in a two step process. For each variable they partitioned the data into sliding windows of a pre determined size, then calculated the NORM of the window and the DONM, which is the difference of the norm in the current timestep and the NORM of the timestep before. They also



(b)

Figure 2.8: Representative illustration of a residual block. Image taken from Bai, Kolter, and Koltun (2018).

calculated statistical features for each window of NORM and DONM, such as: mean, min, max, quantiles, standard deviation and amplitude. These new windowed values resulted in a newly enriched dataset.

Advantages of TCN over other networks (Bai et al., 2018):

- Parallelism: Unlike RNN layers, convolutions can be computed in parallel, since the same filter is used in each layer.
- Flexible receptive field size: TCN have better control of the model's memory and are more easily adapted to different domains because the receptive field size can be changed in multiple ways. For instance, stacking more dilated (causal) convolutional layers, using larger dilation factors, or increasing the filter size.
- Stable gradients: Unlike TCN, the problem of exploding/vanishing gradients is a major issue when dealing with RNN.
- Low memory requirement for training: Due to the shared filters across layers and backpropagation path depending only on network depth, TCN require much less memory during training than RNN.

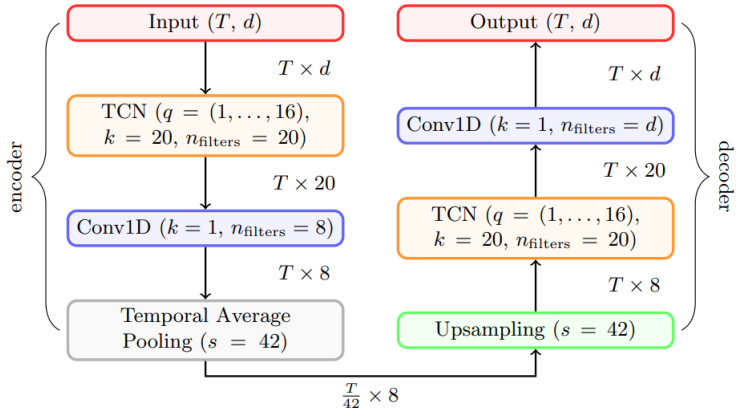


Figure 2.9: Representative illustration of the TCN-AE architecture. Image taken from Thill, Konen, and Bäck (2020). T represents the sequence sequence length, d depicts the data dimensionality/channels, q the dilation rates, k the kernel size and s the downsampling/upsampling factor.

- Variable length inputs: Just like RNN, TCN can also model inputs with arbitrary lengths by sliding the 1D convolutional kernels.

TCN has two disadvantages:

- Data storage during evaluation: While evaluation in RNN only requires its the hidden states to generate a prediction, whereas TCN need to take in the raw sequence up to the effective history length, thus possibly requiring more memory.

- Potential parameter change for a transfer of domain: When transferring a model from a domain where only little memory is needed to a domain where much longer memory is required, TCN may perform poorly for not having a sufficiently large receptive field.

A Temporal Convolutional Networks Autoencoders (TCN-AE) architecture can be observed in image 2.9. It features a TCN network and, in the original papers case, a 1 dimensional convolution for both encoder and decoder. For a higher compression rate the final layer of the encoder has a pooling one, in this case a temporal average is represented but max pooling ones are usually used as well. The first layer of the decoder is an upsampling one, which is the reverse operation of the pooling.

The papers in the literature suggest that the adoption of TCN for time series problems over recurrent ones do not sacrifice much performance, and not only relieves a lot of overload, but also provides a more stable training process. Thus, it is chosen as the second deep learning network candidate for the autoencoder method. Both LSTM and TCN will be experimented because the TCN may not perform well enough when using a large input window frame.

For both LSTM and TCN, the L1 regularizer will be used in order to tackle Overfitting. The regularizer will shift our weights w values to a less than perfect model solution. It is represented by the last element of the cost function:

$$\text{Loss} = \text{Error}(y, \hat{y}) + \lambda \sum_{i=1}^{N} |w_i| \tag{2.10}$$

2.7 Boosting Autoencoder Ensemble

In order to handle Overfitting, some authors tried to train many autoencoders and consider every autoencoder output for time series data prediction and anomaly detection. In Kieu, Yang, Guo, and Jensen (2019), two models are proposed: a sequential one where autoecoders are trained individually from one another and a simultaneous model where all encoders contribute for the same shared latent space. When it comes to scoring, both approaches take the median of all outputs.

The authors of Lorbeer and Botler (2020) also experimented an independent autoencoder framework. This time the autoencoders were let to overfit (by not using any kind of regularization), considering only the highest error value when scoring.

These techniques make use of the subspaces of the problem covered by different autoencoders. The next ensemble will handle this, but will also tackle the problem of having possible anomalies among the data points.

In Sarvari, Domeniconi, Prenkaj, and Stilo (2019) the Boosting-based Autoencoder Ensemble (BAE) method was proposed. It uses boosting to build an adaptive cascade of autoencoders to achieve improved and robust results. Each autoencoder is trained sequentially and its inputs are taken from a resampled dataset (derived from the original one) created based on a probability function that considers the output of the autoencoder trained before. This means that outliers found in the first autoencoder have less probability to appear in the second's dataset, letting the further autoencoders train in less faulty data and turning the training into a completely unsupervised problem. This is illustrated in figure 2.10 for a generic classifier, which in our case would be an autoencoder.

Since an higher reconstruction error also mean an higher anomaly probability, the probability function for the dataset resampling is inversely proportional to the reconstruction error itself:

$$\mathbb{P}_x^{(i+1)} = \frac{1/e_x^{(i)}}{\sum 1/e_x^{(i)}} \tag{2.11}$$

Where $e_x^{(i)}$ represents the reconstruction error generated by the autoencoder i, for the input x:

$$e_x^{(i)} = \left(\left\| x - AE^{(i)}(x) \right\|_2 \right)^2 \tag{2.12}$$

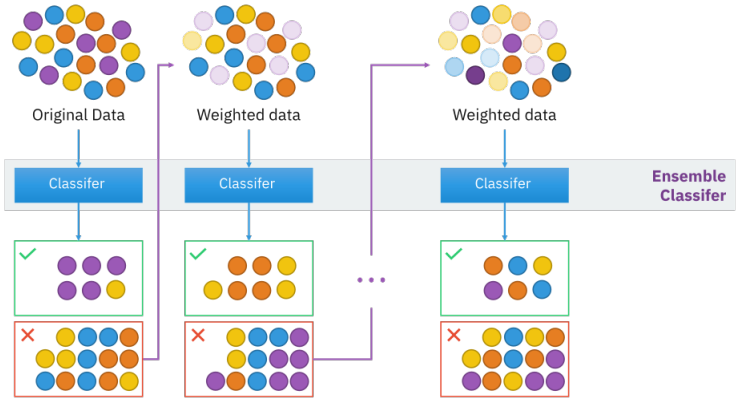


Figure 2.10: Representative illustration of a simple boosting technique. By Sirakorn - Own work, CC BY-SA 4.0

Regarding the overall scoring, for each autoencoder i is attributed a weight based, again, on the inverse of the reconstruction error. This is because the autoencoders are inlier specialized and better suited for detecting outliers. Following this observation, the component that has smaller reconstruction errors has more impact on the final anomaly score. These weights are defined in the next equation:

$$w^{(i)} = \frac{1/\sum_{x \in x^{(i)}} e_x^{(i)}}{\sum_{i=1}^{m-1} \left(1/\sum_{x \in x^{(i)}} e_x^{(i)}\right)} \tag{2.13}$$

Then, the final outlier score assigned to each data input x is:

$$\bar{e}_x = \sum_{i=1}^{m-1} w^{(i)} e_x^{(i)} \tag{2.14}$$

Where m represents the total number of autoencoders. Notice that the sums consider only $m-1$ components because the first autoencoder, which is trained in the original dataset, do not contribute to the final score as it performs much worse than the overall BAE.

This approach will be implemented mostly to tackle the major problem of the dataset: the presence of very likely hidden anomalies. As already mentioned, the intrinsic diversity provided by the many autoencoders usage will also translate into a wider and more resistant to Overfitting solution.

2.8 Parameters and Hyperparameters

As explained in Nyuytiymbiy (2020), parameters are only estimated from the data during training as the algorithm tries to learn the mapping between the input features and the labels or targets, meaning that these are internal to the model. In the case of neural networks these are the layers' weights and biases.

Hyperparameters, on the other hand, are parameters whose values control the learning process and determine the values of model parameters that a learning algorithm ends up learning. These can be the Nu and Gamma parameters for the OCSVM and the quantity of neurons, learning rate and more for the neural networks. The prefix 'hyper' suggests that they are an higher level parameters that control the learning process and the model parameters that result from it.

Regarding hyperparameters, many combinations will be tried out and evaluated using the theoretical metrics mentioned in the next section (2.9) and the testing method depicted in the chapter 4. The ideal would be to design a grid search that incrementally test every combination in a range established for every hyperparameter. Although, this approach is very expensive in terms of computation, which transmits into a very time consuming task. The adopted method runs multiple iterations of combinations between specific and important chosen hyperparameters, each cicle adopting few but more restricted values until performance results are satisfactory. Such methodology's application is described, for each model, in chapter 5.

2.9 Metrics

In order to evaluate the model's performance, the right metric needs to be chosen, which will differ between contexts. But first, there are some definitions to be made: true positives refer to the correctly predicted anomalies; false positives are the non-anomalies that we incorrectly identify as being anomalies; false negatives refer to the anomalies incorrectly identified as non-anomalous instances; and true negatives are healthy points classified as so. Since there are no labels, these metrics are based on the performance of windows containing artificial anomalies (further explained in chapter 4).

For most anomaly detection problems, data is usually imbalanced, meaning that the quantity of labelled normal samples vastly outnumber abnormal samples. This sort of data imbalance introduces problems that make accuracy (which is the ratio of correctly predicted rows to the total observations) an insufficient metric. Considering a naive model that classifies every entry as normal, we would get a really high accuracy, despite being a really unskilled solution.

Therefore, it is very important to consider more appropriate metrics: Precision expresses the percentage of positive predictions that are correct (true positive / true positive + false positive). Recall expresses the proportion of actual positives that were corrected predicted (true positive / true positive + false negative).

Both precision and recall are relevant, hence the use of the F1-score metric, which conveys the balance between both of the earlier mentioned and is defined as:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} = \frac{TP}{TP + \frac{1}{2}(FP + FN)} \tag{2.15}$$

Where TP, FP and FN represent true positives, false positives and false negatives.

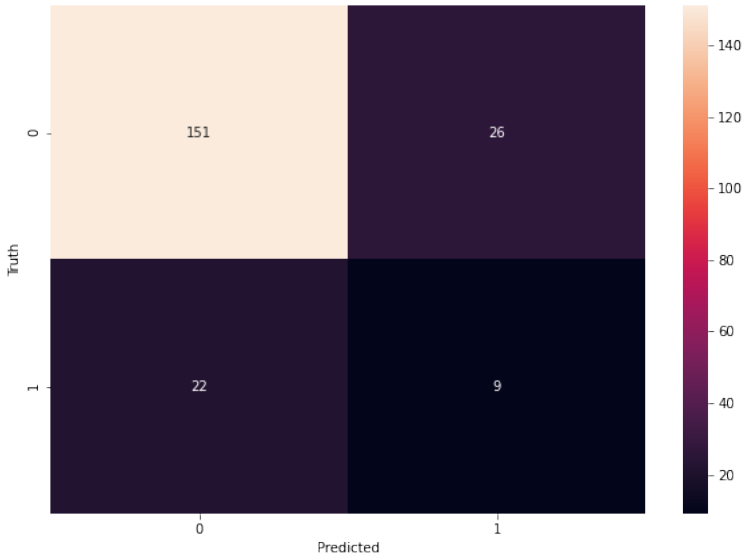


Figure 2.11: Example of a confusion matrix.

Although, it is desired to reduce the false positives because otherwise there could be resources that would be allocated for a problem that does not exist. This means that a high precision is preferred to a high recall.

Confusion matrices, which an example can be observed in figure 2.11, are useful to observe the amount of each prediction category (True positives, False Positives, False Negatives and True Negatives) and compare different model's results.

Because the deep learning models require a threshold to decide which entries are anomalies, plotting the considered values against an useful metric, such as the F1, will be useful. An example can be seen in figure 2.12.

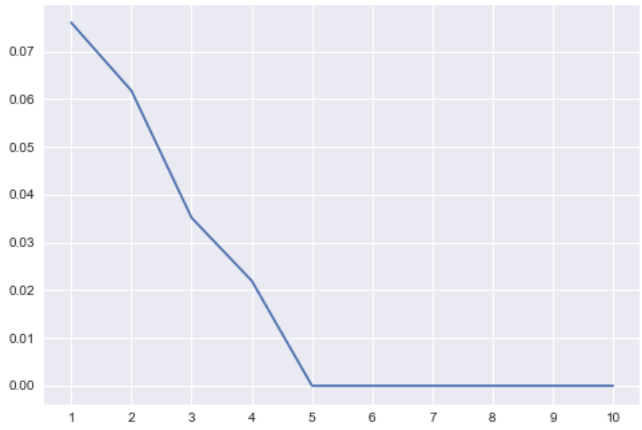


Figure 2.12: Example of threshold-f1 curve. Y-axis represents the F1 metric, while the X-axis plots all tried sigma multipliers.

3

Dataset

In this chapter the datasets will be described and both its exploratory analysis and preprocessing will be demonstrated.

3.1 Description and characterization of the data

Two datasets will be considered, which are maintained by NOS and result from its operating devices already in the production pipeline. One dataset concerns phone calls and the second, larger one refers to router and TV activity, which will be called Zapcae and Testpow respectively. As already mentioned in previous chapters, all data used in this thesis can be categorized as time-series as it implies a sequential order between rows. As mentioned in Andiojaya and Demirhan, 2019, time series data can either be irregular or regular. Since ours is regular (devices are most of the time turned on and regularly sending information), will not be necessary to fill in with entire artificial rows but missing column values might still appear.

3.1.1 Zapcae

The **Zapcae** dataset is rather small, consisting of 3886 entries which represent phone call behavior for roughly 5 months and a half (from 2020-08-18 to 2021-01-27). This one will be used as a preliminary test as it is a lot simpler than the others.

To comprehend the diversity of the data, one must understand its background. The dataset is straight forward but might need some explanations. Calls and clients behavior are quite similar because, in average, clients make two calls while trying to communicate with another person. Not being a linear relationship, both variables are helpful to the model, for example: if something is wrong in the phone infrastructure, one might need to make additional calls before successfully establishing a connection. At first sight, both success and drop rates seem redundant as well, although they are not complementary. Success rate depicts the amount of times the call reached an early stage of connection, while the drop one simply represents all calls that were interrupted at any stage (by

none of the clients). Therefore, both these variables can provide additional valuable information regarding hardware failures.

Since this dataset's features are retrieved as an aggregation from the whole country, identifying anomalous specific to any part of the physical hardware is impossible, serving then as a use case to identify general, serious failures.

3.1.2 Testpow

The **Testpow** dataset can also be depicted as Panel data because each row relates to an independent individual. These are routers and television boxes that amount to 2180951 devices in total. Having a periodicity of 21 entries per day, the dataset is constituted by 55 days of data retrieval.

Note that the aggregated form of the previous dataset allowed the company to keep a longer history of data, while this dataset, although voluminous, has a much smaller time frame.

This dataset relates to house modem devices and most of its columns are Key Performance Indicator (KPI) that can be used to evaluate the hardware's condition. There is a specific power line fork hierarchy that goes from the big centrals to each device at people's home. In our case, it is only relevant from regional control Cable Modem Termination System (CMTS) to the client's end. The company has two CMTS providers, CISCO and ARIS. Either company's apparatus has its own configurations and sensibilities regarding power injections. From the CMTS to the end of the lines, the physical connections are made with Hybrid Fibre Coaxial (HFC), which have a smaller bandwidth with both fibre and metallic cables. These networks use the Data Over Cable Service Interface Specification (DOCSIS) 3.0 framework that, for client's efficiency policy, has 8 downstream (from central to client direction) carriers and only 2 upstream (client to central direction) carriers. The data consists of metrics relative to the weakest carrier at the moment. The weaker the signal power, the more susceptible it is to noise in the network. Since the retrieved data might be bottle necked along the upstream line, apparent home device's problems can also represent a broader issue that affect multiple clients. Just like the earlier one, it will be segmented, this time by its hardware model. By training a model for each possible Model CPE, it is possible then, to use the corresponding model once the data arrives on the server. The used subset concerns the most popular device, which have much higher data quality and in-house expertise, IRIS (dcr7151/24). It amounts to 42000 rows of data.

3.1.3 Raw data

In the following tables: 3.1 and 3.2, raw data's features can be observed as well as their description for the zapcae and testpow, respectively. Note that there is no column that can be considered the target, that is, no column that depicts that row as healthy or anomaly.

Table 3.1: Calls dataset feature description

Designation	Description	Type
Clients	Quantity of clients that attempted to make a call	Discrete
Calls	Quantity of attempted calls	Discrete
Success Rate	Amount of calls that were successfully executed	Continuous
Drop Rate	Amount of calls that unwillingly dropped	Continuous
Time	Date and hour at which these calls occurred	Categorical

3.2 Exploratory analysis of datasets

In this section is shown the exploratory procedures applied to the data. This includes visualizations of the numerical values and the possible values for categorical ones, in order to capture the nature of the data.

3.2.1 Categorical Data

Regarding the Testpow dataset, there are 4 categorical variables in the dataset, the device id, device CMTS, device type and model CPE. As the id is unique for each device, there are 2180951 of these, while there are only 86 different CMTS, which are represented as a reference id as well. Devices with the same CMTS will naturally be physically close to each other in the same region. In figure 3.1 the distribution of rows for each CMTS can be seen. Can be observed that exist a whole gradient of row quantities, from those in most populated areas therefore operating over many devices, to those in remote areas thus receiving a lower quantity of entries.

The device type has 4 distinct possible values: Router, RouterV4 (a special kind of router), STB (TV boxes) and NA (which encompasses a variety of devices that can be either poorly registered in the database or auxiliary instruments like range extenders). Their individual amount of rows can be observed in figure 3.2. While there are many without it, most of them lie in the STB and Router categories.

For the Model CPE, which represent the device hardware specific model, there are 12 possible categories, these are: CVE30360_V3, CVE30360_V2, zd4500zno, dcr8151/24, CVE30360, dcr7151/24, BVW3653_V2, BVW3653, CVE30360_SIP, BVW3653_ZONF, tg2492no and null (for no model registered). For each one of these models, the amount of corresponding data entries can be seen in figure 3.3. There are two very popular models while the rest stays in a more reserved range. Must be noted that there are many missing models as well.

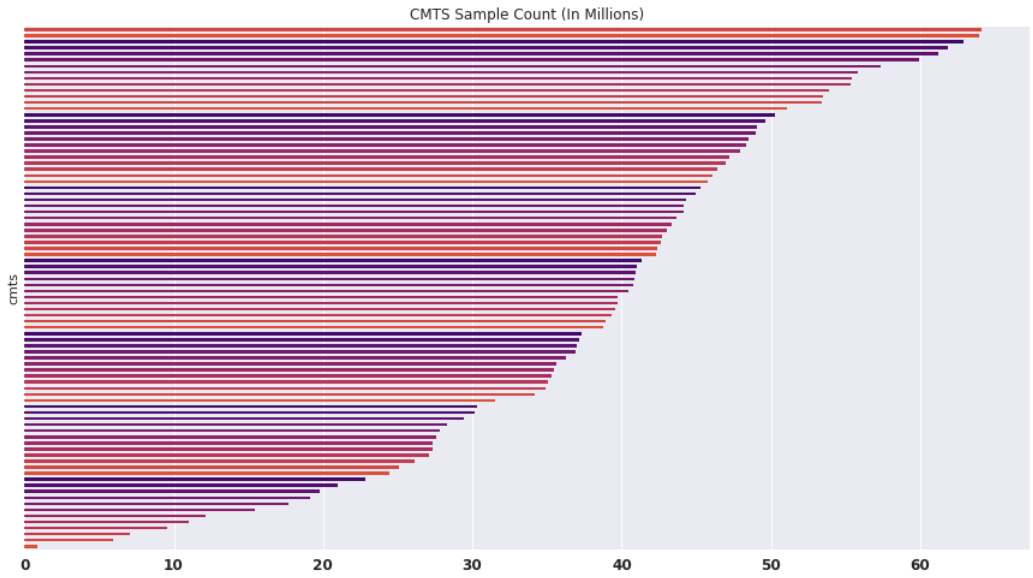


Figure 3.1: Sample count for each CMTS (X Axis: In Millions)

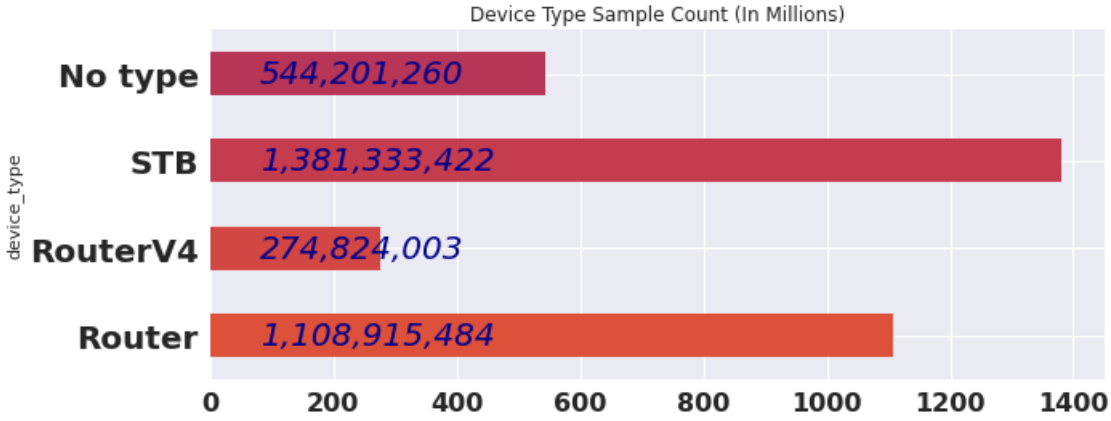


Figure 3.2: Sample count for each Device Type (X Axis: In Millions)

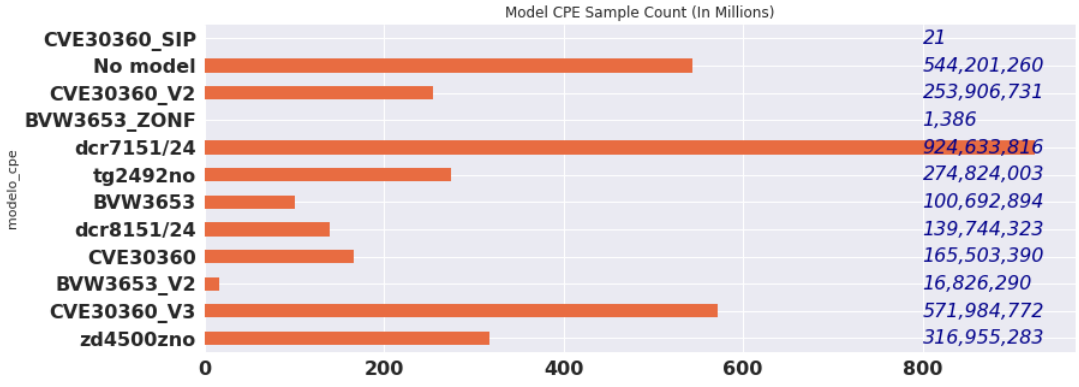


Figure 3.3: Sample count for each Model CPE (X Axis: In Millions)

3.2.2 Numerical Data

In the Zapcae dataset there are both continuous and discrete variables. In figure 3.4, two examples of the call activity can be observed. The first one represents normal data, whereas the second one includes both outliers and an anomaly. The outliers lie around Christmas (December 25th) and New Year (December 31st to January 1st), which are predictable special occasions. The anomaly can be found in January 6th, at the end of the day there is an observable data drop. Both graphs also depict the seasonality of the dataset, having five similar week days and two further consecutive days decreasing in activity, the weekend. Its distribution can be observed in figure 3.7(a).

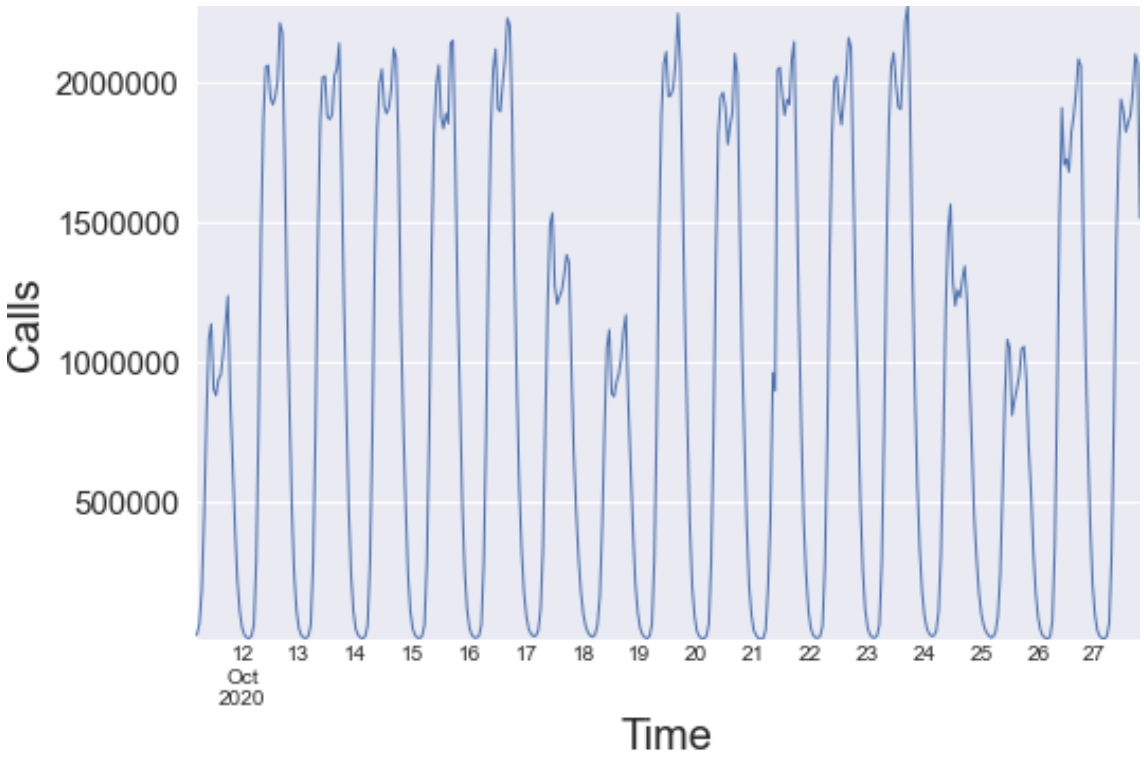
In figure 3.5, the clients activity can be seen. It is particular similar to the call one because they are closely tied. Such is depicted in the distribution 3.7(b).

Both success and drop rates activity can be watched in figures 3.6(a) and 3.6(b). Usually, success rates are rather high and close to the maximum of 100%. On the other hand, drop rates are quite close to zero. Their distributions are available in figures 3.8(a) and 3.8(b), for success and drop rates, respectively. Successes, in general, keep much closer values to 100% depicting the relaxed nature of the attempts that are considered a success. Whereas, in the drops case, the distribution is flatter, meaning that there are more dropped calls than non-successes (complementary value of success).

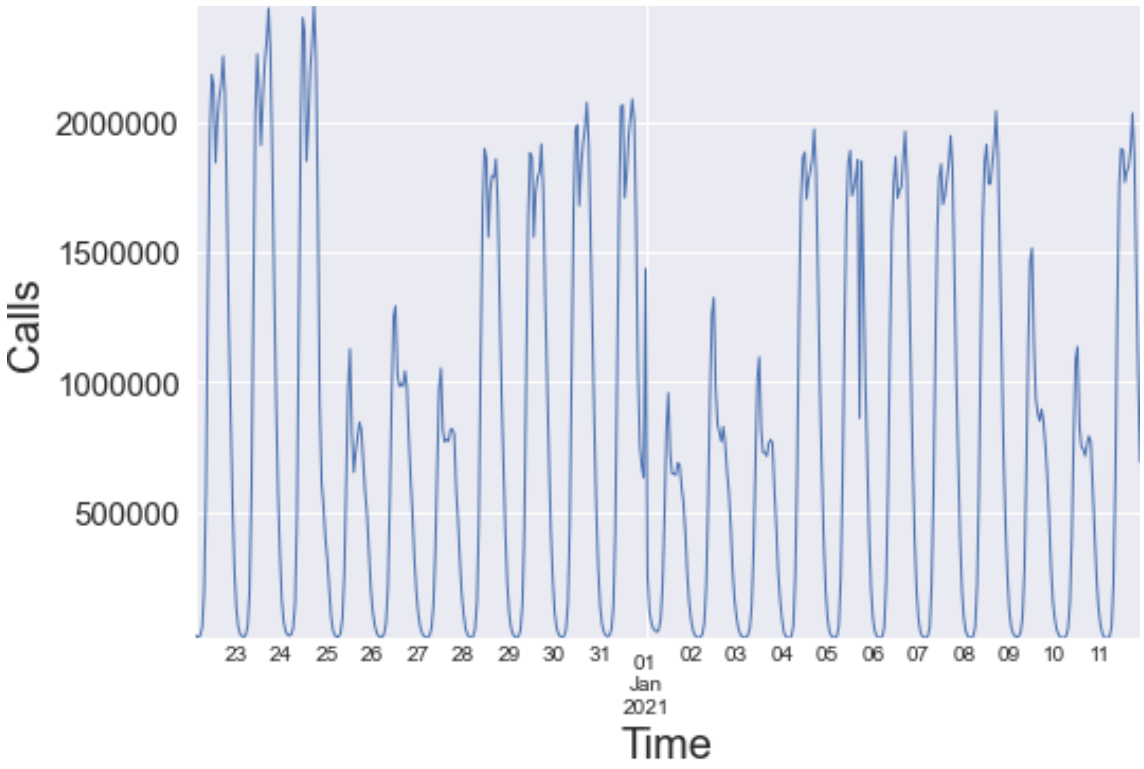
As the router/TV dataset is Panel data and very large, seems appropriate to fix a predetermined device and visualize its points through time for numerical values. Notice that all of these numerical variables are also continuous.

In figure A.1 (in the appendix) is depicted the behavior of the first four variables (Downstream RXpower, Upstream RXpower, Upstream TXpower and Pathloss Up) for a single device and a some time. Both Downstream RXpower and Upstream RXpower are measured in dBm, which explains the difference in value scale when compared to Upstream TXpower and Pathloss Up, that are measured in dBv. An interesting aspect to note is that since the Downstream RXpower is registered right off the server, it is less prone to error, resulting into a more detailed and accurate data. Both RXpowers reveal seasonality and trending while the rest adopt more predictable and less diverse values.

The distribution histogram was plotted for each of these variables (figure 3.9, 3.10, A.4 and A.5), after grouping by device and calculating their means and standard deviation. Overall, we can observe spikes in certain values for every one of them, these seem to be adopted default values. The Downstream RXpower has only one because it is less prone to error as said before.It also follows a gaussian curve because it is a signal strength KPI and its measurement is made at the router. Therefore, the greater the distance between the client's home and the distribution point, the lower the signal. The values adopted by the Upstream RXpower rounds around two main values, meaning that they belong to two different families of models. These two first variable distribution histograms are in this chapter while the others can be found in the appendix A because they are too many to exhaustively present in this chapter.



(a) Normal call activity sample.



(b) Abnormal call activity sample. Outliers around Christmas and new year's eve, whereas there is an anomaly in January 5th.

Figure 3.4: Examples of call activity.

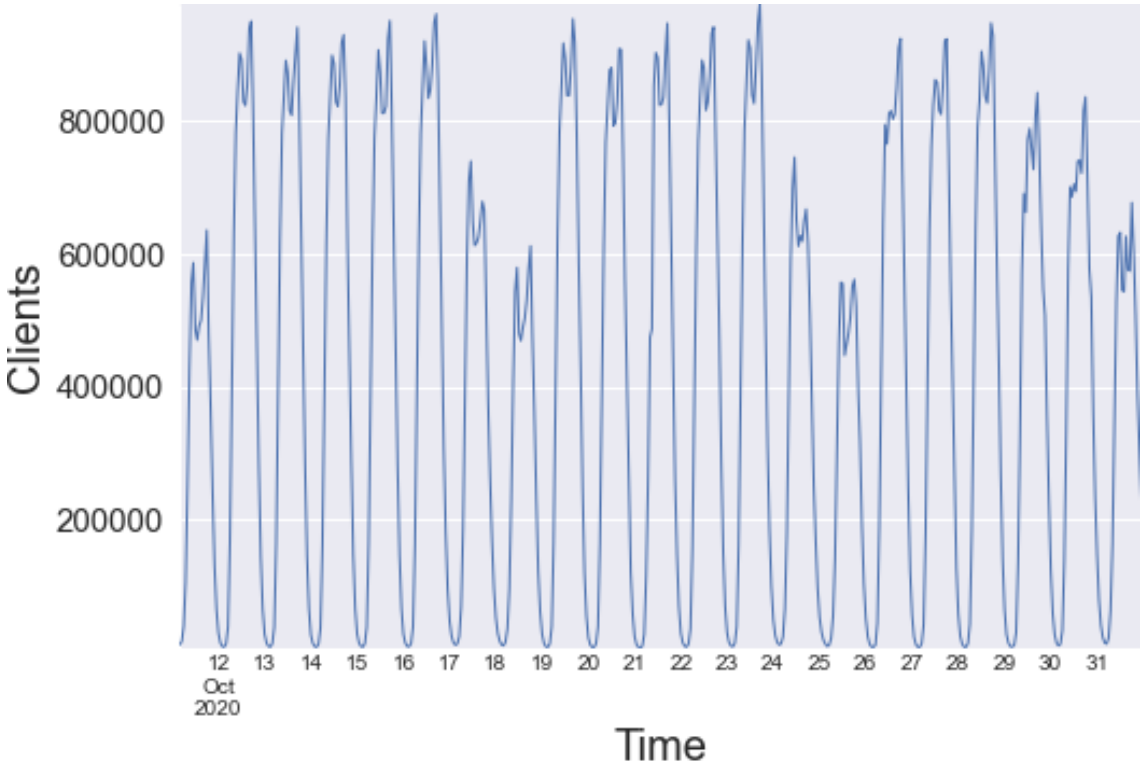


Figure 3.5: Clients sample.

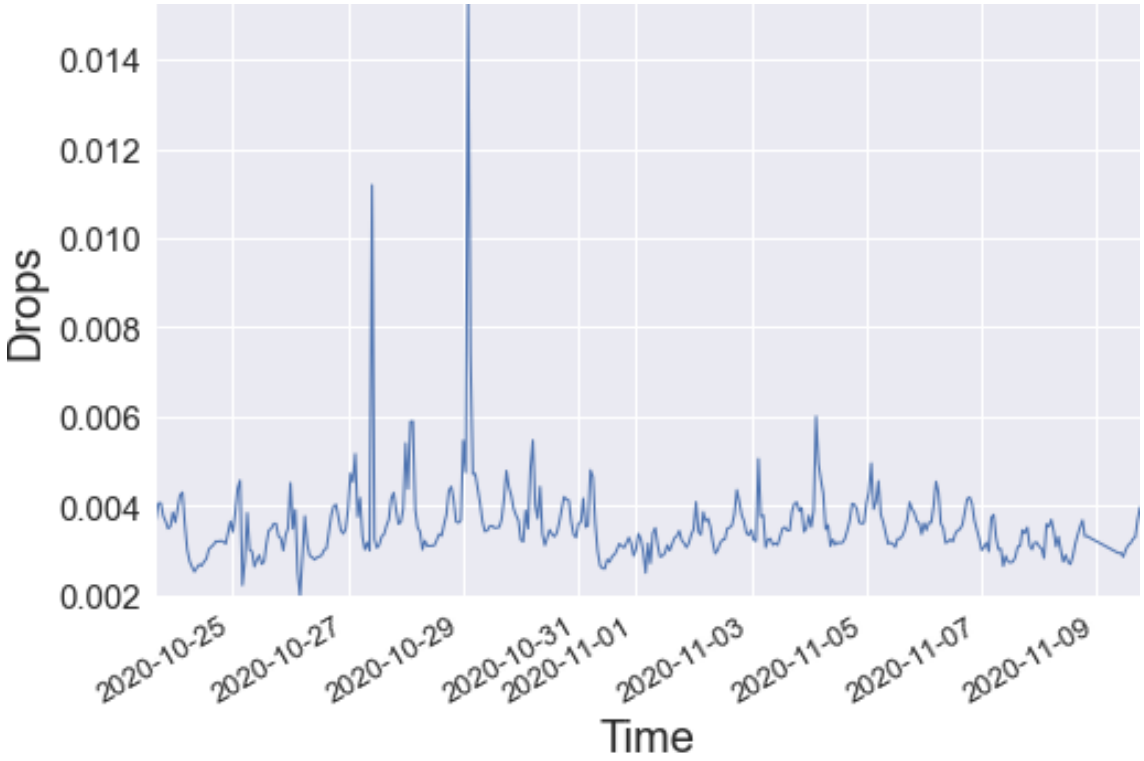
Both Downstream CER and Upstream CER are ratios that represent the amount of corruption there is in the package, where 100 represents package completely corrupted and 0 the lack of corruption. The two CCERs (Downstream CCER and Upstream CCER) represent the quantity of the corrupted parts that the signal recovering algorithm managed to recuperate.

The depiction of the example of behavior for variables Downstream CER, Upstream CER, Downstream CCER and Upstream CCER can be observed in figure A.2 (in the appendix). The Downstream CER showed to be always 0 (or very close to 0) meaning that all information packages were not corrupted, whereas in the Upstream CER there were some corrupted rows. The Downstream CCER shows some minor (close to 0) corrections made, where the Upstream CCER depict an higher need for corrections to be made. This reveals a practical example of the earlier statement regarding upstreams' value quality when compared to the downstream.

Through figures A.6 to A.9 the mean and standard deviation's distribution histograms can be seen for each variable. Since these columns concern the corruption of the packages, we should employ care when representing numerical values to the range of the magnitude of the data being analysed.

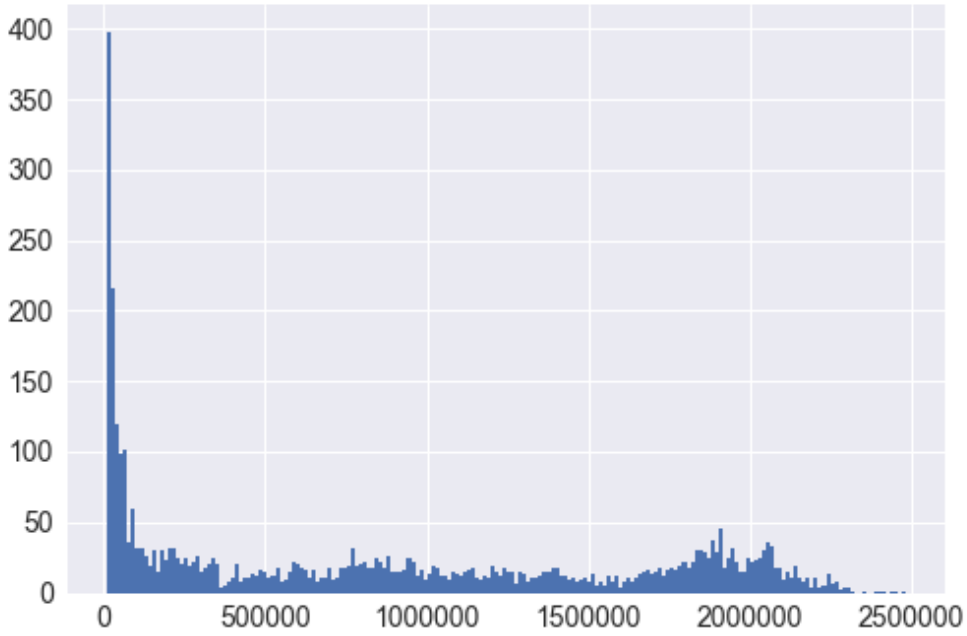
The behavior for the last variables is depicted in figure A.3. Because the Downstream SNR and Upstream SNR are ratios of how much noise exists in the transmission, both occur in a narrow range of values, from 40 to 35 and 40 to 30 (mainly) respectively. This is

(a) Success call activity rate sample.

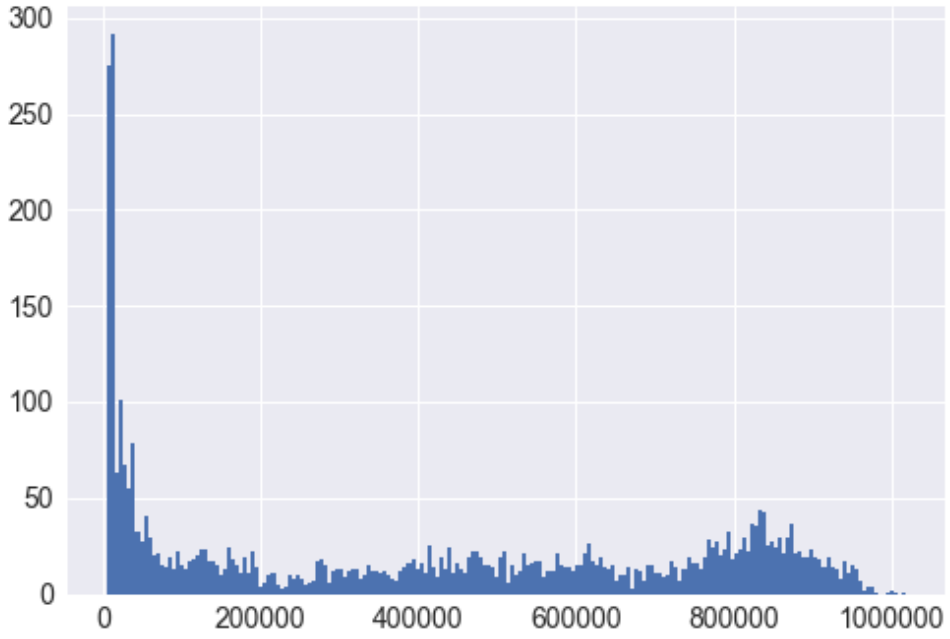


(b) Dropped call activity rate sample.

Figure 3.6: Examples of success and dropped call rates.

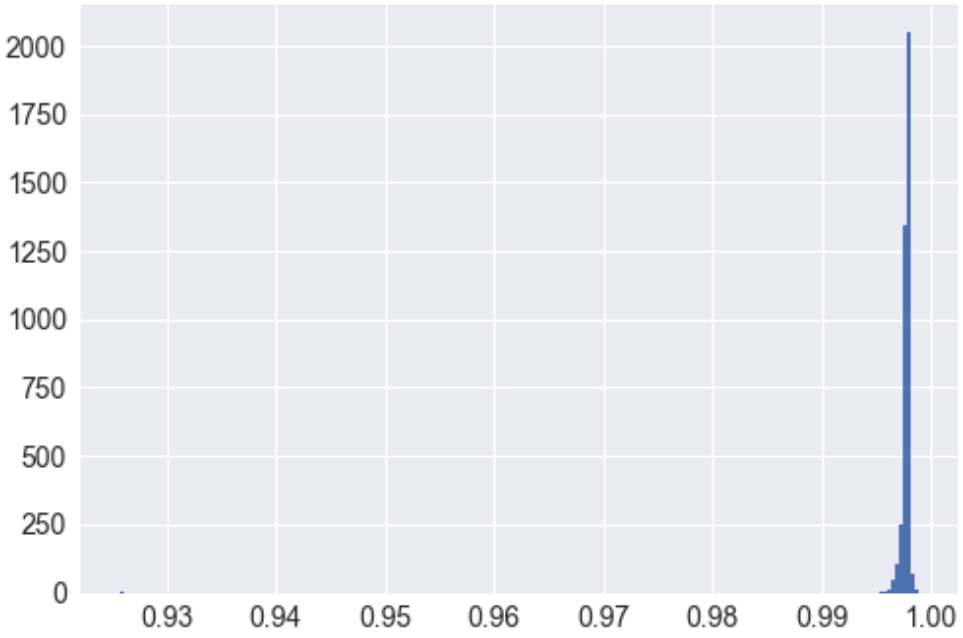


(a) Histogram for the distribution of observed calls, hourly aggregated.

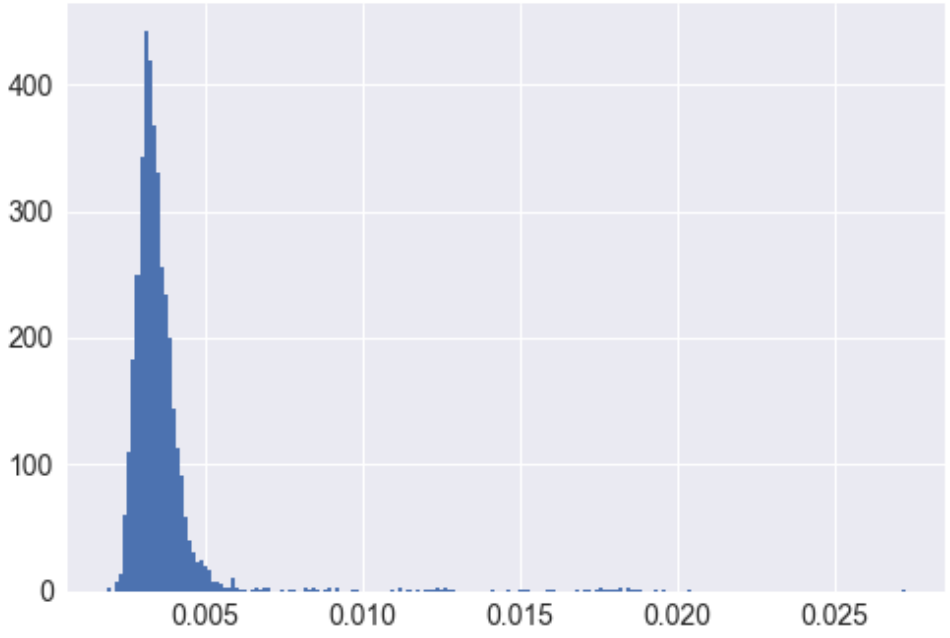


(b) Histogram for the distribution of observed clients, hourly aggregated.

Figure 3.7: Calls and Clients distributions.

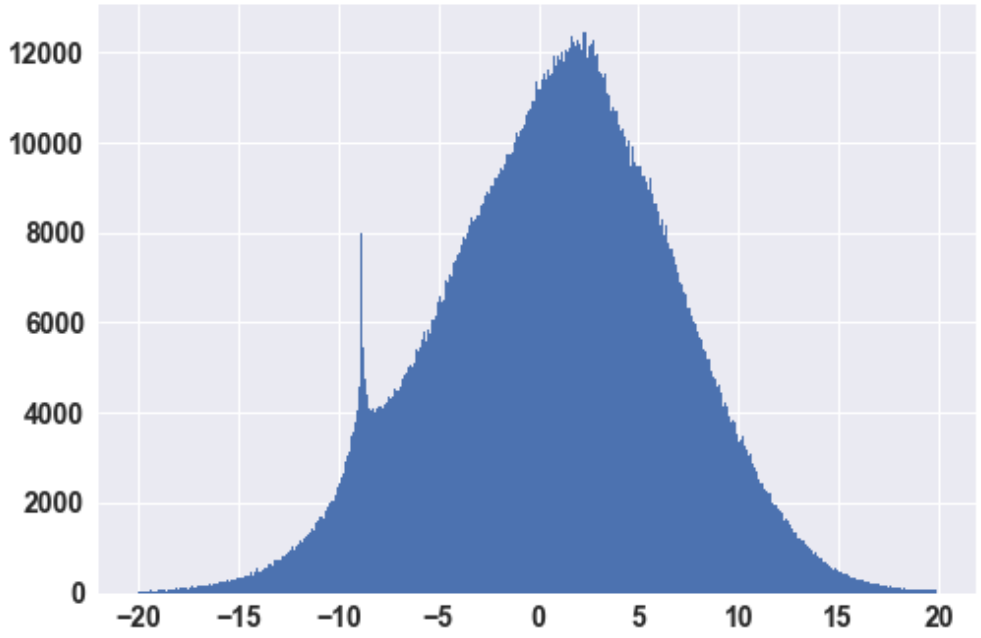


(a) Histogram for the distribution of observed success rates, hourly aggregated.

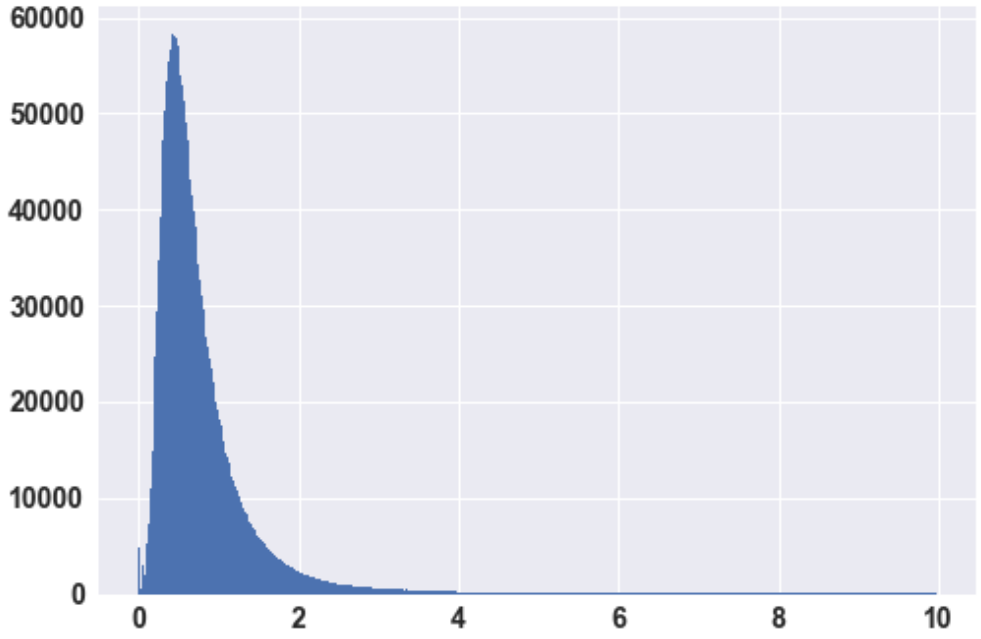


(b) Histogram for the distribution of observed drop rates, hourly aggregated.

Figure 3.8: Success and drop rates distributions.

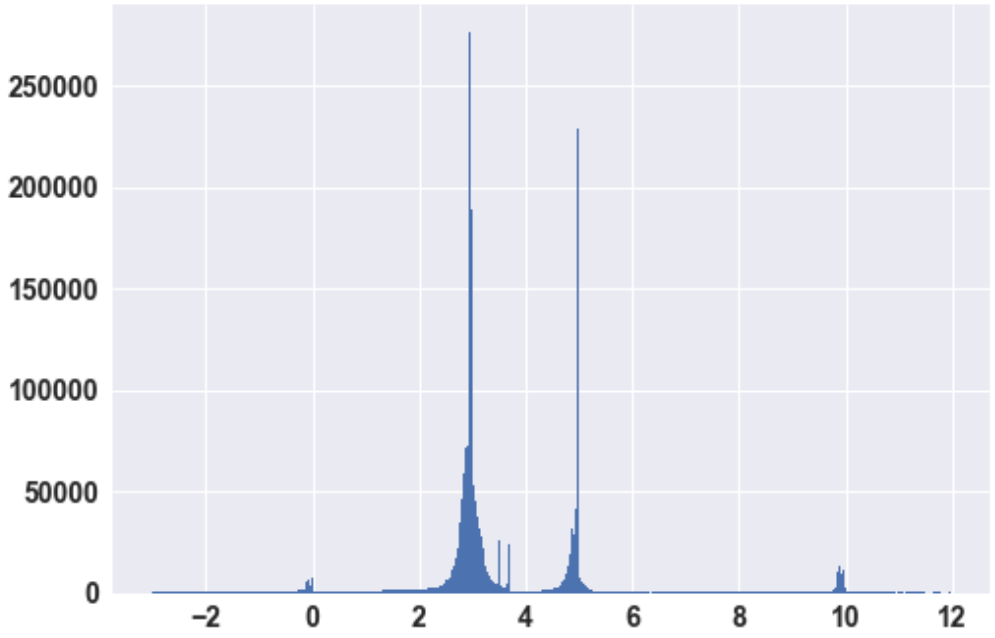


(a) Mean distribution of Downstream RXpower

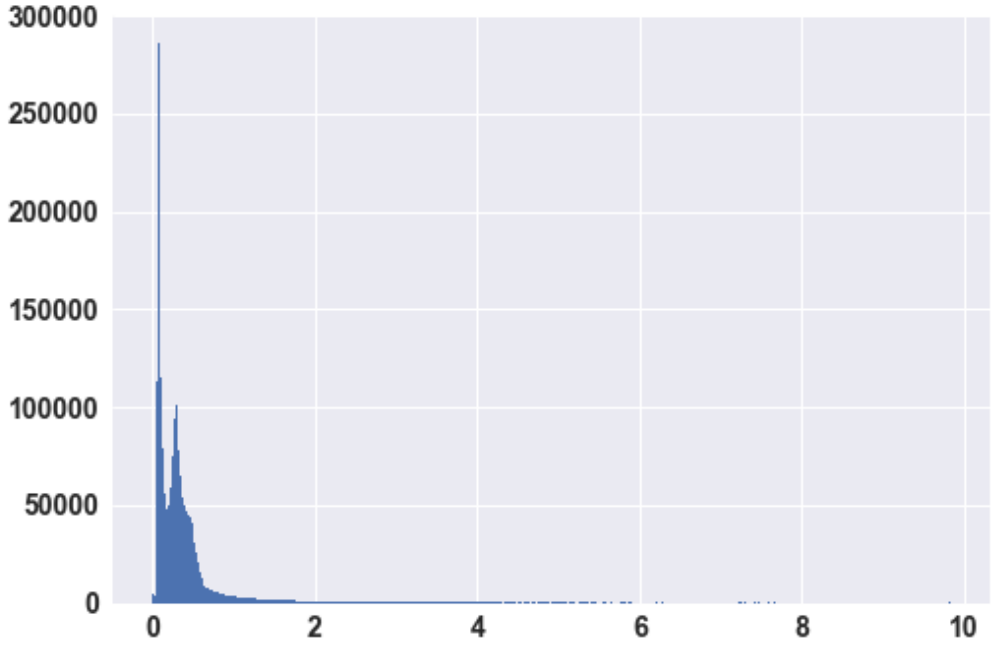


(b) Standard deviation distribution of Downstream RXpower

Figure 3.9: Mean and standard deviation distribution for Downstream RXpower



(a) Mean distribution of Upstream RXpower column



(b) Standard deviation distribution of Upstream RXpower column

Figure 3.10: Mean and standard deviation distribution for Upstream RXpower

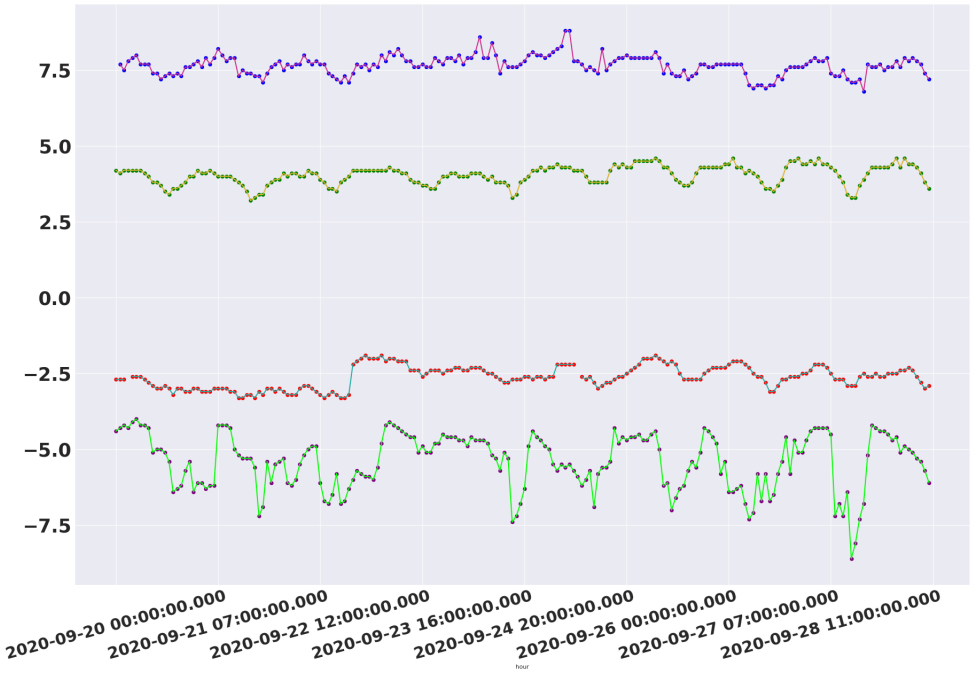


Figure 3.11: One device example through time for first four numeric variables

proved by their mean and standard deviation histograms (figures A.10 and A.11), having a much greater variation in the Upstream data. These values represent a percentage, being already naturally bound between the 0 and 100 mark.

Regarding the Sysuptime, it is the accumulated time that the device has been turned on. We can observe the periodic restarts of the device and the very high attributed values which are in milliseconds and proportional to the time axis. Like earlier variables, in figure A.12 one can observe Sysuptime distributions, noting that the mean histogram's X axis is represented in millions.

The last two (Sum Bytes Up and Sum Bytes Down) track the space necessary for the transmission. These also are in a high number scale because bytes are a very low storage unit, as can be observed in figures A.13 and A.14.

To get a sense of the data's regularity, and because the earlier plots are interpolating missing points, one important measure (Downstream RXpower) is chosen alongside a few more complete (less null data present) individuals to be plotted through time again. This time a scatter plot was also plotted over the line graphic and can be visualized in figure 3.11. For legibility reasons only 8 days of data were displayed. Another interesting aspect of this visualization is the inter-device value register at which this data operates.

3.3 Preprocessing

Before feeding the data to the models, it needs to be put in the most convenient form to models.

Since models can only process numbers, all categorical variables (except for the device id) will be transformed using the One Hot Encoding (OHE) method. This method creates additional columns, one for each possible entry for the target categorical variable. Rows will have the value of 0 for all the columns except the one that represent its original category (Brownlee, 2020). In our case, the Device Type and Model CPE will create 4 and 12 new columns, respectively. Whereas CMTS will add 84 new entries.

Numerical values will be normalized in order to make sure that the data is internally consistent, that is, each data type has the same content and format. This means that data will be rescalled so that all values are within the range of 0 and 1. A value is normalized as follows:

$$y = \frac{x - min}{max - min} \tag{3.1}$$

Where min and max correspond to the minimum and maximum value in the dataset.

Another transformation to make is the splitting of the time variable. Usually, each day granularity would be represented by their own column (hour, day, month, year). But due to GDPR rules, there is not enough data throughout the year for the model to learn such a high picture seasonality. The hour and day of week will be used in order to account for a more realistic seasonality period, the week. Leaving these two columns as numerical can cause numerical proximity associations that do not exist between the hours and days, thus the OHE is again used to mitigate this aspect.

One more aspect to consider is that the examples shown earlier in this chapter are from devices that have a high quantity of entries, whereas some devices' row quantity is low. This can be observed in figure 3.12, revealing that the majority of the devices have at least 1000 rows through the given time.

Regarding the rows themselves, there are many that not only contain null values, but their presence stands out in many columns (as can be observed in table 3.3). Since nulls might be related to hardware problems (concerning numerical variables), their presence needs to stand out as well, by attributing a −1 value after data normalization. Whereas for categorical ones, it is better to leave them as a separate category. For categorical ones, it is better to leave them as a separate category. It is also worth to note that only the Testpow dataset suffers from this null issue.

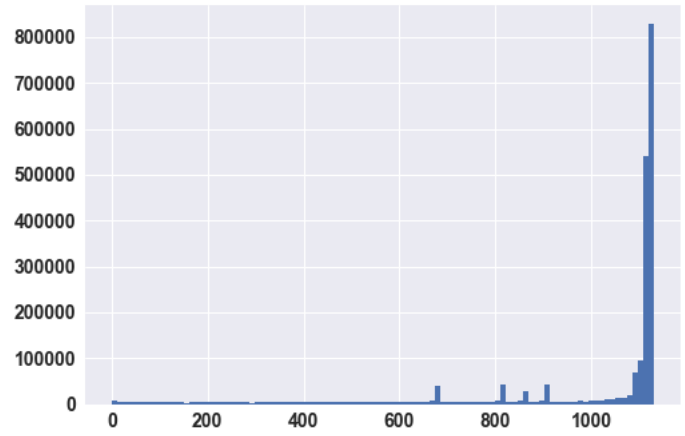


Figure 3.12: Quantity of rows per device distribution

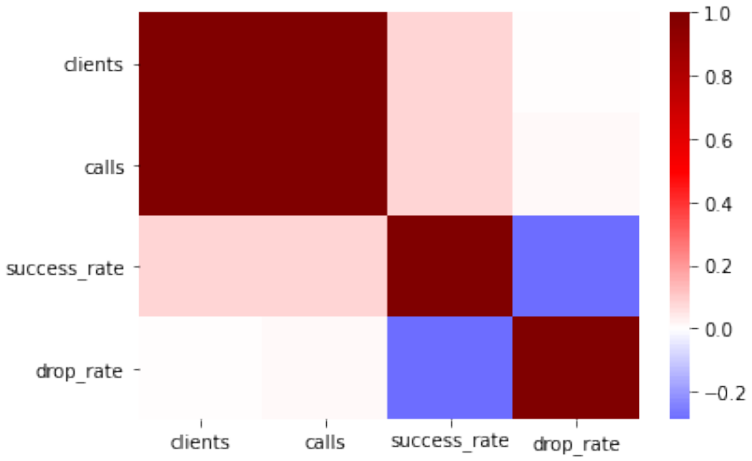


Figure 3.13: Correlation between variables in the Zapcae dataset.

3.4 Feature Selection

Feature selection is a step that will only be used when further strictly specified.

Thanks to the deep learning nature of the chosen models, the more variables that may explain the anomalies, the more possibilities for patterns being recognized. Although, as will be further explained, this processing step will be tried, sometimes because of model adequacy and others due to training time.

Shown in figure 3.13 is the heatmap regarding the Zapcae dataset's variables correlation. Although the clients and calls have slightly different business rules, the correlation between these two is great. Therefore, the clients column is erased. Due to the purpose of reducing the dimensionality of the problem, the categorical variables, which the contribution is rather greater than others, will also be taken.

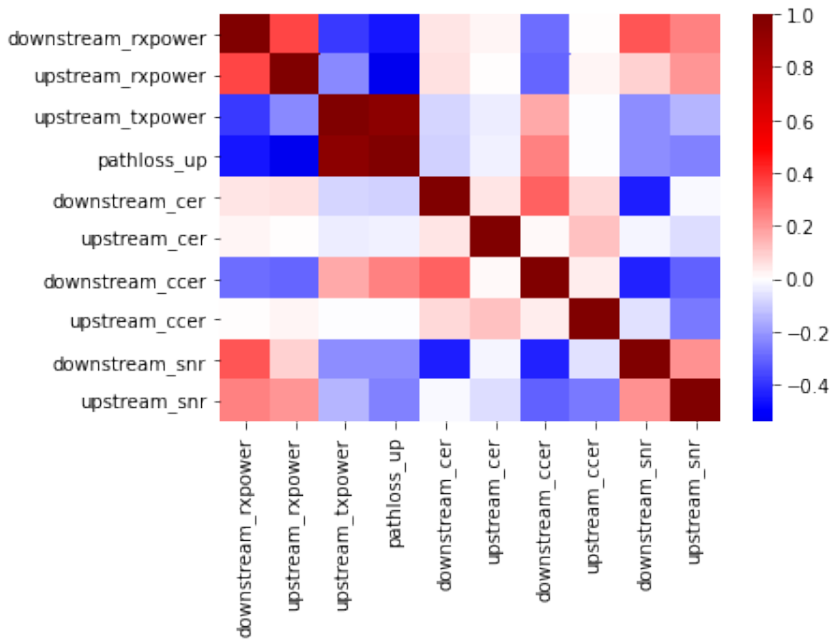


Figure 3.14: Correlation between variables in the signal power dataset.

In figure 3.14, the heat map of the correlations between variables can be observed. The Downstream rxpower, being the signal power received at the home device, can explain anomalies along the downstream hardware. Upstream txpower is the signal power when the upstream package leaves the home device, explaining then problems that only affect the house equipment. While the Pathloss (as the difference between the transmitted and received upstream power) might catch anomalies along the upstream structure that affect multiple devices in the same region. The Downstream SNR (signal to noise ratio) is inversely correlated to the Downstream CER, which (the latter) was removed. Both the CCERs (ratio of correctable errors) were also removed due to their analysis complexity regarding their correction algorithm. The Upstream CER is included because of its low correlation with all other variables and because it, representing the amount of information that can be decoded before any correction, might describe device related problems. Finally, the Upstream SNR is fairly correlated with many other variables, hence its removal. Then, the list of selected columns (aside from categorical ones) is:

- Downstream rxpower
- Upstream txpower
- Pathloss up
- Upstream CER
- Downstream SNR

3.5 Window time frame

The selection of window width is not an easy task. If the width is too large, it will increase the computation complexity and time delay by requiring more historical data for analysis. If it is too small, time-series patterns might be lost.

In You et al. (2019), which tested a few LSTM based models, also tried three different window widths. Since their time scope is much smaller, 10 seconds will be considered a small window, whereas 60 will represent a large one, a 30 second window was also tested. The wider frame yielded better results in general, but also lost much of its precision compared to the 30 window, so this choice also depends on the preferred metrics.

In our case, because we are interested in the model's performance with the information regarding a day's worth of data, each time window will contain 21 points, which is the amount of hours within a day's worth of data in the dataset per day.

These windows are sliding, meaning that the inputs are considerate one time step at a time. For example, if N is the total of inputs through time, and m the size of the window, than the first inputs should cover $\{x_1, x_2, ..., x_m\}$, whereas the second one will handle $\{x_2, x_3, ..., x_{m+1}\}$.

3.6 OCSVM Window Representation

OCSVM considers each data point individually instead of a whole window. If the window representation is an important aspect of the problem, which is the case, one can still do this by creating the mentioned temporal windows and collapse all their rows into one. This row now contains all features of all window's the data points, representing the whole temporal period.

Table 3.2: Routers/TVs dataset feature description

Designation	Description	Type
Device	Id of the device	Categorical
CMTS	CMTS which the device makes a part of	Categorical
Hour	Date and Hour at which the data was reported (e.g. 2020-12-21 10:00:00.000)	Categorical
Downstream rxpower	Received signal power on downstream carrier at home devices (in dBm)	Continuous
Upstream rxpower	Received signal strength on upstream carrier in the servers (in dBm)	Continuous
Upstream txpower	Signal strength transmitted on upstream carrier from home devices (in dBv)	Continuous
Pathloss up	Difference between transmitted (TXPOWER_UP emitted by the client) and received (RX-POWER_UP arrived at the server) upstream powers, for the CM in question (in dBv)	Continuous
Downstream cer	Ratio of Codeword errors received by CM - Ratio of downstream errors present in the cell (in percentage)	Continuous
Upstream cer	Ratio of Codeword errors received by the CMTS relating to the Cable Modem - Ratio of uptream errors present in the cell (in percentage)	Continuous
Downstream ccer	Correctable Codeword error ratio received by CM - Ratio of correctable downstream errors present in the cell (in percentage)	Continuous
Upstream ccer	Ratio of correctable Codeword errors received by the CMTS relating to the CM - Ratio of correctable upstream errors present in the cell (in percentage)	Continuous
Downstream snr	Cable Modem Downstream Signal to Noise Ratio	Continuous
Upstream snr	Cable Modem Upstream Signal to Noise Ratio	Continuous
Sysuptime	Cable Modem Uptime (in milliseconds)	Continuous
Sum bytes up	Total upstream in bytes	Continuous
Sum bytes down	Total downstream in bytes	Continuous
Day part	Daily Split: Date substring for indexing purposes	Categorical
Device Type	Device type between Set-top Box (STB), Router and RouterV4	Categorical
Model CPE	Model of the device at home (12 models)	Categorical

Table 3.3: Amount of NULLS among every data column

Column	Percentage of NULLS
Device	0%
CMTS	0%
Hour	0%
Downstream rxpower	80.9032%
Upstream rxpower	0.0187%
Upstream txpower	80.8574%
Pathloss up	80.8956%
Downstream cer	81.0717%
Upstream cer	2.6170%
Downstream ccer	81.0717%
Upstream ccer	2.6170%
Downstream snr	81.0717%
Upstream snr	0.0187%
Sysuptime	80.8574%
Sum bytes up	2.9628%
Sum bytes down	2.9628%
Device type	0%
Modelo CPE	0%
Product class	0%
Day part	0%

4

Models Testing

In this chapter will be presented the tests preparation which will evaluate the models.

Since there are no labels and no way of telling which rows are anomalous, the only possible method to evaluate the models is to retrieve a sample of the time-series (regarding one device in the Testpow's case) in the test set and produce artificial anomalous entries, which are assigned the label "anomalous". This way, we can calculate the metrics mentioned in section 2.9.

Some different testing scenarios were created, including an unexpected high, low and null values, a persistent value, for a column or many, for both datasets. It is very important to note that each of the following data samples are **subsets**, which will be cut into **multiple windows**. This means that in a single scenario there are some anomalous windows but many healthy ones as well. All test scenarios are shown in the appendix B, from figure B.2 to figure B.11.

In figure B.1 one can observe an example of healthy subset of Zapcae dataset, while in figure B.3 an anomalous one is shown.

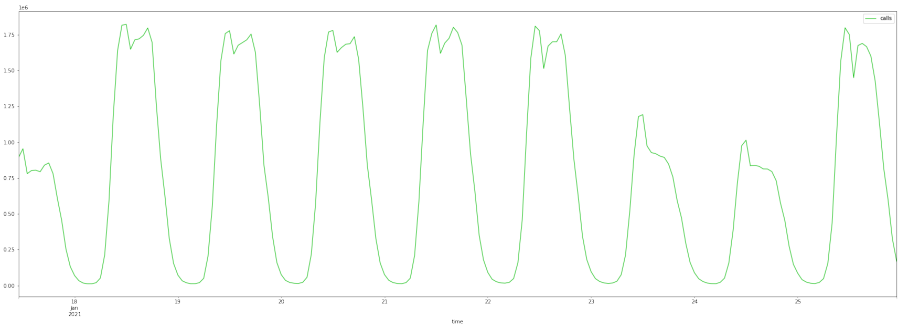


Figure 4.1: Zapcae dataset healthy example of data subsets used for model evaluation

For Testpow, the same healthy case is described in figure B.7, while in figure B.9 an anomalous persistent value case can be observed.

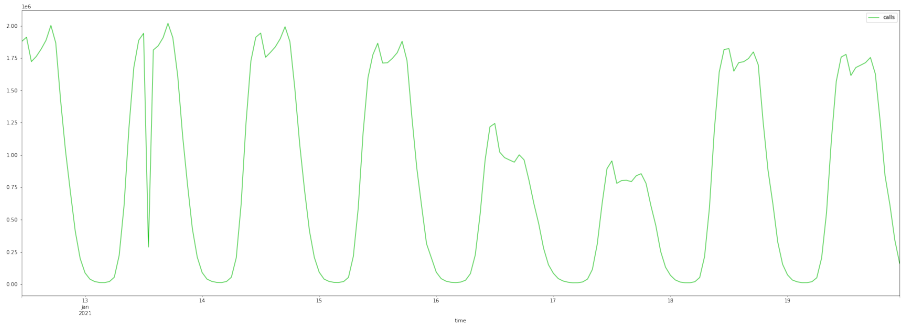


Figure 4.2: Zapcae reduced anomalous value for calls and clients column

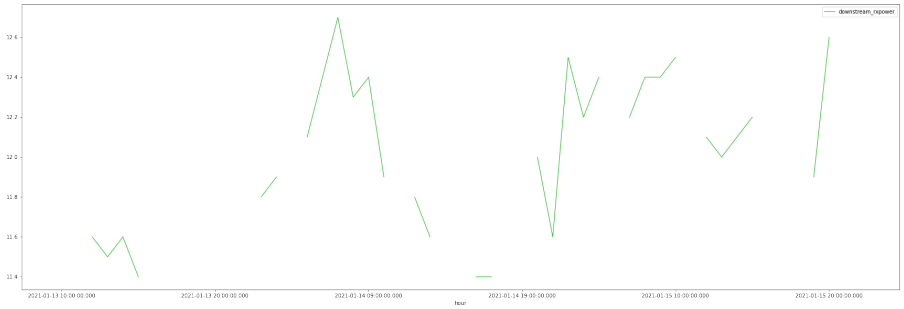


Figure 4.3: Testpow dataset healthy example of data subsets' downstream rxpower column used for model evaluation

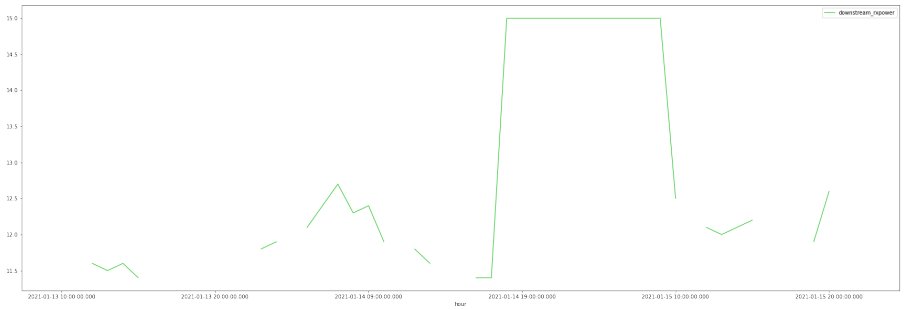


Figure 4.4: Testpow anomalous persistent value for downstream rxpower column

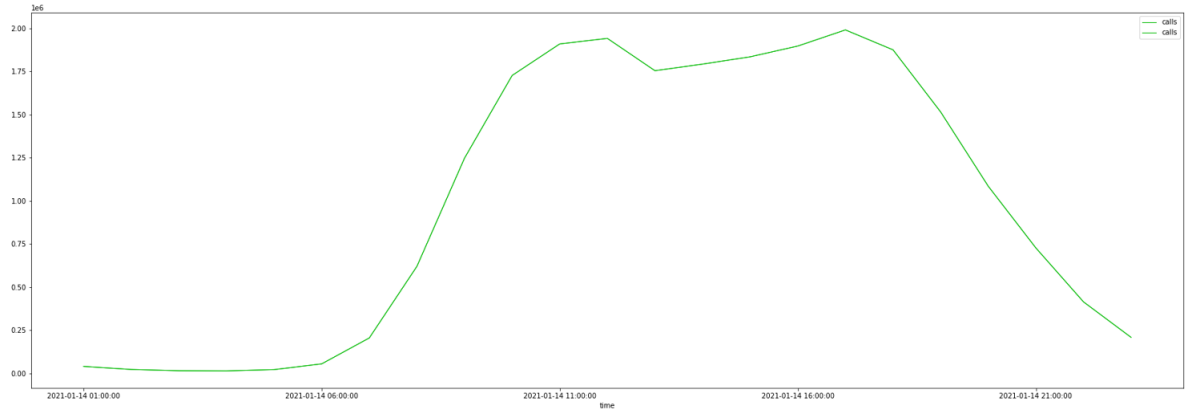


Figure 4.5: Chosen week test healthy window for the zapcae dataset.

Now that we have scenario data with labels, the performance metrics can be calculated, such as the true positives and negatives, and then false positives and negatives. Afterwards, the precision, recall and f1 metrics can be calculated.

The confusion matrices can also be produced for each test. Hypothetically, using a perfect model, the resulting matrix ought to be diagonal. This would mean that the misclassified quadrants should hold the value 0, while the accurately classified ones would have the quantity of healthy/anomalous tests.

Then, to optimize the choice of the threshold, the average F1 metric will be plotted for every sigma multiplier (in the Autoencoder's case). Although a low value seems to always be the best value, it is always tried a range of sigmas between 1 and 10.

In order to assess their performance through observation, two examples of healthy and two others anomalous windows will be chosen. There will also be two situations, one test will use a week window while the other will be a weekend one. For each model type, the hypothesis that yielded better results will classify each scenario.

The following figures (figures 4.5 to 4.12) show the chosen window for each dataset and situation. The Zapcae tests are the kind of anomaly that is the most common: reduced volume in the calls and clients columns.

Since the anomaly threshold is the mean of all scores with the addition of the standard deviation multiplied by some value, the prediction of these windows cannot be done alone. This is because the adopted mean and standard deviation would become the window's own. If the target window is an anomalous one, it won't be flagged as such, because it needs the healthy windows' context considered in the mean in order for the target window score to be above the threshold. These windows were tested among many others, for the threshold value to consider a more wide range of scores.

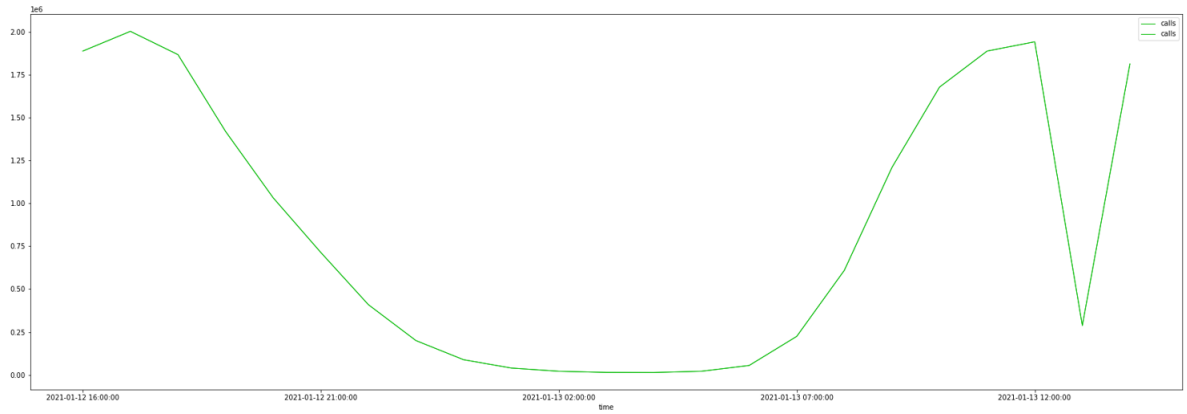


Figure 4.6: Chosen week test anomalous window for the zapcae dataset.

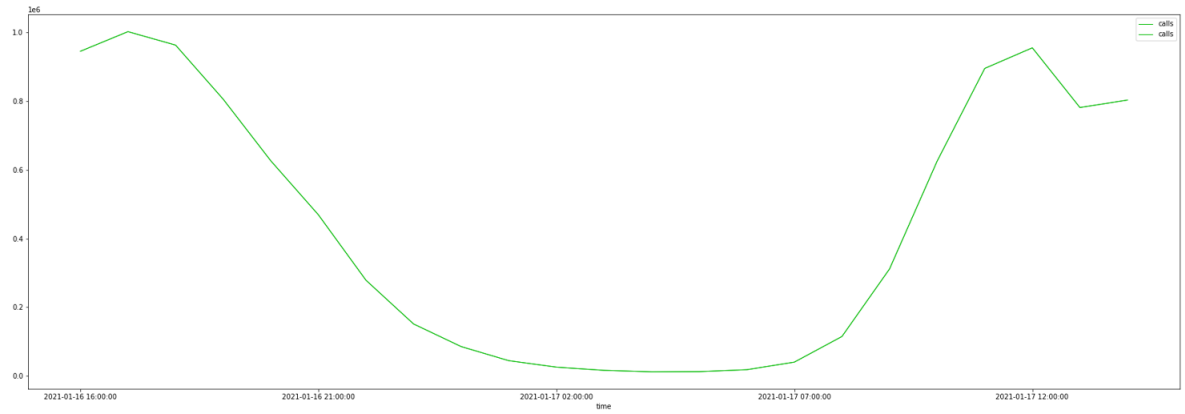


Figure 4.7: Chosen weekend test healthy window for the zapcae dataset.

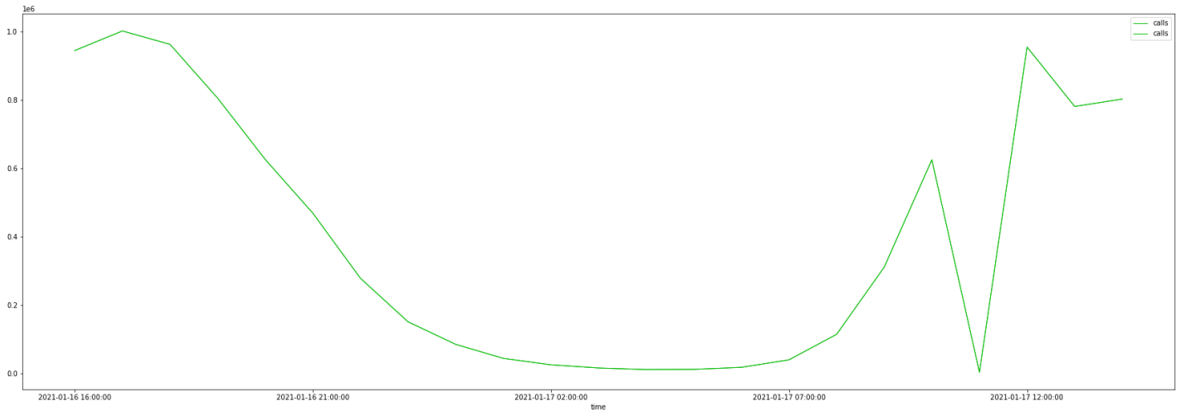


Figure 4.8: Chosen weekend test anomalous window for the zapcae dataset.

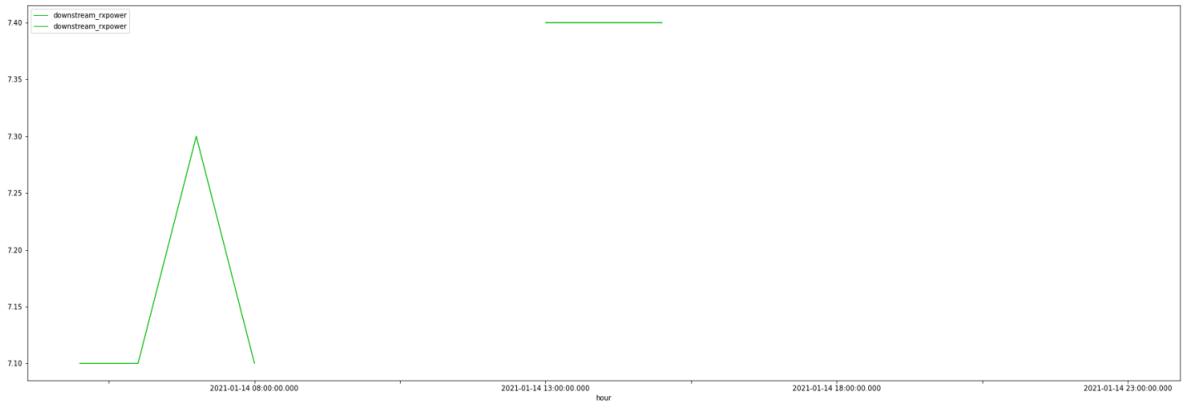


Figure 4.9: Chosen week test healthy window for the testpow dataset.

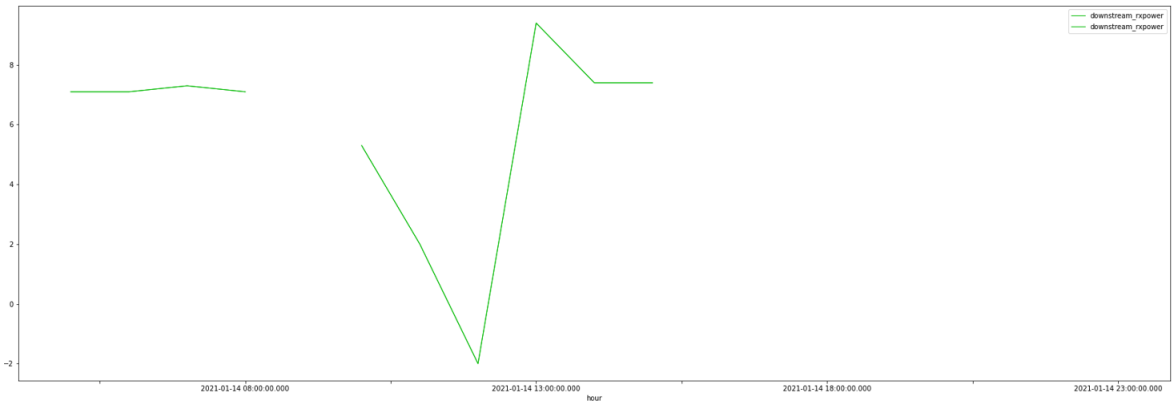


Figure 4.10: Chosen week test anomalous window for the testpow dataset.

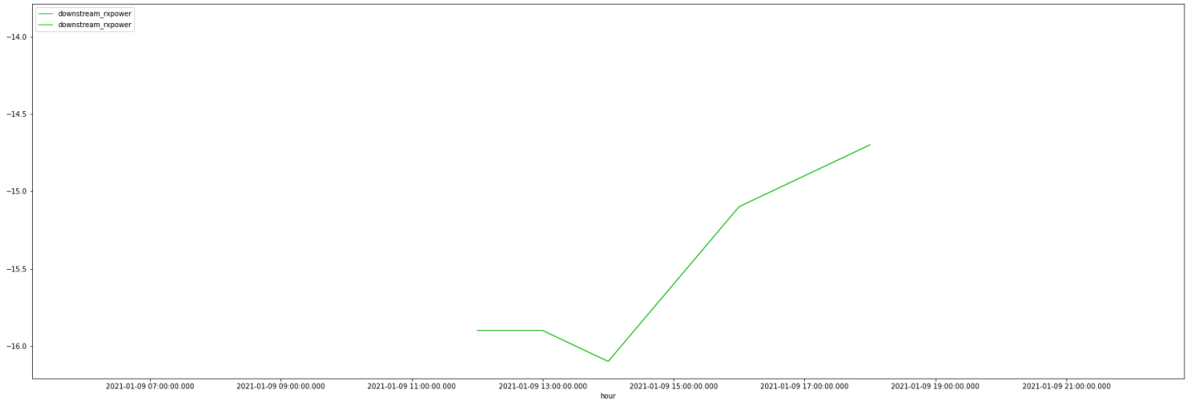


Figure 4.11: Chosen weekend test healthy window for the testpow dataset.

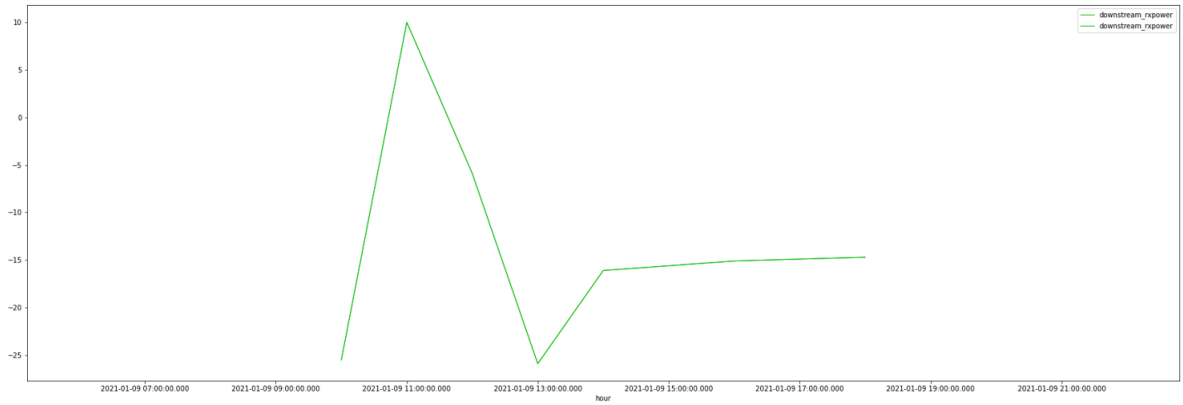


Figure 4.12: Chosen weekend test anomalous window for the testpow dataset.

5

Results

In this chapter will be presented the results of the work carried out in the Dissertation.

Before diving into the specifics of testing and results for each hypothesis, it is relevant to mention that the datasets' train, validation and test split ratios are 80%, 10% and 10%, respectively. The OCSVM is a full unsupervised technique. The only fed data is the training portion, while the test one will be further used in the artificial testing explained in the previous chapter 4. On the other hand, the autoencoders, technically being a supervised problem that tries to reconstruct the inputs, use all portions of the data with a copy of each for its own targets.

In the next section, the hyperparameter optimizations are shown alongside the results it yielded, for all models and both datasets. Each hypothesis is registered as an entry in the respective model's results table.

Most of these results concern a width window of 21 entries but other sizes were tried, and the one that yielded the best results was the window width of 7. Throughout the chapter, it will be mentioned whenever it is relevant.

For an overview of every models' performance, from C.1 to C.11 (in the appendix C), there are tables which display the average of each metric for each hypothesis for all tests.

On the other hand, every model went through all tests designed for both datasets. Then, every test scenario has its corresponding results table for each model type, placed in the appendix C from table C.12 to C.67. The rows shown in these tables relate to a certain hypothesis' extracted metrics.

5.1 OCSVM

The OCSVM hyperparameters to be optimized are the Nu and Gamma. The SVM model needs the tuning of the C parameter, which is essentially a regularization parameter, it trades off between the misclassification of training examples against simplicity of the decision surface. The lower the C, the smoother the decision surface, while a high C aims at classifying all training examples correctly. Although, in the OCSVM this is represented

in the Nu parameter, being related in the following manner: $nu = A + B/C$. Nu is the upper bound on the fraction of margin errors and a lower bound of the fraction of support vectors relative to the total number of training examples. For example, if it is set to 0.05 it is guaranteed to find at most 5% of your training examples being misclassified (or can be considered as outliers by the decision boundary) and at least 5% of your training examples being support vectors. Therefore, an overall lower range of parameters Nu will be tried. The gamma parameter defines how far the influence of a single training example reaches, with low values meaning "far"and high values meaning "close". The Gamma parameter will adopt a wider range of values. The auto gamma mode uses the $\frac{1}{quantity\_of\_features}$.

Although the RBF kernel function usually yields better results, other functions were also tried (RBF, poly and sigmoid). Although, only the best performing function's results will be shown for each context.

In the table C.1 (in the appendix C) display the metric averages, while from C.12 to C.16, all tests' metrics are displayed for the Zapcae dataset using the RBF function.

Both original and feature selected versions of the dataset were tried and these values relate to the latter. It's not that the original's results were worse overall, but that the RBF, with the less dimensions, performed better compared to all others. These values are averages of all test windows created for those specific hyper parameters. The set of hyper parameters that yielded better results are $NU = 0.01$ and $GAMMA = 0.01$, which produced an F1 of roughly 0.34. Most of these results show a recall of 1, meaning that all test windows are being classified as being anomalous, while the best hyper parameters yield a slightly better one (around 0.98). Generally, precision is quite low, and in this case, roughly 0.214.

Concerning the manual single window test mentioned earlier, all healthy and anomalous windows are scored as an anomaly, failing the test.

Regarding the Testpow dataset, its average results can be seen in table C.2 (in the appendix C). From C.17 to C.22 all metrics can be seen for each test.

These were retrieved using the poly kernel, without feature selection. All these results, since recall is 1 or close enough, are quite the same as classifying all windows as anomalous, therefore the very similar and low F1 and precision values.

It seems that even tweaking the Nu value, which controls overfitting, most of entries keep getting signaled as anomalies. Predictably, each manual single test window resulted in an anomaly classification.

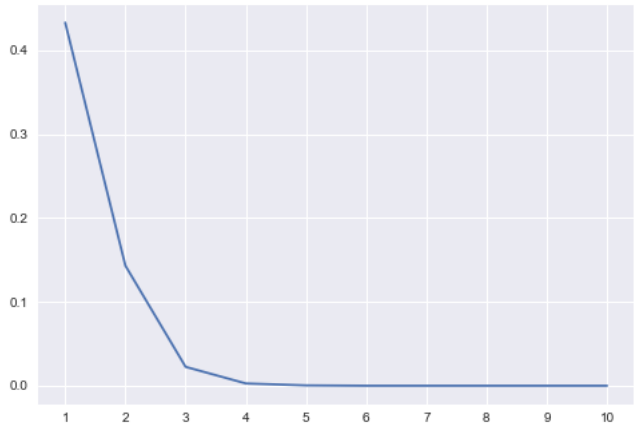


Figure 5.1: F1 measure (Y axis) versus sigma threshold values (X axis) curve.

5.2 LSTM Autoencoder

The most important hyperparameter to be defined in these neural networks is its neurons architecture, which will define how powerful the model is. The learning rate determines how great a leap of parameter optimization the network can take. If given a very low value it does not get enough time to train properly, and can be stuck around a promising minimal value (regarding the loss optimization) if high enough.

The average results for the dataset Zapcae can be seen in the table C.3 (in the appendix C), using a width of 21 and feature selection. For the each individual test, there are the tables C.23 to C.27.

Note that the architecture column represents the neurons from one side of the structure to the bottleneck. The choice of 1 sigma threshold for defining what is an anomaly was chosen based on the best average F1 outputs, which can be seen in figure 5.1.

The best result for the zapcae dataset would be the model 4, with around 0.62 as average f1 value and 0.63 for average precision. It is a small, less powerful architecture, using only 4 layers (32, 16 - 2 layers for the encoder, a bottleneck layer of 16 neurons' outputs, and then another symmetric layer for decoder). There are other more powerful models that almost reached these values, like models 12, and 16. Note that in this case the feature selection was used, meaning that using the whole dataset did worsen the performance.

In figure 5.2, one can see the losses through the training, for all 100 training epochs. Validation data seems to be performing better than the training one, getting a hint of underfitting.

Since the above metrics are an average of all tests, let's take a look into the cases individually. There is a specific kind of anomaly this model seems to get, which is a lower value for the success rate of a call, which the corresponding results table can be observed

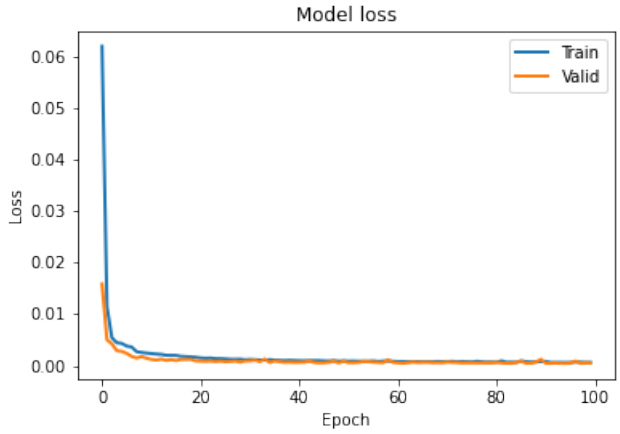


Figure 5.2: LSTM AE best model's loss plotting through the epochs, while training for Zapcae dataset.

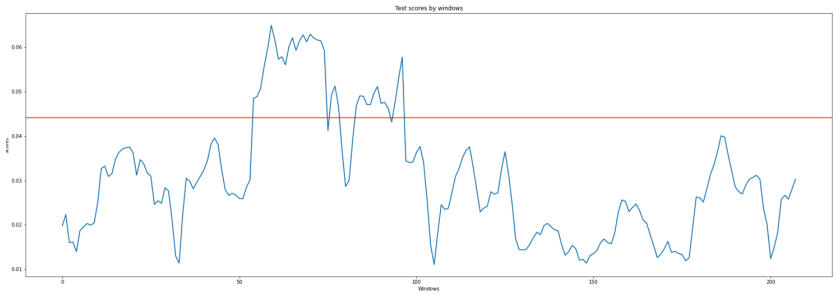


Figure 5.3: LSTM AE anomaly scores through the lower success rates Zapcae test time-series subset.

in tables C.23 and C.24. In figure 5.3, the scores for each window created through the time-series subset are plotted, the first and last value belonging to the first and last test window respectively. The red line delineates the chosen anomaly threshold, if below the window is healthy, if above it is anomalous. The corresponding test subset can be seen in figure B.5, in the appendix B.

It seems that around the timeframe in which the anomaly occurred, the scores are much higher, presumably those are the actual anomalous windows. This scenario yielded an incredible F1 of 0.93, recall of 0.88 and precision of 1. The model appear very sensible to anomalous values in the success rate. The confusion matrix is represented in figure 5.4.

Unfortunately, the other tests were not so successful. As an example I picked the test subset described in picture B.3, in the appendix B. It is a common calls and clients reduced values, which is translated into way fewer entries for phone sites and corresponding results are seen in table C.27. In figure 5.5, we see the scores through the test, and observe that, although there are higher scores among the anomaly, there are other surpassing points which should not. It seems to coincide with the weekends. The confusion

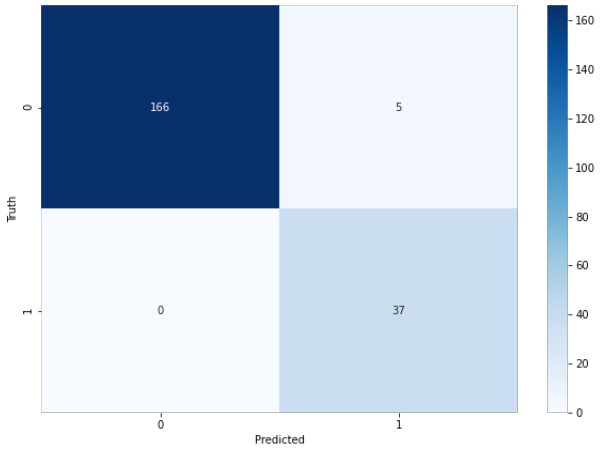


Figure 5.4: Confusion matrix for the lower success rates Zapcae test in the LSTM context.

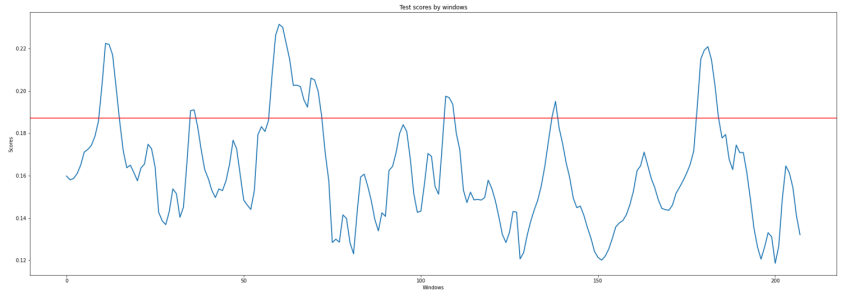


Figure 5.5: LSTM AE anomaly scores through the lower calls and clients Zapcae test time-series subset.

matrix is shown in figure 5.6.

When we look into the chosen more practical tests (mentioned in the previous section), they are of the most common type which is the reduced calls and clients variable. But, unfortunately those are not the best performing tests under this model's evaluation. When prompted with these specific windows, both healthy windows were correctly classified. Concerning the anomalous ones, the week anomaly was correctly signaled as anomalous while the weekend one was incorrectly classified as healthy.

Concerning the Testpow dataset, the average results can be seen in table C.4 (in the appendix), for 21 width windows and no feature selection. All tests are displayed from table C.28 to table C.33. In the figure 5.7 can be observed the average F1 scores for each of the sigmas tried, leading to the choice of 1 sigma.

Although not as good as one would hope for, the best results for this scenario using the LSTM Autoencoder was yielded by the model 19. It got around 0.45 average F1 and 0.37 average precision using 6 layers (512, 128, 8 - one layer for the encoder with a bottleneck of 8 neuron output, and then another, symmetric, layer for decoder).

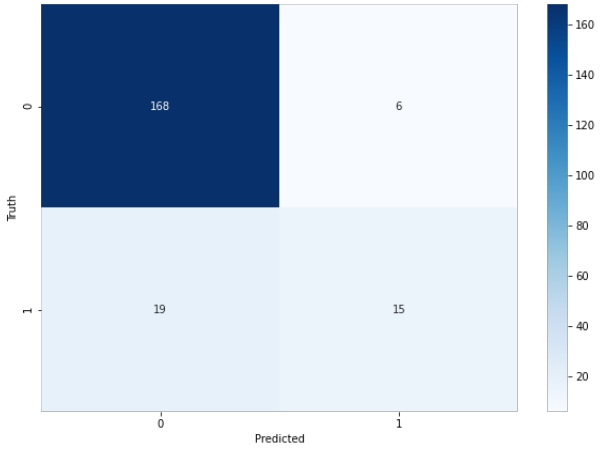


Figure 5.6: Confusion matrix for the lower calls and clients Zapcae test in the LSTM context.

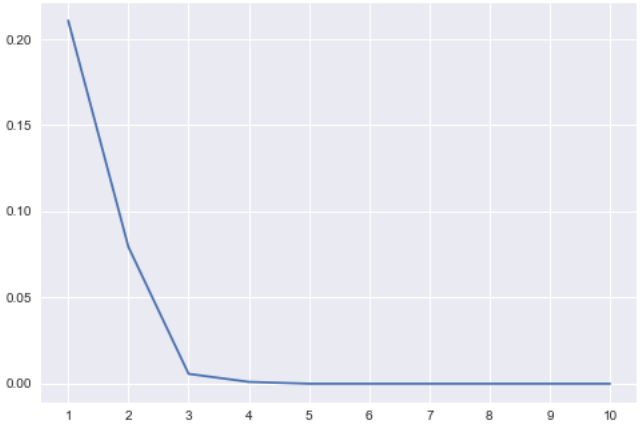


Figure 5.7: F1 measure (Y axis) versus sigma threshold values (X axis) curve.

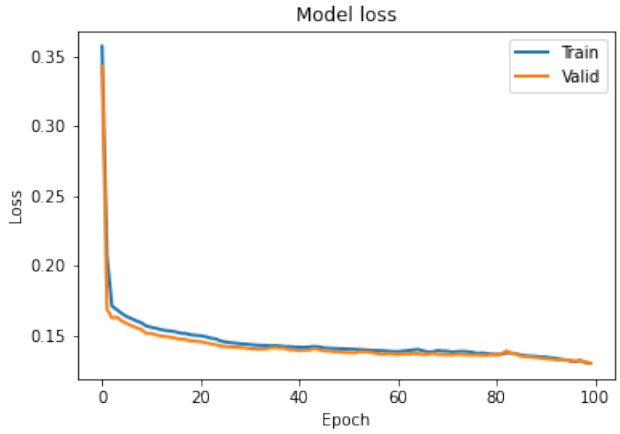


Figure 5.8: LSTM AE best model's loss plotting through the epochs, while training for Testpow dataset.

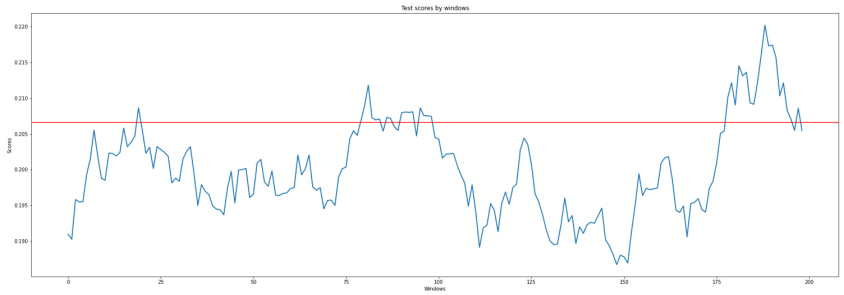


Figure 5.9: LSTM AE anomaly scores through the many nulls in 3 different columns testpow test time-series subset.

In figure 5.2, one can observe the losses through the training, for all 100 training epochs. Again, validation data seems to be producing fewer loss than the training one.

When looking at individual tests, one observes that those which involve many missing values are also the ones that yielded better results. Such was the case for test scenarios like the one described in figure B.8 (displayed in the appendix B), even when the dowstream rxpower is the only anomalous column (tests metrics shown in table C.28) or when there are 2 more (table C.29). The scores through the windows, regarding 3 anomalous columns, are shown in image 5.9 and respective confusion matrix in image 5.10. It got around 0.63 F1 score and precision 0.52.

Persistent value anomalies seem to perform poorly (around 0.47 F1), especially a zero persistent one (0 F1), shown in tables C.30 and C.33. But surprisingly, the anomalies that include a high variation of values for a period of time, when there are 3 affected columns, performed well (around 0.63 F1).

With a model that yielded a lower evaluation metric as these ones, it is expected that will perform worse in the manual tests, which happened. It classified week windows, both healthy and anomalous, as healthy and the two weekend tests as anomalies. Two windows

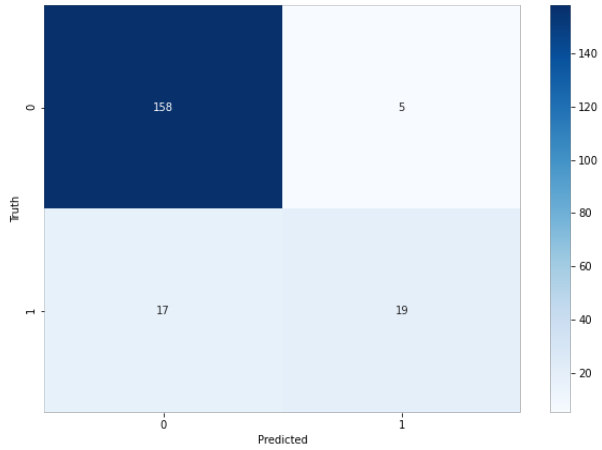


Figure 5.10: Confusion matrix for the many nulls in 3 different columns testpow test in the LSTM context.

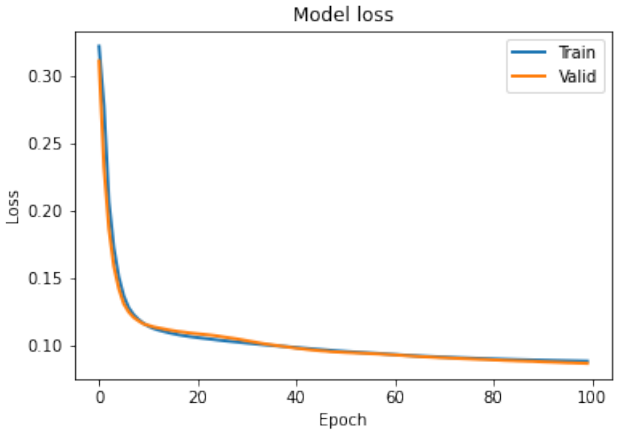


Figure 5.11: LSTM AE best 7 width window model's loss plotting through the epochs, while training for Testpow dataset.

were correctly classified, but it is clear that the model is more sensitive to weekend data.

In this particular case the feature selection did not improve the outcome of the algorithm. On the other hand, a smaller window of width 7 did improve a lot overall, getting around 0.56 F1, 0.5 precision and 0.64 recall with an architecture of 4 layers (16, 8 - layer of encoder with 8 neurons' output as bottleneck, and the decoder symmetric layers) and 0.00001 learning rate (model 2). These models can be seen in table C.5 in the appendix C. The earlier mentioned model has index 2, but there were other improved hypothesis, such as the 17.

Regarding the 7 width window, its training can also be observed in figure 5.11. Its curve seem a lot smoother.

The 7 window average results are displayed in the table C.5, while the corresponding

exhaustive test metrics' table are seen from table C.34 to table C.39. Interestingly, there is the same pattern of types of scenarios that ocurred in the 21 window case. The most performant, the missing values and high value variation for 3 columns had around 0.68 F1 and 0.66 precision, while persistent value cases scored around 0.53 F1 and 0.42 precision. In the persistent zero scenario could be achieved an F1 of 0.35, this time.

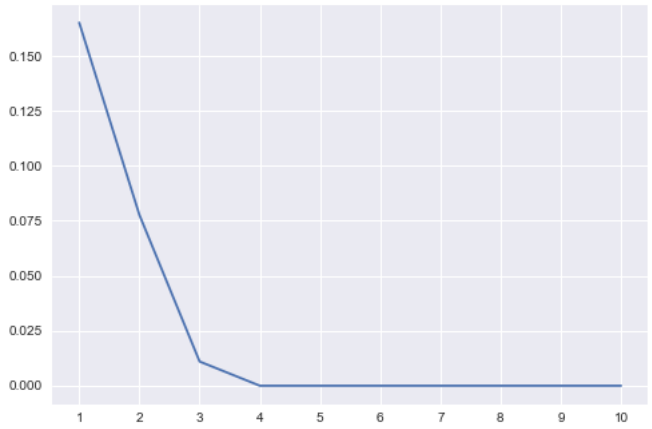


Figure 5.12: F1 measure (Y axis) versus sigma threshold values (X axis) curve.

5.3 TCN Autoencoder

The quantity of filters, in a TCN layer, is similar to the quantity of neurons in the LSTM network, meaning that the filter architecture is a very important hyperparameter. The learning rate is, again, another crucial aspect of the training. Another hyperparameter is the quantity of residual blocks, which will remain one because higher values seemed to worsen the training.

In the table C.6 (in the appendix C) the results of the dataset Zapcae can be observed, for a 21 row window. Note that the architecture represents the neurons from one side of the structure to the bottleneck. In figure 5.12 one can see the average F1 metric for each sigma used to draw the threshold, justifying the choice of the value of 1.

The best result for the zapcae dataset would be the model 8, with around 0.53 as f1 value and 0.49 for precision. The architecture have 6 layers overall (64, 16, 8 - 3 layers for the encoder, a bottleneck layer of 8 neurons' output, and then another 3 layers for decoder), and as more powerful networks were tried the metrics seem to get worse.

For the best model, the training loss through the training for all 100 epochs is shown in figure 5.13.

Concerning the individual tests, they all range between around 0.47 and 0.57 F1 score, being a more balanced model (corresponding metrics shown in tables C.40 to C.44). The same earlier example will be shown, the reduced calls and clients case, which the anomaly scores can be observed in figure 5.14 and respective confusion matrix in figure 5.15.

For the practical tests, just like the LSTM's zapcae model, both the healthy tests were accurately classified as healthy. The weekend anomaly was also signaled as healthy, which is wrong, while the week anomalous window was signaled as it should, an anomaly.

Regarding the testpow dataset, the model did not perform as good with the 21 window time frame, such can be observed in table C.7 (in the appendix C). The best average F1

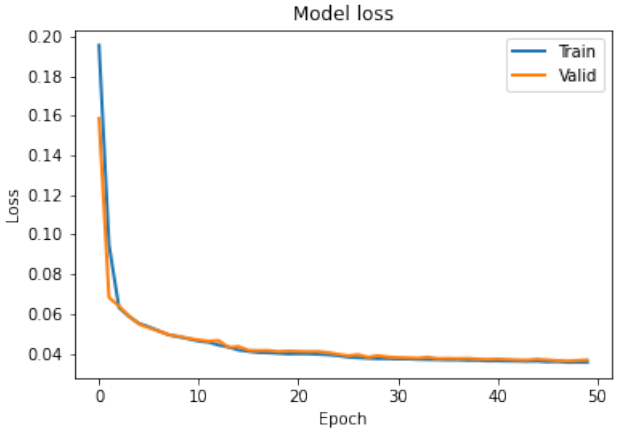


Figure 5.13: TCN AE best model's loss plotting through the epochs, while training for Zapcae dataset.

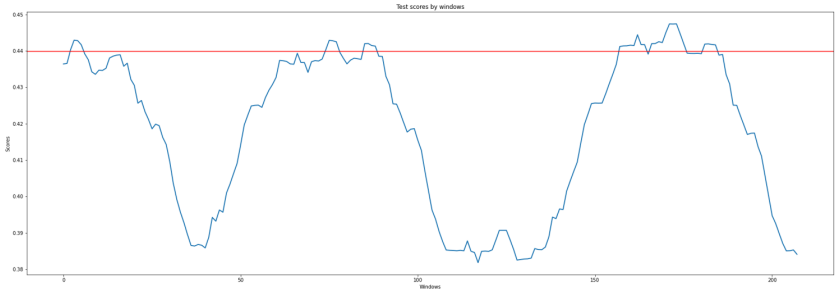


Figure 5.14: TCN AE anomaly scores through the lower success rates zapcae test time-series subset.

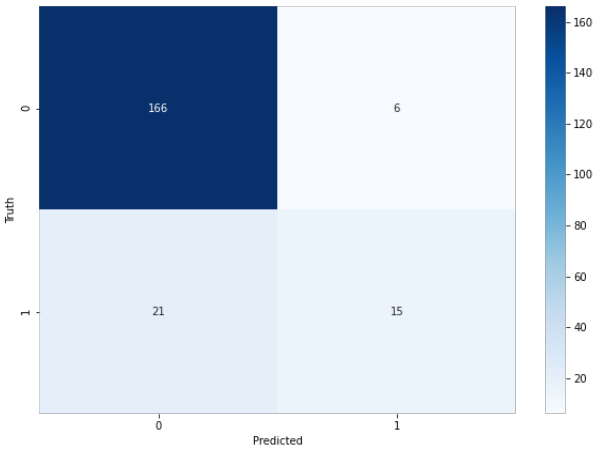


Figure 5.15: Confusion matrix for the lower success rates zapcae test in the TCN context.

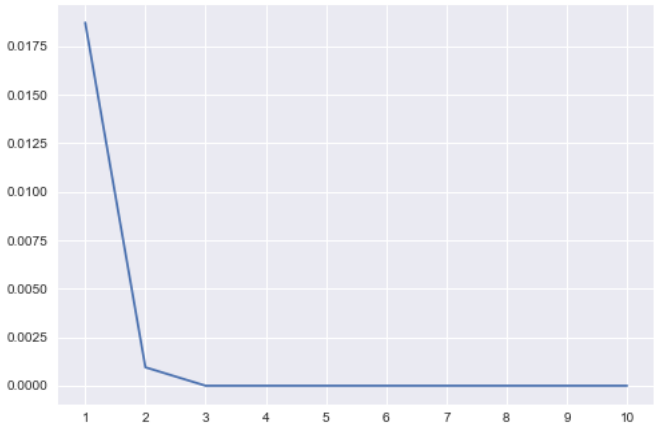


Figure 5.16: F1 measure (Y axis) versus sigma threshold values (X axis) curve.

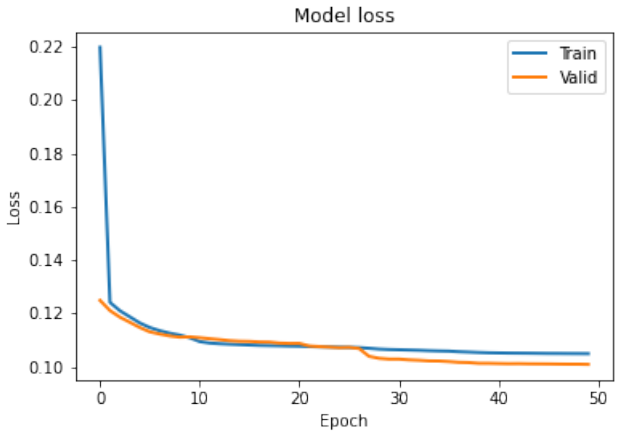


Figure 5.17: TCN AE best model's loss plotting through the epochs, while training for Testpow dataset.

value is around 0.0828 with an average precision of 0.0972, which is quite bad (model index 8).

The training loss through the training for all 100 epochs is shown in figure 5.13, revealing clear signs of underfitting.

As a result of the poorer performance of this network, using all the variables available, the anomaly score graphs are all identical. Moreover, the windows near the end of the subset (those that have the anomaly) actually scored considerably lower than the healthy ones. This is why the metrics are so low. As an example, I will show the scores related to high value variation over time (figure B.10), which can be seen in figure 5.18. This particular test got around 0.066 F1 score and 0.055 precision. The matrix can be seen in figure 5.19. While the model still can figure out many healthy windows, the quantity of true positives is very low and both the false positives and negatives got higher.

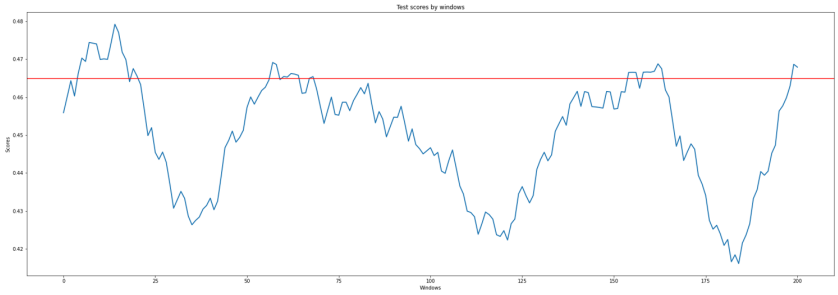


Figure 5.18: TCN AE anomaly scores through the high value variation testpow test time-series subset.

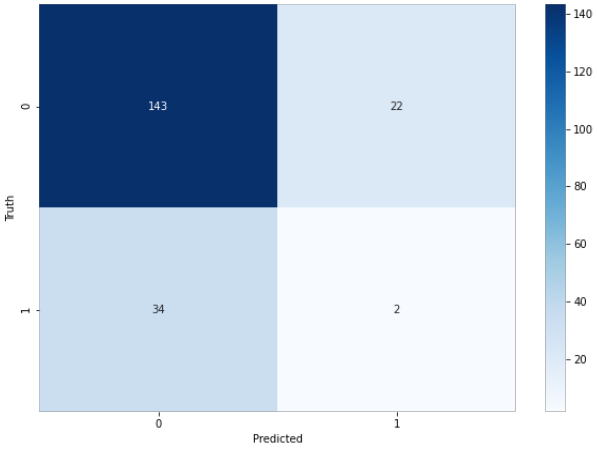


Figure 5.19: Confusion matrix for the high value variation testpow test in the TCN context.

All tests scored around these values, revealing no interesting pattern or specialization.

When feature selection was tried in this particular case, the scenario changed quite a bit, producing a much better predictor. The table C.8 in the appendix C show a more promising hypothesis (model index 1) that yielded 0.4 average F1 and 0.42 average precision. All tests' results can be seen from table C.45 to table C.50.

While the zero persistent value test got a 0 F1 (in table C.50) just like the previous model, the other scenarios hold much brighter results. In the many missing and high variation values' tests that ocurred only in the Downstream RXpower variable, got 0.4 and 0.33 F1 respectively (tables C.45 and C.48). On the other hand, the same tests that span across more 2 features, and the persistent value one, got between 0.55 and 0.6 F1 scores (tables C.46, C.49 and C.47). I show the best performing test anomaly scores as an example in figure 5.20, which is the persistent value scenario. Corresponding confusion matrix can be seen in figure 5.21.

Some models were also tried with the smaller window width of 7 (table C.9, in the appendix C) in pursuit of better results, and achieved an average F1 of around 0.37 and

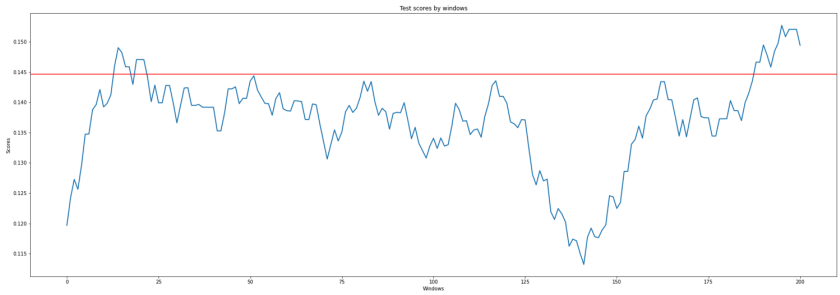


Figure 5.20: TCN AE anomaly scores through the persistent values testpow test time-series subset.

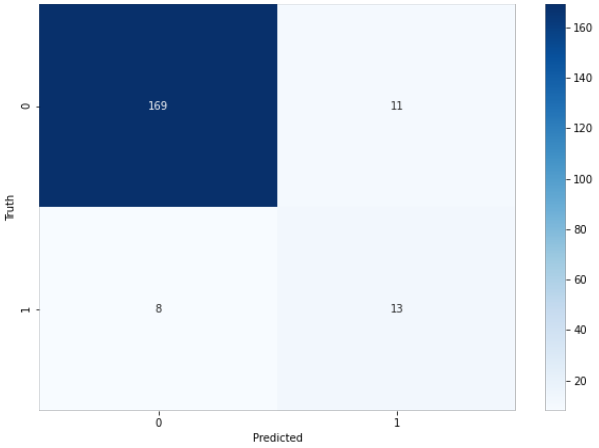


Figure 5.21: Confusion matrix for the persistent value testpow test in the TCN context.

0.293 as average precision (model index 3 in averages table C.9). Corresponding results for all tests can be seen from table C.51 to table C.56.

For the 7 width window best model, the training losses can also be observed, in figure 5.22. This one seems to have trained and adjusted rather well to the data.

Surprisingly, the metrics drastically changed with the 7 width window training. The anomalous windows seem to get a more decent score, which can be seen in figures 5.23 and 5.24, for each test figure B.8 and B.9, respectively (results found in tables C.51 and C.53). The healthy windows seem somewhat similar, but when one looks at the scores near the anomaly there is a different behavior. Both these scenarios' matrices can be observed in figures 5.25 and 5.26. The rest of the tests also scored similar metrics, with the exception of the many zeros one, which got an F1 of 0.09 and precision of 0.08.

Using the TCN model trained with feature selection, both week manual test windows were classified as healthy and the weekend was all signaled as anomalous.

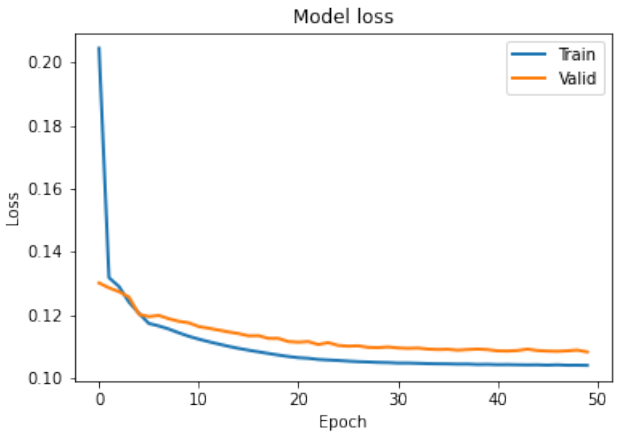


Figure 5.22: TCN AE best 7 width window model's loss plotting through the epochs, while training for Testpow dataset.

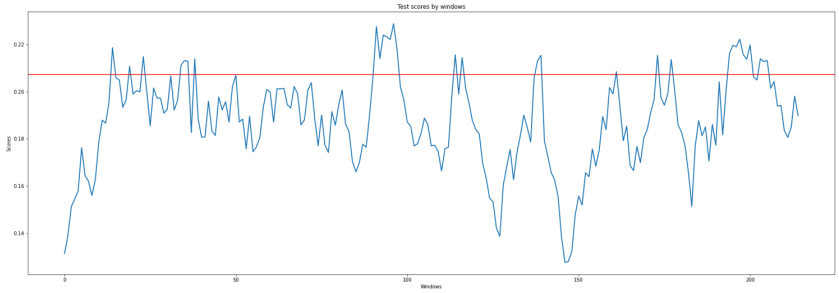


Figure 5.23: TCN AE, using window widths of 7, anomaly scores through the many missing values testpow test time-series subset.

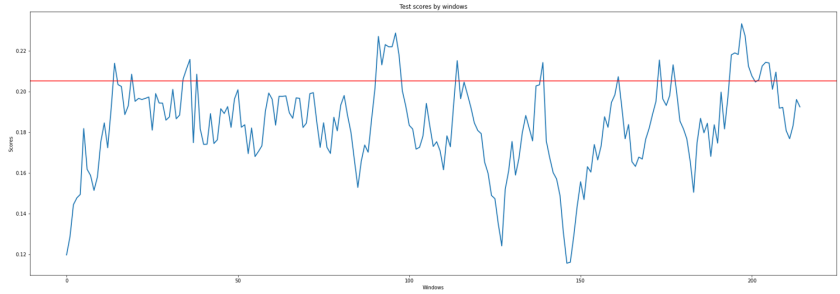


Figure 5.24: TCN AE, using window widths of 7, anomaly scores through the persistent value testpow test time-series subset.

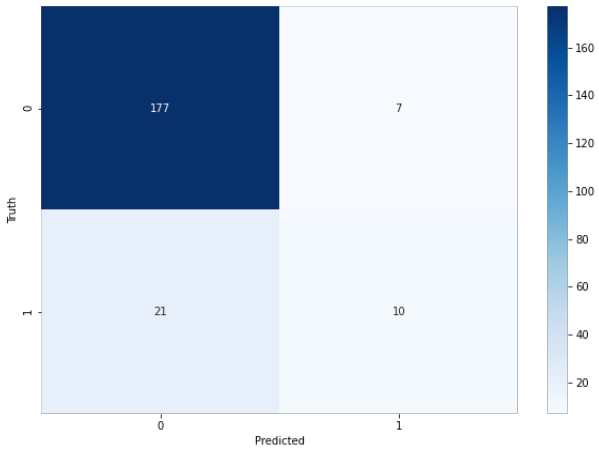


Figure 5.25: Confusion matrix for the many nulls testpow test in the TCN, 7 width window, context.

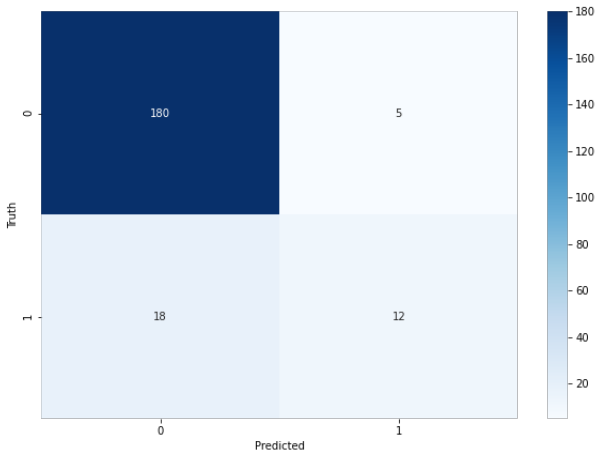


Figure 5.26: Confusion matrix for the persistent values testpow test in the TCN, 7 width window, context.

5.4 Boosting Autoencoder Ensemble

Since the best results were yielded by the LSTM, the BAE algorithm will be implemented using that network. For each model below, there are three chained autoencoders in the manner mentioned in the section 2.7, chapter 2, for 21 width windows.

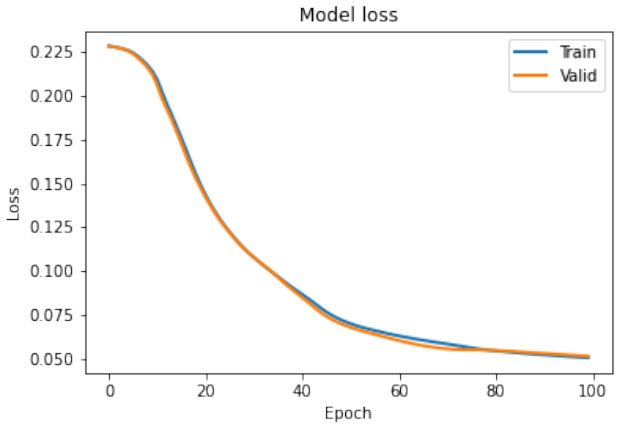
In table C.10 (in the appendix C) is shown the average results for the BAE LSTM, using the Zapcae dataset. Corresponding exhaustive tests' results are displayed in table C.57 to table C.61. It seems that the boosting algorithm did not improve the hypothesis in the Zapcae's case. The best model (index 1) got an incredible 0.92 F1 and 1.0 precision score for tests that involved a drop in success rates (tables C.57). But got a much worse F1 of 0.47 for reductions to the calls, clients or even drop rates, and 0.3 F1 for increased values of calls and clients. It did not seem to solve the unbalanced tests problem, continuing to performing overwhelmingly well in the some tests, while others not so much. This pattern seems to be the same for almost every LSTM model in the zapcae dataset.

Regarding the manual tests, both the healthy tests were correctly classified, while the only anomalous windows signaled as such was the week one. Although not ideal, 3 out of 4 windows were accurately predicted. It seems that the weekend is still a difficult generalization for the current hypothesis.

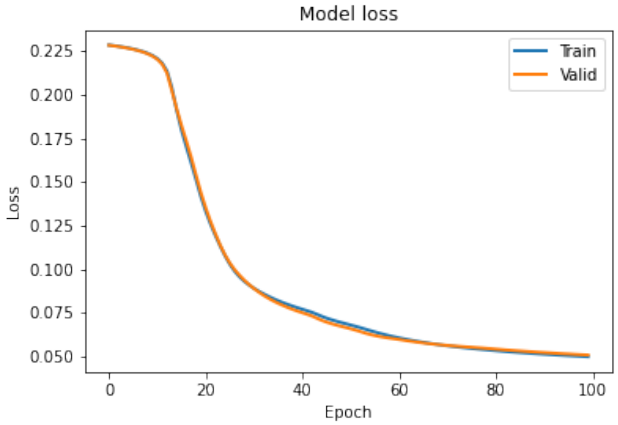
In table C.11 (in the appendix C) is shown the results for the BAE LSTM, using the Testpow dataset. Concerning individual tests' results can be observed from tables C.62 to C.67. When it comes to this dataset, there is an hypothesis that surprise us at first glance, the one with model index 0. It did get an average F1 of around 0.7, which is great compared to the previous models. But it is still not as great as it could be, because it also got a much higher average recall (around 0.9) than average precision (around 0.58). It means that there are still many false positives being generated, which we will see afterwards. Instead of one autoencoder, there were three autoencoders to train and its training losses shown in figure 5.27, which all seem a bit underfitted.

Curiously, the best performing test was the persistent value one, with around 0.82 F1, 1.0 recall, and around 0.7 precision (table C.64). But all the other tests performed similar to the one I am about to show, the many missing data (table C.62). It got around 0.67 F1, 0.87 recall and 0.55 precision. Both scores can be observed in figures 5.28 and 5.29, with its corresponding confusion matrices in figures 5.30 and 5.31. Since the anomaly was set at the end of the subset, it is predictable that the last anomaly scores are the ones above the threshold line, for higher metric values.

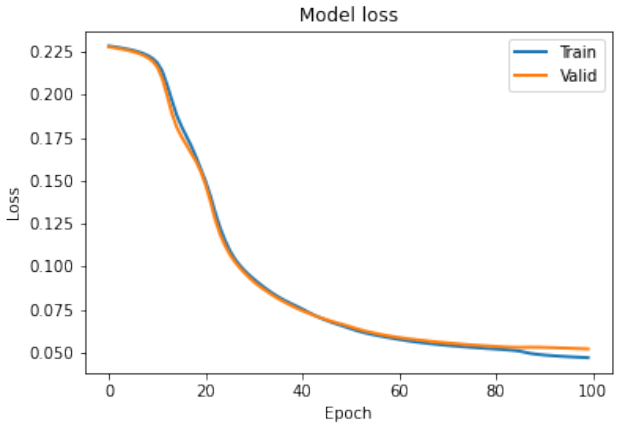
Regarding the 4 windows tests, all were signaled as anomalous, revealing that the much lower precision compared to the recall is not a good sign even if the F1 metric scores high.



(a) First AE training losses



(b) Second AE training losses



(c) Third AE training losses

Figure 5.27: Training losses for all the three autoencoders in the best model of BAE system.

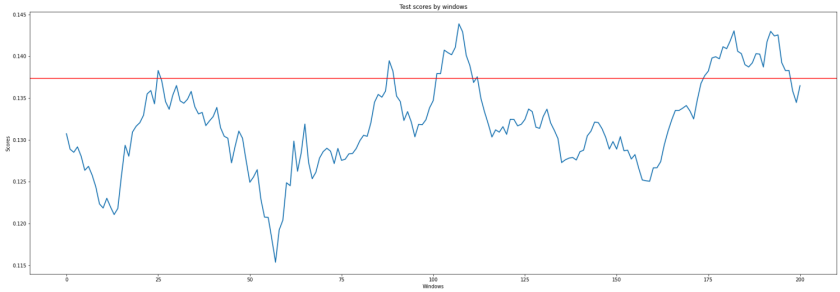


Figure 5.28: LSTM BAE anomaly scores through the persistent values testpow test time-series subset.

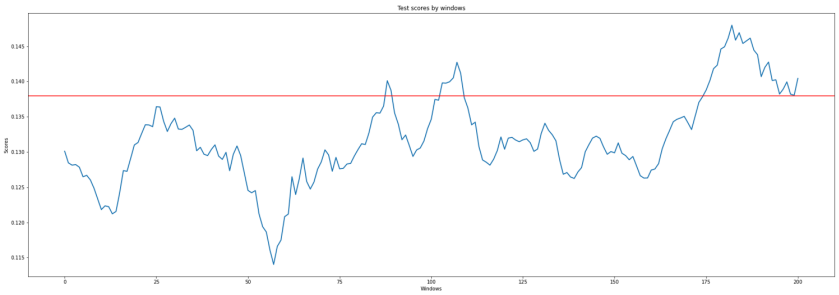


Figure 5.29: LSTM BAE anomaly scores through the missing values testpow test time-series subset.

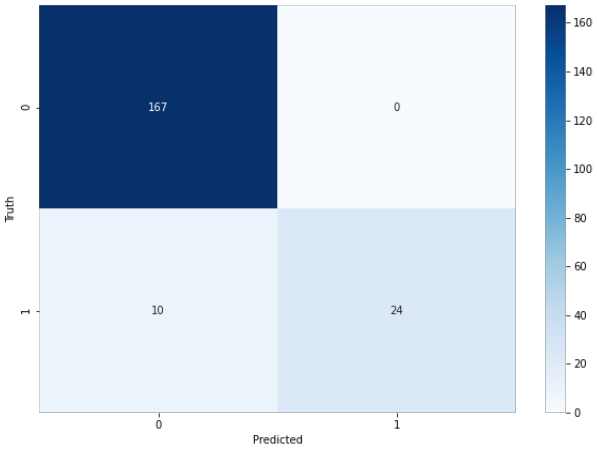


Figure 5.30: Confusion matrix for the persistent values testpow test in the LSTM BAE context.

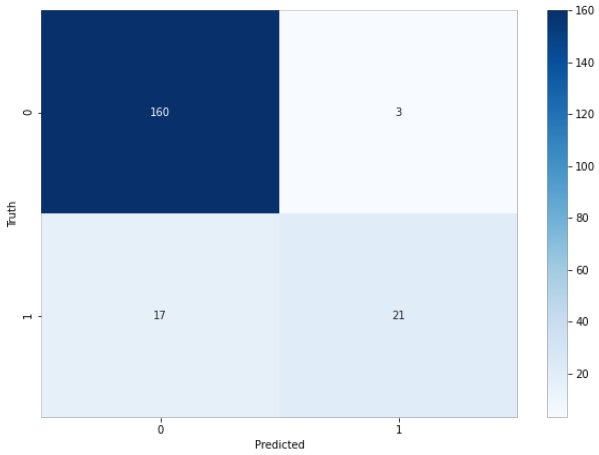


Figure 5.31: Confusion matrix for the missing values testpow test in the LSTM BAE context.

5.5 Manual tests table

For a more convenient way to see how the tests performed in the practical tests, the table 5.1 shows the predictions for each test.

Table 5.1: Results of the practical tests for each best hypothesis, for each model.

Model	Dataset	Week Healthy	Week Anomalous	Weekend Healthy	Weekend Anomalous
OCSVM	Zapcae	Anomalous	Anomalous	Anomalous	Anomalous
OCSVM	Testpow	Anomalous	Anomalous	Anomalous	Anomalous
LSTM AE	Zapcae	Healthy	Anomalous	Healthy	Healthy
LSTM AE	Testpow	Healthy	Healthy	Anomalous	Anomalous
TCN AE	Zapcae	Healthy	Anomalous	Healthy	Healthy
TCN AE	Testpow	Healthy	Healthy	Anomalous	Anomalous
LSTM BAE	Zapcae	Healthy	Anomalous	Healthy	Healthy
LSTM BAE	Testpow	Anomalous	Anomalous	Anomalous	Anomalous

5.6 Brief summary

This chapter covered the results for all implemented techniques. We have seen that the OCSVM performed quite poorly, signaling almost every instance as anomalous. For both the datasets it scored mostly close to a recall of 1, while the other metrics got a much lower value. While still not ideal, the LSTM Autoencoder yielded much better results for the Zapcae dataset, classifying correctly 3 out of 4 manual tests. On the other hand, when it came to the Testpow data, it showed to be very sensitive to the weekend, classifying week tests as healthy and weekend ones as anomalous. Surprisingly, the TCN

Autoencoder couldn't adjust as well to the Testpow dataset unless a smaller window width is used, while kept the same performance for the Zapcae one. The BAE method showed no improvements for both datasets. For this last case, the manual testing showed that the model still needs to improve in order to decently classify this kind of data.

6

Conclusions and future work

In this chapter the observed conclusions and future work aimed at improving the problem's solution will be presented.

The scope of this work was ambitious. When dealing with real data the results are not as good as one would expect. Nevertheless, the objective was to assess the performance of these methods on the provided time-series datasets. The work here reported is the first step towards building a framework of understanding that will, in the near future, bring added value to the company.

Most surprising of all was the behavior of the OCSVM, which did not adjust well to the data at all and kept signaling almost every situation as anomalous, even with a wide range of hyperparameter experimentation.

Regarding the Autoencoder method, results did vastly vary, although, when chosen the right hyperparameters, outcomes seem to get much closer to the wanted conduct. Despite the nature of the models, feature selection seemed to help in most of the scenarios, while a smaller window size seemed to achieve better results in the Testpow dataset specifically. It seems proven that, in a larger dataset, one day of data is not the best amount in order to maximize performance. The LSTM network seemed to adjust better to the data than the TCN, although, I believe further investigation needs to be done using these theoretically promising model.

Another aspect to be considered is that the Zapcae tests seem more realistic than the Testpow ones, since it was possible to identify true positives by hand. The design of the Testpow dataset's tests is not very subtle on purpose, in order to get results with obvious scenarios, and then, move on to more specific ones. This never happened due to the struggle necessary to put these models to work as they are.

Another interesting insight is that tests that span through many columns seem to yield better outcomes. This is good because actual hardware failures often affect many KPIs instead of just one variable.

For the tests done, the BAE algorithm did not seem to improve performance, requiring further experimentation.

Through the week the model seems a lot more robust than in a weekend situation, revealing that these models need more work put onto them in order to generalize the weekend pattern.

Between all models, the recommendation for which is more adequate would concern the LSTM Autoencoder. Regarding its Hyperparameters, those that performed the best (for each dataset) can be used, as is discussed in chapter 5. Although, the relation between Hyperparameters and their performance is not very consistent, meaning that when dealing with other problems and datasets, it is worth to make some experimentations instead of conforming to one hypothesis right away. Both the use of feature selection and a smaller time window width are also worth to try because it might significantly improve the performance of the model.

At the end, a prototype that span through all the necessary steps in order to achieve this recommendation was developed. These steps are the preprocess of the inputs and the multiple hypothesis search, which includes preparation and application of adequate test cases, training of the model and extraction of relevant metrics.

Although there are ways to deal with low quality data (many missing values), companies need to invest more into storing, cataloguing, and governing the data as best as possible, or as expenditures allow.

In a corporate environment, the main goal is to develop some type of alarm or notification system to alert a technical team that would need to follow upon said system. These methods are still not mature enough for production, for there are too many false positives being produced, which would create distrust by the technicians and such techniques would be considered useless very quickly.

Instead of dismissing these algorithms altogether, more work should be put into them. Developing alarm systems case by case is not scalable. The end-goal of developing methods for unsupervised anomaly detection that can learn on their own shows merit. It was already demonstrated they work on some use-cases, shown in chapter 2. Getting such algorithms into a production ready state will require additional investigation.

Future work consists of refining current methods and exploring further alternatives for production-ready algorithms.

Bibliography

A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way. (2018). https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. Accessed: 2021-01-29. (Cit. on p. 13).

Amer, M., Goldstein, M., & Abdennadher, S. (2013). Enhancing one-class support vector machines for unsupervised anomaly detection. In *Odd '13*. (Cit. on p. 7).

Andiojaya, A., & Demirhan, H. (2019). A bagging algorithm for the imputation of missing values in time series. *Expert Syst. Appl.*, *129*, 10–26. (Cit. on p. 23).

Baharani, M., Furgurson, S., Parkhideh, B., & Tabkhi, H. (2020). Atcn: Agile temporal convolutional networks for processing of time series on edge. *ArXiv*, *abs/2011.05260*. (Cit. on p. 8).

Bai, S., Kolter, J. Z., & Koltun, V. (2018). An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *ArXiv*, *abs/1803.01271*. (Cit. on pp. 14, 16).

Berglind, J. (2019). Temporal convolutional networks for forecasting patient volumes in digital healthcare. (Cit. on p. 14).

Bontemps, L., Cao, V. L., McDermott, J., & Le-Khac, N.-A. (2016). Collective anomaly detection based on long short-term memory recurrent neural networks. *ArXiv*, *abs/1703.09752*. (Cit. on pp. 8, 12).

Bounsiar, A., & Madden, M. G. (2014). Kernels for one-class support vector machines. In *2014 international conference on information science applications (icisa)* (pp. 1–4). doi:10.1109/ICISA.2014.6847419. (Cit. on p. 7)

Breunig, M., Kriegel, H., Ng, R., & Sander, J. (2000). Lof: Identifying density-based local outliers. In *Sigmod '00*. (Cit. on p. 6).

Brownlee, J. (2020). Why one-hot encode data in machine learning? Retrieved from https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/. ((accessed: 08.01.2021)). (Cit. on p. 36)

Chalapathy, R., & Chawla, S. (2019). Deep learning for anomaly detection: A survey. doi:https://arxiv.org/pdf/1901.03407.pdf#page=26&zoom=100,96,524. (Cit. on p. 6)

Chen, X., Li, B., Proietti, R., Zhu, Z., & Yoo, S. (2019). Self-taught anomaly detection with hybrid unsupervised/supervised machine learning in optical networks. *Journal of Lightwave Technology*, *37*, 1742–1749. (Cit. on p. 6).

Cheng, Y., Xu, Y., Zhong, H., & Liu, Y. (2019). Hs-tcn: A semi-supervised hierarchical stacking temporal convolutional network for anomaly detection in iot. *2019 IEEE 38th International Performance Computing and Communications Conference (IPCCC)*, 1–7. (Cit. on p. 8).

Duggimpudi, M. B., Abbady, S., Chen, J., & Raghavan, V. V. (2019). Spatio-temporal outlier detection algorithms based on computing behavioral outlierness factor. *Data Knowl. Eng.*, *122*, 1–24. (Cit. on p. 6).

Goldstein, M., & Uchida, S. (2016). A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. *PLOS ONE*, *11*(4), 1–31. doi:10.1371/journal.pone.0152173. (Cit. on p. 2)

Grubbs, F. E. (1969). Procedures for detecting outlying observations in samples. doi:https://scinapse.io/papers/1978239142. (Cit. on pp. 1, 6)

Gupta, M., Gao, J., Aggarwal, C., & Han, J. (2014). Outlier detection for temporal data: A survey. *IEEE Transactions on Knowledge and Data Engineering*, *26*, 2250–2267. (Cit. on p. 9).

Hawkins, D. (1980). *Identification of outliers*. Chapman and Hall. (Cit. on p. 2).

He, Y., & Zhao, J. (2019). Temporal convolutional networks for anomaly detection in time series. *Journal of Physics: Conference Series*, *1213*, 042050. doi:10.1088/1742-6596/1213/4/042050. (Cit. on p. 8)

Hodge, V. J., & Austin, J. (2004). A survey of outlier detection methodologies. *Kluwer Academic Publishers*. doi:https://www-users.cs.york.ac.uk/vicky/myPapers/Hodge+Austin_OutlierDetection_AIRE381.pdf. (Cit. on pp. 2, 6)

How LSTM networks solve the problem of vanishing gradients. (n.d.). https://medium.com/datadriveninvestor/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577. Accessed: 2021-01-29. (Cit. on p. 12).

Karadayi, Y., Aydin, M. N., & Öğrencí, A. S. (2020). A hybrid deep learning framework for unsupervised anomaly detection in multivariate spatio-temporal data. *Applied Sciences*, *10*, 5191. (Cit. on pp. 9, 11, 13, 14).

Kieu, T., Yang, B., Guo, C., & Jensen, C. (2019). Outlier detection for time series with recurrent autoencoder ensembles. (pp. 2725–2732). doi:10.24963/ijcai.2019/378. (Cit. on p. 18)

Lara-Benítez, P., Carranza-García, M., Luna-Romera, J. M., & Riquelme, J. (2020). Temporal convolutional networks applied to energy-related time series forecasting. *Applied Sciences*, *10*, 2322. (Cit. on p. 8).

Lin, S., Clark, R., Birke, R., Schönborn, S., Trigoni, N., & Roberts, S. (2020). Anomaly detection for time series using vae-lstm hybrid model. *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 4322–4326. (Cit. on pp. 2, 10).

Lorbeer, B., & Botler, M. (2020). Anomaly detection with partitioning overfitting autoencoder ensembles. *ArXiv*, *abs/2009.02755*. (Cit. on p. 18).

Malhotra, P., Ramakrishnan, A., Anand, G., Vig, L., Agarwal, P., & Shroff, G. (2016). Lstm-based encoder-decoder for multi-sensor anomaly detection. *ArXiv*, *abs/1607.00148*. (Cit. on p. 12).

Meng, C., Jiang, X., Wei, X. M., & Wei, T. (2020). A time convolutional network based outlier detection for multidimensional time series in cyber-physical-social systems. *IEEE Access*, *8*, 74933–74942. (Cit. on pp. 14, 15).

Nam, H.-s., Jeong, Y.-K., & Park, J. (2020). An anomaly detection scheme based on lstm autoencoder for energy management. *2020 International Conference on Information and Communication Technology Convergence (ICTC)*, 1445–1447. (Cit. on p. 10).

Nascimento, E., de L. Tavares, O., & Souza, A. (2015). A cluster-based algorithm for anomaly detection in time series using mahalanobis distance. (Cit. on p. 6).

Nguyen, H. D., Tran, K., Thomassey, S., & Hamad, M. (2020). Forecasting and anomaly detection approaches using lstm and lstm autoencoder techniques with the applications in supply chain management. *International Journal of Information Management*, 102282. (Cit. on p. 12).

Nyuytiymbiy, K. (2020). Parameters and hyperparameters in machine learning and deep learning. Retrieved from https://towardsdatascience.com/parameters-and-hyperparameters-aa609601a9ac. ((accessed: 28.07.2021)). (Cit. on p. 19)

of Sydney, R. C. U., Centre, C. M. C. R., Institute, S. C. Q. C. R., & Hbku. (2019). Deep learning for anomaly detection: A survey. (Cit. on pp. 8, 10).

Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., ... Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. *ArXiv*, *abs/1609.03499*. (Cit. on p. 15).

Pang, G., Shen, C., Cao, L., & Hengel, A. V. D. (2020). Deep learning for anomaly detection: A review. doi:https://arxiv.org/pdf/2007.02500.pdf. (Cit. on p. 6)

Pereira, J. (2018). Unsupervised anomaly detection in time series data using deep learning. (Cit. on p. 8).

Poor, P. (2019). Historical overview of maintenance management strategies: Development from breakdown maintenance to predictive maintenance in accordance with four industrial revolutions. *ResearchGate*. doi:https://www.researchgate.net/publication/335444202_Historical_Overview_of_Maintenance_Management_Strategies_Development_from_Breakdown_Maintenance_to_Predictive_Maintenance_in_Accordance_with_Four_Industrial_Revolutions. (Cit. on p. 1)

Radford, B. J., Apolonio, L. M., Trias, A. J., & Simpson, J. A. (2018). Network traffic anomaly detection using recurrent neural networks. *ArXiv*, *abs/1803.10769*. (Cit. on p. 10).

Rasheed, W., & Tang, T. (2020). Anomaly detection of moderate traumatic brain injury using auto-regularized multi-instance one-class svm. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, *28*, 83–93. (Cit. on p. 7).

Russo, S., Disch, A., Blumensaat, F., & Villez, K. (2020). Anomaly detection using deep autoencoders for in-situ wastewater systems monitoring data. *ArXiv*, *abs/2002.03843*. (Cit. on pp. 10, 13).

Sainburg, T., Thielk, M., & Gentner, T. (2020). Latent space visualization, characterization, and generation of diverse vocal communication signals. *bioRxiv*. (Cit. on p. 10).

Sarvari, H., Domeniconi, C., Prenkaj, B., & Stilo, G. (2019). Unsupervised boosting-based autoencoder ensembles for outlier detection. arXiv: 1910.09754 [cs.LG]. (Cit. on p. 18)

Saumya, S., & Singh, J. (2020). Spam review detection using lstm autoencoder: An unsupervised approach. *Electronic Commerce Research*, 1–21. (Cit. on p. 12).

Tang, Y., Zhao, L., Zhang, S., Gong, C., Li, G., & Yang, J. (2020). Integrating prediction and reconstruction for anomaly detection. *Pattern Recognition Letters*, *129*, 123–130. doi:https://doi.org/10.1016/j.patrec.2019.11.024. (Cit. on p. 10)

Thill, M., Däubener, S., Konen, W., & Bäck, T. (2019). Anomaly detection in electrocardiogram readings with stacked lstm networks. In *Itat*. (Cit. on p. 8).

Thill, M., Konen, W., & Bäck, T. (2020). Time series encodings with temporal convolutional networks. In *Bioma*. (Cit. on pp. 14, 17).

V., B., & T., L. (1994). *Outliers in statistical data* (Third). J. Wiley & Sons. (Cit. on pp. 1, 6).

Wan, R., Mei, S., Wang, J., Liu, M., & Yang, F. (2019). Multivariate temporal convolutional network: A deep neural networks approach for multivariate time series forecasting. *Electronics*, *8*, 876. (Cit. on p. 14).

Wang, H., Bah, M. J., & Hahhad, M. (2019). Progress in outlier detection techniques: A survey. doi:https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8786096. (Cit. on p. 6)

Wang, Y., Liu, Z., Hu, D., & Zhang, M. (2019). Multivariate time series prediction based on optimized temporal convolutional networks with stacked auto-encoders. In *Acml*. (Cit. on p. 8).

Wen, T., & Keyes, R. (2019). Time series anomaly detection using convolutional neural networks and transfer learning. *ArXiv*, *abs/1905.13628*. (Cit. on pp. 13, 14).

Wu, G., Zhao, Z., Fu, G., Wang, H., Wang, Y., Wang, Z., ... Huang, L. (2019). A fast knn-based approach for time sensitive anomaly detection over data streams. In *Iccs*. (Cit. on p. 6).

Xu, J., & Duraisamy, K. (2019). *Multi-level convolutional autoencoder networks for parametric prediction of spatio-temporal dynamics*. (Cit. on p. 15).

Yaacob, A. H., Tan, I. K., Chien, S. F., & Tan, H. K. (2010). Arima based network anomaly detection. doi:https://ieeexplore.ieee.org/document/5437603. (Cit. on p. 6)

Yao, C., Ma, X., Chen, B., Zhao, X., & Bai, G. (2019). Distribution forest: An anomaly detection method based on isolation forest. In *Appt*. (Cit. on p. 7).

You, C., Wang, Q., & Sun, C. (2019). Sbilsan:stacked bidirectional self-attention lstm network for anomaly detection and diagnosis from system logs. (Cit. on pp. 8, 12, 39).

Yu, Q., Jibin, L., & Jiang, L. (2016). An improved arima-based traffic anomaly detection algorithm for wireless sensor networks. doi:https://journals.sagepub.com/doi/full/10.1155/2016/9653230. (Cit. on p. 6)

Zhikun, H., Zhiwen, C., Weihua, G., & Bin, J. (2013). Adaptive pca based fault diagnosis scheme in imperial smelting process. *2013 10th IEEE International Conference on Control and Automation (ICCA)*, 1447–1453. (Cit. on p. 6).

Zhou, C., & Paffenroth, R. C. (2017). Anomaly detection with robust deep autoencoders. doi:https://dl.acm.org/doi/10.1145/3097983.3098052. (Cit. on p. 6)

A

Appendix 1 - Exploratory data analysis

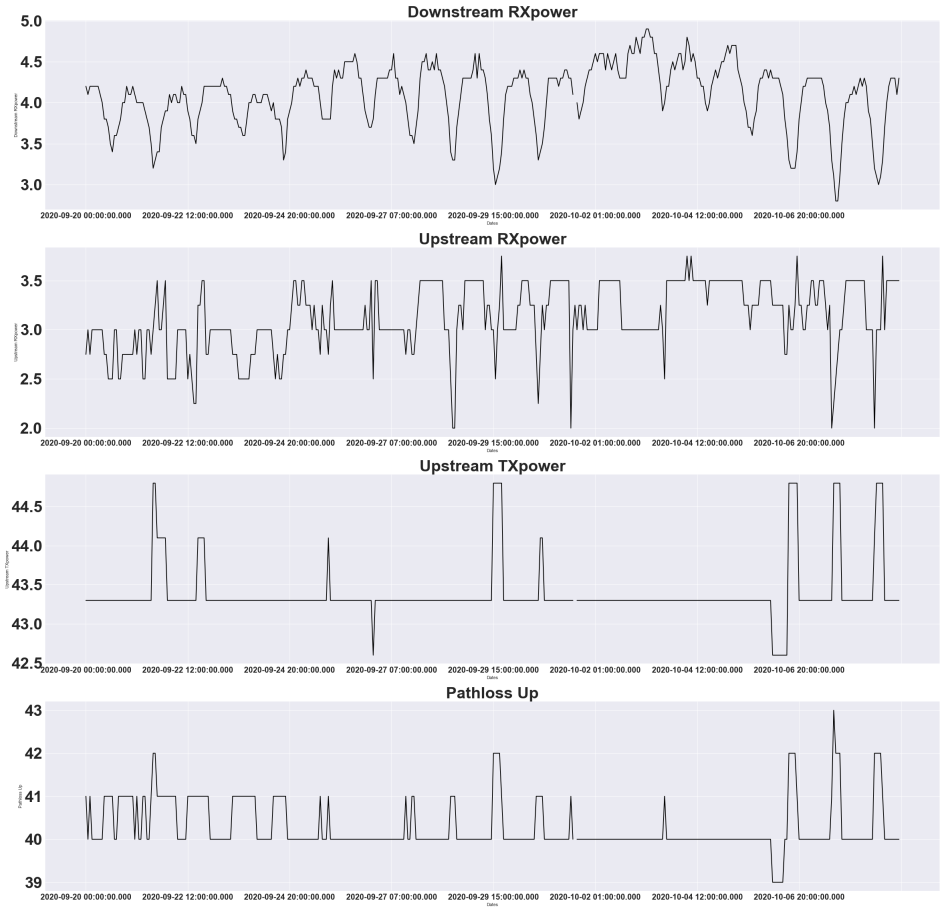


Figure A.1: One device example through time for Downstream RXpower, Upstream RXpower, Upstream TXpower and Pathloss Up

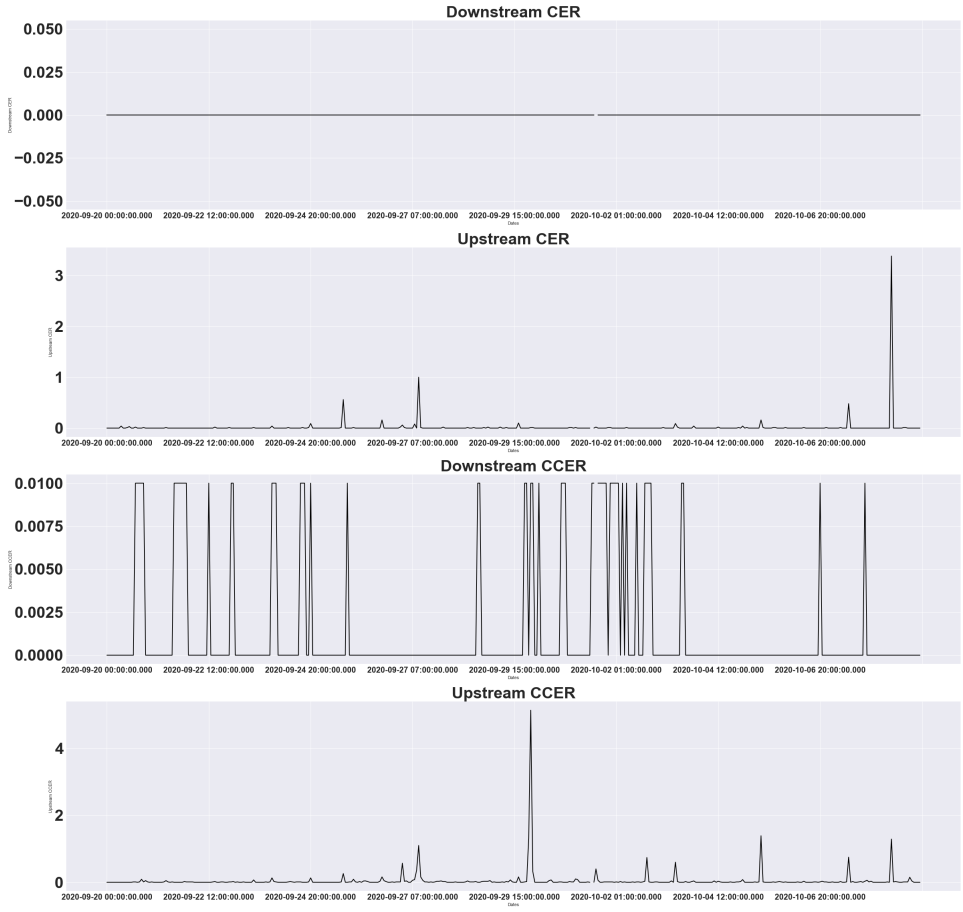


Figure A.2: One device example through time for Downstream CER, Upstream CER, Downstream CCER and Upstream CCER

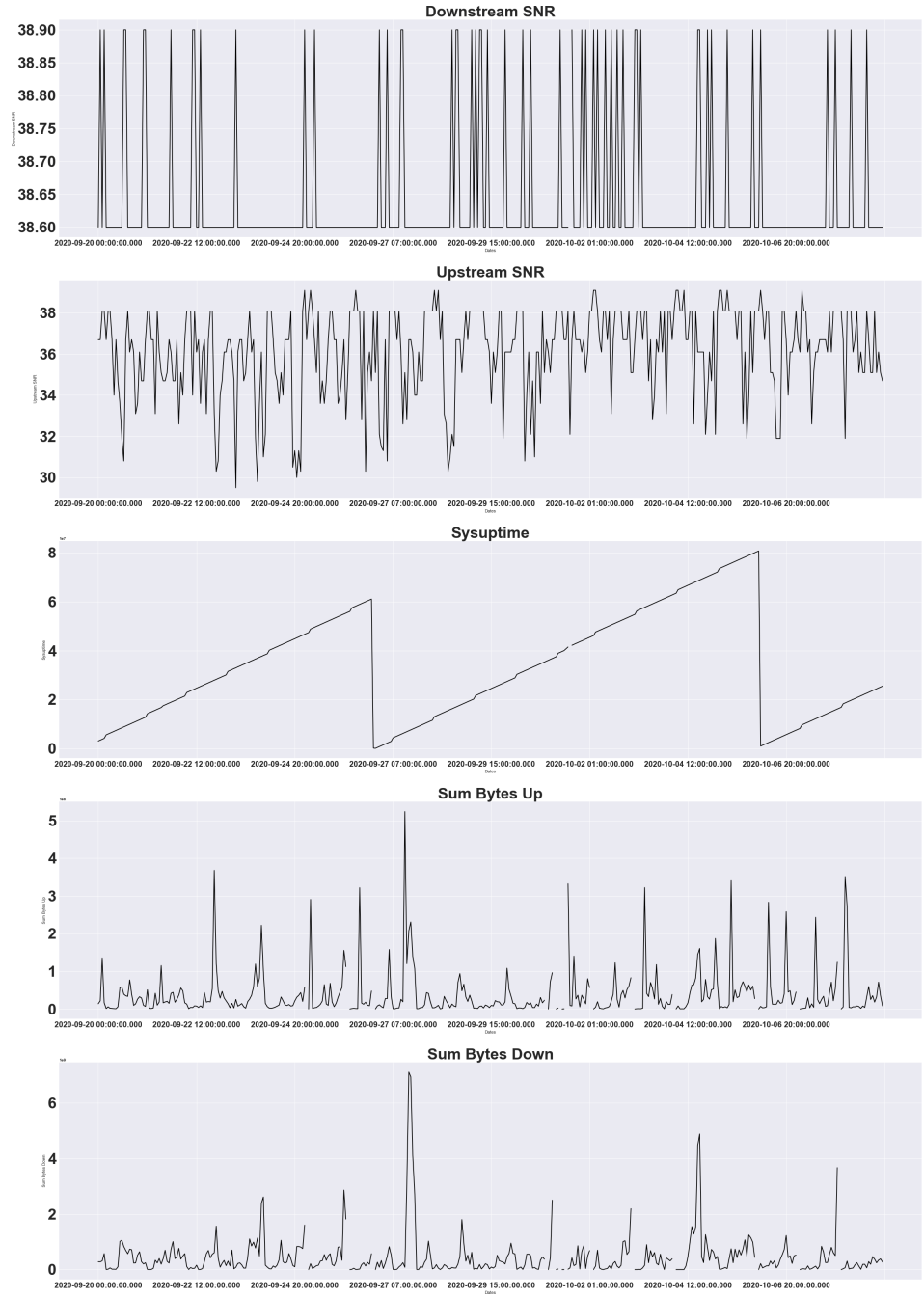
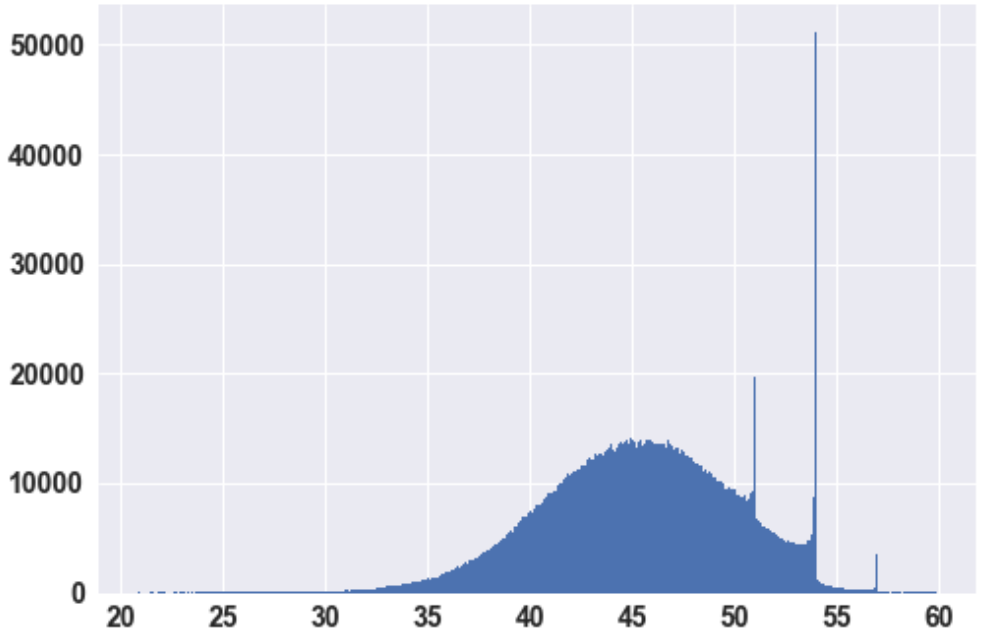
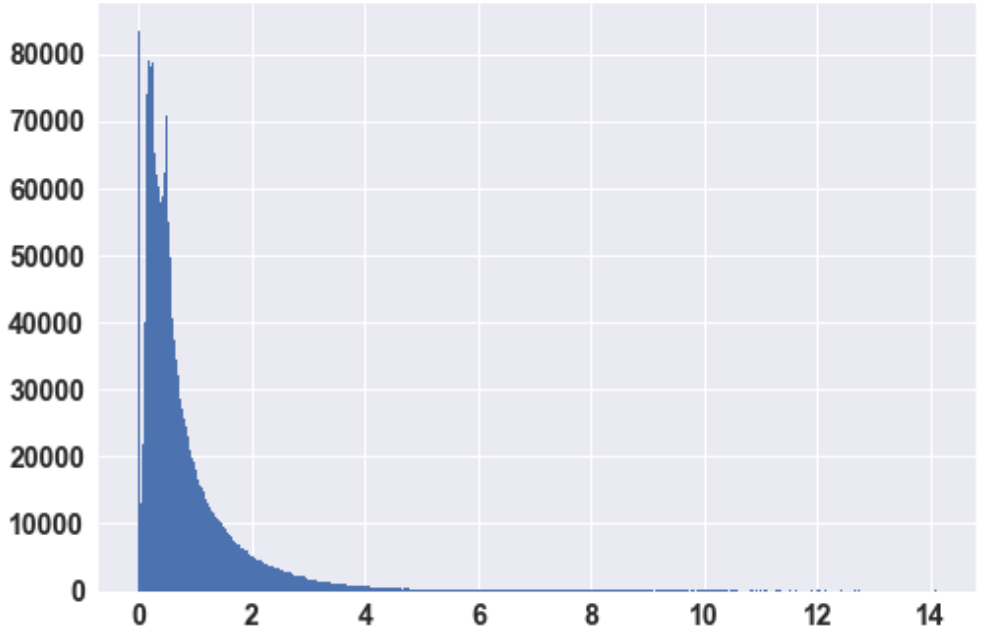


Figure A.3: One device example through time for Downstream SNR, Upstream SNR, Sysuptime, Sum Bytes Up, Sum Bytes Down

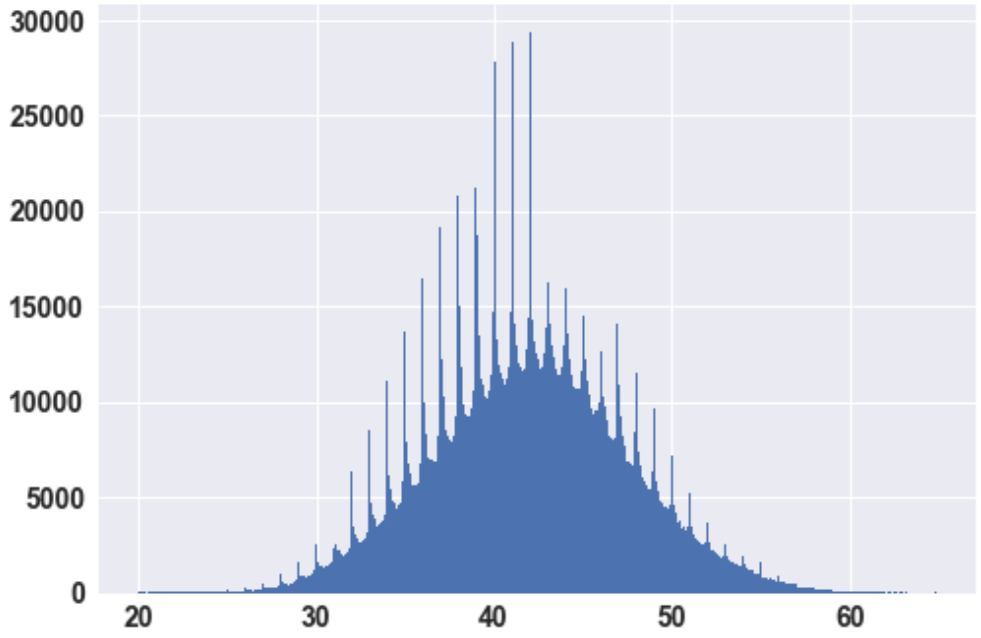


(a) Mean distribution of Upstream TXpower column

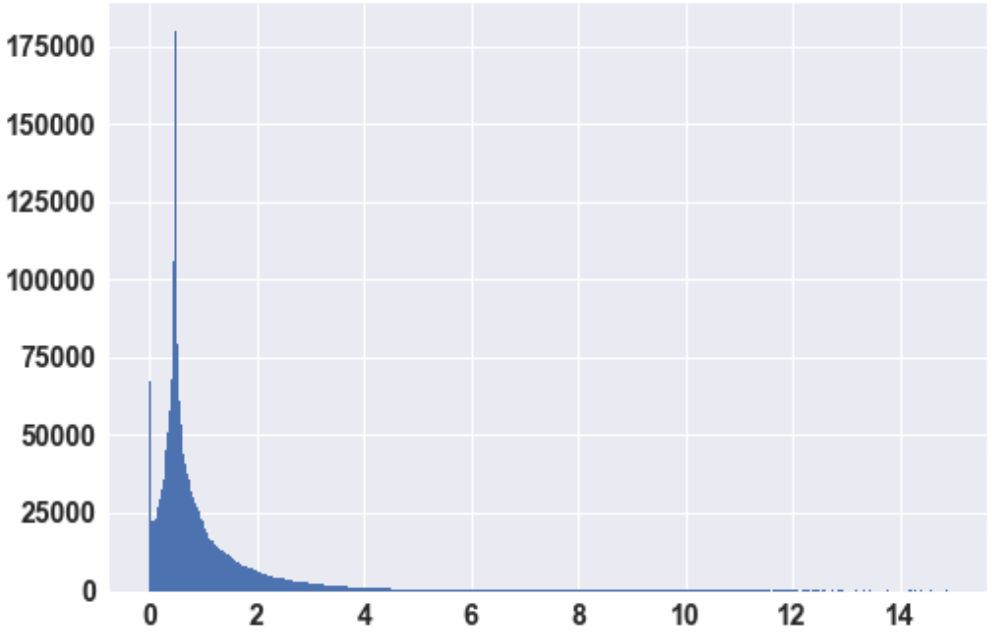


(b) Standard deviation distribution of Upstream TXpower column

Figure A.4: Mean and standard deviation distribution for Upstream TXpower

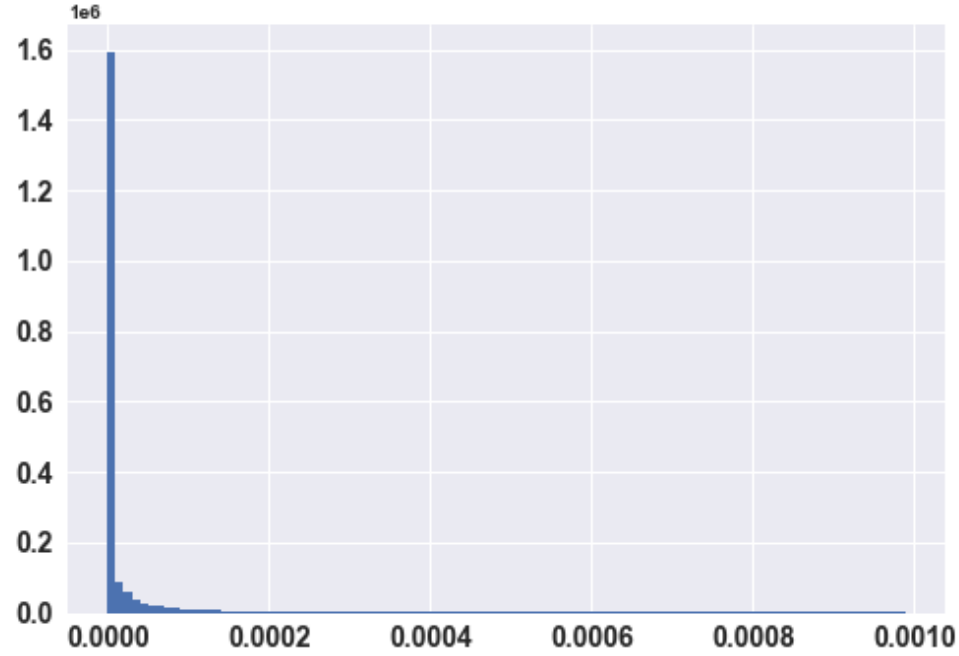


(a) Mean distribution of Pathloss Up column

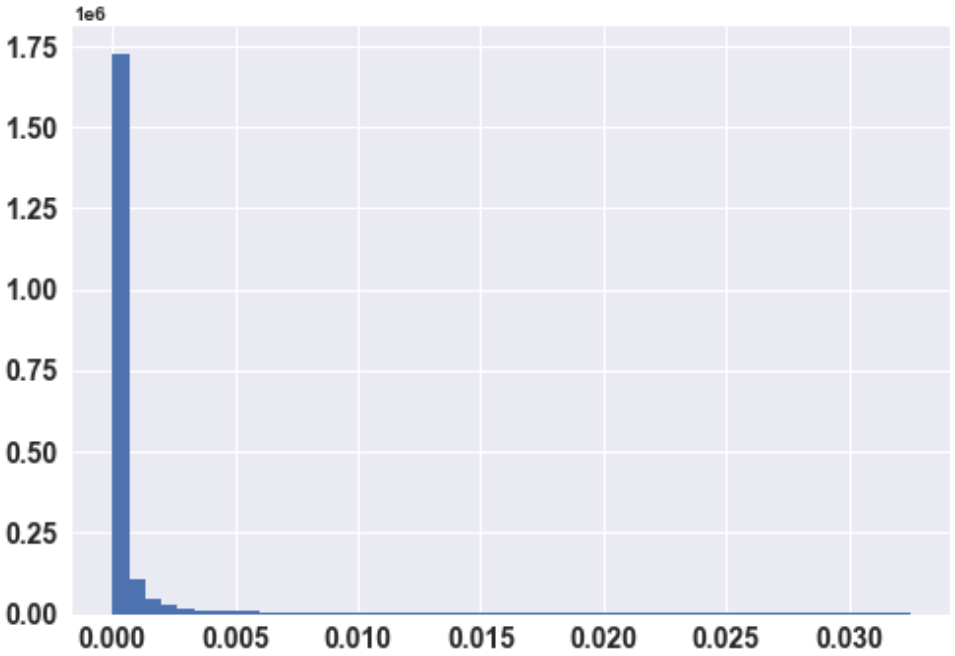


(b) Standard deviation distribution of Pathloss Up column

Figure A.5: Mean and standard deviation distribution for Pathloss Up

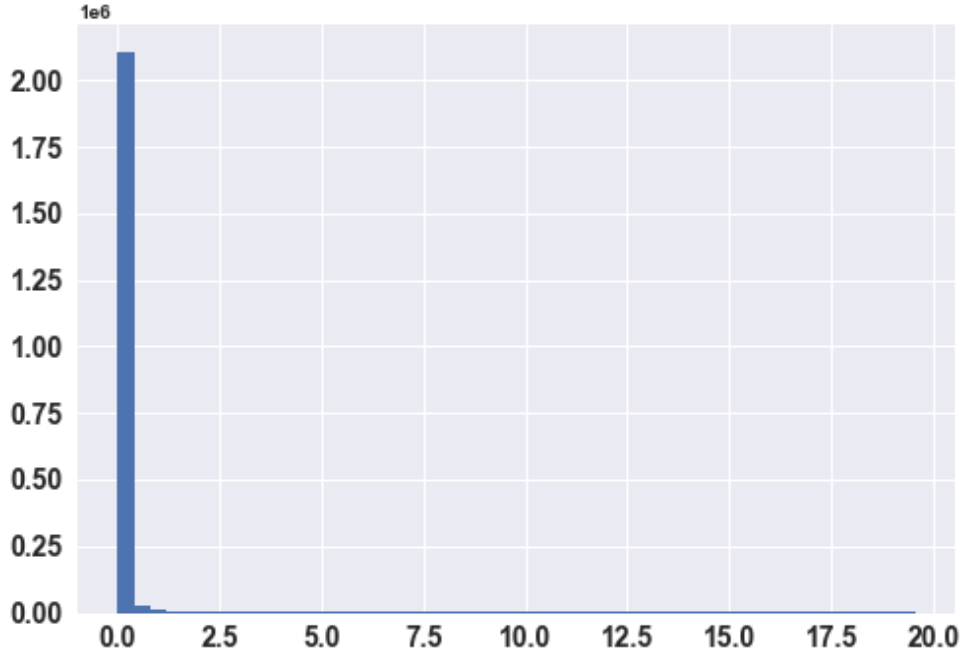


(a) Mean distribution of Downstream CER column(Y Axis: scientific notation)

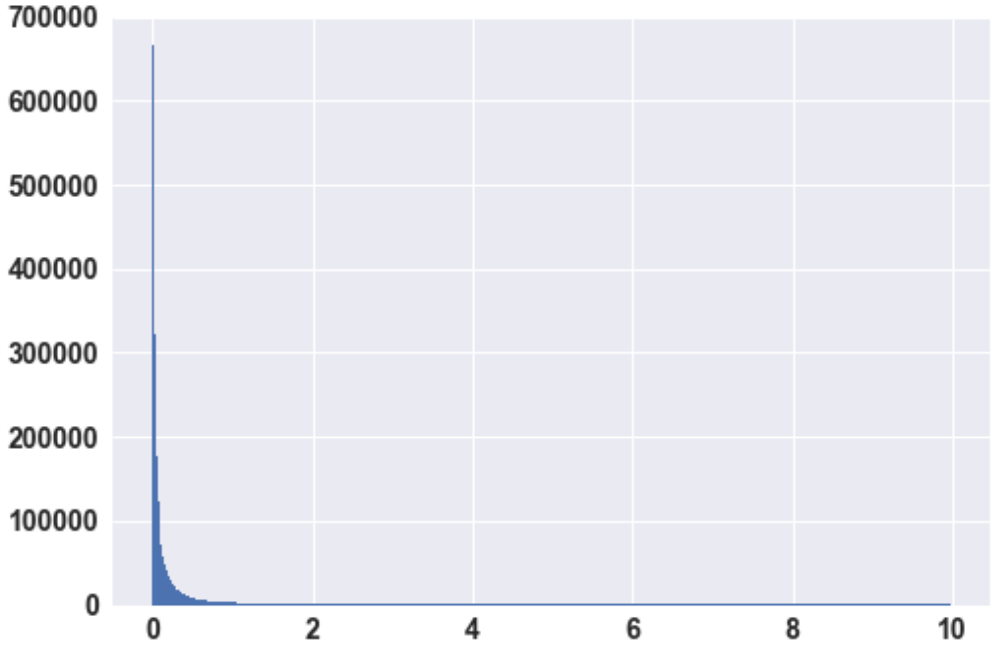


(b) Standard deviation distribution of Downstream CER column(Y Axis: scientific notation)

Figure A.6: Mean and standard deviation distribution for Downstream CER

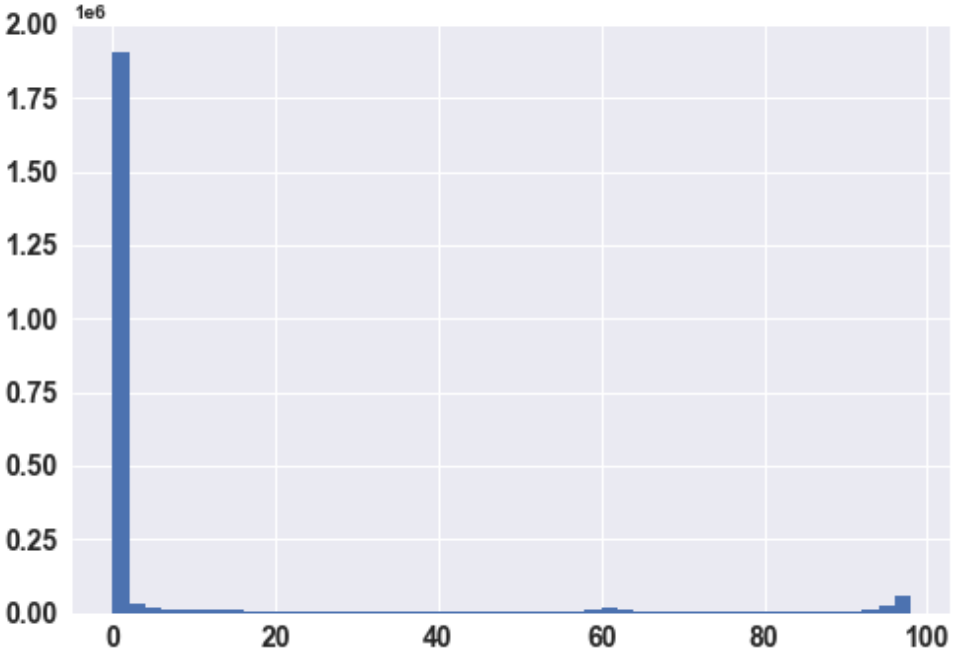


(a) Mean distribution of Upstream CER column(Y Axis: scientific notation)

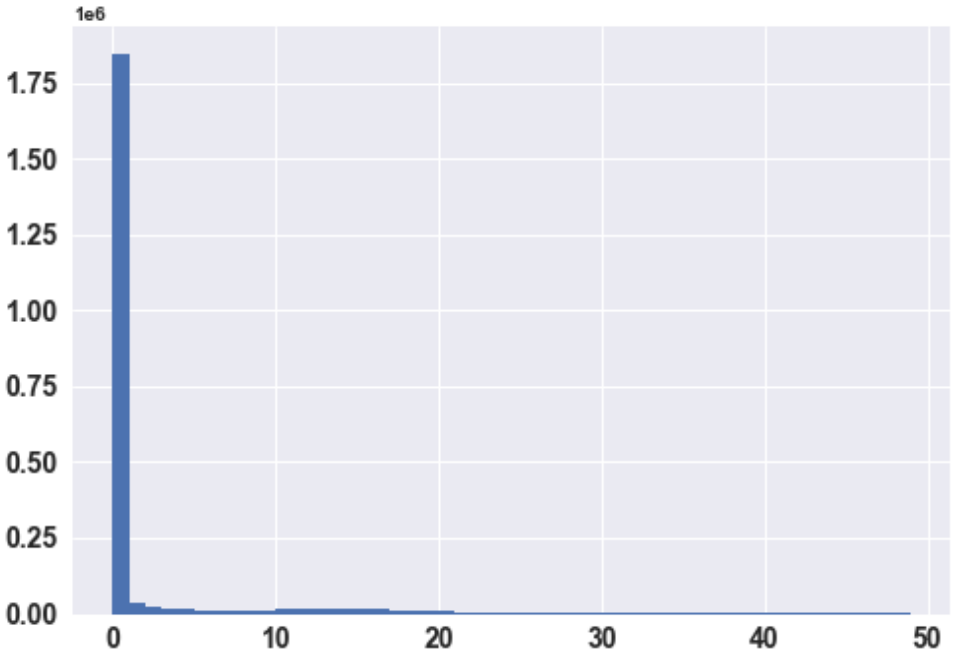


(b) Standard deviation distribution of Upstream CER column(Y Axis: scientific notation)

Figure A.7: Mean and standard deviation distribution for Upstream CER

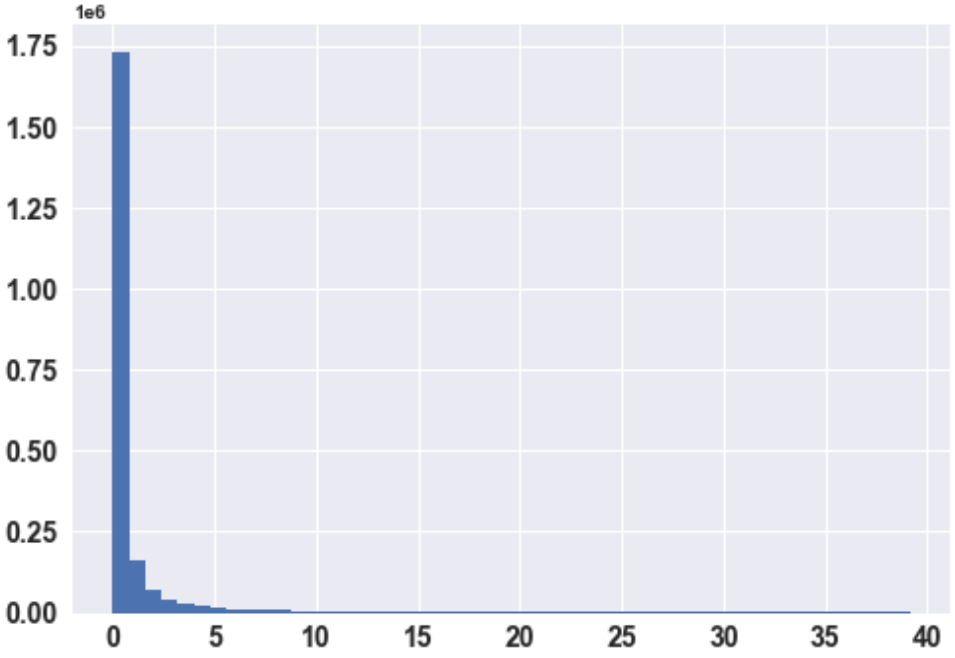


(a) Mean distribution of Downstream CCER column(Y Axis: scientific notation)

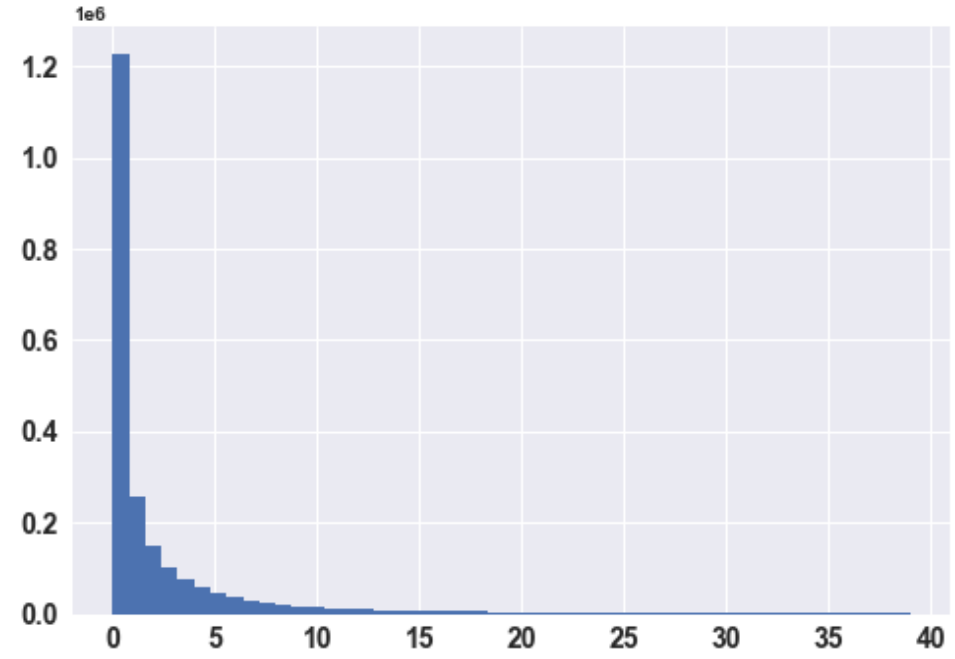


(b) Standard deviation distribution of Downstream CCER column(Y Axis: scientific notation)

Figure A.8: Mean and standard deviation distribution for Downstream CCER

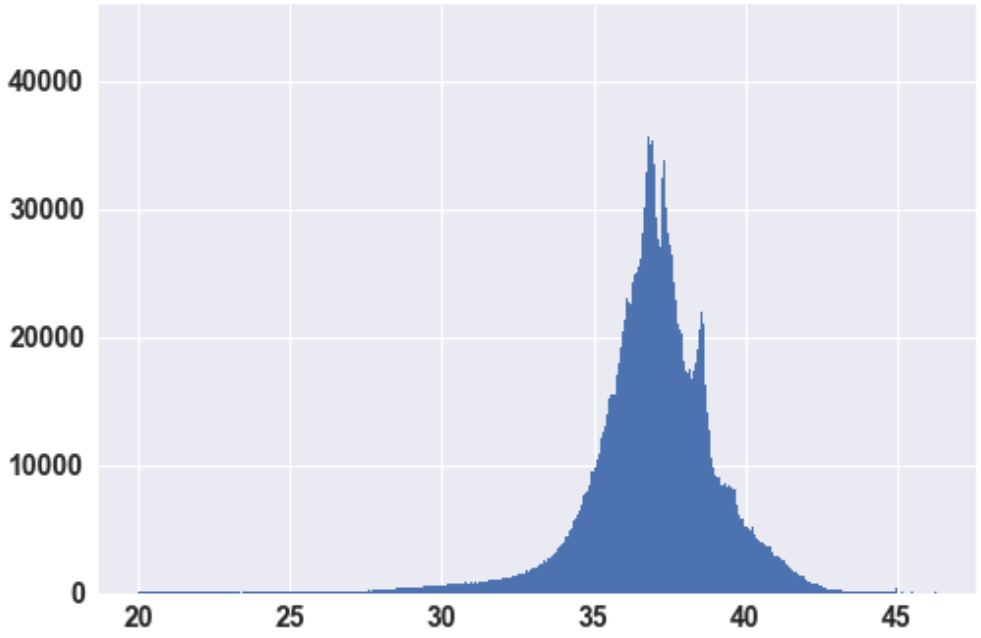


(a) Mean distribution of Upstream CCER column(Y Axis: scientific notation)

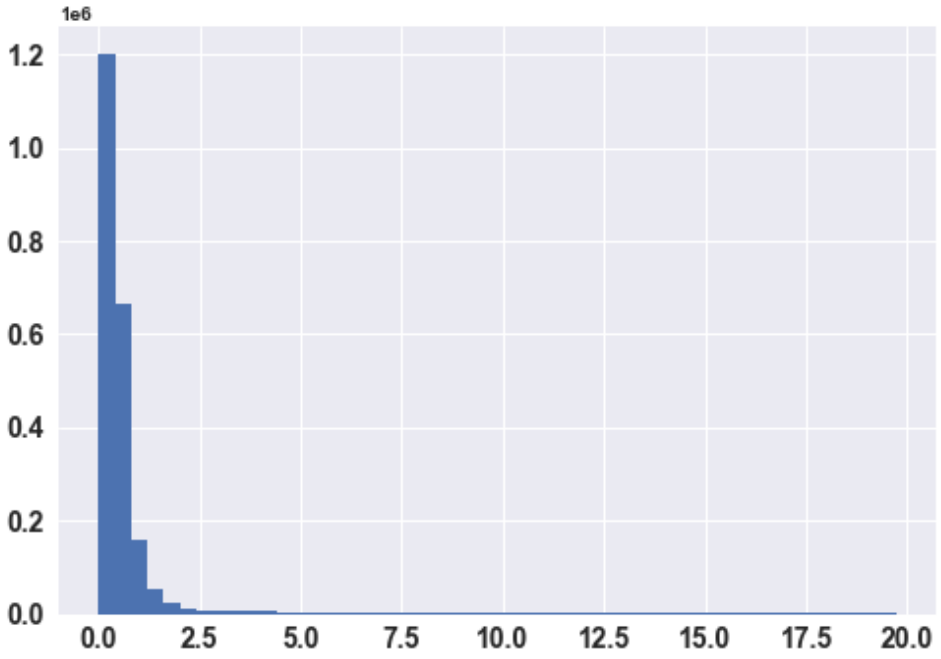


(b) Standard deviation distribution of Upstream CCER column(Y Axis: scientific notation)

Figure A.9: Mean and standard deviation distribution for Upstream CCER

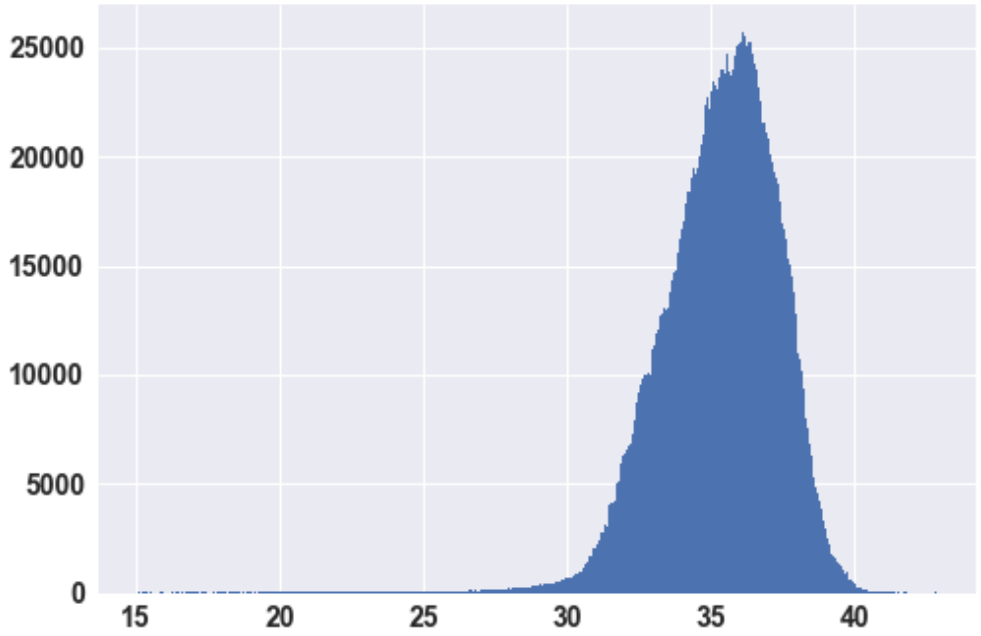


(a) Mean distribution of Downstream SNR column

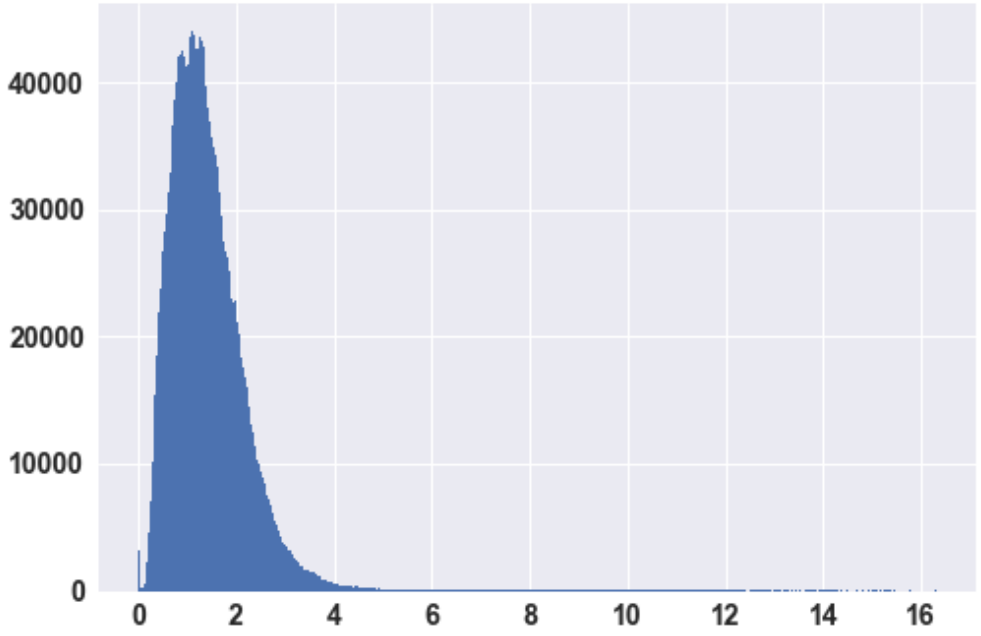


(b) Standard deviation distribution of Downstream SNR column

Figure A.10: Mean and standard deviation distribution for Downstream SNR

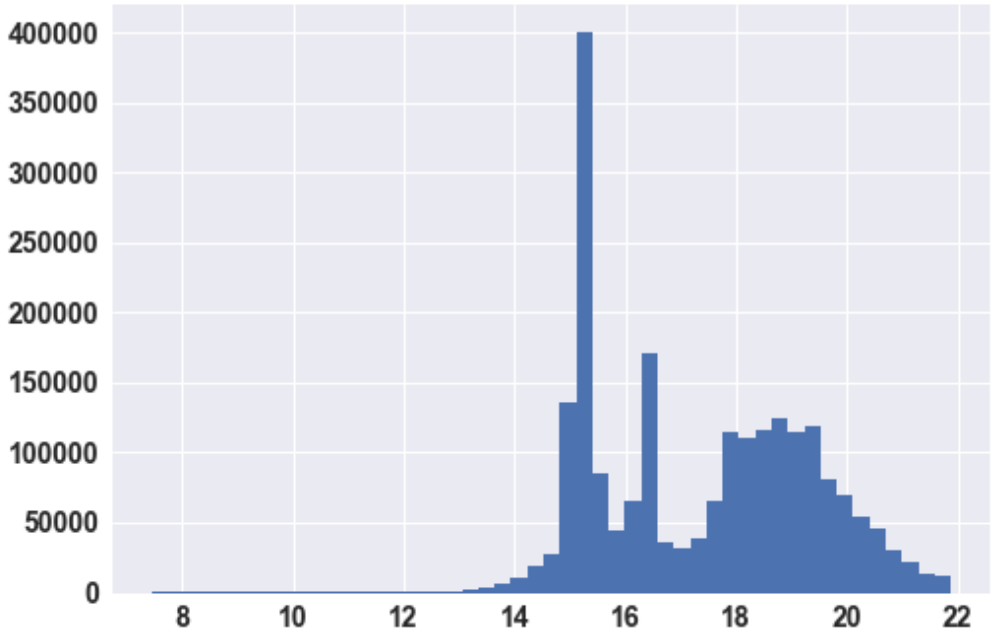


(a) Mean distribution of Upstream SNR column

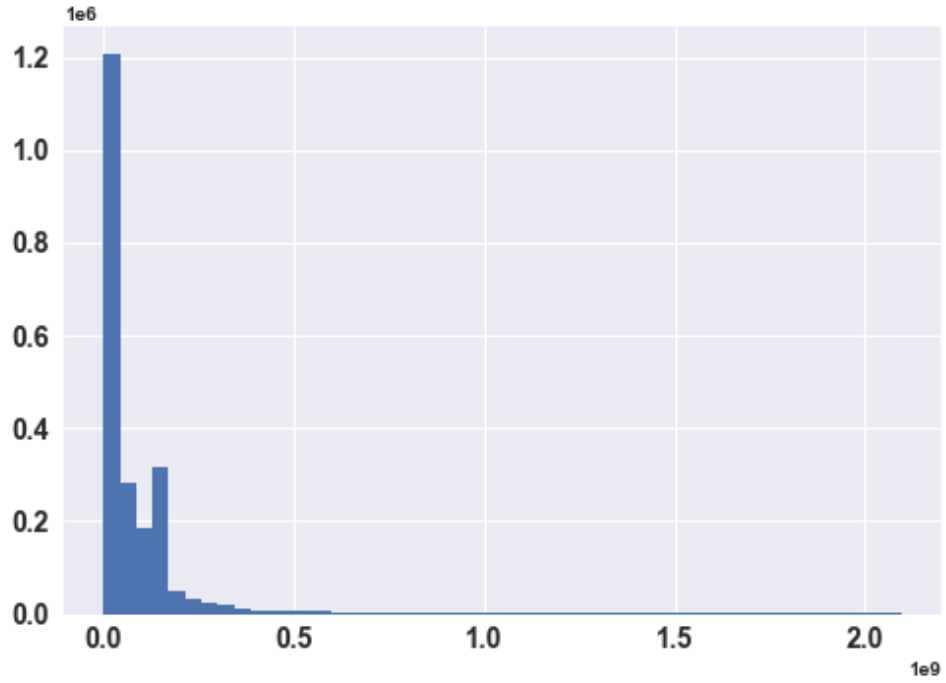


(b) Standard deviation distribution of Upstream SNR column

Figure A.11: Mean and standard deviation distribution for Upstream SNR

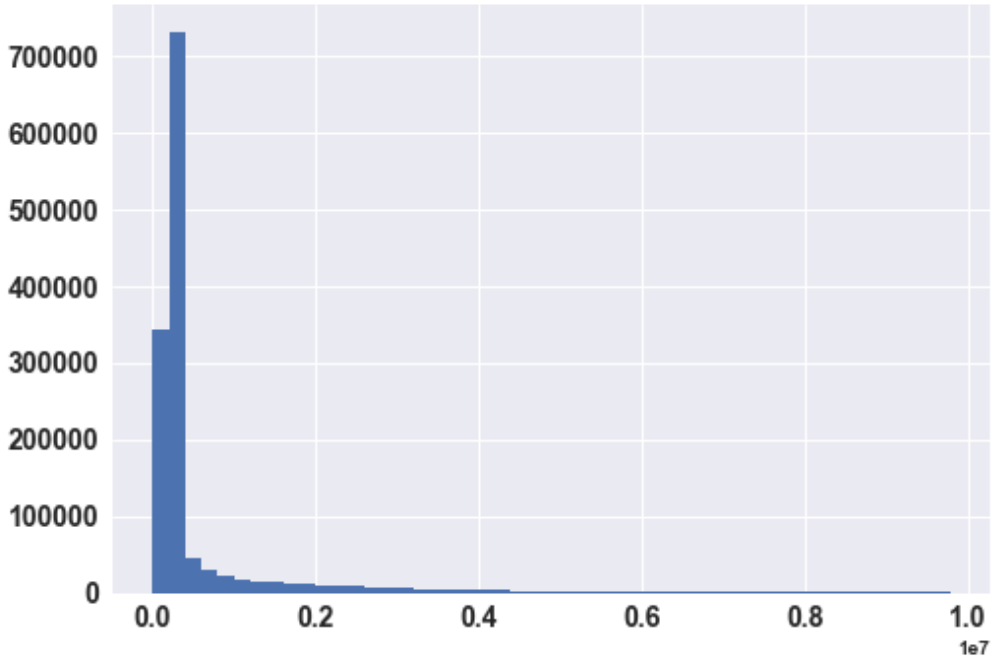


(a) Mean distribution of Sysuptime column(X Axis: in millions)

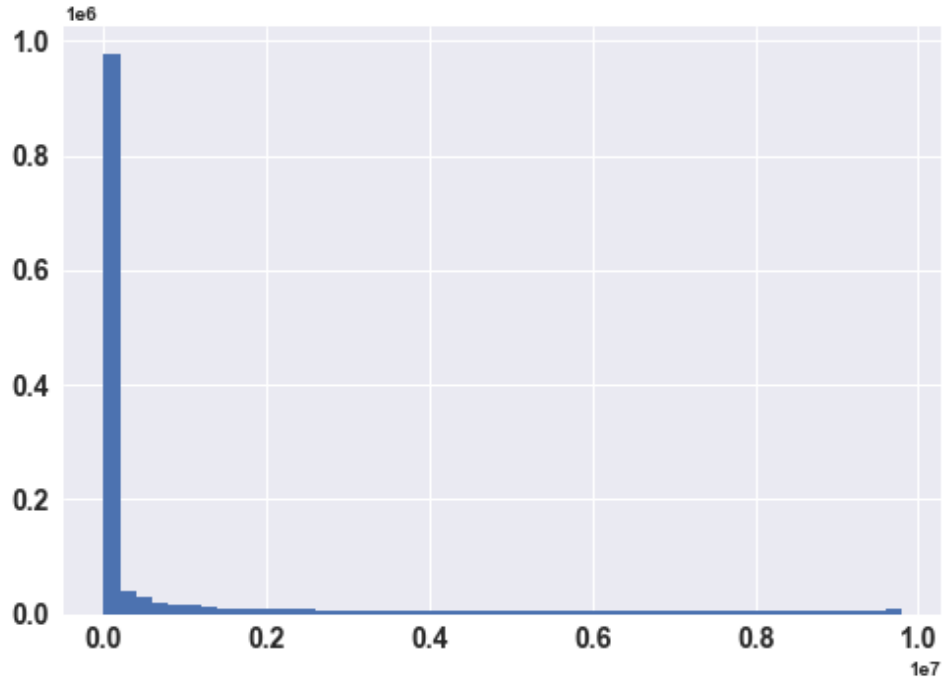


(b) Standard deviation distribution of Sysuptime column(Y Axis: scientific notation)

Figure A.12: Mean and standard deviation distribution for Sysuptime

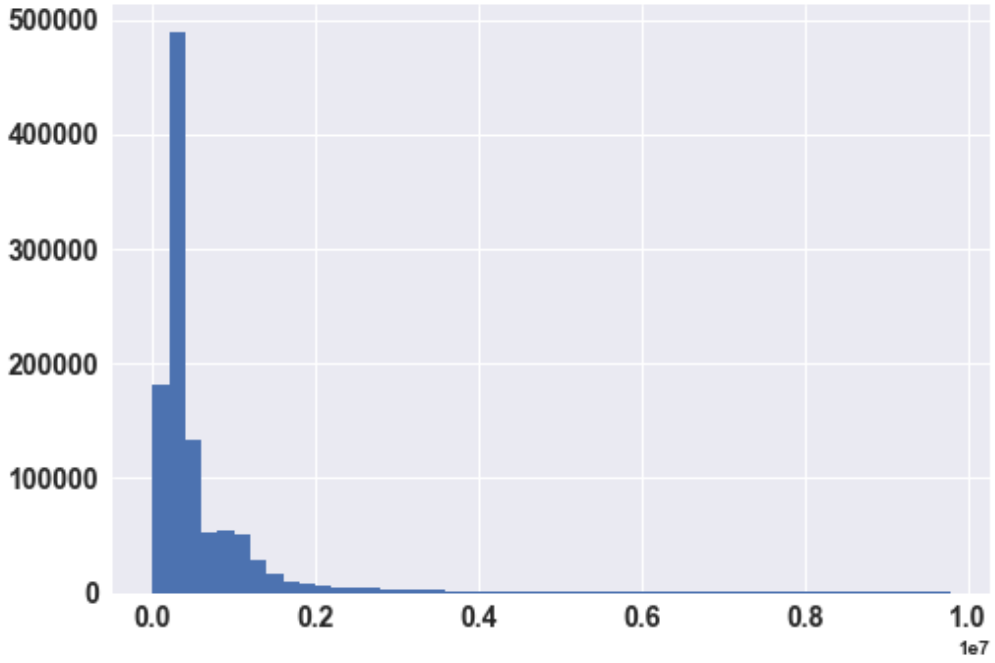


(a) Mean distribution of Sum Bytes Up column(X Axis: scientific notation)

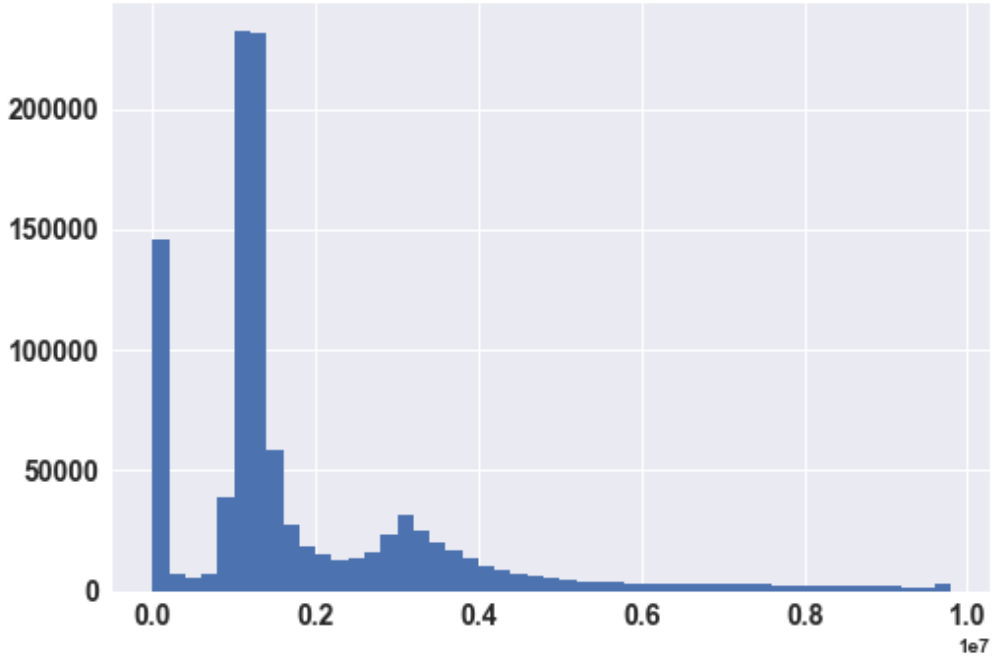


(b) Standard deviation distribution of Sum Bytes Up column(X and Y Axis: scientific notation)

Figure A.13: Mean and standard deviation distribution for Sum Bytes Up



(a) Mean distribution of Sum Bytes Down column(X Axis: scientific notation)



(b) Standard deviation distribution of Sum Bytes Down column(X Axis: scientific notation)

Figure A.14: Mean and standard deviation distribution for Sum Bytes Down

B

Appendix 2 - Evaluation scenarios

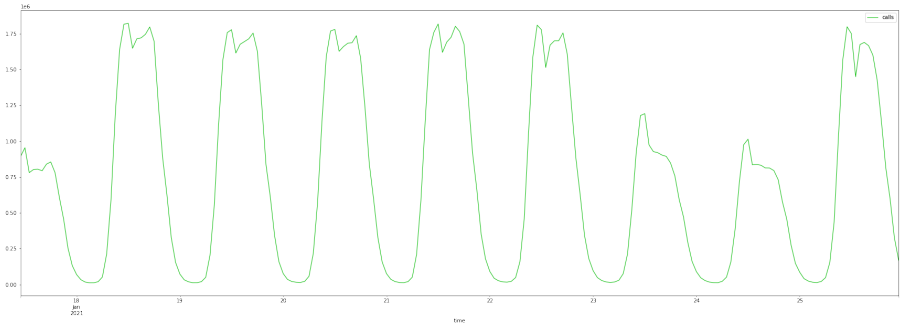


Figure B.1: Zapcae dataset healthy example of data subsets used for model evaluation

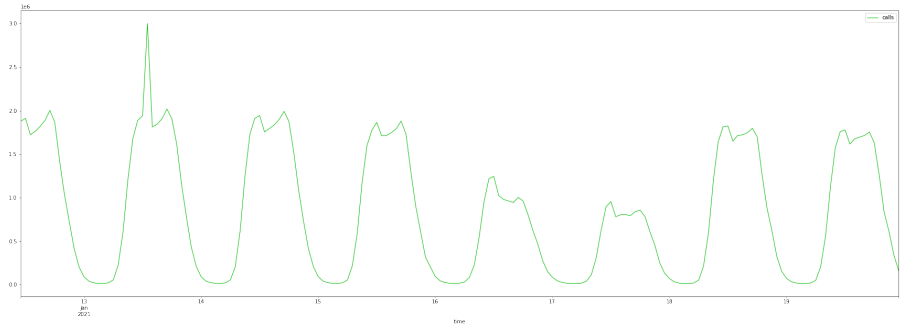


Figure B.2: Zapcae increased anomalous value for calls and clients column

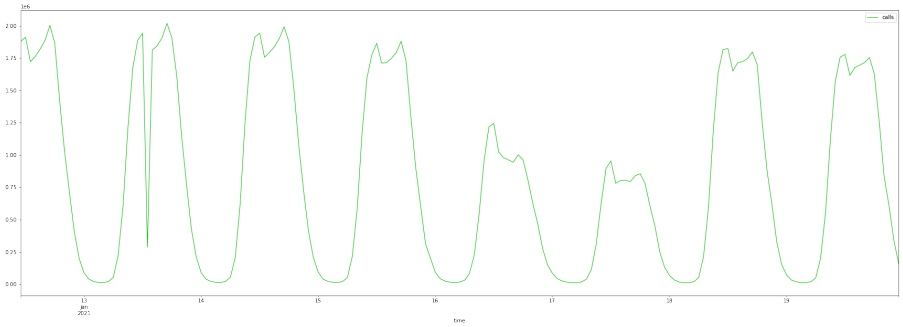


Figure B.3: Zapcae reduced anomalous value for calls and clients column

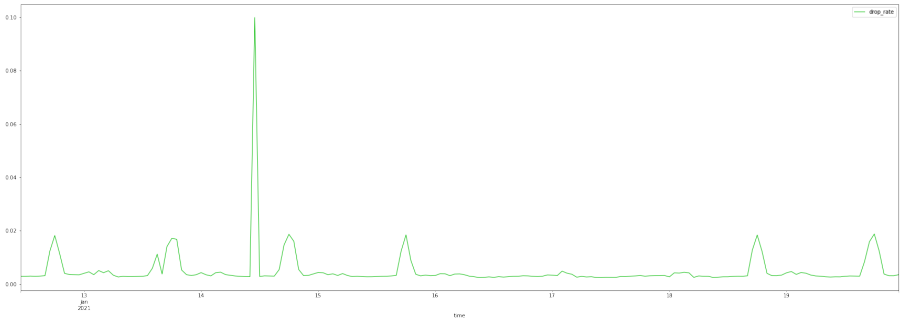


Figure B.4: Zapcae increased anomalous value for drop rates column

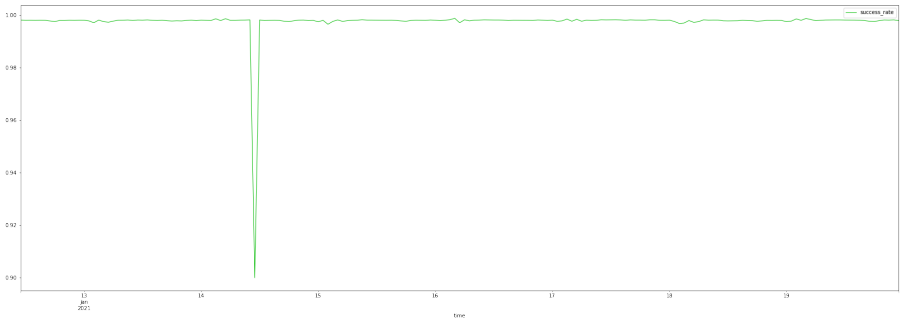


Figure B.5: Zapcae reduced anomalous value for success rates column

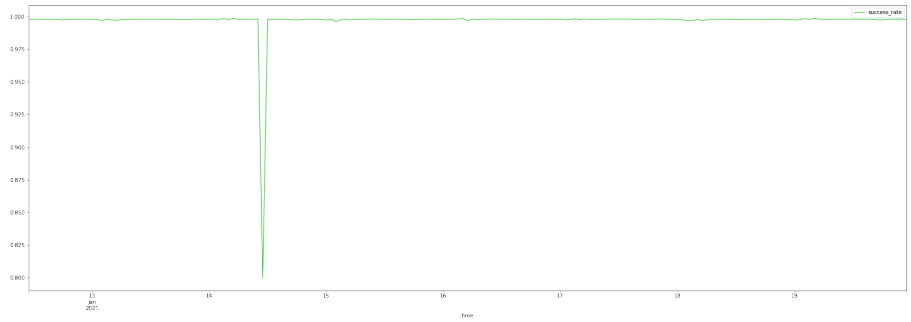


Figure B.6: Zapcae very low anomalous value for success rates column

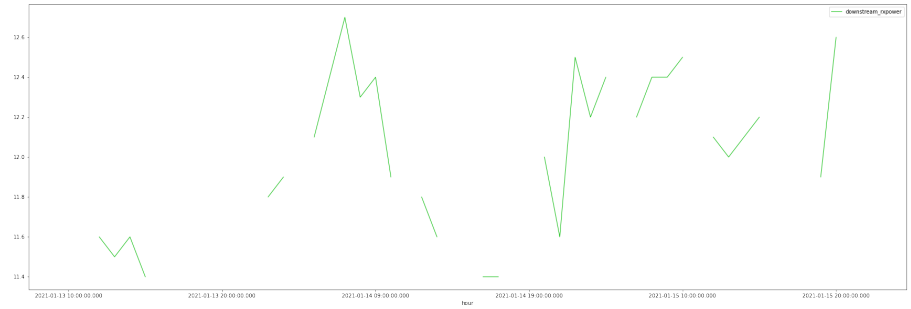


Figure B.7: Testpow dataset healthy example of data subsets' downstream rxpower column used for model evaluation

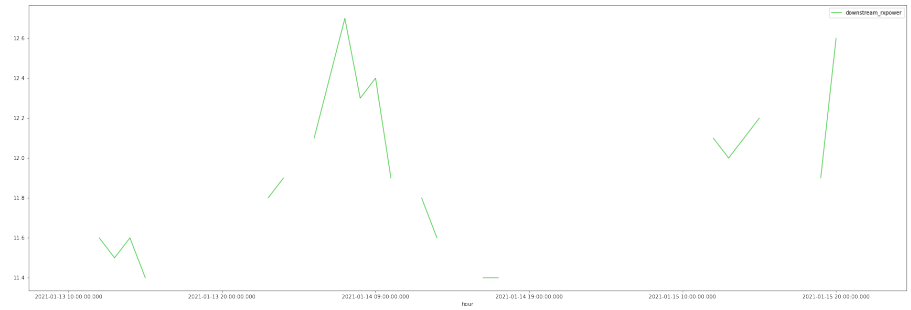


Figure B.8: Testpow many sequential nulls for downstream rxpower column

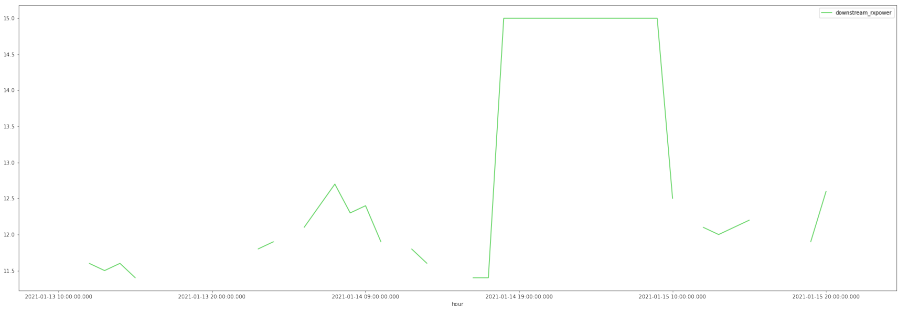


Figure B.9: Testpow anomalous persistent value for downstream rxpower column

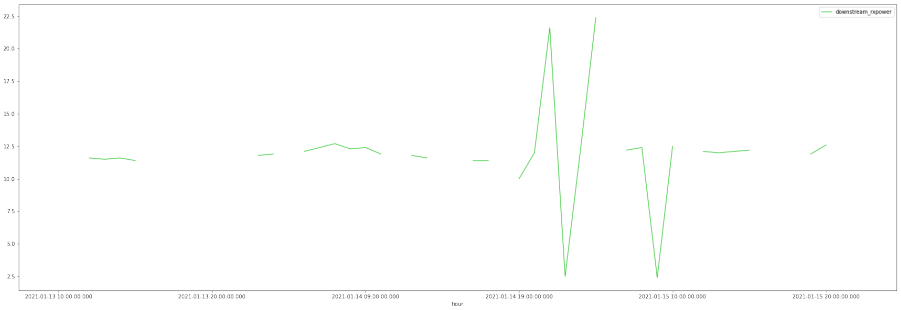


Figure B.10: Testpow high variation anomalous values for downstream rxpower column

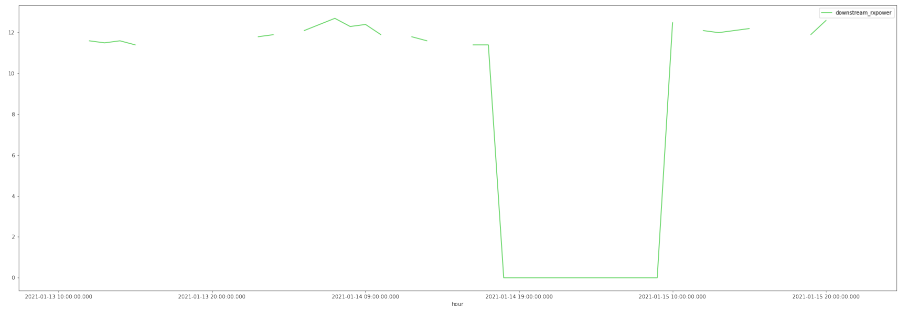


Figure B.11: Testpow consistent zero anomalous value for downstream rxpower column

C

Appendix 3 - Result tables

Table C.1: OCSVM results associated with the Zapcae dataset.

Index	Nu	Gamma	Avg Precision	Avg Recall	Avg F1
0	1.0	1	0.1615	1.0	0.2749
1	1.0	0.1	0.1615	1.0	0.2749
2	1.0	0.01	0.1615	1.0	0.2749
3	1.0	0.5	0.1615	1.0	0.2749
4	1.0	0.7	0.1615	1.0	0.2749
5	1.0	0.9	0.1615	1.0	0.2749
6	1.0	auto	0.1615	1.0	0.2749
7	0.1	1	0.1857	1.0	0.3068
8	0.1	0.1	0.1898	1.0	0.3119
9	0.1	0.01	0.1955	1.0	0.3200
10	0.1	0.5	0.1864	1.0	0.3077
11	0.1	0.7	0.1857	1.0	0.3068
12	0.1	0.9	0.1857	1.0	0.3068
13	0.1	auto	0.1951	1.0	0.3193
14	0.01	1	0.1931	1.0	0.3160
15	0.01	0.1	0.2051	1.0	0.3313
16	0.01	0.01	0.2142	0.9809	0.3440
17	0.01	0.5	0.2033	1.0	0.3282
18	0.01	0.7	0.1970	1.0	0.3207
19	0.01	0.9	0.1935	1.0	0.3164
20	0.01	auto	0.2136	0.9809	0.3432
21	0.5	1	0.1698	1.0	0.2863
22	0.5	0.1	0.1864	1.0	0.3077
23	0.5	0.01	0.1870	1.0	0.3085
24	0.5	0.5	0.1776	1.0	0.2964
25	0.5	0.7	0.1701	1.0	0.2866
26	0.5	0.9	0.1694	1.0	0.2857
27	0.5	auto	0.1870	1.0	0.3085
28	0.7	1	0.1635	1.0	0.2777
29	0.7	0.1	0.1825	1.0	0.3028
30	0.7	0.01	0.1813	1.0	0.3013
31	0.7	0.5	0.1671	1.0	0.2826
32	0.7	0.7	0.1647	1.0	0.2794
33	0.7	0.9	0.1637	1.0	0.2779
34	0.7	auto	0.1813	1.0	0.3013
35	0.9	1	0.1615	1.0	0.2749
36	0.9	0.1	0.1637	1.0	0.2779
37	0.9	0.01	0.1643	1.0	0.2788
38	0.9	0.5	0.1627	1.0	0.2765
39	0.9	0.7	0.1623	1.0	0.2760
40	0.9	0.9	0.1617	1.0	0.2752
41	0.9	auto	0.1643	1.0	0.2788

Table C.2: OCSVM results associated with the Testpow dataset.

Index	Nu	Gamma	Avg Precision	Avg Recall	Avg F1
0	0.1	0.1	0.1357	1.0	0.2390
1	0.1	0.5	0.1357	1.0	0.2390
2	0.1	0.7	0.1357	1.0	0.2390
3	0.1	0.9	0.1357	1.0	0.2390
4	0.1	auto	0.1357	1.0	0.2390
5	0.01	0.1	0.1324	0.9722	0.2331
6	0.01	0.5	0.1324	0.9722	0.2331
7	0.01	0.7	0.1324	0.9722	0.2331
8	0.01	0.9	0.1324	0.9722	0.2331
9	0.01	auto	0.1321	0.9861	0.2330
10	0.001	0.1	0.1315	0.9722	0.2318
11	0.001	0.5	0.1315	0.9722	0.2318
12	0.001	0.7	0.1315	0.9722	0.2318
13	0.001	0.9	0.1315	0.9722	0.2318
14	0.001	auto	0.1213	1.0	0.2163
15	0.2	0.1	0.1323	1.0	0.2337
16	0.2	0.5	0.1323	1.0	0.2337
17	0.2	0.7	0.1323	1.0	0.2337
18	0.2	0.9	0.1323	1.0	0.2337
19	0.2	auto	0.1323	1.0	0.2337
20	0.01	0.1	0.1324	0.9722	0.2331
21	0.01	auto	0.1321	0.9861	0.2330

Table C.3: Results of LSTM, for the Zapcae dataset, in the feature selection form, using a 21 row window and threshold of 1 sigma.

Index	Architecture	Learning Rate	Avg Precision	Avg Recall	Avg F1
0	16 8	0.001	0.4447	0.4523	0.4396
1	16 8	0.0001	0.5635	0.5428	0.5317
2	16 8	0.00001	0.3653	0.3619	0.3469
3	16 8	0.000001	0.2657	0.1809	0.2084
4	32 16	0.001	0.6292	0.6523	0.6265
5	32 16	0.0001	0.4287	0.3714	0.3930
6	32 16	0.00001	0.3093	0.3476	0.3129
7	32 16	0.000001	0.2911	0.2380	0.2575
8	32 16 8	0.001	0.5136	0.5761	0.5254
9	32 16 8	0.0001	0.5444	0.5428	0.5249
10	32 16 8	0.00001	0.5759	0.5190	0.5287
11	32 16 8	0.000001	0.2403	0.1666	0.1937
12	512 32	0.001	0.6030	0.6190	0.6017
13	512 32	0.0001	0.4740	0.5095	0.4785
14	512 32	0.00001	0.5384	0.4904	0.4887
15	512 32	0.000001	0.3597	0.4238	0.3715
16	512 128 8	0.001	0.5962	0.5428	0.5561
17	512 128 8	0.0001	0.4283	0.3619	0.3862
18	512 128 8	0.00001	0.5047	0.5380	0.5074
19	512 128 8	0.000001	0.4152	0.5	0.4368
20	512 256 128 65 32 8	0.001	0.5353	0.5380	0.5237
21	512 256 128 65 32 8	0.0001	0.4095	0.4523	0.4220
22	512 256 128 65 32 8	0.00001	0.6015	0.4666	0.4947
23	512 256 128 65 32 8	0.000001	0.2493	0.2333	0.2385

Table C.4: Results of LSTM, for the Testpow dataset using a 21 row window and threshold of 1 sigma.

Index	Architecture	Learning Rate	Avg Precision	Avg Recall	Avg F1
0	16 8	0.001	0.1839	0.2916	0.2254
1	16 8	0.0001	0.2456	0.3402	0.2852
2	16 8	0.00001	0.0119	0.0138	0.0128
3	16 8	0.000001	0.2642	0.3611	0.3050
4	32 16	0.001	0.2432	0.4513	0.3159
5	32 16	0.0001	0.1485	0.2013	0.1707
6	32 16	0.00001	0.3030	0.4166	0.3505
7	32 16	0.000001	0.2218	0.2847	0.2493
8	32 16 8	0.001	0.3707	0.5138	0.4306
9	32 16 8	0.0001	0.3181	0.4999	0.3887
10	32 16 8	0.00001	0.1970	0.2708	0.2279
11	32 16 8	0.000001	0.0780	0.1111	0.0916
12	512 32	0.001	0.0749	0.1111	0.0894
13	512 32	0.0001	0.0101	0.0138	0.0116
14	512 32	0.00001	0.0861	0.1319	0.1042
15	512 32	0.000001	0.2503	0.3402	0.2882
16	512 128 8	0.001	0.0660	0.0833	0.0736
17	512 128 8	0.0001	0.0875	0.1111	0.0978
18	512 128 8	0.00001	0.3015	0.3819	0.3342
19	512 128 8	0.000001	0.3779	0.5763	0.4564
20	512 256 128 65 32 8	0.001	0.0885	0.1319	0.1059
21	512 256 128 65 32 8	0.0001	0.0885	0.1319	0.1059
22	512 256 128 65 32 8	0.00001	0.0870	0.1319	0.1048
23	512 256 128 65 32 8	0.000001	0.1926	0.2916	0.2319

Table C.5: LSTM results associated with the Testpow dataset using a window width of 7.

Index	Architecture	Learning Rate	Avg Precision	Avg Recall	Avg F1
0	16 8	0.001	0.3027	0.4509	0.3601
1	16 8	0.0001	0.3405	0.5	0.4006
2	16 8	0.00001	0.5010	0.6470	0.5606
3	16 8	0.000001	0.4046	0.5784	0.4698
4	32 16	0.001	0.3761	0.5882	0.4586
5	32 16	0.0001	0.3128	0.4901	0.3807
6	32 16	0.00001	0.4118	0.5588	0.4686
7	32 16	0.000001	0.3249	0.5	0.3888
8	32 16 8	0.001	0.3581	0.5196	0.4214
9	32 16 8	0.0001	0.2576	0.4411	0.3252
10	32 16 8	0.00001	0.3187	0.4313	0.3663
11	32 16 8	0.000001	0.3749	0.5490	0.4420
12	512 32	0.001	0.2625	0.4411	0.3286
13	512 32	0.0001	0.3672	0.5980	0.4548
14	512 32	0.00001	0.2289	0.4607	0.3057
15	512 32	0.000001	0.2843	0.4607	0.3516
16	512 128 8	0.001	0.2625	0.4411	0.3286
17	512 128 8	0.0001	0.4553	0.6078	0.5131
18	512 128 8	0.00001	0.2254	0.3823	0.2835
19	512 128 8	0.000001	0.2479	0.4117	0.3085
20	512 256 128 65 32 8	0.001	0.2625	0.4411	0.3286
21	512 256 128 65 32 8	0.0001	0.2625	0.4019	0.3171
22	512 256 128 65 32 8	0.00001	0.2968	0.4019	0.3405
23	512 256 128 65 32 8	0.000001	0.2885	0.4607	0.3540

Table C.6: Results of TCN, for the Zapcae dataset using a 21 row window and threshold of 1 sigma.

Index	Architecture	Learning Rate	Avg Precision	Avg Recall	Avg F1
0	32 8	0.001	0.1606	0.1809	0.1696
1	32 8	0.0001	0.0180	0.0171	0.0175
2	64 8	0.001	0.1894	0.2400	0.2110
3	64 8	0.0001	0.3194	0.3638	0.3363
4	32 16 8	0.001	0.0	0.0	0.0
5	32 16 8	0.0001	0.2447	0.2400	0.2423
6	128 8	0.001	0.2133	0.2057	0.2092
7	128 8	0.0001	0.0851	0.0990	0.0905
8	64 16 8	0.001	0.4971	0.6057	0.5370
9	64 16 8	0.0001	0.4199	0.2400	0.3054
10	512 8	0.001	0.2495	0.2400	0.2446
11	512 8	0.0001	0.2312	0.2400	0.2355
12	512 128 8	0.001	0.0	0.0	0.0
13	512 128 8	0.0001	0.1017	0.1523	0.1201
14	512 254 64 8	0.001	0.1132	0.2076	0.1440
15	512 254 64 8	0.0001	0.0	0.0	0.0

Table C.7: Results of TCN, for the Testpow dataset using a 21 row window and threshold of 1 sigma.

Index	Architecture	Learning Rate	Avg Precision	Avg Recall	Avg F1
0	32 8	0.0001	0.0	0.0	0.0
1	32 8	0.00001	0.0378	0.0486	0.0425
2	32 8	0.001	0.0222	0.0347	0.0271
3	32 16 8	0.0001	0.0	0.0	0.0
4	32 16 8	0.00001	0.0079	0.0138	0.0101
5	32 16 8	0.001	0.0	0.0	0.0
6	128 16	0.0001	0.0	0.0	0.0
7	128 16	0.00001	0.0050	0.0069	0.0058
8	128 16	0.001	0.0723	0.0972	0.0828

Table C.8: Results of TCN, for the Testpow dataset using a 21 row window, feature selection, and threshold of 1 sigma.

Index	Architecture	Learning Rate	Avg Precision	Avg Recall	Avg F1
0	32 8	0.001	0.0	0.0	0.0
1	32 8	0.0001	0.4204	0.3888	0.4029
2	32 16 8	0.001	0.0931	0.1041	0.0982
3	32 16 8	0.0001	0.0	0.0	0.0
4	128 16	0.001	0.0	0.0	0.0
5	128 16	0.0001	0.1959	0.3125	0.2407

Table C.9: TCN results associated with the Testpow dataset, using a 7 width window.

Index	Architecture	Learning Rate	Avg Precision	Avg Recall	Avg F1
0	32 8	0.0001	0.1764	0.4117	0.2469
1	32 8	0.00001	0.0987	0.1764	0.1266
2	32 8	0.001	0.2271	0.4117	0.2926
3	32 16 8	0.0001	0.2933	0.5294	0.3773
4	32 16 8	0.00001	0.1667	0.3529	0.2264
5	32 16 8	0.001	0.1138	0.2745	0.1609
6	128 16	0.0001	0.1754	0.3431	0.2321
7	128 16	0.00001	0.1636	0.2941	0.2100
8	128 16	0.001	0.0499	0.1078	0.0682

Table C.10: Results of BAE LSTM, for the Zapcae dataset using a 21 row window and threshold of 1 sigma.

Index	Architecture	Learning Rate	Avg Precision	Avg Recall	Avg F1
0	512 128 8	0.00001	0.2788	0.2114	0.2300
1	16 8	0.001	0.6196	0.6285	0.6130
2	16 8	0.0001	0.4914	0.5619	0.5054
3	16 8	0.00001	0.3972	0.3285	0.3544
4	32 16 8	0.001	0.4474	0.4047	0.4149
5	32 16 8	0.0001	0.6038	0.5142	0.5269
6	32 16 8	0.00001	0.2875	0.3809	0.3193
7	512 32	0.001	0.5562	0.6000	0.5652
8	512 32	0.0001	0.4969	0.5000	0.4913
9	512 32	0.00001	0.4980	0.5142	0.4848
0	512 256 128 65 32 8	0.001	0.6210	0.5761	0.5790
11	512 256 128 65 32 8	0.001	0.5504	0.5428	0.5275
12	512 256 128 65 32 8	0.001	0.5085	0.4476	0.4645

Table C.11: Results of BAE LSTM, for the Testpow dataset using a 21 row window and threshold of 1 sigma.

Index	Architecture	Learning Rate	Avg Precision	Avg Recall	Avg F1
0	32 16	0.000001	0.5825	0.9027	0.7077
1	32 16	0.00001	0.1417	0.1319	0.1366
2	512 128 8	0.000001	0.1496	0.1597	0.1530
3	512 128 8	0.00001	0.1734	0.2569	0.2067
4	16 8	0.000001	0.0324	0.0416	0.0364
5	16 8	0.00001	0.1645	0.2083	0.1835
6	32 16 8	0.000001	0.3258	0.4166	0.3629
7	32 16 8	0.00001	0.4091	0.6319	0.4958
8	512 256 128 65 32 8	0.000001	0.1817	0.2291	0.2022
9	512 256 128 65 32 8	0.00001	0.4767	0.5208	0.4916

Table C.12: Results of OCSVM, for the Zapcae reduced success rates scenario using the RBF kernel, using feature selection.

Nu	Gamma	Precision	Recall	F1
0.01	0.01	0.289855072	0.952380952	0.444444444
0.01	0.1	0.302158273	1	0.464088398
0.01	0.5	0.306569343	1	0.469273743
0.01	0.7	0.291666667	1	0.451612903
0.01	0.9	0.281879195	1	0.439790576
0.01	1	0.281879195	1	0.439790576
0.01	auto	0.289855072	0.952380952	0.444444444
0.1	0.01	0.276315789	1	0.432989691
0.1	0.1	0.272727273	1	0.428571429
0.1	0.5	0.264150943	1	0.417910448
0.1	0.7	0.2625	1	0.415841584
0.1	0.9	0.2625	1	0.415841584
0.1	1	0.2625	1	0.415841584
0.1	auto	0.276315789	1	0.432989691
0.5	0.01	0.265822785	1	0.42
0.5	0.1	0.264150943	1	0.417910448
0.5	0.5	0.245614035	1	0.394366197
0.5	0.7	0.223404255	1	0.365217391
0.5	0.9	0.222222222	1	0.363636364
0.5	1	0.223404255	1	0.365217391
0.5	auto	0.265822785	1	0.42
0.7	0.01	0.251497006	1	0.401913876
0.7	0.1	0.254545455	1	0.405797101
0.7	0.5	0.216494845	1	0.355932203
0.7	0.7	0.21	1	0.347107438
0.7	0.9	0.207920792	1	0.344262295
0.7	1	0.206896552	1	0.342857143
0.7	auto	0.251497006	1	0.401913876
0.9	0.01	0.208955224	1	0.345679012
0.9	0.1	0.207920792	1	0.344262295
0.9	0.5	0.204878049	1	0.340080972
0.9	0.7	0.203883495	1	0.338709677
0.9	0.9	0.202898551	1	0.337349398
0.9	1	0.201923077	1	0.336
0.9	auto	0.208955224	1	0.345679012
1	0.01	0.201923077	1	0.336
1	0.1	0.201923077	1	0.336
1	0.5	0.201923077	1	0.336
1	0.7	0.201923077	1	0.336
1	0.9	0.201923077	1	0.336
1	1	0.201923077	1	0.336
1	auto	0.201923077	1	0.336

Table C.13: Results of OCSVM, for the Zapcae high reduction of success rates scenario using the RBF kernel, using feature selection.

Nu	Gamma	Precision	Recall	F1
0.01	0.01	0.289855072	0.952380952	0.444444444
0.01	0.1	0.302158273	1	0.464088398
0.01	0.5	0.306569343	1	0.469273743
0.01	0.7	0.289655172	1	0.449197861
0.01	0.9	0.281879195	1	0.439790576
0.01	1	0.28	1	0.4375
0.01	auto	0.289855072	0.952380952	0.444444444
0.1	0.01	0.276315789	1	0.432989691
0.1	0.1	0.272727273	1	0.428571429
0.1	0.5	0.264150943	1	0.417910448
0.1	0.7	0.2625	1	0.415841584
0.1	0.9	0.2625	1	0.415841584
0.1	1	0.2625	1	0.415841584
0.1	auto	0.276315789	1	0.432989691
0.5	0.01	0.265822785	1	0.42
0.5	0.1	0.264150943	1	0.417910448
0.5	0.5	0.238636364	1	0.385321101
0.5	0.7	0.223404255	1	0.365217391
0.5	0.9	0.221052632	1	0.362068966
0.5	1	0.222222222	1	0.363636364
0.5	auto	0.265822785	1	0.42
0.7	0.01	0.251497006	1	0.401913876
0.7	0.1	0.254545455	1	0.405797101
0.7	0.5	0.215384615	1	0.35443038
0.7	0.7	0.21	1	0.347107438
0.7	0.9	0.206896552	1	0.342857143
0.7	1	0.206896552	1	0.342857143
0.7	auto	0.251497006	1	0.401913876
0.9	0.01	0.208955224	1	0.345679012
0.9	0.1	0.206896552	1	0.342857143
0.9	0.5	0.204878049	1	0.340080972
0.9	0.7	0.203883495	1	0.338709677
0.9	0.9	0.201923077	1	0.336
0.9	1	0.201923077	1	0.336
0.9	auto	0.208955224	1	0.345679012
1	0.01	0.201923077	1	0.336
1	0.1	0.201923077	1	0.336
1	0.5	0.201923077	1	0.336
1	0.7	0.201923077	1	0.336
1	0.9	0.201923077	1	0.336
1	1	0.201923077	1	0.336
1	auto	0.201923077	1	0.336

Table C.14: Results of OCSVM, for the Zapcae high values of drop rates scenario using the RBF kernel, using feature selection.

Nu	Gamma	Precision	Recall	F1
0.01	0.01	0.24852071	1	0.398104265
0.01	0.1	0.21319797	1	0.351464435
0.01	0.5	0.201923077	1	0.336
0.01	0.7	0.201923077	1	0.336
0.01	0.9	0.201923077	1	0.336
0.01	1	0.201923077	1	0.336
0.01	auto	0.247058824	1	0.396226415
0.1	0.01	0.215384615	1	0.35443038
0.1	0.1	0.201923077	1	0.336
0.1	0.5	0.201923077	1	0.336
0.1	0.7	0.201923077	1	0.336
0.1	0.9	0.201923077	1	0.336
0.1	1	0.201923077	1	0.336
0.1	auto	0.214285714	1	0.352941176
0.5	0.01	0.201923077	1	0.336
0.5	0.1	0.201923077	1	0.336
0.5	0.5	0.201923077	1	0.336
0.5	0.7	0.201923077	1	0.336
0.5	0.9	0.201923077	1	0.336
0.5	1	0.201923077	1	0.336
0.5	auto	0.201923077	1	0.336
0.7	0.01	0.201923077	1	0.336
0.7	0.1	0.201923077	1	0.336
0.7	0.5	0.201923077	1	0.336
0.7	0.7	0.201923077	1	0.336
0.7	0.9	0.201923077	1	0.336
0.7	1	0.201923077	1	0.336
0.7	auto	0.201923077	1	0.336
0.9	0.01	0.201923077	1	0.336
0.9	0.1	0.201923077	1	0.336
0.9	0.5	0.201923077	1	0.336
0.9	0.7	0.201923077	1	0.336
0.9	0.9	0.201923077	1	0.336
0.9	1	0.201923077	1	0.336
0.9	auto	0.201923077	1	0.336
1	0.01	0.201923077	1	0.336
1	0.1	0.201923077	1	0.336
1	0.5	0.201923077	1	0.336
1	0.7	0.201923077	1	0.336
1	0.9	0.201923077	1	0.336
1	1	0.201923077	1	0.336
1	auto	0.201923077	1	0.336

Table C.15: Results of OCSVM, for the Zapcae increased values for calls and clients scenario using the RBF kernel, using feature selection.

Nu	Gamma	Precision	Recall	F1
0.01	0.01	0.118644068	1	0.212121212
0.01	0.1	0.100961538	1	0.183406114
0.01	0.5	0.100961538	1	0.183406114
0.01	0.7	0.100961538	1	0.183406114
0.01	0.9	0.100961538	1	0.183406114
0.01	1	0.100961538	1	0.183406114
0.01	auto	0.117318436	1	0.21
0.1	0.01	0.100961538	1	0.183406114
0.1	0.1	0.100961538	1	0.183406114
0.1	0.5	0.100961538	1	0.183406114
0.1	0.7	0.100961538	1	0.183406114
0.1	0.9	0.100961538	1	0.183406114
0.1	1	0.100961538	1	0.183406114
0.1	auto	0.100961538	1	0.183406114
0.5	0.01	0.100961538	1	0.183406114
0.5	0.1	0.100961538	1	0.183406114
0.5	0.5	0.100961538	1	0.183406114
0.5	0.7	0.100961538	1	0.183406114
0.5	0.9	0.100961538	1	0.183406114
0.5	1	0.100961538	1	0.183406114
0.5	auto	0.100961538	1	0.183406114
0.7	0.01	0.100961538	1	0.183406114
0.7	0.1	0.100961538	1	0.183406114
0.7	0.5	0.100961538	1	0.183406114
0.7	0.7	0.100961538	1	0.183406114
0.7	0.9	0.100961538	1	0.183406114
0.7	1	0.100961538	1	0.183406114
0.7	auto	0.100961538	1	0.183406114
0.9	0.01	0.100961538	1	0.183406114
0.9	0.1	0.100961538	1	0.183406114
0.9	0.5	0.100961538	1	0.183406114
0.9	0.7	0.100961538	1	0.183406114
0.9	0.9	0.100961538	1	0.183406114
0.9	1	0.100961538	1	0.183406114
0.9	auto	0.100961538	1	0.183406114
1	0.01	0.100961538	1	0.183406114
1	0.1	0.100961538	1	0.183406114
1	0.5	0.100961538	1	0.183406114
1	0.7	0.100961538	1	0.183406114
1	0.9	0.100961538	1	0.183406114
1	1	0.100961538	1	0.183406114
1	auto	0.100961538	1	0.183406114

Table C.16: Results of OCSVM, for the Zapcae reduced values for calls and clients scenario using the RBF kernel, using feature selection.

Nu	Gamma	Precision	Recall	F1
0.01	0.01	0.124260355	1	0.221052632
0.01	0.1	0.107142857	1	0.193548387
0.01	0.5	0.100961538	1	0.183406114
0.01	0.7	0.100961538	1	0.183406114
0.01	0.9	0.100961538	1	0.183406114
0.01	1	0.100961538	1	0.183406114
0.01	auto	0.124260355	1	0.221052632
0.1	0.01	0.10880829	1	0.196261682
0.1	0.1	0.100961538	1	0.183406114
0.1	0.5	0.100961538	1	0.183406114
0.1	0.7	0.100961538	1	0.183406114
0.1	0.9	0.100961538	1	0.183406114
0.1	1	0.100961538	1	0.183406114
0.1	auto	0.107692308	1	0.194444444
0.5	0.01	0.100961538	1	0.183406114
0.5	0.1	0.100961538	1	0.183406114
0.5	0.5	0.100961538	1	0.183406114
0.5	0.7	0.100961538	1	0.183406114
0.5	0.9	0.100961538	1	0.183406114
0.5	1	0.100961538	1	0.183406114
0.5	auto	0.100961538	1	0.183406114
0.7	0.01	0.100961538	1	0.183406114
0.7	0.1	0.100961538	1	0.183406114
0.7	0.5	0.100961538	1	0.183406114
0.7	0.7	0.100961538	1	0.183406114
0.7	0.9	0.100961538	1	0.183406114
0.7	1	0.100961538	1	0.183406114
0.7	auto	0.100961538	1	0.183406114
0.9	0.01	0.100961538	1	0.183406114
0.9	0.1	0.100961538	1	0.183406114
0.9	0.5	0.100961538	1	0.183406114
0.9	0.7	0.100961538	1	0.183406114
0.9	0.9	0.100961538	1	0.183406114
0.9	1	0.100961538	1	0.183406114
0.9	auto	0.100961538	1	0.183406114
1	0.01	0.100961538	1	0.183406114
1	0.1	0.100961538	1	0.183406114
1	0.5	0.100961538	1	0.183406114
1	0.7	0.100961538	1	0.183406114
1	0.9	0.100961538	1	0.183406114
1	1	0.100961538	1	0.183406114
1	auto	0.100961538	1	0.183406114

Table C.17: Results of OCSVM, for the Testpow many nulls for a single column scenario using the Poly kernel.

Nu	Gamma	Precision	Recall	F1
0.001	0.1	0.134831461	1	0.237623762
0.001	0.5	0.134831461	1	0.237623762
0.001	0.7	0.134831461	1	0.237623762
0.001	0.9	0.134831461	1	0.237623762
0.001	auto	0.121212121	1	0.216216216
0.01	0.1	0.136363636	1	0.24
0.01	0.5	0.136363636	1	0.24
0.01	0.7	0.136363636	1	0.24
0.01	0.9	0.136363636	1	0.24
0.01	auto	0.133333333	1	0.235294118
0.1	0.1	0.13559322	1	0.23880597
0.1	0.5	0.13559322	1	0.23880597
0.1	0.7	0.13559322	1	0.23880597
0.1	0.9	0.13559322	1	0.23880597
0.1	auto	0.13559322	1	0.23880597
0.2	0.1	0.132596685	1	0.234146341
0.2	0.5	0.132596685	1	0.234146341
0.2	0.7	0.132596685	1	0.234146341
0.2	0.9	0.132596685	1	0.234146341
0.2	auto	0.132596685	1	0.234146341

Table C.18: Results of OCSVM, for the Testpow many nulls for multiple columns scenario using the Poly kernel.

Nu	Gamma	Precision	Recall	F1
0.001	0.1 0.134831461	1	0.237623762	
0.001	0.5 0.134831461	1	0.237623762	
0.001	0.7 0.134831461	1	0.237623762	
0.001	0.9 0.134831461	1	0.237623762	
0.001	auto 0.121212121	1	0.216216216	
0.01	0.1 0.136363636	1	0.24	
0.01	0.5 0.136363636	1	0.24	
0.01	0.7 0.136363636	1	0.24	
0.01	0.9 0.136363636	1	0.24	
0.01	auto 0.133333333	1	0.235294118	
0.1	0.1 0.13559322	1	0.23880597	
0.1	0.5 0.13559322	1	0.23880597	
0.1	0.7 0.13559322	1	0.23880597	
0.1	0.9 0.13559322	1	0.23880597	
0.1	auto 0.13559322	1	0.23880597	
0.2	0.1 0.132596685	1	0.234146341	
0.2	0.5 0.132596685	1	0.234146341	
0.2	0.7 0.132596685	1	0.234146341	
0.2	0.9 0.132596685	1	0.234146341	
0.2	auto 0.132596685	1	0.234146341	

Table C.19: Results of OCSVM, for the Testpow persistent value scenario using the Poly kernel.

Nu	Gamma	Precision	Recall	F1
0.001	0.1	0.12568306	0.958333333	0.222222222
0.001	0.5	0.12568306	0.958333333	0.222222222
0.001	0.7	0.12568306	0.958333333	0.222222222
0.001	0.9	0.12568306	0.958333333	0.222222222
0.001	auto	0.12	1	0.214285714
0.01	0.1	0.12568306	0.958333333	0.222222222
0.01	0.5	0.12568306	0.958333333	0.222222222
0.01	0.7	0.12568306	0.958333333	0.222222222
0.01	0.9	0.12568306	0.958333333	0.222222222
0.01	auto	0.125	0.958333333	0.221153846
0.1	0.1	0.132596685	1	0.234146341
0.1	0.5	0.132596685	1	0.234146341
0.1	0.7	0.132596685	1	0.234146341
0.1	0.9	0.132596685	1	0.234146341
0.1	auto	0.132596685	1	0.234146341
0.2	0.1	0.131147541	1	0.231884058
0.2	0.5	0.131147541	1	0.231884058
0.2	0.7	0.131147541	1	0.231884058
0.2	0.9	0.131147541	1	0.231884058
0.2	auto	0.131147541	1	0.231884058

Table C.20: Results of OCSVM, for the Testpow high variation values for a single column scenario using the Poly kernel.

Nu	Gamma	Precision	Recall	F1
0.001	0.1	0.126436782	0.916666667	0.222222222
0.001	0.5	0.126436782	0.916666667	0.222222222
0.001	0.7	0.126436782	0.916666667	0.222222222
0.001	0.9	0.126436782	0.916666667	0.222222222
0.001	auto	0.121212121	1	0.216216216
0.01	0.1	0.12716763	0.916666667	0.223350254
0.01	0.5	0.12716763	0.916666667	0.223350254
0.01	0.7	0.12716763	0.916666667	0.223350254
0.01	0.9	0.12716763	0.916666667	0.223350254
0.01	auto	0.130681818	0.958333333	0.23
0.1	0.1	0.13559322	1	0.23880597
0.1	0.5	0.13559322	1	0.23880597
0.1	0.7	0.13559322	1	0.23880597
0.1	0.9	0.13559322	1	0.23880597
0.1	auto	0.13559322	1	0.23880597
0.2	0.1	0.132596685	1	0.234146341
0.2	0.5	0.132596685	1	0.234146341
0.2	0.7	0.132596685	1	0.234146341
0.2	0.9	0.132596685	1	0.234146341
0.2	auto	0.132596685	1	0.234146341

Table C.21: Results of OCSVM, for the Testpow high variation values for multiple columns scenario using the Poly kernel.

Nu	Gamma	Precision	Recall	F1
0.001	0.1	0.134831461	1	0.237623762
0.001	0.5	0.134831461	1	0.237623762
0.001	0.7	0.134831461	1	0.237623762
0.001	0.9	0.134831461	1	0.237623762
0.001	auto	0.121212121	1	0.216216216
0.01	0.1	0.136363636	1	0.24
0.01	0.5	0.136363636	1	0.24
0.01	0.7	0.136363636	1	0.24
0.01	0.9	0.136363636	1	0.24
0.01	auto	0.133333333	1	0.235294118
0.1	0.1	0.13559322	1	0.23880597
0.1	0.5	0.13559322	1	0.23880597
0.1	0.7	0.13559322	1	0.23880597
0.1	0.9	0.13559322	1	0.23880597
0.1	auto	0.13559322	1	0.23880597
0.2	0.1	0.132596685	1	0.234146341
0.2	0.5	0.132596685	1	0.234146341
0.2	0.7	0.132596685	1	0.234146341
0.2	0.9	0.132596685	1	0.234146341
0.2	auto	0.132596685	1	0.234146341

Table C.22: Results of OCSVM, for the Testpow many zero values scenario using the Poly kernel.

Nu	Gamma	Precision	Recall	F1
0.001	0.1	0.132947977	0.958333333	0.233502538
0.001	0.5	0.132947977	0.958333333	0.233502538
0.001	0.7	0.132947977	0.958333333	0.233502538
0.001	0.9	0.132947977	0.958333333	0.233502538
0.001	auto	0.123076923	1	0.219178082
0.01	0.1	0.132947977	0.958333333	0.233502538
0.01	0.5	0.132947977	0.958333333	0.233502538
0.01	0.7	0.132947977	0.958333333	0.233502538
0.01	0.9	0.132947977	0.958333333	0.233502538
0.01	auto	0.137142857	1	0.24120603
0.1	0.1	0.139534884	1	0.244897959
0.1	0.5	0.139534884	1	0.244897959
0.1	0.7	0.139534884	1	0.244897959
0.1	0.9	0.139534884	1	0.244897959
0.1	auto	0.139534884	1	0.244897959
0.2	0.1	0.132596685	1	0.234146341
0.2	0.5	0.132596685	1	0.234146341
0.2	0.7	0.132596685	1	0.234146341
0.2	0.9	0.132596685	1	0.234146341
0.2	auto	0.132596685	1	0.234146341

Table C.23: Results of LSTM AE, for the Zapcae reduced success rates scenario, using feature selection.

Index	Architecture	Learning Rate	Precision	Recall	F1
0	16 8	0.001	0.75	0.6429	0.6923
1	16 8	0.0001	1.0	0.7619	0.8649
2	16 8	0.00001	0.6562	0.5	0.5676
3	16 8	0.000001	0.3913	0.2143	0.2769
4	32 16	0.001	1.0	0.881	0.9367
5	32 16	0.0001	0.8788	0.6905	0.7733
6	32 16	0.00001	0.3548	0.2619	0.3014
7	32 16	0.000001	0.4688	0.3571	0.4054
8	32 16 8	0.001	0.8919	0.7857	0.8354
9	32 16 8	0.0001	0.9412	0.7619	0.8421
10	32 16 8	0.00001	0.8214	0.5476	0.6571
11	32 16 8	0.000001	0.36	0.2143	0.2687
12	512 32	0.001	0.85	0.8095	0.8293
13	512 32	0.0001	0.8378	0.7381	0.7848
14	512 32	0.00001	1.0	0.6905	0.8169
15	512 32	0.000001	0.4	0.2857	0.3333
16	512 128 8	0.001	0.9355	0.6905	0.7945
17	512 128 8	0.0001	0.8125	0.619	0.7027
18	512 128 8	0.00001	0.8611	0.7381	0.7949
19	512 128 8	0.000001	0.5676	0.5	0.5316
20	512 256 128 65 32 8	0.001	0.8333	0.7143	0.7692
21	512 256 128 65 32 8	0.0001	0.7857	0.7857	0.7857
22	512 256 128 65 32 8	0.00001	0.913	0.5	0.6462
23	512 256 128 65 32 8	0.000001	0.4595	0.4048	0.4304

Table C.24: Results of LSTM AE, for the Zapcae high reduction of success rates scenario, using feature selection.

Index	Architecture	Learning Rate	Precision	Recall	F1
0	16 8	0.001	0.7368	0.6667	0.7
1	16 8	0.0001	1.0	0.7381	0.8493
2	16 8	0.00001	0.6562	0.5	0.5676
3	16 8	0.000001	0.3913	0.2143	0.2769
4	32 16	0.001	1.0	0.881	0.9367
5	32 16	0.0001	0.8824	0.7143	0.7895
6	32 16	0.00001	0.375	0.2857	0.3243
7	32 16	0.000001	0.4516	0.3333	0.3836
8	32 16 8	0.001	0.8919	0.7857	0.8354
9	32 16 8	0.0001	0.9375	0.7143	0.8108
10	32 16 8	0.00001	0.7931	0.5476	0.6479
11	32 16 8	0.000001	0.36	0.2143	0.2687
12	512 32	0.001	0.8293	0.8095	0.8193
13	512 32	0.0001	0.8378	0.7381	0.7848
14	512 32	0.00001	1.0	0.6667	0.8
15	512 32	0.000001	0.3871	0.2857	0.3288
16	512 128 8	0	0.9355	0.6905	0.7945
17	512 128 8	0	0.8125	0.619	0.7027
18	512 128 8	0	0.8611	0.7381	0.7949
19	512 128 8	0	0.5556	0.4762	0.5128
20	512 256 128 65 32 8	0.001	0.8333	0.7143	0.7692
21	512 256 128 65 32 8	0.0001	0.775	0.7381	0.7561
22	512 256 128 65 32 8	0.00001	0.913	0.5	0.6462
23	512 256 128 65 32 8	0.000001	0.4737	0.4286	0.45

Table C.25: Results of LSTM AE, for the Zapcae increased drop rates scenario, using feature selection.

Index	Architecture	Learning Rate	Precision	Recall	F1
0	16 8	0.001	0.2143	0.1429	0.1714
1	16 8	0.0001	0.3846	0.3571	0.3704
2	16 8	0.00001	0.1481	0.0952	0.1159
3	16 8	0.000001	0.4286	0.2857	0.3429
4	32 16	0.001	0.5	0.4048	0.4474
5	32 16	0.0001	0.2059	0.1667	0.1842
6	32 16	0.00001	0.3684	0.3333	0.35
7	32 16	0.000001	0.3793	0.2619	0.3099
8	32 16 8	0.001	0.3023	0.3095	0.3059
9	32 16 8	0.0001	0.3243	0.2857	0.3038
10	32 16 8	0.00001	0.4545	0.3571	0.4
11	32 16 8	0.000001	0.4194	0.3095	0.3562
12	512 32	0.001	0.75	0.5714	0.6486
13	512 32	0.0001	0.1892	0.1667	0.1772
14	512 32	0.00001	0.2857	0.2857	0.2857
15	512 32	0.000001	0.3659	0.3571	0.3614
16	512 128 8	0	0.5789	0.5238	0.55
17	512 128 8	0	0.2143	0.1429	0.1714
18	512 128 8	0	0.3409	0.3571	0.3488
19	512 128 8	0	0.3902	0.381	0.3855
20	512 256 128 65 32 8	0.001	0.3333	0.2143	0.2609
21	512 256 128 65 32 8	0.0001	0.1515	0.119	0.1333
22	512 256 128 65 32 8	0.00001	0.5	0.2857	0.3636
23	512 256 128 65 32 8	0.000001	0.1429	0.0952	0.1143

Table C.26: Results of LSTM AE, for the Zapcae increased calls and clients scenario, using feature selection.

Index	Architecture	Learning Rate	Precision	Recall	F1
0	16 8	0.001	0.3226	0.4762	0.3846
1	16 8	0.0001	0.2667	0.5714	0.3636
2	16 8	0.00001	0.1951	0.381	0.2581
3	16 8	0.000001	0.0	0.0	0.0
4	32 16	0.001	0.2051	0.381	0.2667
5	32 16	0.0001	0.0909	0.1429	0.1111
6	32 16	0.00001	0.2821	0.5238	0.3667
7	32 16	0.000001	0.0	0.0	0.0
8	32 16 8	0.001	0.2093	0.4286	0.2813
9	32 16 8	0.0001	0.3256	0.6667	0.4375
10	32 16 8	0.00001	0.6154	0.7619	0.6809
11	32 16 8	0.000001	0.0	0.0	0.0
12	512 32	0.001	0.2424	0.381	0.2963
13	512 32	0.0001	0.2326	0.4762	0.3125
14	512 32	0.00001	0.2439	0.4762	0.3226
15	512 32	0.000001	0.4286	0.7143	0.5357
16	512 128 8	0	0.0312	0.0476	0.0377
17	512 128 8	0	0.2333	0.3333	0.2745
18	512 128 8	0	0.2703	0.4762	0.3448
19	512 128 8	0	0.3488	0.7143	0.4688
20	512 256 128 65 32 8	0.001	0.2973	0.5238	0.3793
21	512 256 128 65 32 8	0.0001	0.1463	0.2857	0.1935
22	512 256 128 65 32 8	0.00001	0.5	0.8571	0.6316
23	512 256 128 65 32 8	0.000001	0.0417	0.0476	0.0444

Table C.27: Results of LSTM AE, for the Zapcae reduced calls and clients scenario, using feature selection.

Index	Architecture	Learning Rate	Precision	Recall	F1
0	16 8	0.001	0.2143	0.1429	0.1714
1	16 8	0.0001	0.3846	0.3571	0.3704
2	16 8	0.00001	0.1481	0.0952	0.1159
3	16 8	0.000001	0.4286	0.2857	0.3429
4	32 16	0.001	0.5	0.4048	0.4474
5	32 16	0.0001	0.2059	0.1667	0.1842
6	32 16	0.00001	0.3684	0.3333	0.35
7	32 16	0.000001	0.3793	0.2619	0.3099
8	32 16 8	0.001	0.3023	0.3095	0.3059
9	32 16 8	0.0001	0.3243	0.2857	0.3038
10	32 16 8	0.00001	0.4545	0.3571	0.4
11	32 16 8	0.000001	0.4194	0.3095	0.3562
12	512 32	0.001	0.75	0.5714	0.6486
13	512 32	0.0001	0.1892	0.1667	0.1772
14	512 32	0.00001	0.2857	0.2857	0.2857
15	512 32	0.000001	0.3659	0.3571	0.3614
16	512 128 8	0	0.5789	0.5238	0.55
17	512 128 8	0	0.2143	0.1429	0.1714
18	512 128 8	0	0.3409	0.3571	0.3488
19	512 128 8	0	0.3902	0.381	0.3855
20	512 256 128 65 32 8	0.001	0.3333	0.2143	0.2609
21	512 256 128 65 32 8	0.0001	0.1515	0.119	0.1333
22	512 256 128 65 32 8	0.00001	0.5	0.2857	0.3636
23	512 256 128 65 32 8	0.000001	0.1429	0.0952	0.1143

Table C.28: Results of LSTM AE, for the Testpow many nulls for a single column scenario.

Index	Architecture	Learning Rate	Precision	Recall	F1
0	16 8	0.001	0.2051	0.3333	0.254
1	16 8	0.0001	0.2424	0.3333	0.2807
2	16 8	0.00001	0.0	0.0	0.0
3	16 8	0.000001	0.303	0.4167	0.3509
4	32 16	0.001	0.25	0.4583	0.3235
5	32 16	0.0001	0.2059	0.2917	0.2414
6	32 16	0.00001	0.3438	0.4583	0.3929
7	32 16	0.000001	0.4194	0.5417	0.4727
8	32 16 8	0.001	0.3939	0.5417	0.4561
9	32 16 8	0.0001	0.3514	0.5417	0.4262
10	32 16 8	0.00001	0.2353	0.3333	0.2759
11	32 16 8	0.000001	0.0882	0.125	0.1034
12	512 32	0.001	0.0857	0.125	0.1017
13	512 32	0.0001	0.0	0.0	0.0
14	512 32	0.00001	0.1081	0.1667	0.1311
15	512 32	0.000001	0.2812	0.375	0.3214
16	512 128 8	0	0.069	0.0833	0.0755
17	512 128 8	0	0.0938	0.125	0.1071
18	512 128 8	0	0.303	0.4167	0.3509
19	512 128 8	0	0.5278	0.7917	0.6333
20	512 256 128 65 32 8	0.001	0.1111	0.1667	0.1333
21	512 256 128 65 32 8	0.0001	0.1111	0.1667	0.1333
22	512 256 128 65 32 8	0.00001	0.1111	0.1667	0.1333
23	512 256 128 65 32 8	0.000001	0.1944	0.2917	0.2333

Table C.29: Results of LSTM AE, for the Testpow many nulls for multiple columns scenario.

Index	Architecture	Learning Rate	Precision	Recall	F1
0	16 8	0.001	0.2051	0.3333	0.254
1	16 8	0.0001	0.2424	0.3333	0.2807
2	16 8	0.00001	0.0	0.0	0.0
3	16 8	0.000001	0.303	0.4167	0.3509
4	32 16	0.001	0.25	0.4583	0.3235
5	32 16	0.0001	0.2059	0.2917	0.2414
6	32 16	0.00001	0.3438	0.4583	0.3929
7	32 16	0.000001	0.4194	0.5417	0.4727
8	32 16 8	0.001	0.3939	0.5417	0.4561
9	32 16 8	0.0001	0.3514	0.5417	0.4262
10	32 16 8	0.00001	0.2353	0.3333	0.2759
11	32 16 8	0.000001	0.0882	0.125	0.1034
12	512 32	0.001	0.0857	0.125	0.1017
13	512 32	0.0001	0.0	0.0	0.0
14	512 32	0.00001	0.1081	0.1667	0.1311
15	512 32	0.000001	0.2812	0.375	0.3214
16	512 128 8	0	0.069	0.0833	0.0755
17	512 128 8	0	0.0938	0.125	0.1071
18	512 128 8	0	0.303	0.4167	0.3509
19	512 128 8	0	0.5278	0.7917	0.6333
20	512 256 128 65 32 8	0.001	0.1111	0.1667	0.1333
21	512 256 128 65 32 8	0.0001	0.1111	0.1667	0.1333
22	512 256 128 65 32 8	0.00001	0.1111	0.1667	0.1333
23	512 256 128 65 32 8	0.000001	0.1944	0.2917	0.2333

Table C.30: Results of LSTM AE, for the Testpow persistent value scenario.

Index	Architecture	Learning Rate	Precision	Recall	F1
0	16 8	0.001	0.2308	0.375	0.2857
1	16 8	0.0001	0.2857	0.4167	0.339
2	16 8	0.00001	0.0714	0.0833	0.0769
3	16 8	0.000001	0.3143	0.4583	0.3729
4	32 16	0.001	0.2683	0.4583	0.3385
5	32 16	0.0001	0.1379	0.1667	0.1509
6	32 16	0.00001	0.375	0.5	0.4286
7	32 16	0.000001	0.0357	0.0417	0.0385
8	32 16 8	0.001	0.4688	0.625	0.5357
9	32 16 8	0.0001	0.35	0.5833	0.4375
10	32 16 8	0.00001	0.2727	0.375	0.3158
11	32 16 8	0.000001	0.1714	0.25	0.2034
12	512 32	0.001	0.1622	0.25	0.1967
13	512 32	0.0001	0.0606	0.0833	0.0702
14	512 32	0.00001	0.1622	0.25	0.1967
15	512 32	0.000001	0.4	0.5833	0.4746
16	512 128 8	0	0.125	0.1667	0.1429
17	512 128 8	0	0.1724	0.2083	0.1887
18	512 128 8	0	0.5	0.4583	0.4783
19	512 128 8	0	0.3947	0.625	0.4839
20	512 256 128 65 32 8	0.001	0.1667	0.25	0.2
21	512 256 128 65 32 8	0.0001	0.1667	0.25	0.2
22	512 256 128 65 32 8	0.00001	0.1579	0.25	0.1935
23	512 256 128 65 32 8	0.000001	0.3421	0.5417	0.4194

Table C.31: Results of LSTM AE, for the Testpow high variation values for a single column scenario.

Index	Architecture	Learning Rate	Precision	Recall	F1
0	16 8	0.001	0.1667	0.25	0.2
1	16 8	0.0001	0.2424	0.3333	0.2807
2	16 8	0.00001	0.0	0.0	0.0
3	16 8	0.000001	0.2333	0.2917	0.2593
4	32 16	0.001	0.25	0.5	0.3333
5	32 16	0.0001	0.1034	0.125	0.1132
6	32 16	0.00001	0.3143	0.4583	0.3729
7	32 16	0.000001	0.037	0.0417	0.0392
8	32 16 8	0.001	0.4118	0.5833	0.4828
9	32 16 8	0.0001	0.3158	0.5	0.3871
10	32 16 8	0.00001	0.1667	0.2083	0.1852
11	32 16 8	0.000001	0.0323	0.0417	0.0364
12	512 32	0.001	0.0303	0.0417	0.0351
13	512 32	0.0001	0.0	0.0	0.0
14	512 32	0.00001	0.0303	0.0417	0.0351
15	512 32	0.000001	0.2581	0.3333	0.2909
16	512 128 8	0	0.0645	0.0833	0.0727
17	512 128 8	0	0.0714	0.0833	0.0769
18	512 128 8	0	0.3143	0.4583	0.3729
19	512 128 8	0	0.2895	0.4583	0.3548
20	512 256 128 65 32 8	0.001	0.0312	0.0417	0.0357
21	512 256 128 65 32 8	0.0001	0.0312	0.0417	0.0357
22	512 256 128 65 32 8	0.00001	0.0312	0.0417	0.0357
23	512 256 128 65 32 8	0.000001	0.1765	0.25	0.2069

Table C.32: Results of LSTM AE, for the Testpow high variation values for multiple columns scenario.

Index	Architecture	Learning Rate	Precision	Recall	F1
0	16 8	0.001	0.2051	0.3333	0.254
1	16 8	0.0001	0.2424	0.3333	0.2807
2	16 8	0.00001	0.0	0.0	0.0
3	16 8	0.000001	0.303	0.4167	0.3509
4	32 16	0.001	0.25	0.4583	0.3235
5	32 16	0.0001	0.2059	0.2917	0.2414
6	32 16	0.00001	0.3438	0.4583	0.3929
7	32 16	0.000001	0.4194	0.5417	0.4727
8	32 16 8	0.001	0.3939	0.5417	0.4561
9	32 16 8	0.0001	0.3514	0.5417	0.4262
10	32 16 8	0.00001	0.2353	0.3333	0.2759
11	32 16 8	0.000001	0.0882	0.125	0.1034
12	512 32	0.001	0.0857	0.125	0.1017
13	512 32	0.0001	0.0	0.0	0.0
14	512 32	0.00001	0.1081	0.1667	0.1311
15	512 32	0.000001	0.2812	0.375	0.3214
16	512 128 8	0	0.069	0.0833	0.0755
17	512 128 8	0	0.0938	0.125	0.1071
18	512 128 8	0	0.303	0.4167	0.3509
19	512 128 8	0	0.5278	0.7917	0.6333
20	512 256 128 65 32 8	0.001	0.1111	0.1667	0.1333
21	512 256 128 65 32 8	0.0001	0.1111	0.1667	0.1333
22	512 256 128 65 32 8	0.00001	0.1111	0.1667	0.1333
23	512 256 128 65 32 8	0.000001	0.1944	0.2917	0.2333

Table C.33: Results of LSTM AE, for the Testpow many zero values scenario.

Index	Architecture	Learning Rate	Precision	Recall	F1
0	16 8	0.001	0.0909	0.125	0.1053
1	16 8	0.0001	0.2188	0.2917	0.25
2	16 8	0.00001	0.0	0.0	0.0
3	16 8	0.000001	0.129	0.1667	0.1455
4	32 16	0.001	0.1915	0.375	0.2535
5	32 16	0.0001	0.0323	0.0417	0.0364
6	32 16	0.00001	0.0976	0.1667	0.1231
7	32 16	0.000001	0.0	0.0	0.0
8	32 16 8	0.001	0.1622	0.25	0.1967
9	32 16 8	0.0001	0.1892	0.2917	0.2295
10	32 16 8	0.00001	0.037	0.0417	0.0392
11	32 16 8	0.000001	0.0	0.0	0.0
12	512 32	0.001	0.0	0.0	0.0
13	512 32	0.0001	0.0	0.0	0.0
14	512 32	0.00001	0.0	0.0	0.0
15	512 32	0.000001	0.0	0.0	0.0
16	512 128 8	0	0.0	0.0	0.0
17	512 128 8	0	0.0	0.0	0.0
18	512 128 8	0	0.0857	0.125	0.1017
19	512 128 8	0	0.0	0.0	0.0
20	512 256 128 65 32 8	0.001	0.0	0.0	0.0
21	512 256 128 65 32 8	0.0001	0.0	0.0	0.0
22	512 256 128 65 32 8	0.00001	0.0	0.0	0.0
23	512 256 128 65 32 8	0.000001	0.0541	0.0833	0.0656

Table C.34: Results of LSTM AE, for the Testpow many nulls for a single column scenario.

Index	Architecture	Learning Rate	Precision	Recall	F1
0	16 8	0.001	0.0	0.0	0.0
1	16 8	0.0001	0.2857	0.5	0.3636
2	16 8	0.00001	0.0345	0.0417	0.0377
3	16 8	0.000001	0.0526	0.0833	0.0645
4	32 16	0.001	0.1429	0.1667	0.1538
5	32 16	0.0001	0.1875	0.25	0.2143
6	32 16	0.00001	0.1667	0.2083	0.1852
7	32 16	0.000001	0.2162	0.3333	0.2623
8	32 16 8	0.001	0.0	0.0	0.0
9	32 16 8	0.0001	0.0	0.0	0.0
10	32 16 8	0.00001	0.0	0.0	0.0
11	32 16 8	0.000001	0.0833	0.125	0.1
12	512 32	0.001	0.1724	0.2083	0.1887
13	512 32	0.0001	0.2593	0.2917	0.2745
14	512 32	0.00001	0.0909	0.125	0.1053
15	512 32	0.000001	0.0588	0.0833	0.069
16	512 128 8	0	0.1724	0.2083	0.1887
17	512 128 8	0	0.1724	0.2083	0.1887
18	512 128 8	0	0.129	0.1667	0.1455
19	512 128 8	0	0.125	0.1667	0.1429
20	512 256 128 65 32 8	0.001	0.1724	0.2083	0.1887
21	512 256 128 65 32 8	0.0001	0.1724	0.2083	0.1887
22	512 256 128 65 32 8	0.00001	0.1613	0.2083	0.1818
23	512 256 128 65 32 8	0.000001	0.1379	0.1667	0.1509

Table C.35: Results of LSTM AE, for the Testpow many nulls for multiple columns scenario.

Index	Architecture	Learning Rate	Precision	Recall	F1
0	16 8	0.001	0.0	0.0	0.0
1	16 8	0.0001	0.4839	0.625	0.5455
2	16 8	0.00001	0.0667	0.0833	0.0741
3	16 8	0.000001	0.0526	0.0833	0.0645
4	32 16	0.001	0.2692	0.2917	0.28
5	32 16	0.0001	0.2333	0.2917	0.2593
6	32 16	0.00001	0.1935	0.25	0.2182
7	32 16	0.000001	0.2973	0.4583	0.3607
8	32 16 8	0.001	0.16	0.1667	0.1633
9	32 16 8	0.0001	0.0938	0.125	0.1071
10	32 16 8	0.00001	0.0	0.0	0.0
11	32 16 8	0.000001	0.0278	0.0417	0.0333
12	512 32	0.001	0.2727	0.375	0.3158
13	512 32	0.0001	0.24	0.25	0.2449
14	512 32	0.00001	0.3611	0.5417	0.4333
15	512 32	0.000001	0.1818	0.25	0.2105
16	512 128 8	0	0.2727	0.375	0.3158
17	512 128 8	0	0.2727	0.375	0.3158
18	512 128 8	0	0.2727	0.375	0.3158
19	512 128 8	0	0.2647	0.375	0.3103
20	512 256 128 65 32 8	0.001	0.2727	0.375	0.3158
21	512 256 128 65 32 8	0.0001	0.2727	0.375	0.3158
22	512 256 128 65 32 8	0.00001	0.2941	0.4167	0.3448
23	512 256 128 65 32 8	0.000001	0.2581	0.3333	0.2909

Table C.36: Results of LSTM AE, for the Testpow persistent value scenario.

Index	Architecture	Learning Rate	Precision	Recall	F1
0	16 8	0.001	0.0	0.0	0.0
1	16 8	0.0001	0.5357	0.625	0.5769
2	16 8	0.00001	0.45	0.375	0.4091
3	16 8	0.000001	0.2667	0.5	0.3478
4	32 16	0.001	0.25	0.25	0.25
5	32 16	0.0001	0.2581	0.3333	0.2909
6	32 16	0.00001	0.3636	0.5	0.4211
7	32 16	0.000001	0.375	0.5	0.4286
8	32 16 8	0.001	0.16	0.1667	0.1633
9	32 16 8	0.0001	0.037	0.0417	0.0392
10	32 16 8	0.00001	0.0	0.0	0.0
11	32 16 8	0.000001	0.0	0.0	0.0
12	512 32	0.001	0.2647	0.375	0.3103
13	512 32	0.0001	0.0	0.0	0.0
14	512 32	0.00001	0.4	0.5833	0.4746
15	512 32	0.000001	0.225	0.375	0.2813
16	512 128 8	0	0.2647	0.375	0.3103
17	512 128 8	0	0.2647	0.375	0.3103
18	512 128 8	0	0.2857	0.4167	0.339
19	512 128 8	0	0.2632	0.4167	0.3226
20	512 256 128 65 32 8	0.001	0.2647	0.375	0.3103
21	512 256 128 65 32 8	0.0001	0.2647	0.375	0.3103
22	512 256 128 65 32 8	0.00001	0.2973	0.4583	0.3607
23	512 256 128 65 32 8	0.000001	0.2286	0.3333	0.2712

Table C.37: Results of LSTM AE, for the Testpow high variation values for a single column scenario.

Index	Architecture	Learning Rate	Precision	Recall	F1
0	16 8	0.001	0.0	0.0	0.0
1	16 8	0.0001	0.2	0.25	0.2222
2	16 8	0.00001	0.1875	0.25	0.2143
3	16 8	0.000001	0.075	0.125	0.0937
4	32 16	0.001	0.1071	0.125	0.1154
5	32 16	0.0001	0.0	0.0	0.0
6	32 16	0.00001	0.1379	0.1667	0.1509
7	32 16	0.000001	0.0769	0.0833	0.08
8	32 16 8	0.001	0.0	0.0	0.0
9	32 16 8	0.0001	0.0	0.0	0.0
10	32 16 8	0.00001	0.0	0.0	0.0
11	32 16 8	0.000001	0.0526	0.0833	0.0645
12	512 32	0.001	0.037	0.0417	0.0392
13	512 32	0.0001	0.0	0.0	0.0
14	512 32	0.00001	0.0556	0.0833	0.0667
15	512 32	0.000001	0.0556	0.0833	0.0667
16	512 128 8	0	0.0357	0.0417	0.0385
17	512 128 8	0	0.0357	0.0417	0.0385
18	512 128 8	0	0.037	0.0417	0.0392
19	512 128 8	0	0.0333	0.0417	0.037
20	512 256 128 65 32 8	0.001	0.0357	0.0417	0.0385
21	512 256 128 65 32 8	0.0001	0.0357	0.0417	0.0385
22	512 256 128 65 32 8	0.00001	0.0385	0.0417	0.04
23	512 256 128 65 32 8	0.000001	0.0357	0.0417	0.0385

Table C.38: Results of LSTM AE, for the Testpow high variation values for multiple columns scenario.

Index	Architecture	Learning Rate	Precision	Recall	F1
0	16 8	0.001	0.0	0.0	0.0
1	16 8	0.0001	0.4839	0.625	0.5455
2	16 8	0.00001	0.0667	0.0833	0.0741
3	16 8	0.000001	0.0526	0.0833	0.0645
4	32 16	0.001	0.2692	0.2917	0.28
5	32 16	0.0001	0.2333	0.2917	0.2593
6	32 16	0.00001	0.1935	0.25	0.2182
7	32 16	0.000001	0.2973	0.4583	0.3607
8	32 16 8	0.001	0.16	0.1667	0.1633
9	32 16 8	0.0001	0.0938	0.125	0.1071
10	32 16 8	0.00001	0.0	0.0	0.0
11	32 16 8	0.000001	0.0278	0.0417	0.0333
12	512 32	0.001	0.2727	0.375	0.3158
13	512 32	0.0001	0.24	0.25	0.2449
14	512 32	0.00001	0.3611	0.5417	0.4333
15	512 32	0.000001	0.1818	0.25	0.2105
16	512 128 8	0	0.2727	0.375	0.3158
17	512 128 8	0	0.2727	0.375	0.3158
18	512 128 8	0	0.2727	0.375	0.3158
19	512 128 8	0	0.2647	0.375	0.3103
20	512 256 128 65 32 8	0.001	0.2727	0.375	0.3158
21	512 256 128 65 32 8	0.0001	0.2727	0.375	0.3158
22	512 256 128 65 32 8	0.00001	0.2941	0.4167	0.3448
23	512 256 128 65 32 8	0.000001	0.2581	0.3333	0.2909

Table C.39: Results of LSTM AE, for the Testpow many zero values scenario.

Index	Architecture	Learning Rate	Precision	Recall	F1
0	16 8	0.001	0.0	0.0	0.0
1	16 8	0.0001	0.0	0.0	0.0
2	16 8	0.00001	0.1613	0.2083	0.1818
3	16 8	0.000001	0.0	0.0	0.0
4	32 16	0.001	0.0	0.0	0.0
5	32 16	0.0001	0.0	0.0	0.0
6	32 16	0.00001	0.0	0.0	0.0
7	32 16	0.000001	0.0	0.0	0.0
8	32 16 8	0.001	0.0	0.0	0.0
9	32 16 8	0.0001	0.0	0.0	0.0
10	32 16 8	0.00001	0.0	0.0	0.0
11	32 16 8	0.000001	0.0526	0.0833	0.0645
12	512 32	0.001	0.0	0.0	0.0
13	512 32	0.0001	0.0	0.0	0.0
14	512 32	0.00001	0.0	0.0	0.0
15	512 32	0.000001	0.0	0.0	0.0
16	512 128 8	0	0.0	0.0	0.0
17	512 128 8	0	0.0	0.0	0.0
18	512 128 8	0	0.0	0.0	0.0
19	512 128 8	0	0.0	0.0	0.0
20	512 256 128 65 32 8	0.001	0.0	0.0	0.0
21	512 256 128 65 32 8	0.0001	0.0	0.0	0.0
22	512 256 128 65 32 8	0.00001	0.0	0.0	0.0
23	512 256 128 65 32 8	0.000001	0.0	0.0	0.0

Table C.40: Results of TCN AE, for the Zapcae reduced success rates scenario.

Index	Architecture	Learning Rate	Precision	Recall	F1
0	32 8	0.001	0.25	0.2857	0.2667
1	32 8	0.0001	0.0278	0.0286	0.0282
2	64 8	0.001	0.3415	0.4	0.3684
3	64 8	0.0001	0.4571	0.4571	0.4571
4	32 16 8	0.001	0.0	0.0	0.0
5	32 16 8	0.0001	0.4118	0.4	0.4058
6	128 8	0.001	0.4	0.4	0.4
7	128 8	0.0001	0.1351	0.1429	0.1389
8	64 16 8	0.001	0.5714	0.5714	0.5714
9	64 16 8	0.0001	0.7	0.4	0.5091
10	512 8	0.001	0.4118	0.4	0.4058
11	512 8	0.0001	0.3889	0.4	0.3944
12	512 128 8	0.001	0.0	0.0	0.0
13	512 128 8	0.0001	0.1163	0.1429	0.1282
14	512 254 64 8	0.001	0.12	0.1714	0.1412
15	512 254 64 8	0.0001	0.0	0.0	0.0

Table C.41: Results of TCN AE, for the Zapcae high reduction of success rates scenario.

Index	Architecture	Learning Rate	Precision	Recall	F1
0	32 8	0.001	0.25	0.2857	0.2667
1	32 8	0.0001	0.0278	0.0286	0.0282
2	64 8	0.001	0.3415	0.4	0.3684
3	64 8	0.0001	0.4571	0.4571	0.4571
4	32 16 8	0.001	0.0	0.0	0.0
5	32 16 8	0.0001	0.4118	0.4	0.4058
6	128 8	0.001	0.4	0.4	0.4
7	128 8	0.0001	0.1351	0.1429	0.1389
8	64 16 8	0.001	0.5714	0.5714	0.5714
9	64 16 8	0.0001	0.7	0.4	0.5091
10	512 8	0.001	0.4118	0.4	0.4058
11	512 8	0.0001	0.3889	0.4	0.3944
12	512 128 8	0.001	0.0	0.0	0.0
13	512 128 8	0.0001	0.1163	0.1429	0.1282
14	512 254 64 8	0.001	0.12	0.1714	0.1412
15	512 254 64 8	0.0001	0.0	0.0	0.0

Table C.42: Results of TCN AE, for the Zapcae increased drop rates scenario.

Index	Architecture	Learning Rate	Precision	Recall	F1
0	32 8	0.001	0.2778	0.2857	0.2817
1	32 8	0.0001	0.0345	0.0286	0.0312
2	64 8	0.001	0.2642	0.4	0.3182
3	64 8	0.0001	0.4054	0.4286	0.4167
4	32 16 8	0.001	0.0	0.0	0.0
5	32 16 8	0.0001	0.4	0.4	0.4
6	128 8	0.001	0.2667	0.2286	0.2462
7	128 8	0.0001	0.1053	0.1143	0.1096
8	64 16 8	0.001	0.4848	0.4571	0.4706
9	64 16 8	0.0001	0.7	0.4	0.5091
10	512 8	0.001	0.4242	0.4	0.4118
11	512 8	0.0001	0.3784	0.4	0.3889
12	512 128 8	0.001	0.0	0.0	0.0
13	512 128 8	0.0001	0.1163	0.1429	0.1282
14	512 254 64 8	0.001	0.1176	0.1714	0.1395
15	512 254 64 8	0.0001	0.0	0.0	0.0

Table C.43: Results of TCN AE, for the Zapcae increased calls and clients scenario.

Index	Architecture	Learning Rate	Precision	Recall	F1
0	32 8	0.001	0.0256	0.0476	0.0333
1	32 8	0.0001	0.0	0.0	0.0
2	64 8	0.001	0.0	0.0	0.0
3	64 8	0.0001	0.1667	0.2857	0.2105
4	32 16 8	0.001	0.0	0.0	0.0
5	32 16 8	0.0001	0.0	0.0	0.0
6	128 8	0.001	0.0	0.0	0.0
7	128 8	0.0001	0.05	0.0952	0.0656
8	64 16 8	0.001	0.4167	0.7143	0.5263
9	64 16 8	0.0001	0.0	0.0	0.0
10	512 8	0.001	0.0	0.0	0.0
11	512 8	0.0001	0.0	0.0	0.0
12	512 128 8	0.001	0.0	0.0	0.0
13	512 128 8	0.0001	0.0435	0.0952	0.0597
14	512 254 64 8	0.001	0.0909	0.2381	0.1316
15	512 254 64 8	0.0001	0.0	0.0	0.0

Table C.44: Results of TCN AE, for the Zapcae reduced calls and clients scenario.

Index	Architecture	Learning Rate	Precision	Recall	F1
0	32 8	0.001	0.0	0.0	0.0
1	32 8	0.0001	0.0	0.0	0.0
2	64 8	0.001	0.0	0.0	0.0
3	64 8	0.0001	0.1111	0.1905	0.1404
4	32 16 8	0.001	0.0	0.0	0.0
5	32 16 8	0.0001	0.0	0.0	0.0
6	128 8	0.001	0.0	0.0	0.0
7	128 8	0.0001	0.0	0.0	0.0
8	64 16 8	0.001	0.4412	0.7143	0.5455
9	64 16 8	0.0001	0.0	0.0	0.0
10	512 8	0.001	0.0	0.0	0.0
11	512 8	0.0001	0.0	0.0	0.0
12	512 128 8	0.001	0.0	0.0	0.0
13	512 128 8	0.0001	0.1163	0.2381	0.1562
14	512 254 64 8	0.001	0.1176	0.2857	0.1667
15	512 254 64 8	0.0001	0.0	0.0	0.0

Table C.45: Results of TCN AE, for the Testpow many nulls for a single column scenario, using feature selection.

Index	Architecture	Learning Rate	Precision	Recall	F1
0	32 8	0.001	0.0	0.0	0.0
1	32 8	0.0001	0.3846	0.4167	0.4
2	32 16	0.001	0.08	0.0833	0.0816
3	32 16	0.0001	0.0	0.0	0.0
4	128 16	0.001	0.0	0.0	0.0
5	128 16	0.0001	0.1579	0.25	0.1935

Table C.46: Results of TCN AE, for the Testpow many nulls for multiple columns scenario, using feature selection.

Index	Architecture	Learning Rate	Precision	Recall	F1
0	32 8	0.001	0.0	0.0	0.0
1	32 8	0.0001	0.5652	0.5417	0.5532
2	32 16	0.001	0.08	0.0833	0.0816
3	32 16	0.0001	0.0	0.0	0.0
4	128 16	0.001	0.0	0.0	0.0
5	128 16	0.0001	0.2821	0.4583	0.3492

Table C.47: Results of TCN AE, for the Testpow persistent value scenario, using feature selection.

Index	Architecture	Learning Rate	Precision	Recall	F1
0	32 8	0.001	0.0	0.0	0.0
1	32 8	0.0001	0.619	0.5417	0.5778
2	32 16	0.001	0.1071	0.125	0.1154
3	32 16	0.0001	0.0	0.0	0.0
4	128 16	0.001	0.0	0.0	0.0
5	128 16	0.0001	0.3421	0.5417	0.4194

Table C.48: Results of TCN AE, for the Testpow high variation values for a single column scenario, using feature selection.

Index	Architecture	Learning Rate	Precision	Recall	F1
0	32 8	0.001	0.0	0.0	0.0
1	32 8	0.0001	0.3889	0.2917	0.3333
2	32 16	0.001	0.1379	0.1667	0.1509
3	32 16	0.0001	0.0	0.0	0.0
4	128 16	0.001	0.0	0.0	0.0
5	128 16	0.0001	0.0882	0.125	0.1034

Table C.49: Results of TCN AE, for the Testpow high variation values for multiple columns scenario, using feature selection.

Index	Architecture	Learning Rate	Precision	Recall	F1
0	32 8	0.001	0.0	0.0	0.0
1	32 8	0.0001	0.5652	0.5417	0.5532
2	32 16	0.001	0.08	0.0833	0.0816
3	32 16	0.0001	0.0	0.0	0.0
4	128 16	0.001	0.0	0.0	0.0
5	128 16	0.0001	0.2821	0.4583	0.3492

Table C.50: Results of TCN AE, for the Testpow many zero values scenario, using feature selection.

Index	Architecture	Learning Rate	Precision	Recall	F1
0	32 8	0.001	0.0	0.0	0.0
1	32 8	0.0001	0.0	0.0	0.0
2	32 16	0.001	0.0741	0.0833	0.0784
3	32 16	0.0001	0.0	0.0	0.0
4	128 16	0.001	0.0	0.0	0.0
5	128 16	0.0001	0.0233	0.0417	0.0299

Table C.51: Results of TCN AE, for the Testpow many nulls for a single column scenario, with a window width of 7.

Index	Architecture	Learning Rate	Precision	Recall	F1
0	32 8	0.0001	0.2195	0.5294	0.3103
1	32 8	0.00001	0.1333	0.2353	0.1702
2	32 8	0.001	0.2581	0.4706	0.3333
3	32 16 8	0.0001	0.3226	0.5882	0.4167
4	32 16 8	0.00001	0.1944	0.4118	0.2642
5	32 16 8	0.001	0.1429	0.3529	0.2034
6	128 16	0.0001	0.2121	0.4118	0.28
7	128 16	0.00001	0.2188	0.4118	0.2857
8	128 16	0.001	0.0556	0.1176	0.0755

Table C.52: Results of TCN AE, for the Testpow many nulls for multiple columns scenario, with a window width of 7.

Index	Architecture	Learning Rate	Precision	Recall	F1
0	32 8	0.0001	0.2195	0.5294	0.3103
1	32 8	0.00001	0.1333	0.2353	0.1702
2	32 8	0.001	0.2581	0.4706	0.3333
3	32 16 8	0.0001	0.3226	0.5882	0.4167
4	32 16 8	0.00001	0.1944	0.4118	0.2642
5	32 16 8	0.001	0.1429	0.3529	0.2034
6	128 16	0.0001	0.2121	0.4118	0.28
7	128 16	0.00001	0.2188	0.4118	0.2857
8	128 16	0.001	0.0556	0.1176	0.0755

Table C.53: Results of TCN AE, for the Testpow persistent value scenario, with a window width of 7.

Index	Architecture	Learning Rate	Precision	Recall	F1
0	32 8	0.0001	0.2162	0.4706	0.2963
1	32 8	0.00001	0.1613	0.2941	0.2083
2	32 8	0.001	0.2812	0.5294	0.3673
3	32 16 8	0.0001	0.4	0.7059	0.5106
4	32 16 8	0.00001	0.1944	0.4118	0.2642
5	32 16 8	0.001	0.1538	0.3529	0.2143
6	128 16	0.0001	0.2647	0.5294	0.3529
7	128 16	0.00001	0.2143	0.3529	0.2667
8	128 16	0.001	0.0789	0.1765	0.1091

Table C.54: Results of TCN AE, for the Testpow high variation values for a single column scenario, with a window width of 7.

Index	Architecture	Learning Rate	Precision	Recall	F1
0	32 8	0.0001	0.1579	0.3529	0.2182
1	32 8	0.00001	0.0312	0.0588	0.0408
2	32 8	0.001	0.2333	0.4118	0.2979
3	32 16 8	0.0001	0.3125	0.5882	0.4082
4	32 16 8	0.00001	0.1622	0.3529	0.2222
5	32 16 8	0.001	0.075	0.1765	0.1053
6	128 16	0.0001	0.1515	0.2941	0.2
7	128 16	0.00001	0.1111	0.1765	0.1364
8	128 16	0.001	0.0541	0.1176	0.0741

Table C.55: Results of TCN AE, for the Testpow high variation values for multiple columns scenario, with a window width of 7.

Index	Architecture	Learning Rate	Precision	Recall	F1
0	32 8	0.0001	0.2195	0.5294	0.3103
1	32 8	0.00001	0.1333	0.2353	0.1702
2	32 8	0.001	0.2581	0.4706	0.3333
3	32 16 8	0.0001	0.3226	0.5882	0.4167
4	32 16 8	0.00001	0.1944	0.4118	0.2642
5	32 16 8	0.001	0.1429	0.3529	0.2034
6	128 16	0.0001	0.2121	0.4118	0.28
7	128 16	0.00001	0.2188	0.4118	0.2857
8	128 16	0.001	0.0556	0.1176	0.0755

Table C.56: Results of TCN AE, for the Testpow many zero values scenario, with a window width of 7.

Index	Architecture	Learning Rate	Precision	Recall	F1
0	32 8	0.0001	0.0263	0.0588	0.0364
1	32 8	0.00001	0.0	0.0	0.0
2	32 8	0.001	0.0741	0.1176	0.0909
3	32 16 8	0.0001	0.08	0.1176	0.0952
4	32 16 8	0.00001	0.0606	0.1176	0.08
5	32 16 8	0.001	0.0256	0.0588	0.0357
6	128 16	0.0001	0.0	0.0	0.0
7	128 16	0.00001	0.0	0.0	0.0
8	128 16	0.001	0.0	0.0	0.0

Table C.57: Results of LSTM BAE, for the Zapcae reduced success rates scenario.

Index	Architecture	Learning Rate	Precision	Recall	F1
0	512 128 8	0.00001	0.4	0.4	0.4
1	16 8	0.001	1.0	0.8571	0.9231
2	16 8	0.0001	0.8889	0.7619	0.8205
3	16 8	0.00001	0.7333	0.5238	0.6111
4	32 16 8	0.001	0.8438	0.6429	0.7297
5	32 16 8	0.0001	1.0	0.5714	0.7273
6	32 16 8	0.00001	0.3191	0.3571	0.3371
7	512 32	0.001	0.8421	0.7619	0.8
8	512 32	0.0001	0.8378	0.7381	0.7848
9	512 32	0.00001	0.9394	0.7381	0.8267
0	512 256 128 65 32 8	0.001	0.9355	0.6905	0.7945
11	512 256 128 65 32 8	0.0001	0.9677	0.7143	0.8219
12	512 256 128 65 32 8	0.00001	0.7188	0.5476	0.6216

Table C.58: Results of LSTM BAE, for the Zapcae high reduction of success rates scenario.

Index	Architecture	Learning Rate	Precision	Recall	F1
0	512 128 8	0.00001	0.4	0.4	0.4
1	16 8	0.001	1.0	0.8571	0.9231
2	16 8	0.0001	0.8947	0.8095	0.85
3	16 8	0.00001	0.75	0.5714	0.6486
4	32 16 8	0.001	0.8438	0.6429	0.7297
5	32 16 8	0.0001	1.0	0.5952	0.7463
6	32 16 8	0.00001	0.3043	0.3333	0.3182
7	512 32	0.001	0.8421	0.7619	0.8
8	512 32	0.0001	0.8421	0.7619	0.8
9	512 32	0.00001	0.9412	0.7619	0.8421
0	512 256 128 65 32 8	0.001	0.9355	0.6905	0.7945
11	512 256 128 65 32 8	0.0001	0.9375	0.7143	0.8108
12	512 256 128 65 32 8	0.00001	0.7742	0.5714	0.6575

Table C.59: Results of LSTM BAE, for the Zapcae increased drop rates scenario.

Index	Architecture	Learning Rate	Precision	Recall	F1
0	512 128 8	0.00001	0.3636	0.1143	0.1739
1	16 8	0.001	0.4737	0.4286	0.45
2	16 8	0.0001	0.2692	0.3333	0.2979
3	16 8	0.00001	0.2727	0.2143	0.24
4	32 16 8	0.001	0.2	0.1667	0.1818
5	32 16 8	0.0001	0.4048	0.4048	0.4048
6	32 16 8	0.00001	0.3514	0.3095	0.3291
7	512 32	0.001	0.5263	0.4762	0.5
8	512 32	0.0001	0.3429	0.2857	0.3117
9	512 32	0.00001	0.3061	0.3571	0.3297
0	512 256 128 65 32 8	0.001	0.4643	0.3095	0.3714
11	512 256 128 65 32 8	0.0001	0.2439	0.2381	0.241
12	512 256 128 65 32 8	0.00001	0.375	0.2143	0.2727

Table C.60: Results of LSTM BAE, for the Zapcae increased calls and clients scenario.

Index	Architecture	Learning Rate	Precision	Recall	F1
0	512 128 8	0.00001	0.2308	0.1429	0.1765
1	16 8	0.001	0.2424	0.381	0.2963
2	16 8	0.0001	0.2128	0.4762	0.2941
3	16 8	0.00001	0.1613	0.2381	0.1923
4	32 16 8	0.001	0.1	0.1429	0.1176
5	32 16 8	0.0001	0.4	0.5714	0.4706
6	32 16 8	0.00001	0.2381	0.4762	0.3175
7	512 32	0.001	0.1538	0.2857	0.2
8	512 32	0.0001	0.2286	0.381	0.2857
9	512 32	0.00001	0.1731	0.4286	0.2466
0	512 256 128 65 32 8	0.001	0.3636	0.5714	0.4444
11	512 256 128 65 32 8	0.0001	0.303	0.4762	0.3704
12	512 256 128 65 32 8	0.00001	0.3548	0.5238	0.4231

Table C.61: Results of LSTM BAE, for the Zapcae reduced calls and clients scenario.

Index	Architecture	Learning Rate	Precision	Recall	F1
0	512 128 8	0.00001	0.0	0.0	0.0
1	16 8	0.001	0.3824	0.619	0.4727
2	16 8	0.0001	0.1915	0.4286	0.2647
3	16 8	0.00001	0.069	0.0952	0.08
4	32 16 8	0.001	0.25	0.4286	0.3158
5	32 16 8	0.0001	0.2143	0.4286	0.2857
6	32 16 8	0.00001	0.225	0.4286	0.2951
7	512 32	0.001	0.4167	0.7143	0.5263
8	512 32	0.0001	0.2333	0.3333	0.2745
9	512 32	0.00001	0.1304	0.2857	0.1791
0	512 256 128 65 32 8	0.001	0.4062	0.619	0.4906
11	512 256 128 65 32 8	0.0001	0.3	0.5714	0.3934
12	512 256 128 65 32 8	0.00001	0.32	0.381	0.3478

Table C.62: Results of LSTM BAE, for the Testpow many nulls for a single column scenario.

Index	Architecture	Learning Rate	Precision	Recall	F1
0	32 16	0.000001	0.5526	0.875	0.6774
1	32 16	0.00001	0.1304	0.125	0.1277
2	512 128 16	0.000001	0.1	0.0833	0.0909
3	512 128 16	0.00001	0.0968	0.125	0.1091
4	16 8	0.000001	0.0312	0.0417	0.0357
5	16 8	0.00001	0.037	0.0417	0.0392
6	32 16 8	0.000001	0.2353	0.3333	0.2759
7	32 16 8	0.00001	0.3889	0.5833	0.4667
8	512 256 128 65 32 8	0.000001	0.0714	0.0833	0.0769
9	512 256 128 65 32 8	0.00001	0.3333	0.4167	0.3704

Table C.63: Results of LSTM BAE, for the Testpow many nulls for multiple columns scenario.

Index	Architecture	Learning Rate	Precision	Recall	F1
0	32 16	0.000001	0.5526	0.875	0.6774
1	32 16	0.00001	0.1304	0.125	0.1277
2	512 128 16	0.000001	0.0526	0.0417	0.0465
3	512 128 16	0.00001	0.0968	0.125	0.1091
4	16 8	0.000001	0.0323	0.0417	0.0364
5	16 8	0.00001	0.037	0.0417	0.0392
6	32 16 8	0.000001	0.2353	0.3333	0.2759
7	32 16 8	0.00001	0.4118	0.5833	0.4828
8	512 256 128 65 32 8	0.000001	0.037	0.0417	0.0392
9	512 256 128 65 32 8	0.00001	0.3125	0.4167	0.3571

Table C.64: Results of LSTM BAE, for the Testpow persistent value scenario.

Index	Architecture	Learning Rate	Precision	Recall	F1
0	32 16	0.000001	0.7059	1.0	0.8276
1	32 16	0.00001	0.4091	0.375	0.3913
2	512 128 16	0.000001	0.5	0.625	0.5556
3	512 128 16	0.00001	0.3514	0.5417	0.4262
4	16 8	0.000001	0.0333	0.0417	0.037
5	16 8	0.00001	0.4242	0.5833	0.4912
6	32 16 8	0.000001	0.88	0.9167	0.898
7	32 16 8	0.00001	0.5278	0.7917	0.6333
8	512 256 128 65 32 8	0.000001	0.5455	0.75	0.6316
9	512 256 128 65 32 8	0.00001	1.0	0.7917	0.8837

Table C.65: Results of LSTM BAE, for the Testpow high variation values for a single column scenario.

Index	Architecture	Learning Rate	Precision	Recall	F1
0	32 16	0.000001	0.5641	0.9167	0.6984
1	32 16	0.00001	0.05	0.0417	0.0455
2	512 128 16	0.000001	0.1429	0.125	0.1333
3	512 128 16	0.00001	0.2368	0.375	0.2903
4	16 8	0.000001	0.0312	0.0417	0.0357
5	16 8	0.00001	0.2667	0.3333	0.2963
6	32 16 8	0.000001	0.3421	0.5417	0.4194
7	32 16 8	0.00001	0.3659	0.625	0.4615
8	512 256 128 65 32 8	0.000001	0.2143	0.25	0.2308
9	512 256 128 65 32 8	0.00001	0.6667	0.75	0.7059

Table C.66: Results of LSTM BAE, for the Testpow high variation values for multiple columns scenario.

Index	Architecture	Learning Rate	Precision	Recall	F1
0	32 16	0.000001	0.5526	0.875	0.6774
1	32 16	0.00001	0.1304	0.125	0.1277
2	512 128 16	0.000001	0.0526	0.0417	0.0465
3	512 128 16	0.00001	0.0968	0.125	0.1091
4	16 8	0.000001	0.0323	0.0417	0.0364
5	16 8	0.00001	0.037	0.0417	0.0392
6	32 16 8	0.000001	0.2353	0.3333	0.2759
7	32 16 8	0.00001	0.4118	0.5833	0.4828
8	512 256 128 65 32 8	0.000001	0.037	0.0417	0.0392
9	512 256 128 65 32 8	0.00001	0.3125	0.4167	0.3571

Table C.67: Results of LSTM BAE, for the Testpow many zero values scenario.

Index	Architecture	Learning Rate	Precision	Recall	F1
0	32 16	0.000001	0.5676	0.875	0.6885
1	32 16	0.00001	0.0	0.0	0.0
2	512 128 16	0.000001	0.05	0.0417	0.0455
3	512 128 16	0.00001	0.1622	0.25	0.1967
4	16 8	0.000001	0.0345	0.0417	0.0377
5	16 8	0.00001	0.1852	0.2083	0.1961
6	32 16 8	0.000001	0.027	0.0417	0.0328
7	32 16 8	0.00001	0.3488	0.625	0.4478
8	512 256 128 65 32 8	0.000001	0.1852	0.2083	0.1961
9	512 256 128 65 32 8	0.00001	0.2353	0.3333	0.2759