Anomaly detection in AWS load balancer logs

Jankuleski, Marko

University of Split
Faculty of Science

GRADUATE THESIS

ANOMALY DETECTION IN AWS LOAD BALANCER LOGS

Marko Jankuleski

Split, 2023.

Basic documentation card

Graduate thesis

University of Split

Faculty of Science

Department of Computer Science

Ruđera Boškovića 33, 21000 Split, Croatia

ANOMALY DETECTION IN AWS LOAD BALANCER LOGS

Marko Jankuleski

ABSTRACT

Amazon Web Services (AWS) is a widely used cloud platform. Load Balancing service that it uses is the Elastic Load Balancing service. Among its load-balancing options, the Application Load Balancer (ALB) is crucial for businesses as it routes incoming requests to one or multiple targets. In this thesis we aimed to explore the current scope of research on anomaly detection in ALB logs and have concluded that there is limited research evaluating ALB's anomaly detection in log files, and that no benchmark dataset for AWS ALB research is widely available. Additionally, we wanted to look at some computational approaches with logs and anomaly detection and have thus assembled a set of log files from an AWS Application Load Balancer from a private company and have touched upon various anomaly detection methods applied to these logs considering that we have not had enough time to label the full data we have collected. In our work, we have been guided by a systematic review of literature in the fields of time series anomaly detection and log file anomaly detection.

Keywords: anomaly detection, Amazon Web Services, log files, statistical method, ARIMA, Autoencoder

Graduate thesis deposited in library of Faculty of science, University of Split

Thesis consists of: 72 pages, 28 figures, 9 tables and 21 references

Original language: English

Mentor: **Ivo Ugrina, Ph.D.** *Assistant Professor, Faculty of Science, University of Split*

Reviewers: **Ivo Ugrina, Ph.D.** *Assistant Professor, Faculty of Science, University of Split*

Goran Zaharija, Ph.D. *Assistant Professor, Faculty of Science, University of Split*

Milica Klaričić Bakula, Ph.D. *Full Professor, Faculty of Science, University of Split*

Thesis accepted: **October 2023**

Temeljna dokumentacijska kartica

Diplomski rad

Sveučilište u Splitu

Prirodoslovno-matematički fakultet

Odjel za informatiku

Ruđera Boškovića 33, 21000 Split, Hrvatska

DETEKCIJA ANOMALIJA U LOGOVIMA AWS BALANSERA OPTEREĆENJA

Marko Jankuleski

SAŽETAK

Amazon Web Services (AWS) široko je korištena platforma u oblaku. Za balansiranje opterećenja koristi Elastic Load Balancing. Među opcijama za balansiranje opterećenja, Application Load Balancer ključan je za poslovanje tvrtki jer upravlja dolaznim zahtjevima na način da ih prosljeđuje jednom ili vise ciljeva. Nismo pronašli znanstvene radove koji se bave detekcijom anomalija koristeći Amazon CloudWatch sustav za detekciju anomalija i detekcijom anomalija u logovima AWS balansera opterećenja te nismo pronašli skup podatka za AWS koji bi se koristio kao referentni skup podataka. Kako bi to riješili, u okviru istraživanja smo prikupili logove AWS Application Load Balancera i procijenili smo različite metode detekcije anomalija, vođeni sustavnim pregledom istraženosti područja detekcije anomalija u vremenskim nizovima i detekcije anomalija u logovima balansera opterećenja.

Ključne riječi: detekcija anomalija, Amazon web servisi, log datoteke, statistička metoda, ARIMA, autoenkoder

Rad je pohranjen u knjižnici Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Rad sadrži: 72 stranice, 28 grafičkih prikaza, 9 tablica i 21 literaturnih navoda. Izvornik je na engleskom jeziku.

Mentor: **Dr. sc. Ivo Ugrina,** *docent, Prirodoslovno-matematički fakultet, Sveučilište u Splitu*

Ocjenjivači: **Dr. sc. Ivo Ugrina,** *docent, Prirodoslovno-matematički fakultet, Sveučilište u Splitu*

Dr. sc. Goran Zaharija, *docent, Prirodoslovno-matematički fakultet, Sveučilište u Splitu*

Dr. sc. Milica Klaričić Bakula, *redoviti profesor, Prirodoslovno-matematički fakultet, Sveučilište u Splitu*

Rad prihvaćen: **listopad 2023**

Table of Contents

DECLARATION

of the independent preparation of the graduate thesis

I declare under full material and moral responsibility that I have independently created this work and that it does not contain copied or duplicated parts of text from the works of others unless appropriately marked as quotations with the specified source from which they were taken.

In Split, 04.10.2023 Marko Jankuleski

(student)

Introduction

Amazon Web Services (AWS) is the world's most all-encompassing and broadly adopted cloud. It offers its solutions for load balancing, called Elastic Load Balancing. From the perspective of businesses and clients, probably the most significant one is the Application Load Balancer. Application Load Balancer serves as a single point of contact for clients as it distributes incoming requests to one or multiple targets. Since application availability time is crucial for business, the load balancer would benefit from a check, or from a connected service to execute checks, of all requests it receives before sending them to targets as anomalous requests can be the cause of the downtime.

We looked at the existing literature on anomaly detection from AWS load balancer logs and through a systematic review of the literature we did not find publications that address anomaly detection in log files of the AWS load balancer. Furthermore, we did not find a benchmark dataset for this challenge either.

With the mentioned disadvantages considered, we have collected log files from an AWS Application Load Balancer of a privately owned company and devised three methods for anomaly detection. Methods and evaluation metrics were chosen based on the systematic review of literature in the areas of anomaly detection and outlier detection for time series and anomaly detection and outlier detection for log files.

We would like to note that this work is not a comprehensive overview of the field and that as a starting point for future research it might omit relevant information unknown to us at the time of writing this thesis.

Additionally, this research is not a full-fledged machine/statistical learning analysis of the possibilities and challenges of presented approaches but an introduction to the topic.

Having a real-world time series dataset as the first dataset to work on is quite challenging and can be discouraging in some ways but that approach mimics reality for many professions and serves as a well-prepared introduction to industry for soon to be graduates. Thus, the findings presented here might be far away from the ideal case scenario but at least are a start.

1. Systematic literature review

To position our research in the corresponding state-of-the-art, we planned and conducted a systematic review and meta-analysis of the literature, following guidelines of "Preferred Reporting Items for Systematic Reviews and Meta-Analyses" (PRISMA) [1].

In the following sections we will outline the process of conducting and reporting a systematic review of publications, starting with the formulation of 7 research questions.

1.1. Planning

The systematic literature review and empirical study that will be conducted afterward intend to address the subsequent research questions:

1. RQ1: What methods are usually used to detect anomalies in AWS load balancer logs?
2. RQ2: What measures are usually used to evaluate the accuracy of anomaly detection models in AWS load balancer logs?
3. RQ3: What datasets are usually used when comparing different models?
4. RQ4: Which method is the most effective for anomaly detection in AWS load balancer logs?
5. RQ5: What are the advantages and disadvantages of the methods used to detect anomalies in AWS load balancer logs?
6. RQ6: What methods are used as baseline methods for anomaly detection in AWS load balancer logs?
7. RQ7: How to apply statistical, probabilistic, and deep learning methods on AWS ALB logs to detect anomalies in different metrics presented in the logs?

Research questions were formulated by writing down questions of interest for anomaly detection of AWS load balancer logs and using the publication's supplement [2].

After searching the electronic databases, we applied the selection criteria outlined in Table 1.1. to check the eligibility of retrieved publications. Selection criteria are exclusion criteria and operate in a way that, if a piece of work meets a criterion, it is excluded from the subsequent activities in the systematic literature review.

Exclusion criteria
Publications that perform only univariate time series analysis
Publications that do not suit research questions
Editorials, presentations, prefaces, awards notes, conference summaries, keynote talks, and tutorials
Not publicly available publications
Publications authored in a non-English language.

Table 1.1 Exclusion criteria for the systematic literature review.

After the application of exclusion criteria, by using specific criteria guidelines from [2], we adopted specific criteria for the evaluation of the methodological quality of retrieved publications. Specific criteria are outlined in Table 1.2.

Specific criteria
Is the paper categorized as an empirical study or a review article?
Are the computational frameworks and datasets used in the study publicly available?
Is there a valid justification for the parameter settings used in the predictive models from the publication?
Does the publication mention advantages, disadvantages, limitations, and implementations issues of the statistical or machine learning methods for anomaly detection?

Table 1.2 Specific criteria for checking methodological quality of retrieved publications.

1.2. Conducting

By applying exclusion criteria and sorting retrieved publications from the newest to the oldest, we have chosen 2 publications outlined in Table 1.3 as the latest systematic review publications.

Publication information	Extracted Information
Title	A Comprehensive Review of Anomaly Detection in Web Logs
Authors	M. Majd, P. Najafi, S. A. Alhosseini, F. Cheng and C. Meinel
Name of Journal	2022 IEEE/ACM International Conference on Big Data Computing, Applications and Technologies (BDCAT)
Publication Date	06-09 December 2022 (date of conference), 13 March 2023 (added to IEEE Xplore)
Title	A Survey of Deep Anomaly Detection for System Logs
Authors	X. Zhao, Z. Jiang and J. Ma
Name of Journal	2022 International Joint Conference on Neural Networks (IJCNN)
Publication Date	18-32 July 2022 (date of conference), 30 September 2022 (added to IEEE Xplore)

Table 1.3 Latest systematic review publications.

We used a systematic review protocol to select publications from these. By applying methods from the protocol, 49 publications were identified and reduced to 6 publications after the application of exclusion criteria. All 6 publications satisfied the specific criteria and all of them will be included in our systematic review and meta-analysis. Information for every publication is presented in Table 1.4.

Publication information	Extracted information
Title	Anomaly detection on OpenStack logs based on an improved robust principal component analysis model and its projection onto column space
Authors	Kalaki, Parisa & Shameli-Sendi, Alireza & Abbasi, Behzad
Name of Journal	Software: Practice and Experience
Publication Date	November 2022
Title	Design and Evaluation of Unsupervised Machine Learning Models for Anomaly Detection in Streaming Cybersecurity Logs
Authors	Sánchez-Zas, C.; Larriva-Novo, X.; Villagrá, V.A.; Rodrigo, M.S.; Moreno, J.I.
Name of Journal	Mathematics 2022
Publication Date	31 October 2022
Title	Utility Analysis about Log Data Anomaly Detection Based on Federated Learning
Authors	Shin, T.-H.; Kim, S.-H.
Name of Journal	Appl. Sci. 2023
Publication Date	1 April 2023
Title	Semi-supervised and unsupervised anomaly detection by mining numerical workflow relations from system logs

Authors	Zhang, Bo & Zhang, Hongyu & Le, Van-Hoang & Moscato, Pablo & Zhang, Aozhong
Name of Journal	Automated Software Engineering
Publication Date	December 2022
Title	BERT-Log: Anomaly Detection for System Logs Based on Pre-trained Language Model
Authors	Song Chen & Hai Liao
Name of Journal	Applied Artificial Intelligence
Publication Date	17 November 2022
Title	Learning-Based Anomaly Detection Using Log Files with Sequential Relationships
Authors	M. Fält, S. Forsström, Q. He and T. Zhang
Name of Journal	2022 6th International Conference on System Reliability and Safety (ICSRS)
Publication Date	23-25 November 2022 (Date of conference), 21 March 2023 (added to IEEE Xplore)

Table 1.4 Selected publications

1.3. Meta-analysis

By analyzing six chosen publications, we were able to conduct the following meta-analysis:

- Anomaly detection methods - Given the frequency at which methods were referenced in the publications, 4 out of 6 publications (66.6%) employed supervised learning methods, 1 out of 6 publications (16.6%) employed unsupervised learning methods, 1 out of 6 (16.6%) publications employed semi-supervised learning methods, 5 out

of 6 (83.3%) publications employed Deep Learning methods, 2 out of 6 publications (33.3%) employed probabilistic methods, 1 out of 6 (16.6%) methods comprised clustering methods, and 1 out of 6 publications (16.6%) employed hybrid model.

- Evaluation measures - Given the regularity with which the evaluation measures were used in the published literature, 4 out of 6 (66.6%) publications used recall, 4 out of 6 (66.6%) publications used precision, 4 out of 6 (66.6%) publications used F1-score, 3 out of 6 (50%) publications used accuracy, 1 out of 6 (16.6%) publications used weighted F1-score, and 1 out of 6 (16.6%) publications used WSSE (Within-Cluster Sum of Squares Error).

- Parameters setting – 2 out of 6 (33.3%) publications explicitly mentioned the used search technique for parameters setting, 1 of them employed k-Fold Cross Validation, and the other employed Grid Search, 2 out of 6 (33.3%) publications only mentioned the split of the dataset on training data and validation data, and the remaining 2 out of 6 (33.3%) publications did not mention anything related to the search for parameter setting.

- Baseline methods – 3 out of 6 (50%) publications compared the results of proposed methods with the baseline or baseline methods, 3 out of 6 (50%) publications did not compare the results with baseline methods; of the publications that employed baseline method, one of them (16.6%) uses DeepLog as a baseline method, one of them (16.6%) used all the methods mentioned in the paper trained with centralized learning as a baseline method, and one of them (16.6%) used Standard Vector Machines, LogRobust, A2Log, and HitAnomaly methods as baseline methods.

- Datasets used – 5 out of 6 (83.3%) publications used labeled datasets, 1 out of 6 (16.6%) publications used unlabeled datasets, 3 out of 6 (50%) publications used publicly available datasets, other half used datasets that are not publicly available, 1 out of 6 (16.6%) publications used the combination of a real dataset and a synthetic dataset, all datasets that are publicly available and that were used were Hadoop Distributed File System and Blue Gene/L log datasets, one of the mentioned datasets is comprised of logs of Hadoop Distributed File System that was run of Amazon EC2 Platform.

1.4. Summary of the selected publications

In this section, we provide an overview of the six identified publications:

1. New methods for anomaly detection in OpenStack log files were investigated in [3], along with the systematic review of related work. Since there was no suitable dataset of OpenStack logs, the authors made a dataset of 25,000 logs accompanied by the injection of three anomalies. The authors evaluated PCA, DeepLog, and variation of PCA PRPCA-CS on the generated dataset. Authors improved the anomaly detection in OpenStack logs in terms of F1-score, recall, and precision relative to identified recent studies that applied PCA and DeepLog algorithm. Furthermore, the running time in relation to the amount of log data has been decreased by 30 seconds in comparison to the DeepLog approach.
2. The objective of [4] was to introduce a practical environment that enables the real-time management of substantial data volumes and that was built upon a scalable open-source system. The aim was to introduce a system capable of processing logs from heterogenous cybersecurity devices in real-time. This system utilizes similarly trained models to detect anomalies in communications. In addition to that, the authors developed an unsupervised learning system that can detect anomalies in a set of data from different devices in real-time. The developed anomaly detection system is based on the development of the thresholding system in conjunction with the clustering algorithm selected. Three clustering algorithms were explored, namely K-Means, Bisecting K-Means, and GMM. WSSE and Silhouette scores were used to optimize the hyperparameters of the model. The results of the experiment highlight the K-Means as optimal, obtaining better metrics and prediction results close to 99%.
3. A comparative study of deep learning models trained for anomaly detection with centralized learning and of deep learning models trained for anomaly detection with federated learning was conducted in [5]. The following deep learning models were trained: CNN1D-based log anomaly detection model with convolutional layers and LSTM-based log anomaly detection model with LSTM cells. The models were trained by the application of federated learning to analyze the applicability of federated learning for log anomaly detection. The authors demonstrated that the hybrid model combining the two models performs better than the application of a single deep learning algorithm, CNN1D, LSTM, in log anomaly detection.

4. In [6], authors proposed ADR (an acronym that stands for Anomaly Detection by workflow Relations), a novel approach to mining numerical relations from logs and using the relations for anomaly detection. Authors highlighted that ADR requires a very small size of training data and that it can produce comparable results with the state-of-the-art approaches. Furthermore, authors highlighted that for training of ADR only logs that are produced when the system is running normally are required. ADR was evaluated on log data from the HDFS dataset and BGL dataset in an online and offline manner. ADR achieved high precision and recall scores (all greater than 0.9) for offline anomaly detection and outperformed the state-of-the-art methods.

5. Authors in [7] proposed BERT-Log, which regards the log sequence as a natural language sequence. The authors employed a pre-trained language model to acquire semantic representations of both normal and anomalous logs. Subsequently, they fine-tuned the BERT model using a fully connected neural network to fine-tune its ability to detect anomalies. A new log feature extractor on the BGL dataset was proposed by authors to obtain log sequence by sliding window. The new proposed method achieved an F1-score of 98.9% with 1% of the training ratio on the BGL dataset. As compared to other related works, it had smaller parameters and stronger generalization ability.

6. The work in [8] aimed to examine log sequence anomaly detection and to present a method that can compete with state-of-the-art methods. The authors showed how self-attention transformer networks compare to LSTM networks in the task of log sequence anomaly detection. The key finding was that the self-attention transformer model attained a higher recall score compared to the LSTM model, but both models achieved an identical precision score, indicating that they were equally good at finding the anomalous events. The main conclusion was that the self-attention transformer model can achieve a better or similar results to LSTM-based models.

2. Anomaly detection

In the subsequent sections, we will describe the procedure for anomaly detection.

2.1. Outliers and anomalies

One of the best definitions of outliers was provided by Hawkins in [9]: “An outlier is an observation which deviates so much from the other observations as to arouse suspicions that it was generated by a different mechanism.”

Here are some definitions of outliers from the literature:

- Outliers can be seen as data points that diverge from the anticipated pattern or behavior [10].
- A data point that exhibits notable dissimilarity from the rest of the dataset is identified as an outlier, as per [11].
- An outlier is defined as a data point that is very different from the rest of the data based on some measure [12].

Considering the previously provided definitions, we will adopt a definition of outliers as observations that demonstrate a significant dissimilarity from most of the data, as quantified by a particular measure.

Data is typically generated in most applications by one or multiple data generation processes. When the generative process displays unusual behavior, outliers are created as a result [11].

In the areas of application of outlier detection techniques, such as credit card fraud, network intrusion detection, financial applications, and marketing, the data often follows a *normal* model, with the outliers being identified as departures from the mentioned model [11].

It is often a subjective judgment, as to what constitutes a “sufficient” deviation for a point to be considered an outlier. In real applications, the data may be embedded in a significant amount of noise, and such noise may not be of any interest to the analyst as it is usually the significantly interesting deviations that are of interest. The distinction between normal data and outliers, as well as noise and anomalies, is regulated by the analyst's specific interests and objectives [11].

The mentioned can be explained using the Figure 2.1:

- The primary patterns or clusters within the data remain the same in both situations.
- The notable differences exist beyond the primary clusters.
- In Figure 2.1 (a), a data point designated as “A” appears to display substantial dissimilarity from the remaining data points, resulting in its categorization as an outlier.
- When referring to Figure 2.1 (b) the data point labeled as “A” also resides in a sparsely populated region of the data as on (a), nevertheless, it is much harder to confidently state that it is an outlier. The data point labeled as “A” is more likely a noise in the data as it fits a pattern represented by other points that seem to be randomly distributed [11].

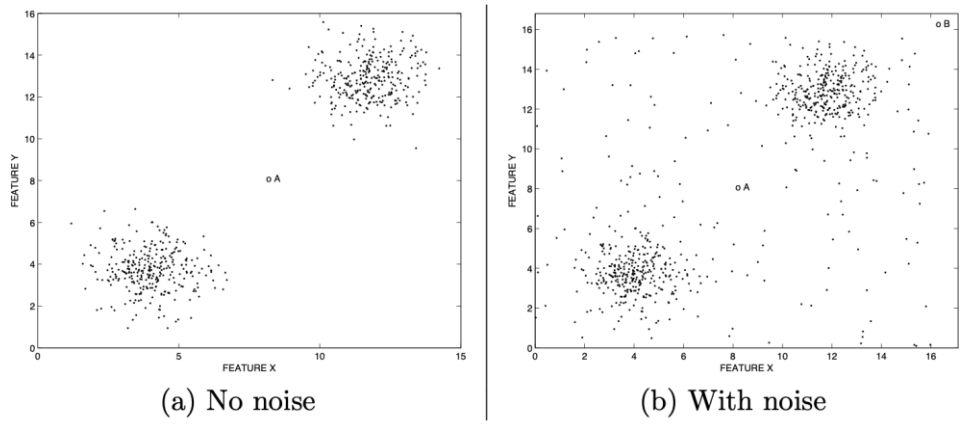

Figure 2.1 Comparison between noise and anomalies [11]

Figure 2.2 illustrates two meanings of outliers.

The outliers labeled as "Unwanted data" are associated with noise, errors, or undesired data. These data points are not of interest to the analyst, and their removal or correction is necessary to enhance data quality [10]. The process of deletion or correction results in a cleaner dataset that can be effectively utilized by data mining algorithms [10].

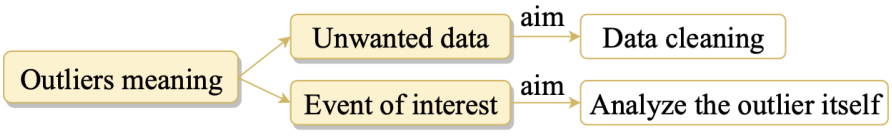

Figure 2.2 Meaning of outliers based on the analyst's judgment [10]

The distinction between normal data and weak or strong outliers is illustrated in **Error! Reference source not found.**. Every data point lies on a continuous spectrum from normal data to noise [11].

The separation of distinct regions within this spectrum lacks a precise definition and is determined in an ad hoc manner, based on application-specific criteria [11]. In unsupervised scenarios, noise serves as the semantic threshold differentiating normal data from genuine anomalies [11].

Commonly, noise is conceptualized as a less severe form of outliers, occasionally failing to meet the strict criteria needed to classify a data point as adequately interesting [11].

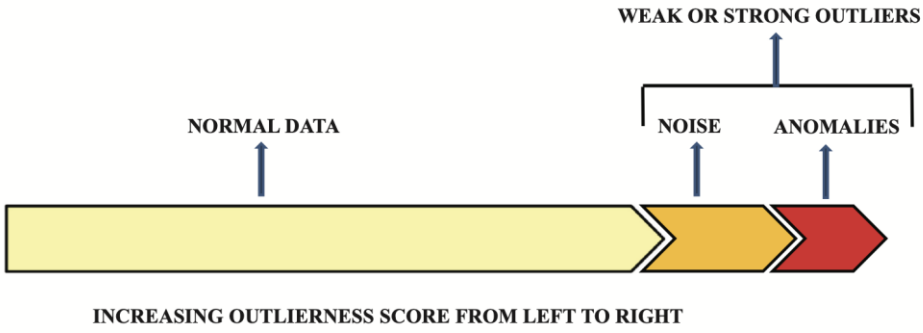

Figure 2.3 The spectrum from normal data to outliers [11]

Here are some definitions of anomalies from the literature:

- An anomaly is a pattern that does not conform to past patterns of behavior [13].
- Anomalies are patterns in data that do not conform to a well-defined notion of normal behavior [11].

We will accept the following high-level definitions of outliers and anomalies, based on definitions provided in [11]:

- To define the concept of "outlier," one can consider it as either an anomaly or noise when referring to a data point.
- The term "anomaly" refers to a unique subset of outliers that holds particular significance for an analyst.

Although we accepted the definitions mentioned above, we must acknowledge that determining anomalies in a suitable manner is highly application-dependent, necessitating that the analyst possesses a good understanding of the application domain [11].

2.2. Outliers and anomalies for time series data

A data series is an ordered sequence of data points [15].

Now, we will define univariate and multivariate time series based on definitions provided in [10]. Here are the definitions:

- A univariate time series X, denoted as $\{x_t\}$ for $t \in T$, represents a sequential collection of real-valued observations, with each observation recorded at a distinct time point t from a set T, which belongs to the positive integers Z+.
- A multivariate time series X, represented as $\{x_t\}$ for $t \in T$, comprises an arranged collection of k-dimensional vectors. Each vector is recorded at a particular time point $t \in T$, where T is a subset of positive integers Z+, and these vectors contain k real-valued observations, denoted as $x_t = (x_{1t}, ..., x_{kt})$.

Here we will deal with both univariate and multivariate time series as log files of the AWS load balancer can have both.

The inclusion of temporal dependencies significantly alters the anomaly detection process, even when considering its definition [11].

Here are the reasons why [11]:

- A time series consists of a sequence of values often obtained through continuous measurement over a duration of time.
- The values in successive time intervals exhibit either minimal fluctuations or change in a gradual manner.
- In such scenarios, abrupt alterations in the fundamental data records may be seen as extraordinary events.

When temporal dependencies are present, the assumption of temporal continuity becomes pivotal in the identification of outliers [11]. Temporal continuity represents the fact that patterns within the data typically do not undergo abrupt changes unless abnormal processes are in operation [11]. In the realm of time-series data, one anticipates a strong and immediate presence of temporal continuity [11].

Two different scenarios can arise for temporal data [11]:

- Sudden change in time series – In such cases, the outlier is characterized as atypical as it demonstrates a discontinuity regarding either its latest or extended history. Illustrations include abrupt shifts in time-series values in relation to the recent past and distinctive subsequence patterns within the time series with regard to its extended historical context.

- Novelty and variations within multivariate data - In such cases, the data consists of independent multivariate points and temporal continuity is considerably less prominent in comparison to time series data.

The example makes it clear that temporal data offers several approaches for defining anomalies.

It's worth emphasizing that while the analysis of temporal data for changes and detection of outliers share a close relationship, they are not identical problems [11].

In temporal data, change can happen in two ways [11]:

- The data stream experiences gradual changes in values and trends over time, a phenomenon known as concept drift [11]. In such instances, detecting concept drift requires thorough analysis over an extended period and often goes unnoticed in many scenarios.
- The data stream's values and patterns undergo sudden shifts, prompting immediate suspicion that the fundamental data generation process has fundamentally changed.

Between the two scenarios, only the second one is applicable for identifying outliers [11].

Based on the [10] and [11], we have two types of time series outliers or anomalies:

- Point or contextual outliers or anomalies – They refer to values that display abnormal behavior at specific time instances[10]. This abnormality can be in relation to other values within the entire time series (global outlier or anomaly) or its neighboring data points (local outlier or anomaly).
- Subsequence or collective outliers or anomalies - Consecutive values in a time series that collectively exhibit unusual behavior when contrasted with other values within the same time series (global outlier or anomaly) or with their neighboring data points (local outlier or anomaly), even if each individual value might not qualify as a point outlier.

Subsequence or collective outliers or anomalies can be further divided into the following [11]:

- Complete series anomaly - The time series database is used to compare the entire time series pattern that is categorized as anomaly.
- Anomaly-based on subsequences - When dealing with time series data that spans an extended duration, it might display regular patterns in shorter intervals. In such cases,

we detect unusual shapes by looking at small sections of the time series and identifying deviations from these usual patterns.

Point outliers can be univariate or multivariate. In Figure 2.4 (on the left), there are two instances of univariate point outliers or anomalies represented by O1 and O2 [11]. In Figure 2.5 (on the right), which consists of a multivariate time series involving three variables, there are both single-variable (O3) and multi-variable (O1 and O2) individual point outliers [10].

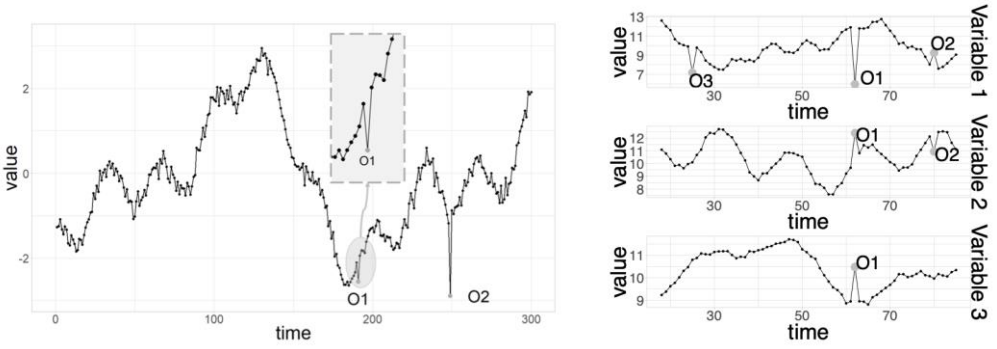

Figure 2.4 Point outliers or anomalies [10]

Figure 2.5 illustrates an example of univariate (O1 and O2 in Figure 2.5 (left), and O3 in Figure 2.5 (right)) and multivariate (O1 and O2 in Figure 2.5 (right)) subsequence outliers or anomalies [10].

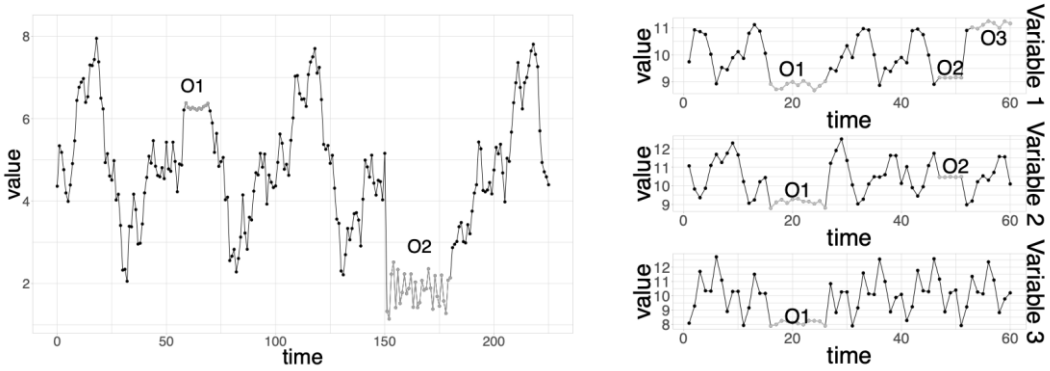

Figure 2.5 Subsequence outliers or anomalies [10]

The difference between local and global outliers is the following, based on [10]:

- When the detection method utilizes the entire time series as contextual information, the detected outliers or anomalies are categorized as global.
- When the detection method focuses solely on a time window, the outliers it identifies are regarded as local, as they exhibit distinct characteristics within that particular neighborhood.

Some local outliers may appear ordinary when considering all the time series data, but they can be seen as unusual when we look only at their nearby context, as shown by O1 in Figure 2.4 [10].

2.3. Anomaly detection methods for time series data

The outcome of the algorithm for detecting anomalies can be divided into one of two groups, based on [11]:

- Outlier score – A score that quantifies the extent of "outlierness" for every data point can be used to sort the data points according to each data point's likelihood of being an outlier. This type of output preserves all the information generated by an algorithm but does not define which data points are outliers.
- Binary label - Some algorithms produce binary labels by applying thresholds to outlier scores, with the threshold determined by the statistical distribution of the scores. This type of output doesn't retain all the information generated by an algorithm but does specify which data points are considered outliers.

Labels can be used to guide the process of detecting anomalies. Within the realm of time series, these labels might be linked to specific time moments, time intervals, or the entire time series [11]. In such instances, anomaly detection problem becomes an instance of rare-class detection problem [11]. In most cases, supervised approaches tend to deliver superior performance compared to unsupervised methods due to their capacity to identify abnormalities tailored to the specific application. Therefore, the common advice is to leverage supervision whenever it is accessible [11].

Temporal continuity plays a crucial role in methods for detecting anomalies in time series data. This is due to the assumption that time series values are tightly interconnected from one time step to the subsequent one, and that temporal patterns don't change suddenly [11].

We will only focus on detecting point outliers therefore an explanation of methods used for full time series anomaly detection and subsequence-based anomaly detection will not be presented.

2.3.1. Prediction-based methods

Temporal outlier detection is most frequently applied to identify deviations at specific time points by utilizing regression-based forecasting models. This approach can fulfill either the objective of foundational process sudden shifts detection or of noise elimination from the core data streams [11]. Outliers based on deviations in time series have a significant connection to the process of time series forecasting, as they are recognized by their deviations from expected (forecasted or predicted) values [11].

Since our dataset is not labeled, as will be described in the section where the technical overview of chosen algorithms and dataset will be presented, here is the explanation of how prediction can be used for unsupervised outlier detection based on [11]:

- Outliers are instances where data dependencies deviate from the expected model.
- A prediction model assists in representing the stated dependencies.
- Breaking these relationships signifies a departure from the normal model of data and, as a result, corresponds to outliers.

In some cases, the input time series consists of multiple variables that may be correlated, rather than being a single-variable time series. When we apply univariate techniques to each of these time-dependent variables independently, we disregard the correlation relationships between them, resulting in a loss of information. In order to tackle this challenge and leverage the extensively developed univariate detection methods, some researchers employ a pre-processing strategy on the multivariate time series [10]. This strategy involves uncovering a new set of uncorrelated variables that are suitable for applying univariate techniques. This is achieved by simplifying the multivariate series into a lower-dimensional representation using dimensionality reduction techniques before applying univariate detection methods. Since the new series are composed of combinations of the original input variables (some dimensionality reduction techniques determine the new set of uncorrelated variables by computing linear combinations of the initial variables), the detected outliers exhibit multivariate characteristics [10].

For multivariable time series, the following methods can be used for anomaly detection based on [10]:

- Anomaly detection methods for univariate time series that are employed individually for each variable.

- Multivariate time series anomaly detection methods.

The connections within an individual time series or among multiple series can be utilized for prediction purposes [11].

Two forms of correlations are employed:

- Correlations across time - Time-based correlations are commonly established through autoregressive modeling and forecasting. Outliers are identified as substantial deviations from the anticipated (predicted or forecasted) values [11].
- Correlations across series - For instance, when a bird call is detected by one sensor, it is likely to be captured by a neighboring sensor as well (time series that often exhibit strong correlations). In such situations, one series can often be employed to forecast another. Anomalies can be identified when there are differences between these anticipated predictions. The concept here is that distinct time series may frequently convey similar information and sometimes exhibit lagged correlations [11].

First, we will explain univariate time series outlier detection methods. Their division is illustrated in Figure 2.6.

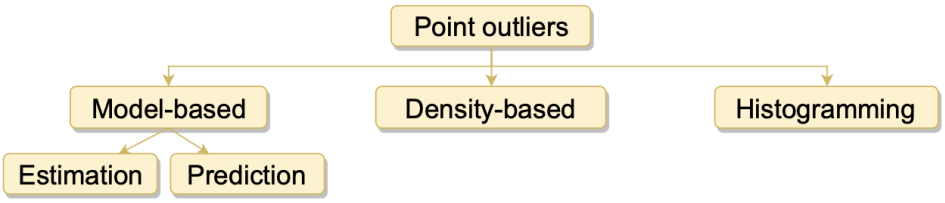

Figure 2.6 Univariate time series point outlier detection methods [10]

As per the definition of point outliers, when working with a univariate time series, a point at time t can be identified as an outlier if the absolute difference between its observed value and the expected value exceeds a predefined threshold τ: $|x_t - \hat{x}_t| > \tau$ [10].

Methods for detecting outliers that rely on the concept mentioned earlier are known as model-based methods and are widely used in research literature, based on [10].

The following are their characteristics [10]:

- Each technique computes the expected value $\hat{x}_t$ and threshold τ using unique approaches.
- Each method fits a model to the data.

Here's the differentiation between estimation-based and prediction-based methods, based on [10]:

- When $\hat{x}_t$ is computed by considering both preceding and subsequent observations to x_t (encompassing past, present, and future data), it falls within the category of estimation model-based methods.
- If $\hat{x}_t$ is determined solely using prior observations to x_t (past data), it falls under prediction model-based methods.

The following estimation models are often used [10]:

- Simple estimation models - Using basic models that rely on fundamental statistics like the median or the Median Absolute Deviation (MAD), we calculate the expected value $\hat{x}_t$ [10].
- The approaches that aim to identify data points that would be considered improbable under the assumption that a specific model or distribution generated the data. Such techniques encompass B-splines, kernel methods, various adaptations of the Exponentially Weighted Moving Average (EWMA) approach, Gaussian Mixture Models (GMM), a method that presumes a normal distribution in the absence of outliers, and a method that models alterations in slopes both before and after time t (while assuming minimal slope changes at a particular time point) [10].
- Various approaches examining all residuals derived from different models for outlier identification. Among the techniques employed are STL decomposition, ARIMA models incorporating exogenous factors, linear regression, or Artificial Neural Networks [10]. While many of these models are also capable of prediction, in this context, the focus is on identifying outliers within the residual dataset using historical and future data. After the model is trained, hypothesis testing, such as the ESD test, is applied to the residuals to pinpoint outliers [10].
- Methods that examine all the differences between observed and predicted values obtained from various models to find outliers - Some of the methods used include ARIMA models with external inputs, STL decomposition, Artificial Neural Networks (ANNs), and linear regression [10].

The following prediction models are often used [10]:

- Models that remain constant and do not adapt to changing data over time – Deep learning models (DeepAnT algorithm that uses fixed Convolutional Neural Networks, deep neural networks with LSTM cells) [10].
- Autoregressive models – ARIMA [10]
- Adaptable models – One approach involves forecasting the value $\hat{x}_t$ by employing the median of historical data, while another method relies on an ARIMA model that dynamically adapts its parameters using a sliding window as it progresses forward [10].

Now, we will explain multivariate time series outlier detection methods. Their division is also illustrated in Figure 2.6.

When applying the point outlier definition to a multivariate time series, a point at time t can be classified as an outlier if the distance between its actual k-dimensional data point, x_t, and its expected value, $\hat{x}_t$, surpasses a predefined threshold τ: $||x_t - \hat{x}_t|| > \tau$ [10].

Model-based methods are the prevailing methods frequently encountered in the literature when dealing with multivariate time series. The definitions of estimation-based and prediction-based models are the same, except x_t and $\hat{x}_t$ are k-dimensional data points.

The following estimation models are often used [10]:

- Autoencoders without modeled temporal correlation - Neural networks that prioritize learning the most essential aspects of a training dataset are used as a reference for what's considered normal. As outliers typically involve atypical characteristics, autoencoders struggle to reconstruct them, resulting in significant errors [10].
- Autoencoders with modeled temporal correlation – The methods employed consist of Variational Autoencoders (VAE) in combination with a Gated Recurrent Unit (GRU) [10].

The following prediction models are often used [10]:

- Contextual Hidden Markov Model (CHMM)
- The DeepAnt algorithm

Based on [11], the following methods are used for the prediction-based approach (not to confuse with the division of model-based methods on evaluation and prediction-based methods, prediction based methods in this context mean that the model is fitted and used to predict for outlier detection):

- Autoregressive models - Very useful for single-variable time series data. Commonly employed methods include autoregressive moving average (ARMA) and autoregressive integrated moving average (ARIMA) models [11].
- Time series regression models - The basic idea is to predict the target time series, y, by assuming it has a linear connection with other time series, x [17].
- PCA-based techniques with other models - PCA-based methods are typically more resilient in the presence of noise and outliers. A set of k uncorrelated time series is generated by projecting the d-dimensional values at each timestamp projecting onto the leading k eigenvectors. Variance of the remaining (d−k) time series is minimal and the resulting time series can be considered as invariable time series, without the necessity of forecasting using autoregressive model. Consequently, a single auto-correlation model can be used for analysis using the larger eigenvectors [11].
- Combination of PCA-based models with other models - Combining PCA-based models with other models is often more robust when dealing with noise and outliers. This is achieved by projecting the d-dimensional values at each timestamp onto the top k eigenvectors of the matrix, resulting in a set of k uncorrelated time series. The analysis can be carried using a single auto-correlation model on the larger eigenvectors [11].

When examining techniques employed in complete time series anomaly detection and subsequence-based time series anomaly detection, it's important to note that the division between these two types of problems is somewhat arbitrary [11]. This similarity arises because a significant number of techniques developed for anomaly detection for subsequences can also find use in detecting anomalies in the entire time series. This is because subsequence anomaly detection typically begins with the extraction of windows from the time series, treating these extracted subsequences as complete series for anomaly detection [11].

2.3.2. Distance-based methods

Numerous traditional outlier detection techniques rely on the concepts of density and proximity. Essentially, these methods identify outliers as data points located in sparsely populated regions of the dataset, far from their neighboring points (this is achieved using distance measures) [16].

These methods are based on the computation of dissimilarity between pairs of multivariate data points or their representations, without the need for model fitting. When a predefined threshold τ is established, a point x_t is considered an outlier if the dissimilarity measure $s(x_t, \hat{x}_t)$ between the actual k-dimensional point x_t and its expected value $\hat{x}_t$ exceeds the predefined threshold τ [11].

The basic form of the distance-based approach (without pruning) can be outlined as follows based on [11]:

- Create overlapping subsequences from time series with a length of p. In a time series consisting of n data points, it is possible to generate a total of $(n - p + 1)$ subsequences.
- Compute outlier scores by evaluating the k-nearest neighbor distances from k-nearest neighbors of these subsequences, without including overlapping windows in the calculation.

The following similarity or distance functions are employed based on [11]:

- Euclidean distance
- Dynamic Time Warping
- Other similarity functions

The following are limitations of distance-based methods based on [16]:

- They face a challenge known as the curse of dimensionality, which makes many traditional distance measures ineffective when the dataset has a high number of dimensions. The core issue is that in high-dimensional spaces most distance measures become compressed resulting in almost identical distances between all pairs of data points.

3. An introduction to applied methods and the dataset

In the following sections we will present a short introduction to the applied methods and the dataset we used.

3.1. Methods

We have decided to work with 3 methods:

- A method we will call “Statistical method”,
- A method we will call “Probabilistic method”,
- A deep learning based method.

The statistical method is the simplest method in terms of implementation complexity and the number of parameters. The probabilistic method is a bit more complex than the statistical method while the deep learning-based method is the most complex one, based on the underlying algorithms and the number of parameters.

In all three cases, we used a threshold-based mechanism to mark possible anomalies and outliers.

3.1.1. The statistical method

The idea for the statistical method was to have a method that would “walk” through our time series and collect some type of volatility and if a value goes outside the, loosely defined, “95% confidence interval” the point would be declared as an outlier.

For this task we have chosen the leverage statistic.

High leverage points are data points where the value appears unusual given other values of the same variable [19].

The expression for the leverage statistic for the unidimensional case is

$$h_i = \frac{1}{n} + \frac{(x_i - \bar{x})^2}{\sum_{i'=1}^{n}(x_{i'} - \bar{x})^2}.$$

As x_i moves further away from arithmetic mean, the leverage statistic h_i increases. The value of the statistic is always bounded by 1/n and 1.

The reasons why we have chosen the leverage statistic:

- An example of contextual anomaly is a rapid increase in the values of a time series since it greatly diverges from the most recent time stamps' values. The important insight here is that the latest time records establish the time series' expected value, and departures from expected value are considered anomalies [11].
- h_i increases as x_i moves further away from its arithmetic mean.
- We can modify the threshold value based on the application-specific criteria.

If at least one predictor value for the observation has an extreme value based on the threshold defined by leverage statistic, we classify the observation as anomalous.

We will compute the leverage statistic:

- For unidimensional vectors, if a value of the observation is a high leverage value, then we classify the observation as anomalous (it is possible to have an observation where its values fall well within the range of each predictor's values, except for one predictor).
- For multidimensional vectors as it is possible to have an observation that falls well within the range of each predictor's values individually but is anomalous when considering the values of the entire set of predictors (no predictor has extreme value, but the observation is unusual when all predictor's values are considered).

The pseudocode for the statistical method for the unidimensional case is shown in Table 3.1.

Statistical method unidimensional case

leverage_statistic_values = list()
for every column in log files columns:
for i in range from window_size to number of observations:

```
    leverage statistic = compute leverage statistic for the first window_size values of the column
        add leverage statistic to leverage_statistic_values
    sort leverage_statistic_values from lowest to biggest value
    top_10_leverage_statistic_values = leverage_statistic_values last 10 values
    anomalies_indices = indices of values from top_10_leverage_statistic_values
```

Table 3.1 Pseudocode for the statistical method for unidimensional case

The pseudocode for the statistical method for multidimensional case is shown in Table 3.2.

Statistical method multidimensional case
leverage_statistic_values = list() for i in range from window_size to number of observations: leverage statistic = compute leverage statistic for the first window_size values of all columns add leverage statistic to leverage_statistic_values sort leverage_statistic_values from lowest to biggest value top_10_leverage_statistic_values = leverage_statistic_values last 10 values anomalies_indices = indices of values from top_10_leverage_statistic_values

Table 3.2 Pseudocode for the statistical method for multidimensional case

3.1.2. The probabilistic method

For the probabilistic method, we have chosen the non-seasonal ARIMA model. ARIMA is short for Autoregressive Integrated Moving Average.

We have chosen the non-seasonal ARIMA model because:

- A seasonal pattern arises when a time series is influenced by seasonal factors, such as the time of the year or the day of the week [17]. Seasonality in such cases always follows a fixed and identifiable period [17]. Based on the visual inspection of the time series, no seasonal pattern was observed.

- In a non-seasonal ARIMA model, time stamps are not needed as we are not modeling seasonality as it is non-existent. Since our time series does not have a fixed period, it is harder to represent a non-periodic time series with a seasonal ARIMA model (even if the time series has a fixed period).
- An ARIMA model is typically not easily interpretable in terms of visible data patterns like trends and seasonality. It has the capability to capture a broad range of time series patterns [17]. We have not identified trends or seasonal patterns by visual inspection of our time series.

ARIMA is made up of three parts [17]:

- Autoregressive part (lagged observations are used as inputs)
- Integrated part (differencing to make series stationary)
- Moving Average part (lagged errors are used as inputs)

Autoregressive part explanation:

- In an autoregression model, the prediction for the variable of interest is determined by the linear combination of its past values [17].

Differencing part explanation (integrated same as differencing):

- A stationary time series is one that maintains consistent statistical properties regardless of the time of observation [17]. Time series with trends or seasonality are non-stationary because these patterns can cause variations in the series at different time points [17]. On the other hand, a white noise series is considered stationary because its statistical properties remain the same regardless of when it is observed [17]. To make a non-stationary time series into a stationary one, we often use differencing, which computes the differences between consecutive observations [17].
- Transformations such as logarithms can help to stabilize the variance of a time series [17]. Differencing can help stabilize the mean of a time series by removing changes in the level of a time series and therefore eliminating (or reducing) trend and seasonality [17].
- Taking the logarithm of time series values can stabilize the time series' variance [17]. Differencing can stabilize the mean of a time series by removing level changes, which helps mitigate or eliminate trends and seasonality [17].

Moving Average part explanation:

- Instead of relying on previous values of the forecast variable in a regression, a moving average model incorporates past errors in a regression-like manner [17].

The ARIMA model is commonly denoted as ARIMA (p, d, q) model, where:

- p is the order of the autoregressive part [17].
- d s the order of the differencing [17].
- q is the order of the moving average part [17].

Special cases of ARIMA models are presented in Figure 3.1.

White noise	ARIMA(0,0,0) with no constant
Random walk	ARIMA(0,1,0) with no constant
Random walk with drift	ARIMA(0,1,0) with a constant
Autoregression	ARIMA(p,0,0)
Moving average	ARIMA(0,0,q)

Figure 3.1 Special cases of ARIMA models [17]

We used the ARIMA model to detect anomalies in the following way:

- Using the model structure, we computed the expectation of the variable.
- Using the model structure, we computed the standard error.
- For every data point, if the data point is outside the interval [expectation – 2 * standard error, expectation + 2 * standard error], the data point is declared an anomaly.

We did not consider the distribution of predictor values or the distribution of differenced predictor values (based on the value of d), i.e., we did not select an appropriate probabilistic model for the data (we only know the structure of the model based on the output of ARIMA). The interval [expectation – 2 * standard error, expectation + 2 * standard error] is only an approximate interval because we assumed that the variables are normally distributed (the above interval is obtained for random variables that have a normal distribution and which are standardized), without looking at their histograms or using hypothesis tests.

The method was applied in R as R has an automatic procedure for finding the best possible parameters (p, d, and q) and coefficients for the ARIMA model.

Figure 3.2 illustrates the general forecasting process using an ARIMA model. The automated algorithm that we will use is shown in Figure 3.3. Some of the algorithms' steps will be explained in later chapters. The Hyndman-Khandakar algorithm only deals with steps 3 to 5 described in Figure 3.2 [17].

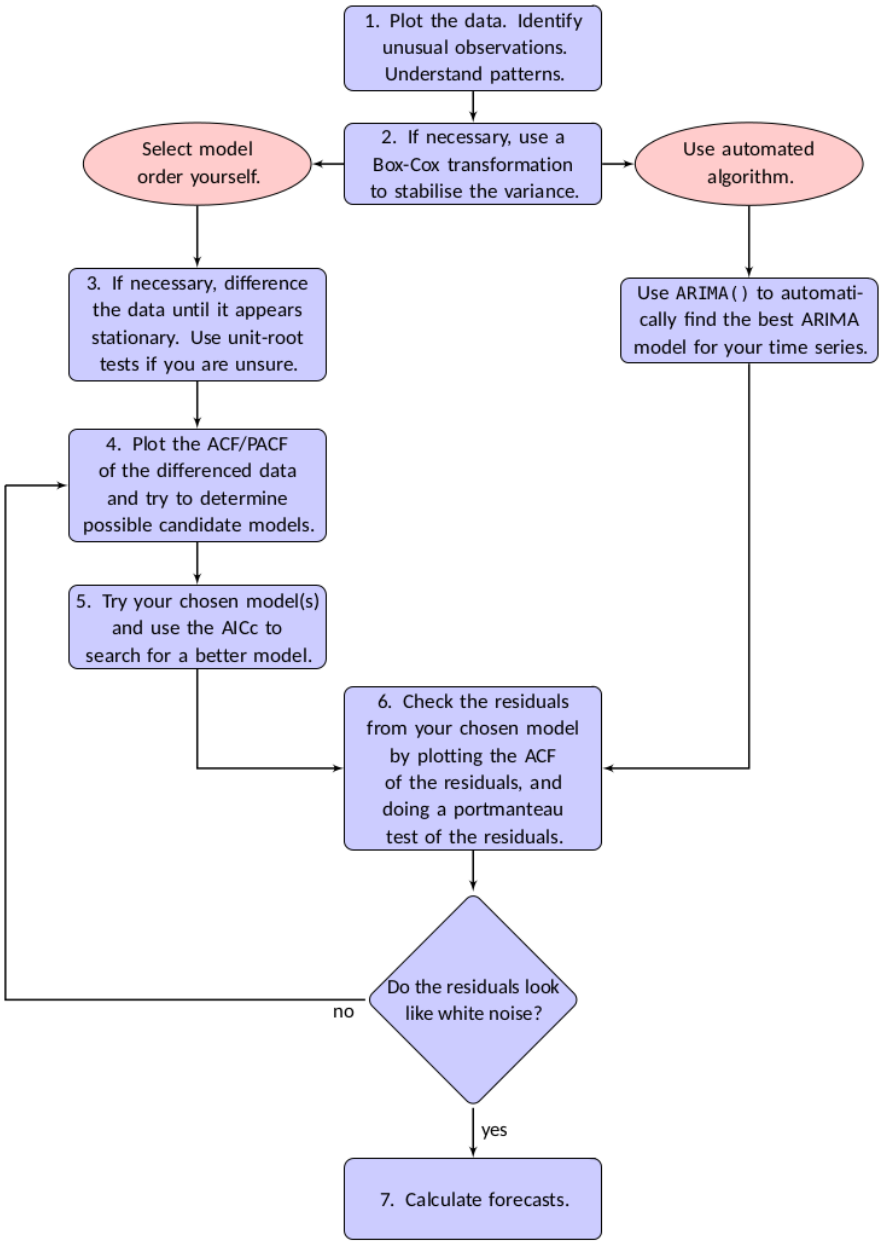

Figure 3.2 General forecasting process using an ARIMA model [17].

The Hyndman-Khandakar algorithm is shown in Figure 3.3.

Hyndman-Khandakar algorithm for automatic ARIMA modelling

1. The number of differences $0 \leq d \leq 2$ is determined using repeated KPSS tests.
2. The values of p and q are then chosen by minimising the AICc after differencing the data d times. Rather than considering every possible combination of p and q, the algorithm uses a stepwise search to traverse the model space.
   a. Four initial models are fitted:
      - ARIMA$(0, d, 0)$,
      - ARIMA$(2, d, 2)$,
      - ARIMA$(1, d, 0)$,
      - ARIMA$(0, d, 1)$.

      A constant is included unless $d = 2$. If $d \leq 1$, an additional model is also fitted:
      - ARIMA$(0, d, 0)$ without a constant.

   b. The best model (with the smallest AICc value) fitted in step (a) is set to be the "current model".

   c. Variations on the current model are considered:
      - vary p and/or q from the current model by ± 1;
      - include/exclude c from the current model.

      The best model considered so far (either the current model or one of these variations) becomes the new current model.

   d. Repeat Step 2(c) until no lower AICc can be found.

Figure 3.3 Hyndman-Khandakar algorithm for automatic ARIMA modeling [17].

Table 3.3 illustrates pseudocode for the probabilistic method.

The probabilistic method
ARIMA model = output of Hyndman-Khandakar algorithm for automatic ARIMA modelling expectation = expectation of a variable based on ARIMA model standard error = standard error of variable based on ARIMA model anomalies indices = list() for every value of time series: if absolute value (value) > expectation + 2 * standard error: add the index of the value to the list of anomalies indices

Table 3.3 Pseudocode for probabilistic method

3.1.3. The autoencoder method

For the deep learning-based method, we have chosen an autoencoder neural network architecture with LSTM cells. Here are the reasons why:

- Based on [10], they are the most mentioned estimation models.
- Autoencoders are a special type of unsupervised neural network that learns during training to effectively compress input data and to reconstruct the data from the compressed representation so that the reconstructed data is as similar as possible to the original data. Autoencoders reduce the dimensionality of the data by learning to ignore the noise contained in the data. As mentioned in [11], in practical scenarios data often contains a considerable amount of noise that might not be relevant to the analyst's objectives. Therefore, the neural network can eliminate the noise and can leave only the deviations that are of interest to the analyst.
- Autoencoder learns to effectively compress the data and to reconstruct the data from the compressed representation. If the autoencoder predominantly (anomalies belong to rare class) learns to compress the data that are generated by data generating process that performs normally (data has a normal model), then the autoencoder will not reconstruct the anomalies effectively, i.e., reconstruction error will be significantly bigger than for normal data points.

The architecture of the autoencoder we used is presented in Figure 3.4.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 encoder_1 (LSTM)            (None, 4, 64)             3289088

 encoder_2 (LSTM)            (None, 4, 32)             12416

 encoder_3 (LSTM)            (None, 16)                3136

 encoder_decoder_bridge (Re  (None, 4, 16)             0
 peatVector)

 decoder_1 (LSTM)            (None, 4, 16)             2112

 decoder_2 (LSTM)            (None, 4, 32)             6272

 decoder_3 (LSTM)            (None, 4, 64)             24832

 time_distributed (TimeDist  (None, 4, 12783)          830895
 ributed)

=================================================================
Total params: 4168751 (15.90 MB)
Trainable params: 4168751 (15.90 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 3.4 Autoencoder architecture used.

We did not know how complex the problem is in advance so we used cells with a higher number of parameters (cell type will be updated during training).

Table 3.4 illustrates pseudocode for the autoencoder method.

The autoencoder method
train the neural network and chose the best model based on validation find residuals use residuals with a pre-defined threshold to mark possible anomalies

Table 3.4 Pseudocode for the autoencoder method

3.2. Dataset

The dataset we have obtained from an ALB from a private company consists of the following 7 files:

- alb-logs-20230503.csv
- alb-logs-20230504.csv
- alb-logs-20230505.csv
- alb-logs-20230506.csv

- alb-logs-20230507.csv
- alb-logs-20230508.csv
- alb-logs-20230509.csv

The mentioned Application Load Balancer balances traffic for only one application.

A load balancer functions as the primary point of contact for clients [20]. Its role is to distribute incoming application traffic across numerous targets, like EC2 instances, situated in different Availability Zones [20].

The basic components of the AWS Application Load Balancer are shown in Figure 3.5.

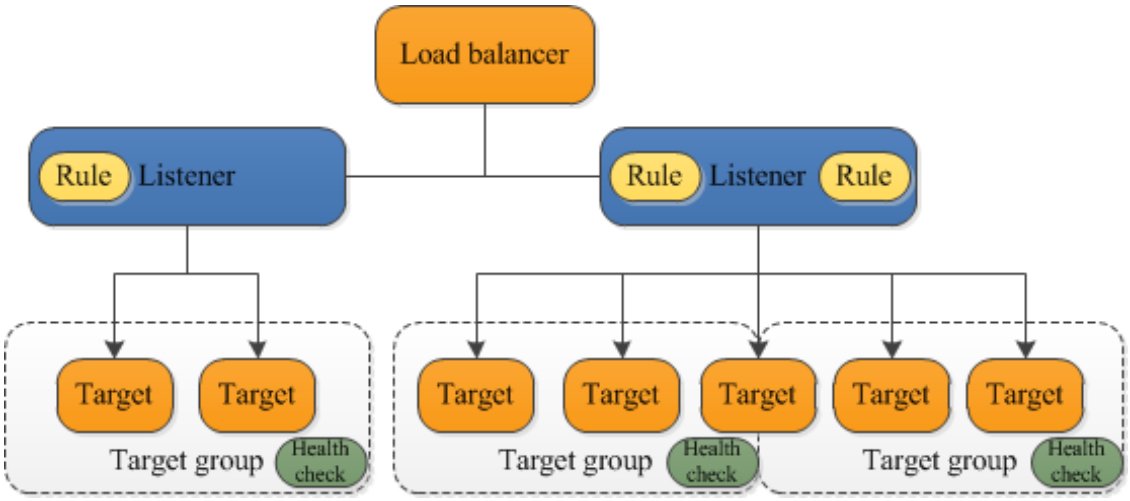

Figure 3.5 Basic components of AWS Application Load Balancer [20]

Here is the description of all basic components of the AWS Application Load Balancer based on [20]:

- A listener is responsible for inspecting connection requests from clients, utilizing the protocol and port settings that are specified [20]. The load balancer's routing of requests to its registered targets is determined by the rules that are established for a listener [20]. Each rule comprises a priority, one or more actions, and one or more conditions [20]. Actions associated with a rule are executed when the rule’s conditions are met [20].
- Each target group routes requests to one or more registered targets, such as EC2 instances, using the protocol and port number that you specify. You can register a target with multiple target groups. You can configure health checks on a per-target group basis. Health checks are performed on all targets registered to a target group that is specified in a listener rule for your load balancer.

- Each target group directs requests to one or more registered targets [20]. It is possible to register a target with multiple target groups [20]. Health checks can be configured separately for each target group [20]. These health checks are conducted on all targets that belong to a specific target group defined in a listener rule for the load balancer [20].

Contents of every file were saved in a unique pandas data frame and afterward, all data frames were concatenated as a single data frame.

Thus, we have obtained a dataset, that without pre-processing, has:

- 2577066 observations
- 28 features

Import timestamps in the dataset are:

- The first timestamp is 2023-05-03 23:55:00.936341
- The last timestamp is 2023-05-09 23:55:00.496609

The dataset is not labeled, since labeling is a daunting process requiring more resources than we had, and therefore we do not have a domain expert's confirmation on which observations are anomalies, and which are not.

Columns of the dataset, along with their descriptions, are shown in Table 3.5.

Field	Description
type	Connection or request type.
time	Timestamp when the response to the client was generated by the load balancer.
elb	Load balancer resource ID.
client:port	Requesting client’s IP address and port.
target:port	Target’s IP address and port. Target is a machine the processed the request.
request_processing_time	The time taken from request received to request sent by the load balancer (in seconds, with millisecond precision).
target_processing_time	The time from request sent by the load balancer to target until target started sending response headers (in seconds, with millisecond precision).
response_processing_time	The time from load balancer receiving response header to sending response to the client, including queuing and connection time (in seconds, with millisecond precision).
elb_status_code	Load balancer response’s status code.
target_status_code	Target’s response status code.
received_bytes	Client’s request size in bytes.
sent_bytes	The response's byte size sent to the client.
request	Client’s request.

user_agent	Client’s user agent.
ssl_cipher	The encryption cipher used for SSL.
ssl_protocol	The SSL communication protocol.
target_group_arn	The ARN for the target group in Amazon Resource Name format.
trace_id	X-Amzn-Trace-Id header’s content.
domain_name	The client's SNI domain during the TLS handshake.
chosen_cert_arn	The certificate's ARN sent to the client.
matched_rule_priority	The matched request's rule priority value.
request_creation_time	Timestamp when client’s request was received by the load balancer.
actions_executed	The steps performed when handling the request.
redirect_url	The URL of the redirect destination in the HTTP response's location header.
error_reason	The error code indicating the reason.
target:port_list	A list of target IP addresses and ports used for processing this request.
target_status_code_list	A list of status codes from target responses.
classification	Desync mitigation classification.
classification_reason	The reason code for classification.

Table 3.5 Columns in the log files of AWS ALBs and their description [21]

4. The experimental part

In the following sections we will describe the experimental part.

4.1. Anomalies for log files of AWS load balancer

First, we have to define anomalies for the log files of the AWS load balancer.

4.1.1. Anomalies for the statistical method

We will use two variants of the statistical method:

- Variant for the unidimensional case
- Variant for the multidimensional case

Anomaly for the unidimensional case is defined as follows:

- If any of the predictor's values leverage statistic is above the predefined threshold for that predictor, the observation is defined as an anomalous observation.

Anomaly for the multidimensional case is defined as follows:

- If the leverage statistic for multidimensional observation is above the predefined threshold, the observation is defined as anomalous observation.

4.1.2. Anomalies for the probabilistic method

The probabilistic method was trained only for single predictors.

Anomalies for the probabilistic method are defined as follows:

- If the predictors' value of the observation is outside the predefined interval, the observation is defined as anomalous observation.

4.1.3. Anomalies for the autoencoder method

Anomalies for the autoencoder are defined as follows:

- If the residual of the observation is higher than the predefined threshold, the observation is defined as anomalous observation.

4.2. Data preparation

Before concatenating data frames, we did the following:

- alb-logs-20230503 data frame – Drop column classification_reason as it had 5 values that were not missing values, and all of them had value “-”; drop zero variance columns as they do not contain information; we are left with 25 columns.
- alb-logs-20230504 data frame - Drop column classification_reason as it had 5 values that were not missing values, and all of them had value “-”; drop zero variance columns as they do not contain information; we are left with 25 columns.
- alb-logs-20230505 data frame - Drop column classification_reason as it had 5 values that were not missing values, and all of them had value “-”; drop zero variance columns as they do not contain information; we are left with 26 columns.
- alb-logs-20230506 data frame - Drop zero variance columns as they do not contain information; we are left with 25 columns.
- alb-logs-20230507 data frame - Drop zero variance columns as they do not contain information; we are left with 25 columns.
- alb-logs-20230508 data frame - Drop zero variance columns as they do not contain information; we are left with 25 columns.
- alb-logs-20230509 data frame - Drop column classification_reason as it had 5 values that were not missing values, and all of them had value “-”; drop zero variance columns as they do not contain information; we are left with 28 columns.

After that, the data frames were concatenated into a single data frame.

Based on the meeting with the domain expert, basic requirements on useful observation windows were agreed upon. The reason for this was the following:

- Size of the data frame when the full set of observations is taken.
- Our constraints with computational power.

We have decided to take the first 100,000 observations.

After that, we’ve dropped the following columns:

- elb as variance = 0 for values that are not nan
- classification as variance = 0 for values that are not nan when we exclude the value "Ambiguous".

Additionally, we dropped the following columns:

- client_port
- target_port
- ssl_protocol
- trace_id
- chosen_cert_arn
- matched_rule_priority
- target_port_list
- target_status_code_list
- request_creation_time

We are well-aware that these features would be useful in anomaly detection but with the resources we had, significant increase in complexity of our research was not possible. However, we are hoping we will continue this research with some of these variables included in our future methods.

Every row of the column request was disassembled into the following parts to reduce the cardinality of the column:

1. HTTP method
2. protocol
3. host
4. port
5. URI
6. HTTP version

After that, rows were reordered by column time.

After we applied everything mentioned above, we were left with the following columns and their data types:

- type – data type is object
- time – data type is datetime64[ns]
- request_processing_time – data type is float64, we will change data type to float16
- target_processing_time – data type is float64, we will change data type to float16
- response_processing_time – data type is float64, we will change data type to float16
- elb_status_code – data type is int64, we will change data type to uint16

- target_status_code – data type is object
- received_bytes – data type is int64, we will change data type to uint16
- sent_bytes – data type is int64, we will change data type to uint16
- request – data type is object
- user_agent – data type is object
- ssl_cipher – data type is object
- target_group_arn – data type is object
- domain_name – data type is object
- actions_executed – data type is object
- redirect_url – data type is object

The last step was to encode categorical features as the statistical method, probabilistic method, and autoencoder expect input that is numerical.

Here is the cardinality of columns that were encoded:

- type – cardinality is 3
- elb_status_code – cardinality is 7
- target_status_code – cardinality is 6
- request – cardinality is 47477
- user_agent – cardinality is 247
- ssl_cipher – cardinality is 2
- target_group_arn – cardinality is 3
- domain_name – cardinality is 3
- actions_executed – cardinality is 2
- redirect_url – cardinality is 3

For encoding, we have chosen one hot encoding.

4.3. The training phase

In the following sections we will describe the training approach if used.

4.3.1. The statistical method

Statistical method was used for detecting anomalies only for the following columns:

- request_processing_time
- target_processing_time
- response_processing_time
- received_bytes
- sent_bytes
- number of requests

For the statistical method, we did the following:

- unidimensional case – Method "walks" through values of all predictors separately and flags anomalous values (belonging to observations) whose leverage statistic is higher than the pre-defined threshold.
- multidimensional case – Method "walks" through observations and flags anomalous observations whose leverage statistic is higher than the defined threshold.

We could have played with the window size as the parameter for our statistical method but have decided not to do so due to time constraints. We have chosen just one window size and no training phase was employed in this case.

4.3.2. The probabilistic method

The probabilistic method was used for detecting anomalies only for the following columns:

- request_processing_time
- response_processing_time
- target_processing_time
- received_bytes
- sent_bytes

The training procedure for ARIMA used The Hyndman-Khandakar algorithm shown in Figure 3.3. We used the implementation of the mentioned algorithm that is implemented as function auto.arima from the forecast package in R.

We have observed the following challenges with the probabilistic method:

- auto.arima can only accept equally-spaced time periods,

- time series we have do not have equally-spaced time periods.

Thus, we've tried to set the period in the following way:

1. We found the greatest common divisor of all differences between timestamps.
2. We expressed every timestamp as a multiple of the greatest common divisor.

Period length was 0.000244140625 milliseconds. The total number of timestamps after setting a period would have been 2123365634282, which is too large, and the data frame would not fit in available RAM.

After that, we did the following:

1. The first and last timestamp will be fixed.
2. The period will be (difference between the first and the last timestamp) / (number of timestamps - 1)

The period is now 201.1589979519837 milliseconds. We used the timestamps we made that way for the statistical method, although dates are not needed for that method.

We are quite aware that this approach is a heuristic and might deviate from common practices, but we wanted to see if a bit of a fresh approach might result in a good method.

4.3.3. The autoencoder method

We used the following columns for the autoencoder:

- time
- request_processing_time
- target_processing_time
- response_processing_time
- received_bytes
- sent_bytes
- type – one hot encoded
- elb_status_code – one hot encoded
- target_status_code – one hot encoded
- request – one hot encoded
- user_agent – one hot encoded
- ssl_cipher – one hot encoded

- target_group_arn – one hot encoded
- domain_name – one hot encoded
- actions_executed – one hot encoded
- redirect_url – one hot encoded

First, we had to generate datasets for training. The process of generating datasets is shown in Figure 4.1.

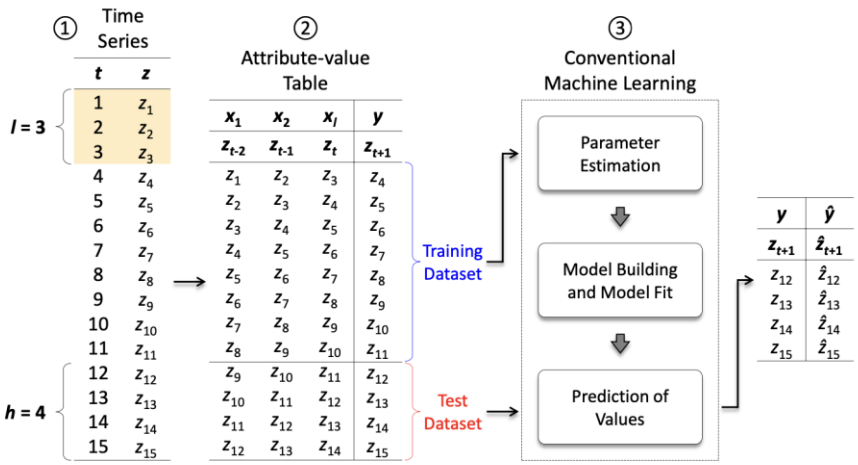

Figure 4.1 Making attribute-value table from time series values [2]

We applied a window size of 4. Since we have a multivariate time series, the window size of 4 is applied to every column so that for every column we have an attribute-value table as in Figure 4.1.

After that, we scaled the data and then we split the dataset into the training dataset and the test dataset.

We used the following hyperparameters for training:

- epochs = 100
- batch_size = 32
- windows_size = 4
- min_delta for early stopping criterion = 10^{-2}

The optimizer that we used was adam and the loss function was the mean squared error.

For the training, we did the following:

- We used the early stopping criterion that stops training if the difference between the mean squared error for the current epoch and the previous epoch is less than or equal to 10^{-2}.
- We used a checkpoint that saves the model that has the lowest validation mean squared error.
- We shuffled the vectors before training to remove the order in the values of the predictors if the order exists.

4.4. Results

In the following sections we will describe the results of the methods we used.

4.4.1. The statistical method

As mentioned earlier, we worked with only one window size (n=50) due to time constraints.

Histograms of the obtained leverage values for all variables used with the statistical method are shown in Figure 4.2, Figure 4.3, Figure 4.4, Figure 4.5, Figure 4.6, and Figure 4.7.

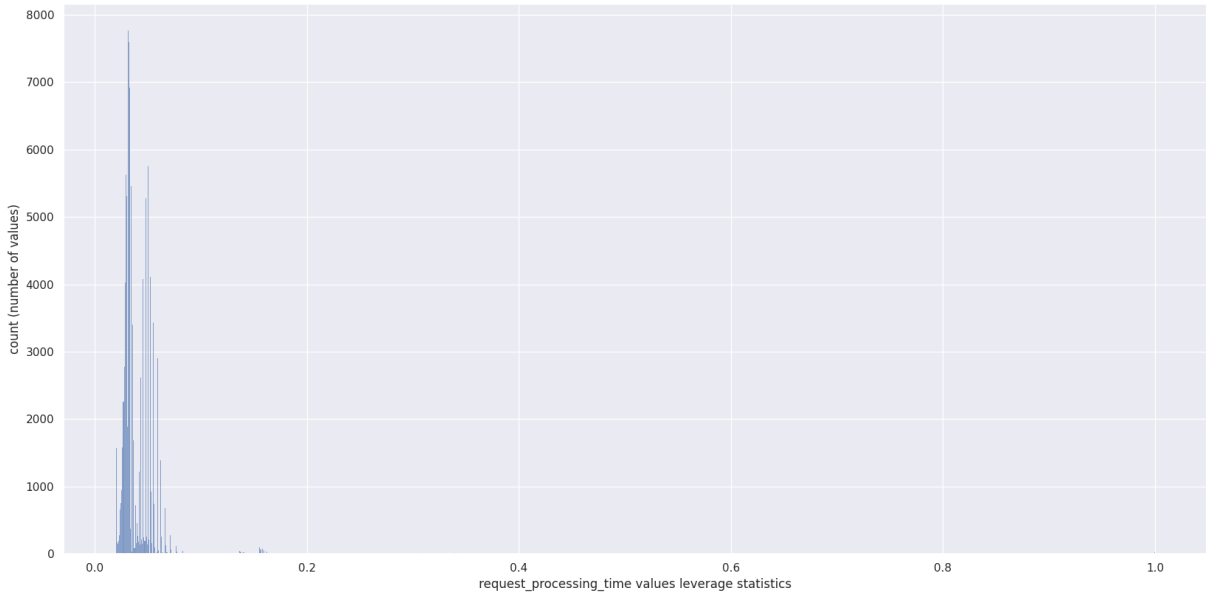

Figure 4.2 request_processing_time leverage statistic values

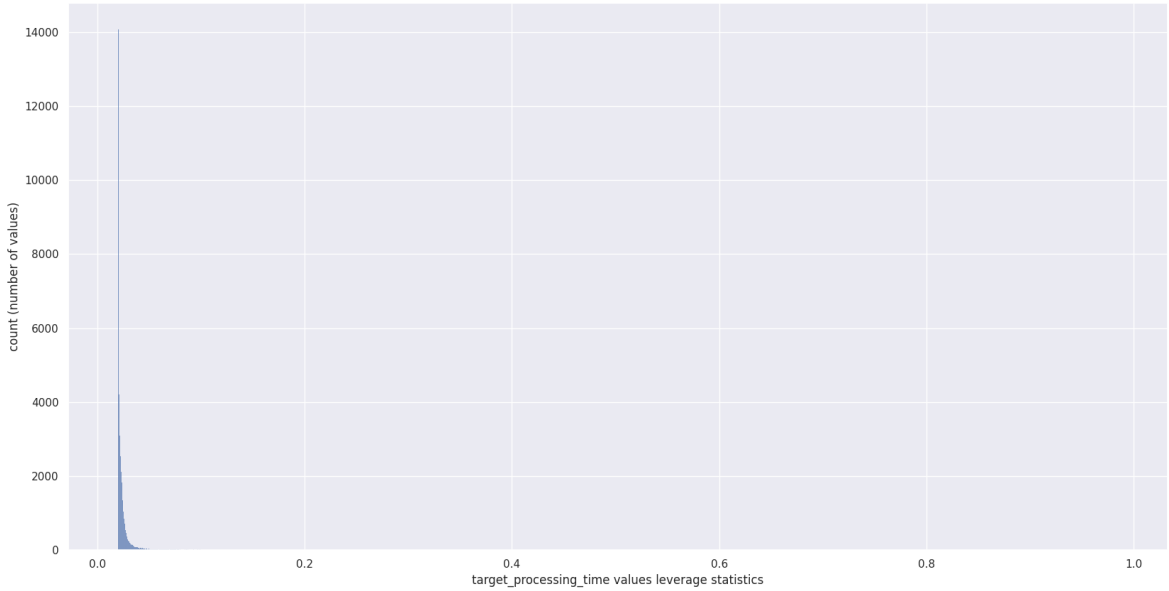

Figure 4.3 target_processing_time leverage statistic values

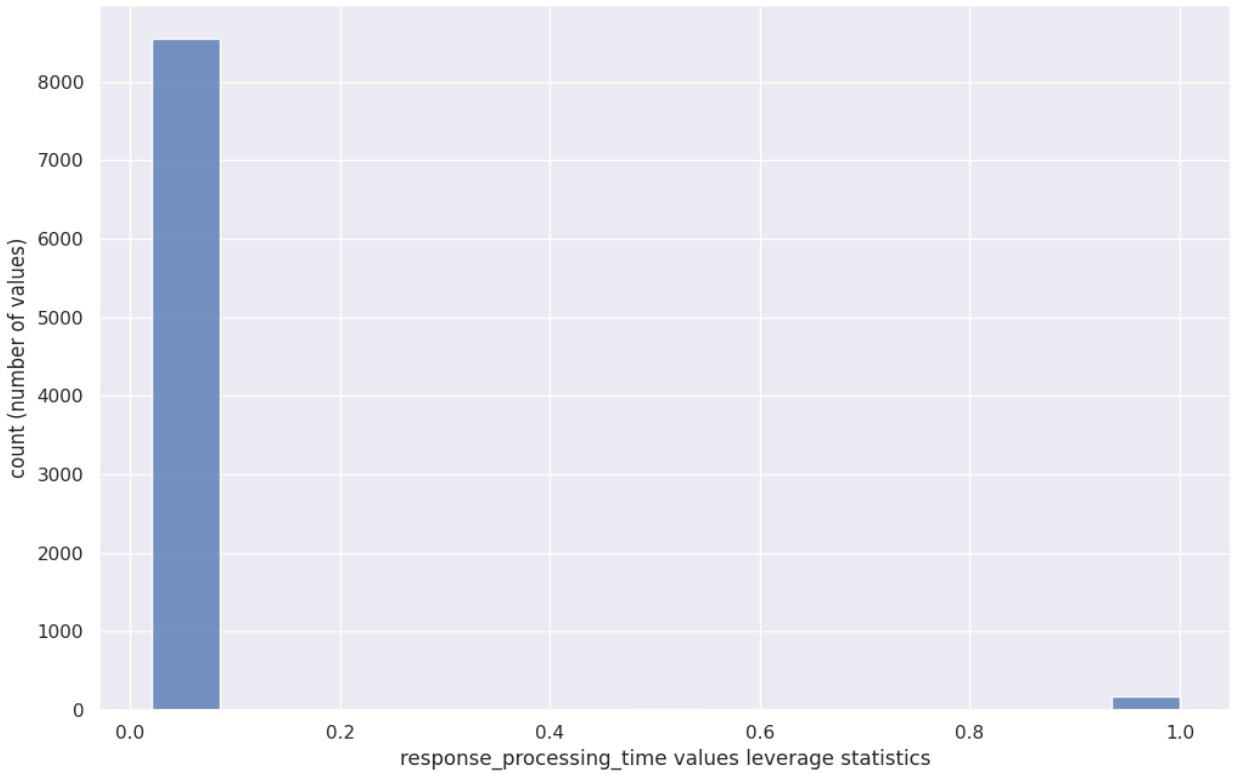

Figure 4.4 response_processing_time leverage statistic values

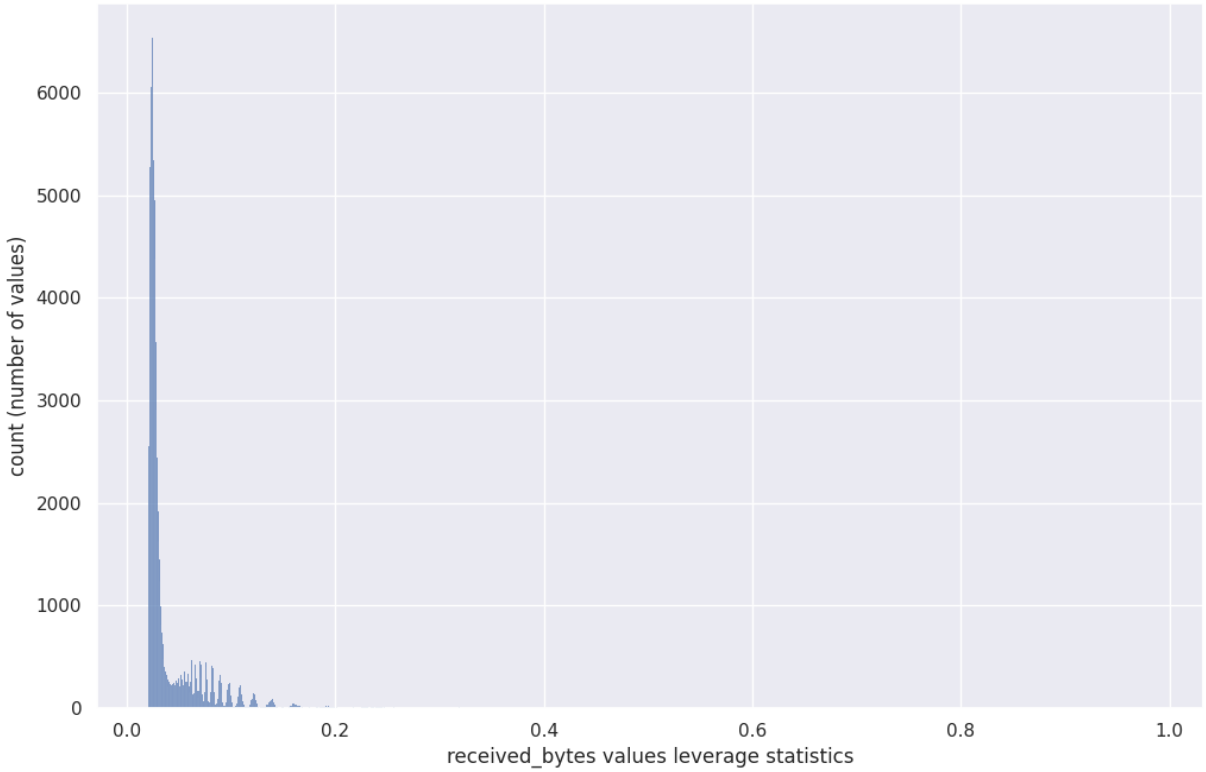

Figure 4.5 received_bytes leverage statistic values

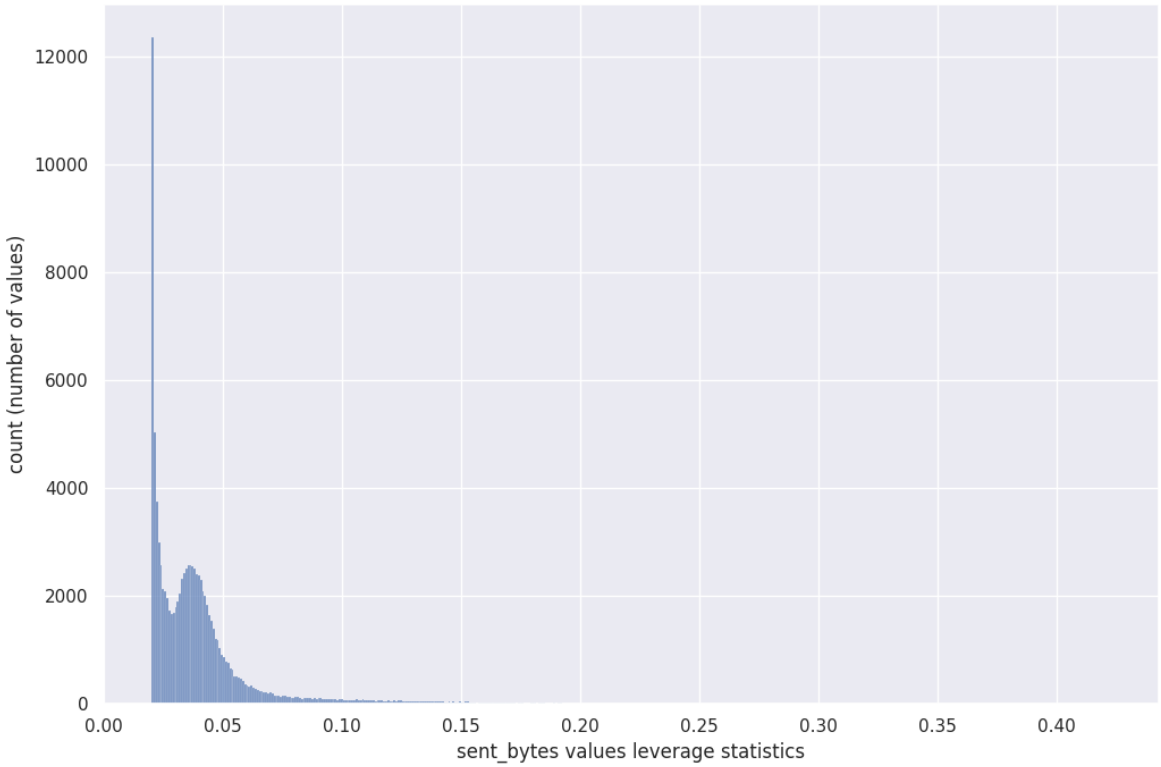

Figure 4.6 sent_bytes leverage statistic values

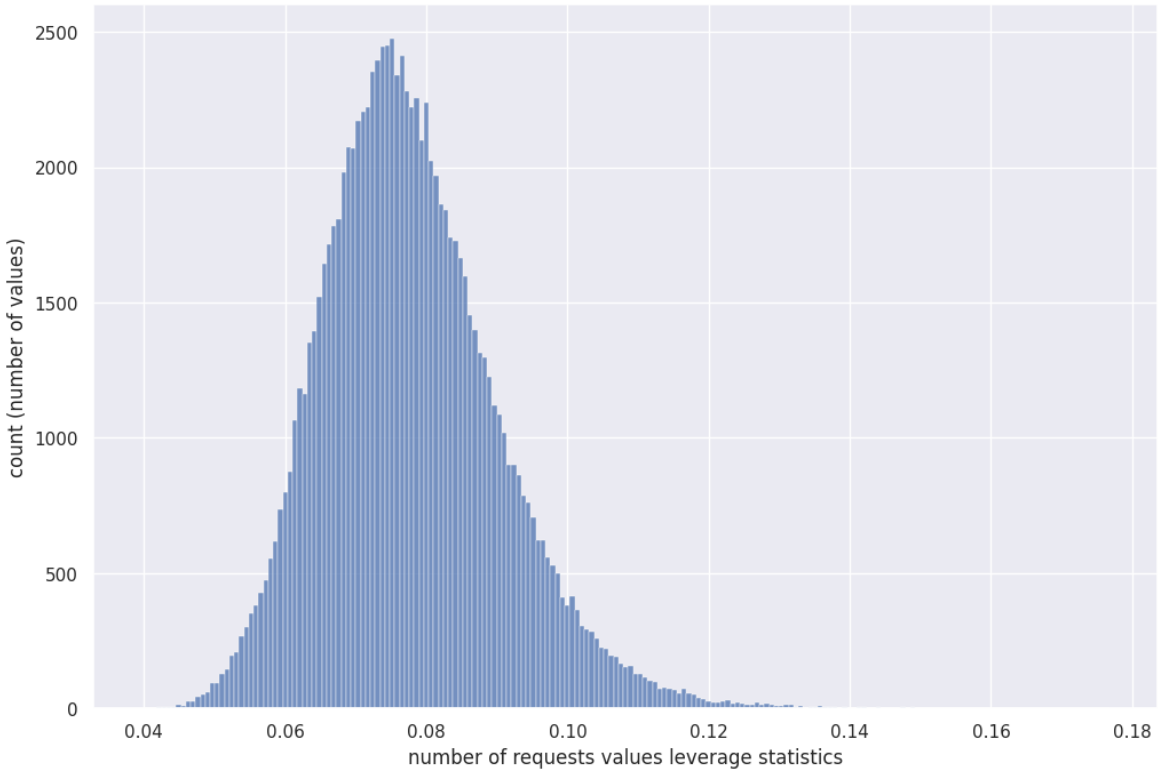

Figure 4.7 number of requests leverage statistic values

Results for the statistical method in the unidimensional case, for the variable request_processing_time:

- 10 highest leverage statistic values relate to 10 distinct observations. Based upon visual inspection of the 10 observations flagged as anomalous, all of them are spikes and we would mark them as true anomalies. All observations flagged as anomalous, i.e., all spikes, are the highest spikes on the time series plot (all other spikes have the same or smaller values).

Results for the statistical method in the unidimensional case, for the variable target_processing_time:

- 10 highest leverage statistic values relate to 10 distinct observations. Based upon visual inspection of the 10 observations flagged as anomalous, all of them are spikes and we would mark them as true anomalies. All observations flagged as anomalous, i.e., all spikes, are the highest spikes on the time series plot (all other spikes have the same or smaller values).

Results for the statistical method in the unidimensional case, for the variable response_processing_time:

- 10 highest leverage statistic values relate to 170 distinct observations. Based upon visual inspection of the 10 chosen observations (among the 170 distinct observations) flagged as anomalous, all of them are spikes and we would mark them as true anomalies. All observations flagged as anomalous, i.e., all spikes, are the highest spikes on the time series plot (all other spikes have the same or smaller values).

Results for the statistical method in the unidimensional case, for the variable received_bytes:

- 10 highest leverage statistic values relate to 10 distinct observations. Based upon visual inspection of the 10 observations flagged as anomalous, all of them are spikes and we would mark them as true anomalies. All observations flagged as anomalous, i.e., all spikes, are the highest spikes on the time series plot (all other spikes have the same or smaller values).

Results for the statistical method in the unidimensional case, for the variable sent_bytes:

- 10 highest leverage statistic values relate to 10 distinct observations. Based upon visual inspection of the 10 observations flagged as anomalous, all of them are spikes and we would mark them as true anomalies. All observations flagged as anomalous, i.e., all spikes, are the highest spikes on the time series plot (all other spikes have the same or smaller values).

Results for the statistical method in the unidimensional case, for the variable number_of_requests:

- 10 highest leverage statistic values relate to 10 distinct observations. Based upon visual inspection of the 10 observations flagged as anomalous, all of them are spikes and we would mark them as true anomalies. All observations flagged as anomalous, i.e., all spikes, are the highest spikes on the time series plot (all other spikes have the same or smaller values).

Results for statistical method in the multidimensional case:

- 10 highest leverage statistic values relate to 10 distinct observations. Based upon visual inspection of the 10 observations flagged as anomalous, all of them are spikes and we would mark them as true anomalies.

The statistical method worked well for the top leverage values which is great since we could use it in future as the baseline method. Especially since the computational complexity is low and we can raise an alarm on anomalies without looking at future values.

4.4.2. The probabilistic method

The probabilistic method for request_processing_time resulted in:

- ARIMA(0, 0, 0) with constant $4 * 10^{-4}$.
- Our model flagged 0 observations as anomalies.
- Based on Figure 4.8, there are visible spikes
- We conclude that either the assumption about the interval is wrong or the model is not working properly for this time series.

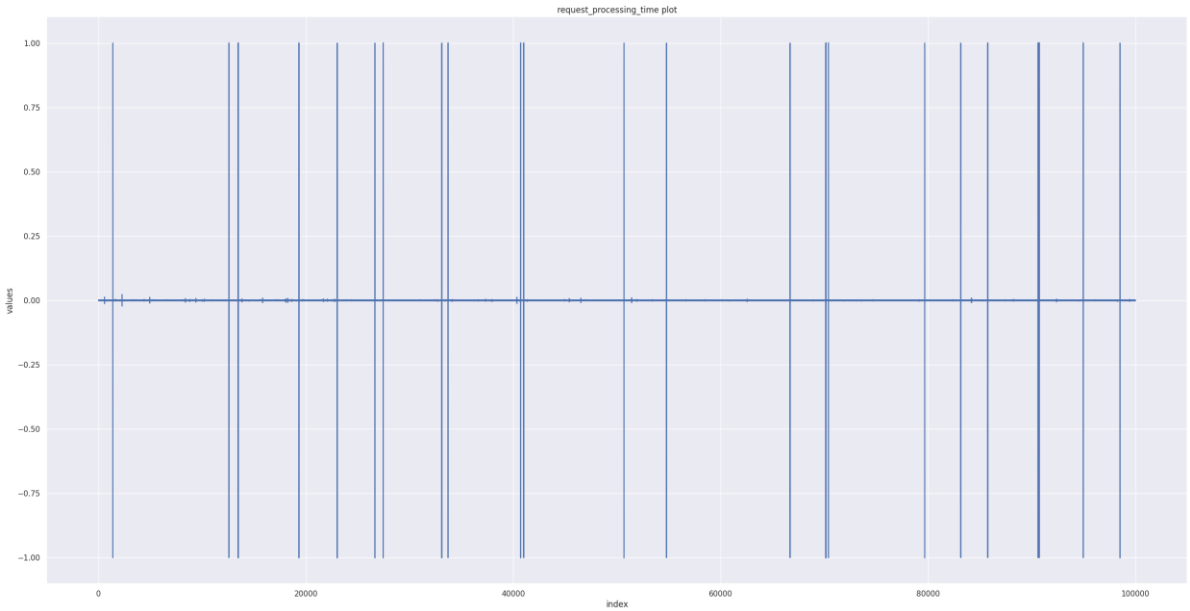

Figure 4.8 request_processing_time plot

The probabilistic method for response_processing_time time resulted in:

- ARIMA(5, 1, 0).
- Our method flagged 0 observations as anomalies.
- Based on Figure 4.9, there are visible spikes.
- We conclude that either the assumption about the interval is wrong or the model is not working properly for this time series.

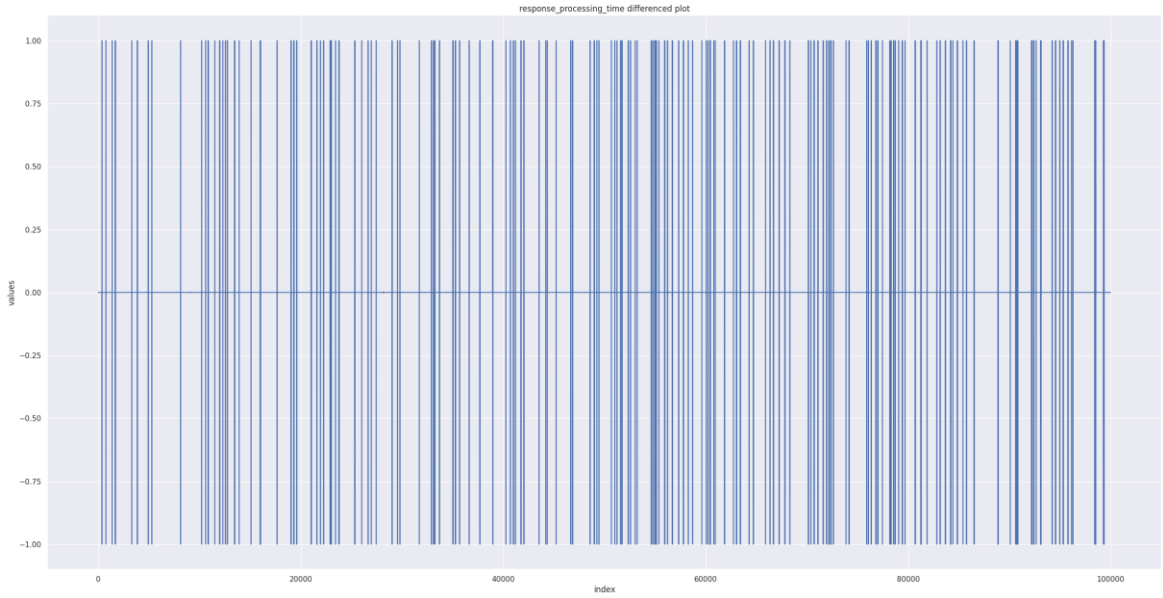

Figure 4.9 response_processing_time differenced one time plot

The probabilistic method for target_processing_time time resulted in:

- ARIMA(1, 1, 0) with drift.
- Our model flagged 4366 observations as anomalies, as shown in Figure 4.8.
- Based on Figure 4.8, almost all real spikes are flagged as anomalies.
- Based on Figure 4.8, some values that are not spikes are also flagged as anomalies.

Figure 4.8 target_processing_time anomalies plot

The probabilistic method for received_bytes time resulted in:

- ARIMA(0, 1, 1).
- Our flagged 86763 observations as anomalies, as shown in Figure 4.10.
- Based on Figure 4.10, almost all spikes are flagged as anomalies.
- Based on the number of observations that are flagged as anomalies and based on Figure 4.10, a lot of values that are not spikes are also flagged as anomalies. That is, we get a lot of false positives.

Figure 4.10 received_bytes anomalies plot

The probabilistic method for sent_bytes time resulted in:

- ARIMA(5, 1, 0).
- Our method flagged 8 observations as anomalies, as shown in Figure 4.9.
- Based on the number of observations flagged as anomalies and based on Figure 4.9, only significant spikes were flagged as anomalies.

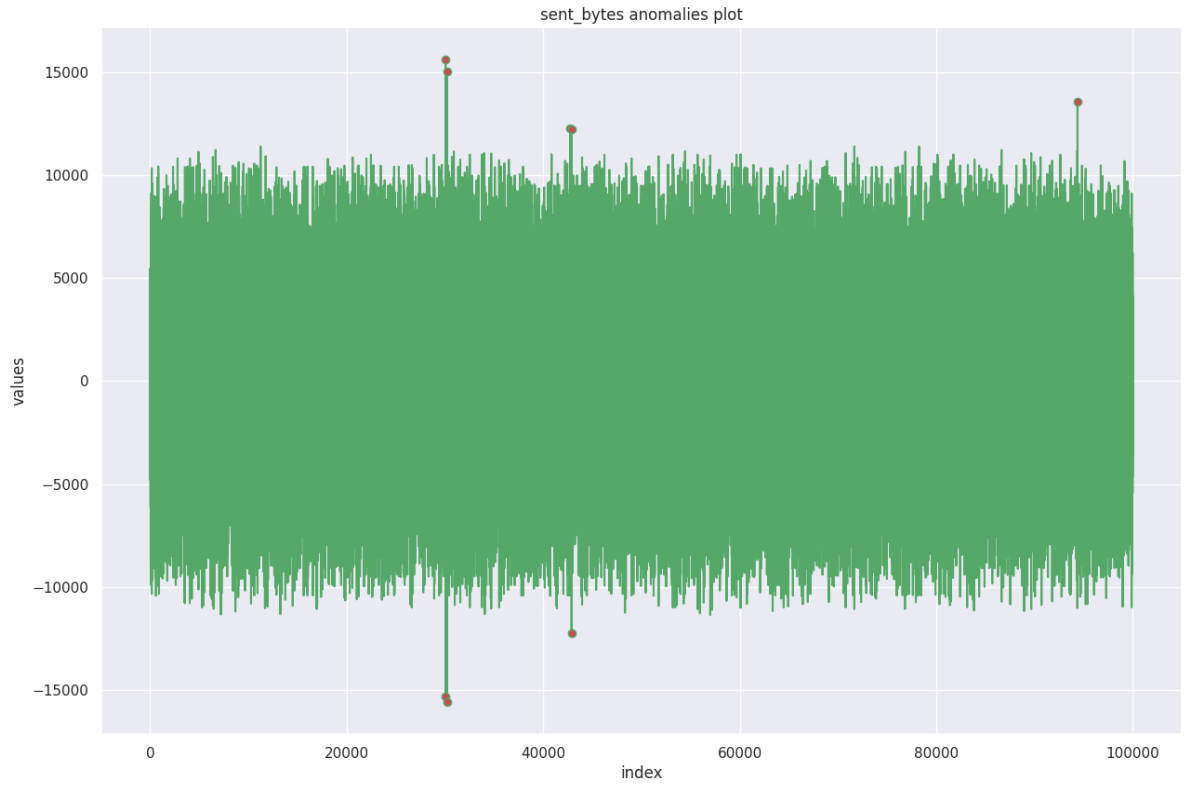

Figure 4.9 sent_bytes anomalies plot

It is interesting to notice that our approach with the probabilistic method resulted in a wide range of results: from 0 findings to a significant number of findings. It could be that our method assumptions were completely wrong but also that the ARIMA model works on some time series and for others it is not suitable, as is already well-known in the literature and, to be honest, expected.

We also made some changes to the time series time values (x's) and we are aware that this means that the ARIMA model obtained this way might not be appropriate for other time series or new periods of activity of the same time series.

Additionally, the threshold of 2 * standard error is based on normality and should be explored with a wider range of values in future research.

4.4.3. The autoencoder method

For the training and validation of the autoencoder, we only used the first 200,000 observations (before sliding the window through data) since the computational resources we

used were budget constrained. From the technical perspective, we were renting an NVIDIA A100 Tensor Core GPU for the training and validation processes.

Figure 4.10 shows the output of training and validation mean squared error for epochs while Figure 4.11 shows a plot of training and validation mean squared error for epochs. From the figures mentioned, we can see that six epochs were used.

From Figure 4.10, we can also see that:

- As the number of epochs increases, the training mean squared error decreases monotonically.
- As the number of epochs increases, validation mean squared error decreases monotonically.

We believe that overfitting did not happen based on the following definition from [19]:

- When a method produces a low training mean squared error and a high validation mean squared error, it is referred to as overfitting the data [19].
- Our method achieves a small training mean squared error and a small test mean squared error.

It is worth noting that, whether or not overfitting has happened, we typically anticipate the training mean squared error (MSE) to be lower than the test MSE [19]. This is because most statistical learning methods aim to minimize the training MSE, either directly or indirectly [19].

Some additional comments based on Figure 4.10 (from [19]):

- A monotone decrease in the training mean squared errors and a U-shape in validation mean squared errors is observed as the flexibility of the statistical learning method increases. This is a fundamental aspect of statistical learning, applicable regardless of the dataset or the specific statistical method used [19].
- We have not observed a U-shape in the validation mean squared error.

The mean squared error of the best model we obtained was 0.9992541670799255 on validation data.

```
Epoch 1/100
468/469 [==========================>.] - ETA: 0s - loss: 0.9992
Epoch 1: val_loss improved from inf to 1.00060, saving model to LSTM_best_val_mse.h5
469/469 [==============================] - 29s 29ms/step - loss: 0.9991 - val_loss: 1.0006
Epoch 2/100
  5/469 [..............................] - ETA: 6s - loss: 1.0074/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3000: UserWarning: You are savi
  saving_api.save_model(
469/469 [==============================] - ETA: 0s - loss: 0.9987
Epoch 2: val_loss improved from 1.00060 to 1.00030, saving model to LSTM_best_val_mse.h5
469/469 [==============================] - 8s 16ms/step - loss: 0.9987 - val_loss: 1.0003
Epoch 3/100
466/469 [==========================>.] - ETA: 0s - loss: 0.9982
Epoch 3: val_loss improved from 1.00030 to 1.00000, saving model to LSTM_best_val_mse.h5
469/469 [==============================] - 8s 16ms/step - loss: 0.9981 - val_loss: 1.0000
Epoch 4/100
469/469 [==============================] - ETA: 0s - loss: 0.9974
Epoch 4: val_loss improved from 1.00000 to 0.99968, saving model to LSTM_best_val_mse.h5
469/469 [==============================] - 8s 16ms/step - loss: 0.9974 - val_loss: 0.9997
Epoch 5/100
469/469 [==============================] - ETA: 0s - loss: 0.9966
Epoch 5: val_loss improved from 0.99968 to 0.99942, saving model to LSTM_best_val_mse.h5
469/469 [==============================] - 8s 16ms/step - loss: 0.9966 - val_loss: 0.9994
Epoch 6/100
466/469 [==========================>.] - ETA: 0s - loss: 0.9963
Epoch 6: val_loss improved from 0.99942 to 0.99925, saving model to LSTM_best_val_mse.h5
469/469 [==============================] - 8s 17ms/step - loss: 0.9960 - val_loss: 0.9993
```

Figure 4.10 Training and validation mean squared errors for epochs

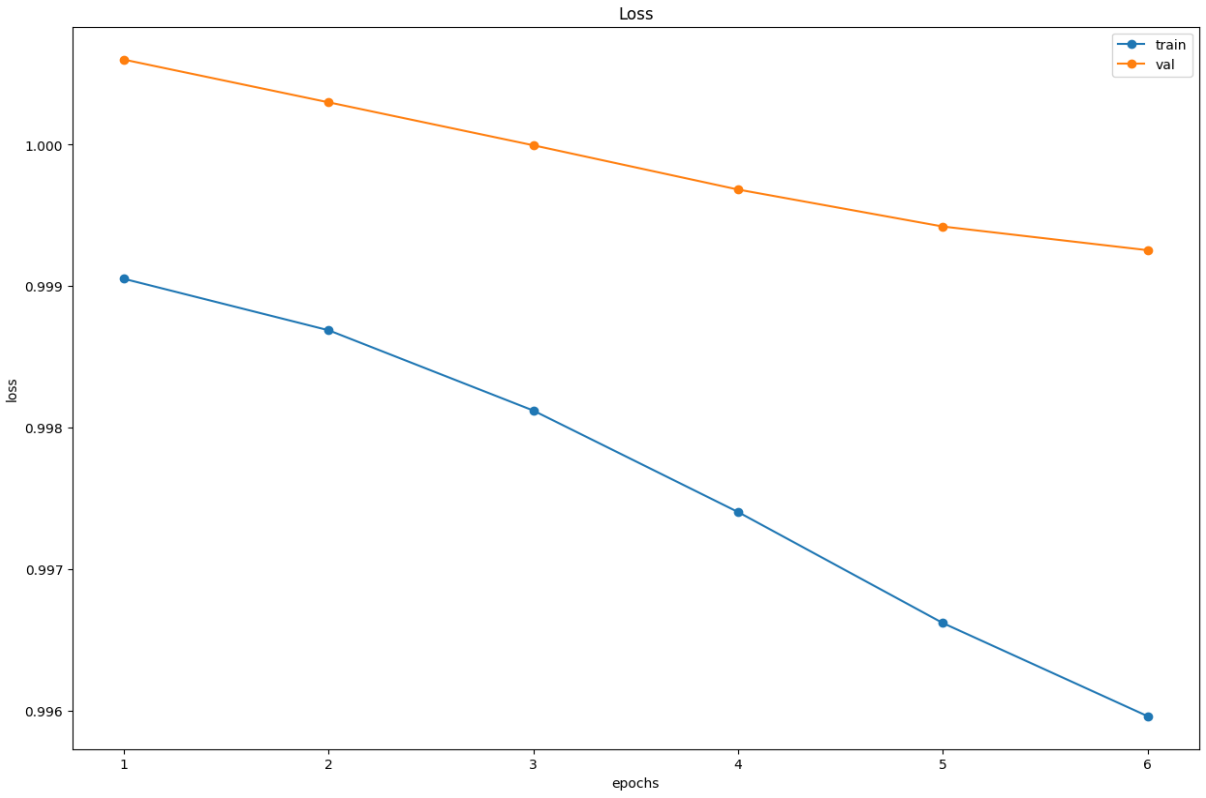

Figure 4.11 Plot of training and validation mean squared error for epochs

Figure 4.12 shows the histogram of residuals. Imposing a threshold on residuals is a challenging task but we need to start with something. Thus, based on Figure 4.12, a good starting point seems to be value 6 (all points that have residual higher than 6).

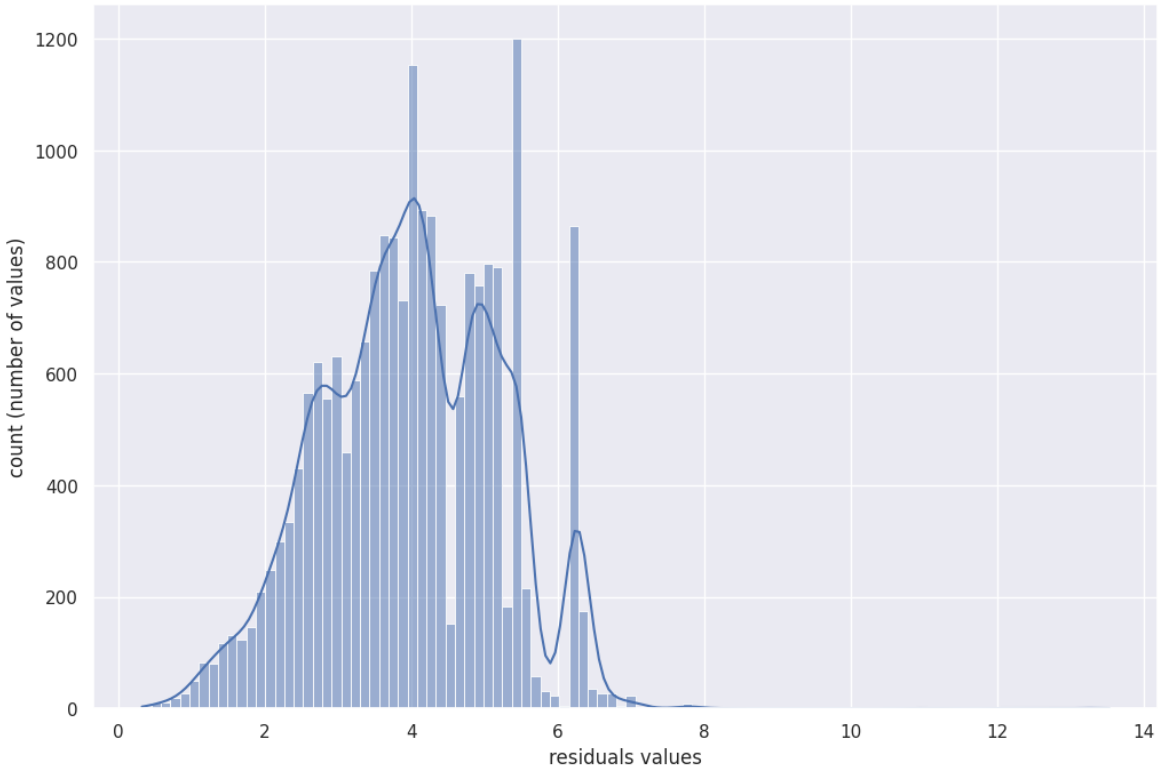

Figure 4.12 Histogram of residuals values

Results when 6 is used as the threshold:

- Our method flagged 1188 windows as anomalies.

Results when 7 is used as the threshold:

- Our method flagged 42 windows as anomalies.

Results when 8 is used as the threshold:

- Our method flagged 9 windows as anomalies.

By increasing the number used as the threshold less windows were marked as outliers and thus, we had more reason to believe these are anomalies.

If a window is flagged as anomalous, all observations that belong to that window will be considered as anomalous. If there is a value in the window that can be identified as a spike on the plot, we will accept that window as being anomalous. Based on the interpretation of the results and based on the visual inspection of each time series, the autoencoder method with the threshold value equal to 7 flagged 9 windows correctly as anomalous, of 42 windows flagged in total.

Plots used for visual inspection can be seen in Figure 4.13, Figure 4.14, Figure 4.15, Figure 4.16, and Figure 4.17.

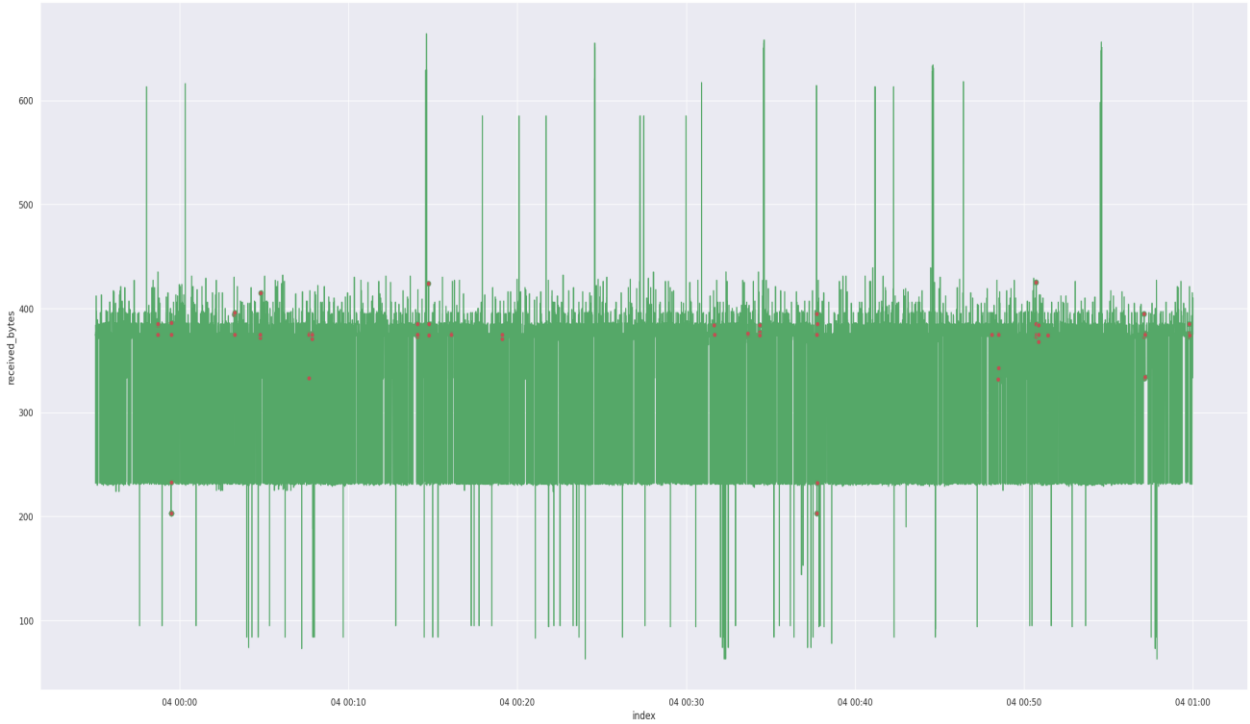

Figure 4.13 Received_bytes variable flagged observations

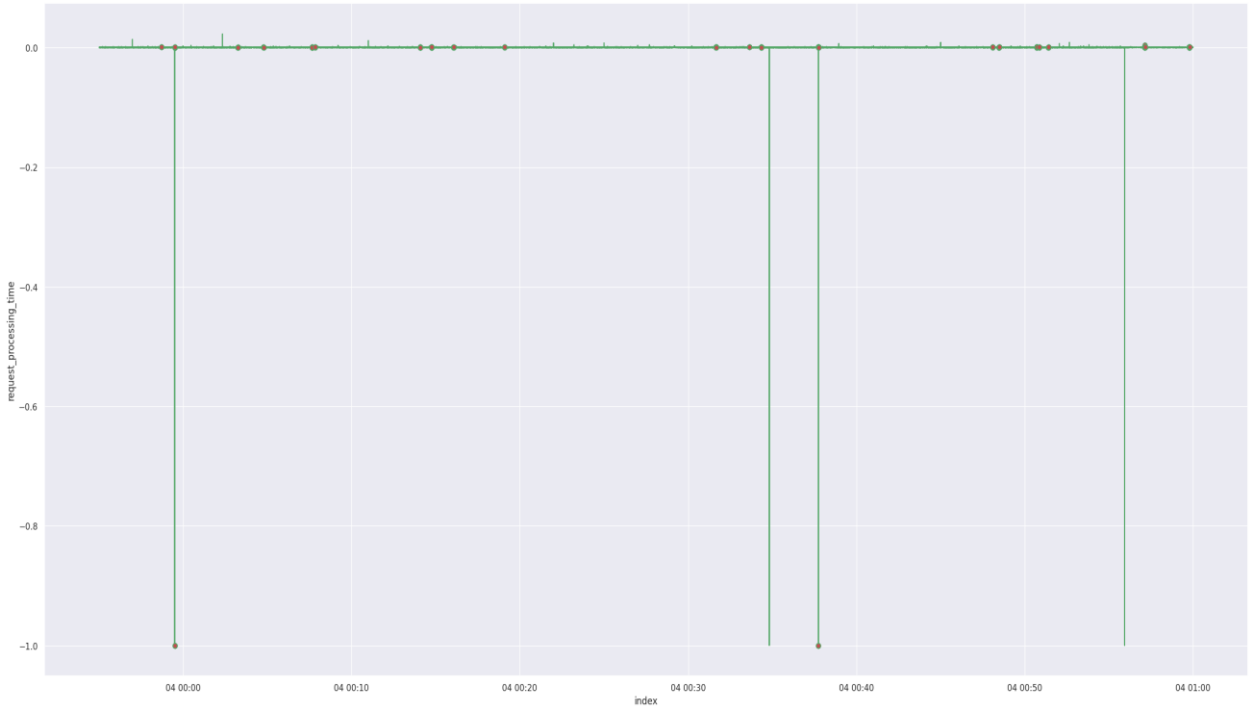

Figure 4.14 Request_processing_time variable flagged observations

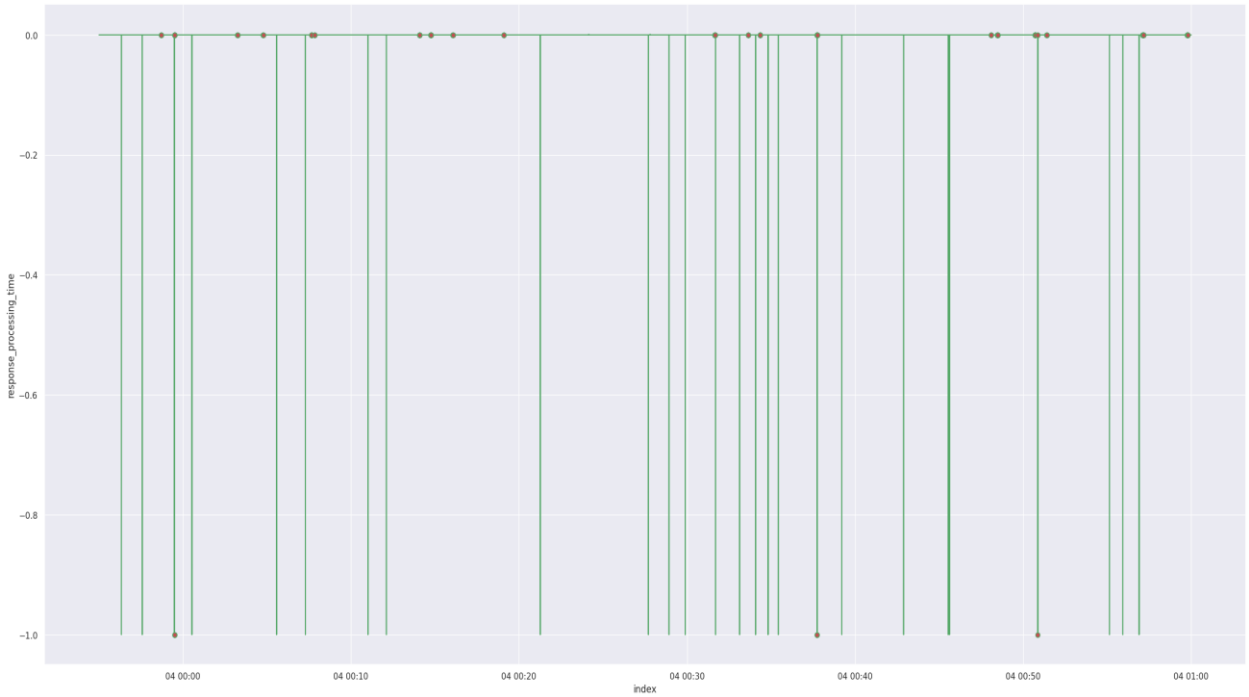

Figure 4.15 Response_processing_time flagged observations

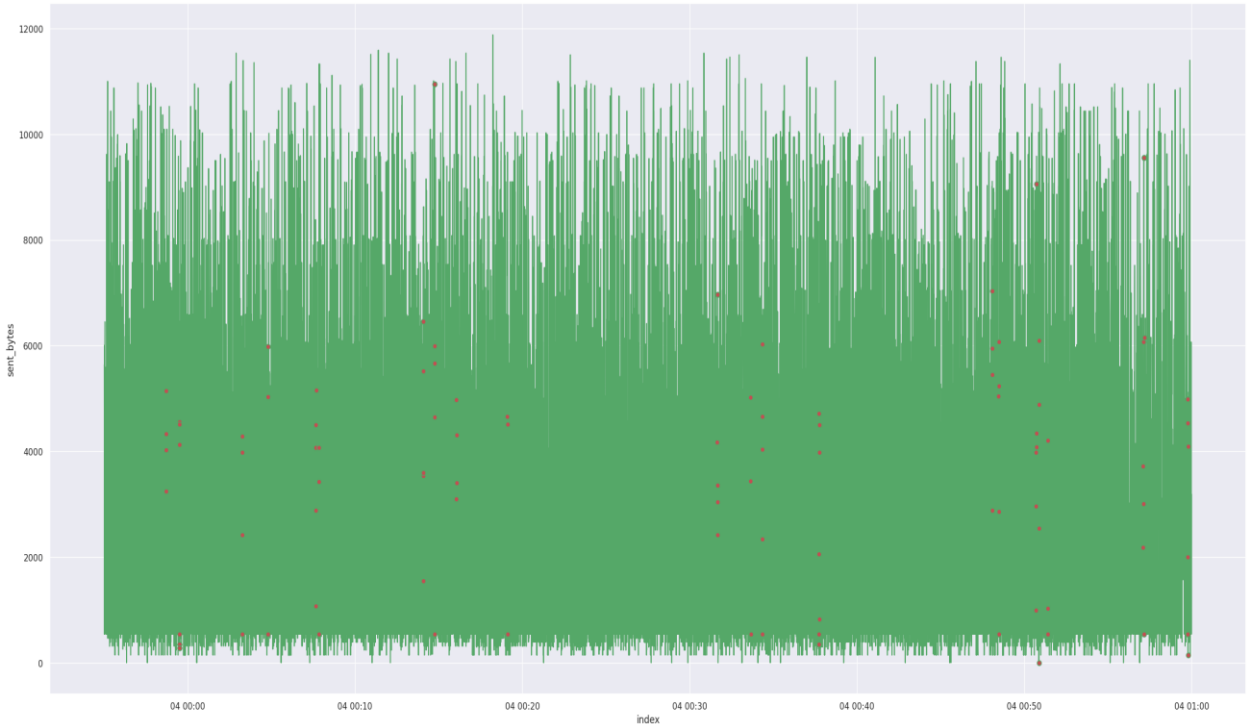

Figure 4.16 Sent_bytes flagged observation

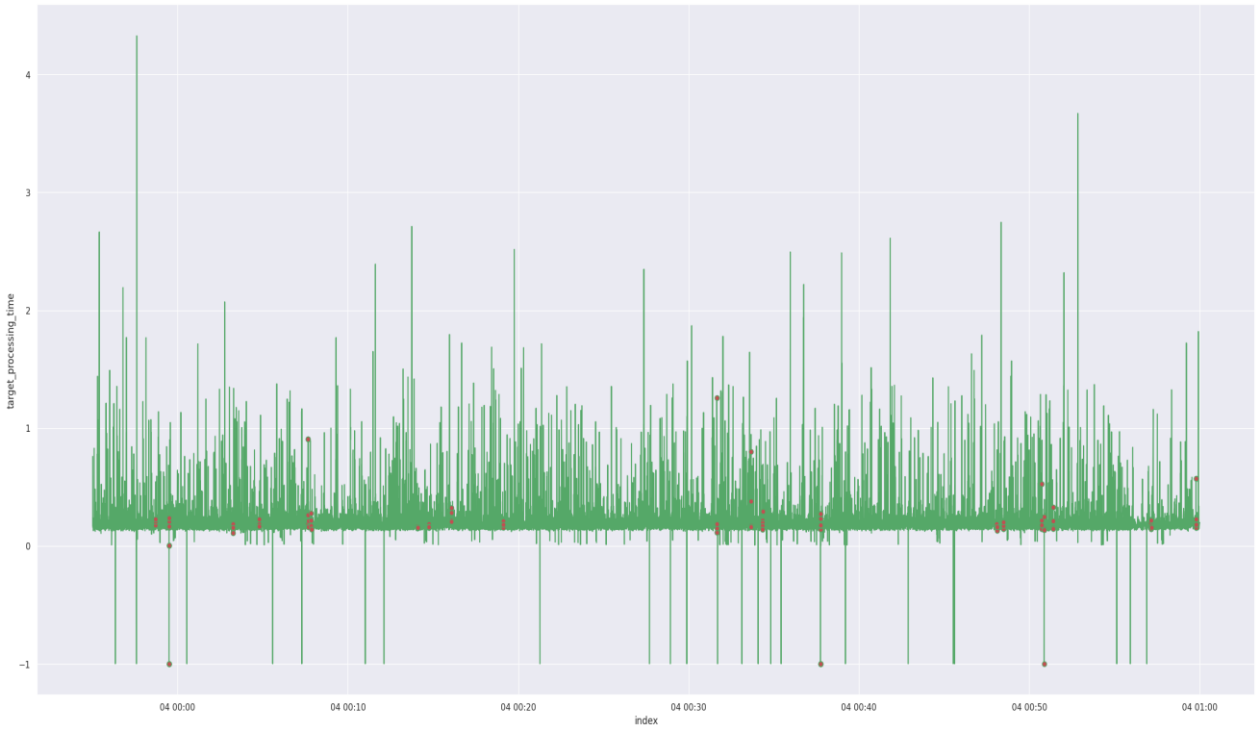

Figure 4.17 target_processing_time flagged observations

4.4.4. Answers to research questions

Since we have not found publications that directly deal with the detection of anomalies in AWS load balancer logs, we cannot answer research questions RQ1-RQ6 with more than the fact that such research is yet to be published. Regardless of that, these research questions have guided us through our own research and have proved extremely valuable in defining our approach to the subject and the research question RQ7.

Although this thesis is not meant to server as a scientific breakthrough in the subject, we think that it can serve on its own as an introductory example on how to answer the research question RQ7, i.e., by utilizing a systematic review of the literature and devising methods that can be applied within the given budget restraints. This should not be the end of course; other methods could have been applied and we have no doubt that they will be in future research on the subject.

Conclusions

The result of the systematic literature review described in Chapter 1 is the absence of publications that directly deal with the detection of anomalies in AWS load balancer logs and the absence of a benchmark dataset that would be used for the evaluation of existing and new methods. Therefore, there exists a gap in the literature that needs to be completed with a publication that will extensively evaluate the collection of chosen methods on the benchmark dataset, based on a systematic literature review encompassing the domains of time series outlier and anomaly detection, as well as log file anomaly and outlier detection.

By systematically reviewing the literature in the fields of anomaly detection, anomaly identification in time series data, and both anomaly and outlier detection within log files, we observed the need to define a baseline method that will be highly interpretable because the existing methods are based on probabilistic and deep learning methods that have many parameters and are, therefore, often uninterpretable.

By conducting the experimental part, we saw a need for domain expertise in data preprocessing, defining anomalies, and setting threshold values for all models, an act that would save a significant amount of time.

Although we believe everyone should give preference to supervised methods, if possible, we used unsupervised methods because our data set was not labeled. The above points out to the need for a labeled dataset that will be labeled by different domain experts, whose decisions will be agreed into a single decision.

By interpreting the results of the statistical method, we can conclude that something as simple as the method we've applied is a good choice for the baseline method and probably the first thing that should be applied for a proof-of-concept or a minimum viable product in industry.

The probabilistic method had variable results, but many prerequisites for ARIMA models were not thoroughly checked and thus the method we have presented should be taken as a heuristic at its best.

The autoencoder method had a lot of false positives and based on our experience with the computational complexity and requirements and the overall interpretability of autoencoders we would not recommend it as the first step in applications. The results could have been

better with more time invested into the autoencoder architecture and with more expertise in the subject itself, but it is not probable that such an investment is feasible for everyone in today's world. The shortage of AI engineers is still quite observable in many areas of the world.

Finding the appropriate threshold is hard and we should have some prior knowledge about the expected number of anomalies. On the other hand, as an exploratory technique (i.e. without raising alarms for current anomalies) or a post-mortem technique the thresholds can be varied and fine-tuned over time.

We have found that the visual introspection of potential anomalies should be based on some ranking to make it easier for users to mark findings as anomalies or just outliers. Otherwise, it is quite a tedious job as we can confirm.

The aim of this thesis was to lay a foundation for systematic literature review and to experiment with a personally collected dataset, i.e., to check the possibilities of certain computer methods in detecting anomalies for metrics from AWS ALB logs. Unfortunately, we cannot share the data we collected publicly but at least we believe we were able to present an introduction to the subject and open a path to future research and applications.

Literature

[1] Systematic Reviews (OPEN ACCESS) Page MJ, McKenzie JE, Bossuyt PM, Boutron I, Hoffmann TC, Mulrow CD, et al. The PRISMA 2020 statement: an updated guideline for reporting systematic reviews. Systematic Reviews 2021;10:89

[2] Parmezan, Antonio & Alves de Souza, Vinícius & Batista, Gustavo. (2019). Evaluation of statistical and machine learning models for time series prediction: Identifying the state-of-the-art and the best conditions for the use of each model. Information Sciences. 10.1016/j.ins.2019.01.076.

[3] Kalaki, Parisa & Shameli-Sendi, Alireza & Abbasi, Behzad. (2022). Anomaly detection on OpenStack logs based on an improved robust principal component analysis model and its projection onto column space. Software: Practice and Experience. 53. 10.1002/spe.3164.

[4] Sánchez-Zas, C.; Larriva-Novo, X.; Villagrá, V.A.; Rodrigo, M.S.; Moreno, J.I. Design and Evaluation of Unsupervised Machine Learning Models for Anomaly Detection in Streaming Cybersecurity Logs. Mathematics 2022, 10, 4043. https://doi.org/10.3390/math10214043

[5] Shin, T.-H.; Kim, S.-H. Utility Analysis about Log Data Anomaly Detection Based on Federated Learning. Appl. Sci. 2023, 13, 4495. https://doi.org/10.3390/app13074495

[6] Zhang, Bo & Zhang, Hongyu & Moscato, Pablo. (2020). Anomaly Detection via Mining Numerical Workflow Relations from Logs. 10.36227/techrxiv.12570926.v1.

[7] Caiping Hu, Xuekui Sun, Hua Dai, Hangchuan Zhang & Haiqiang Liu. (2023) Research on Log Anomaly Detection Based on Sentence-BERT. Electronics 12:17, pages 3580.

[8] M. Fält, S. Forsström, Q. He and T. Zhang, "Learning-Based Anomaly Detection Using Log Files with Sequential Relationships," 2022 6th International Conference on System Reliability and Safety (ICSRS), Venice, Italy, 2022, pp. 337-342, doi: 10.1109/ICSRS56243.2022.10067856.

[9] D. Hawkins. Identification of Outliers, Chapman and Hall, 1980.

[10] Blázquez-García, Ane & Conde, Angel & Mori, Usue & Lozano, Jose. (2021). A Review on Outlier/Anomaly Detection in Time Series Data. ACM Computing Surveys. 54. 1-33. 10.1145/3444690.

[11] Aggarwal, Charu C., and Charu C. Aggarwal. An introduction to outlier analysis. Springer International Publishing, 2017.

[12] Aggarwal, Charu C., and Philip S. Yu. "Outlier detection for high dimensional data." Proceedings of the 2001 ACM SIGMOD international conference on Management of data. 2001.

[13] Freeman, Cynthia & Merriman, Jonathan & Beaver, Ian & Mueen, Abdullah. (2021). Experimental Comparison and Survey of Twelve Time Series Anomaly Detection Algorithms. Journal of Artificial Intelligence Research. 72. 849-899. 10.1613/jair.1.12698.

[14] Chandola, Varun & Banerjee, Arindam & Kumar, Vipin. (2009). Anomaly Detection: A Survey. ACM Comput. Surv.. 41. 10.1145/1541880.1541882.

[15] Schmidl, Sebastian & Wenig, Phillip & Papenbrock, Thorsten. (2022). Anomaly detection in time series: a comprehensive evaluation. Proceedings of the VLDB Endowment. 15. 1779-1797. 10.14778/3538598.3538602.

[16] Paulheim, Heiko & Meusel, Robert. (2015). A decomposition of the outlier detection problem into a set of supervised learning problems. Machine Learning. 100. 509-531. 10.1007/s10994-015-5507-y.

[17] Hyndman, Rob J., and George Athanasopoulos. Forecasting: principles and practice. OTexts, 2018.

[18] PMF Sveučilište u Zagrebu, Bojan Basrak. "Statistika s primjerima u R-u", Available Online https://web.math.pmf.unizg.hr/~bbasrak/pdf_files/BiostatNotes0216.pdf

[19] James, Gareth, et al. An introduction to statistical learning. Vol. 112. New York: Springer, 2013.

[20] Amazon Web Services. "What is an application load balancer", Amazon Web Services Documentation. Available Online https://docs.aws.amazon.com/elasticloadbalancing/latest/application/introduction.html

[21] Amazon Web Services. "Access logs for your Application Load Balancer", Amazon Web Services Documentation. Available Online https://docs.aws.amazon.com/elasticloadbalancing/latest/application/load-balancer-access-logs.html

List of figures

List of tables