ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

JOHN VON NEUMANN
FACULTY OF INFORMATICS

THESIS

OE-NIK	Student's name:	**Laryssa Karen Almeida Duarte Ferreira**
2021	Student's registration number:	**T/005507/FI12904/N**

Óbudai University
John von Neumann Faculty of Informatics
Software Engineering Institute

THESIS WORK SPECIFICATION

Name of the student:	**Laryssa Karen Almeida Duarte Ferreira**
Identification number of the student:	T/005507/FI12904/N
Neptun's code:	[redacted]

Title of the thesis:

Automatic Major Incident Recognition Based on Alert Logs
Súlyos események automatikus felismerése riasztási naplók alapján

Internal supervisor	Dr. Gábor Kertész
External supervisor:	József Kertész (SAP Hungary)
Deadline:	15 December 2021
Subjects of the final exam:	Computer Architectures Cloud service technologies and IT security specialization

The task:

The monitoring tool created by SAP for internal use is an easy to use single pane of glass for SAP's infrastructure monitoring, it aggregates alerts from various monitoring tools into a single tool. This monitoring tool aims to help recognize, react, respond, fix major infrastructure issues faster.
Currently the classification of major incidents (MI) is done in a hard-coded way. The tool gets alerts from several monitoring tools and creates indexes with all the information about those alerts. If the alerts hit a threshold, it will create an alert notification. A person will review the notification, and if an MI is detected, responsible engineers are addressed.
In this thesis, the aim is to substitute the hardcoded threshold with an intelligent solution. Based on the alerts that the tool gets from several monitoring tools, an artificial intelligence (AI) model automatically decides if an MI is going on or not, and if necessary, handles the incident similarly as given above.

The thesis must contain:

- detailed literature review of relevant topics,
- introduction of similar methods, intelligent solutions,
- detailed description of the existing environment,
- the design of the system, including the plans for the AI,
- implementation and detailed description of the solution,
- test documentation,
- user guide,
- summary of results, further plans.

Place of stamp

ÓBUDAI EGYETEM Neumann János Informatikai Kar 1. Budapest

.....
Dr. Zoltán Vámossy
head of institute

This specification is valid until: **15 December 2023**
(According to OE TVSz 55.§)

I consider the thesis suitable for submission:

.....
external supervisor

.....
internal supervisor

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

John von Neumann Faculty of Informatics

DECLARATION

I hereby declare that this thesis is the result of my own work, every tool used and every source referred is cited identifiably. The results presented in the thesis can be used by the University and the institute that gave the task free of charge.

Budapest, 2021/12/15.....

Ramyna Karen A. D. Fonseca
.....

Signature of the student

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

John von Neumann Faculty of Informatics

Log of consultations

Name of the student: Laryssa Karen Almeida Duarte Ferreira

Neptun code: [redacted]

Program: Computer Engineering BSc

Phone number: [redacted]

Address: [redacted]

Title of the thesis work:

Automatic Major Incident Recognition Based on Alert Logs

Internal supervisor: Dr. Gábor Kertész

External supervisor: József Kertész

Please use BLOCK CAPITAL LETTERS to fill in this form!

Occasion	Date	Topics discussed	Signature
1.	04/10/2021	REFINE USE CASE	
2.	17/11/2021	DATA TRANSFORMATIONS	
3.	23/11/2021	PINGDOM DATA	
4.	07/12/2021	ISOLATION FOREST	

This log should be signed on each consultation (by a supervisor), 4 times altogether.

The student fulfilled the requirements of the Thesis work subject, I allow his/her participation on the presentation/defence[1].

.....

Internal supervisor

Budapest, 20.21. 12. 13.

[1] Underline the appropriate one.

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

John von Neumann Faculty of Informatics

Log of consultations

Name of the student: Laryssa Karen Almeida Duarte Ferreira

Neptun code: [redacted]

Program: Computer Engineering BSc

Phone number: [redacted]

Address: [redacted]

Title of the thesis work:

Automatic Major Incident Recognition Based on Alert Logs

Internal supervisor: Dr. Gábor Kertész

External supervisor: József Kertész

Please use BLOCK CAPITAL LETTERS to fill in this form!

Occasion	Date	Topics discussed	Signature
1.	11/02/2021	MOM ALARMS DASHBOARD	
2.	26/02/2021	WORKFLOW OF MOM	
3.	13/03/2021	LABELLING SYSTEM	
4.	20/04/2021	ACCESSING ELASTICSEARCH	

This log should be signed on each consultation (by a supervisor), 4 times altogether.

The student fulfilled the requirements of the Thesis work subject, I allow his/her participation on the presentation/defence[1].

.....

Internal supervisor

Budapest, 20.21. 05. 10.

[1] Underline the appropriate one.

Contents

Abstract

Machine Learning is a rapidly emerging topic that is now receiving a lot of attention. Major Incident Detection is one of its applications, which may be used to monitor systems in order to proactively detect and avoid failures. When a significant catastrophe occurs, each minute that a system fails might cost a lot of money. It can reduce the impact on a business and its customers by minimizing the time it takes to detect it and restore services. The goal of this thesis was to enhance a major incident management tool by employing machine learning techniques to create an early recognition system for major incidents.

1. Introduction

Although each organization might define major incident (MI) differently, a MI can be defined as an event with important impact or urgency for the business and it demands a response beyond the routine of incident management.

When a major incident happens, every minute costs money. We can minimize the impact on a business and its customers by reducing the amount of time it takes to identify it and restore services. That is one of the reasons why a monitoring tool was developed to be used inside SAP.

The monitoring tool is an easy-to-use "single pane of glass" for SAP's infrastructure, it aggregates alerts from various monitoring tools into a single one with the purpose of helping recognize, react, respond, and fix major infrastructure issues faster.

This same system and processes may hold clues that could have helped predict and prevent a major incident. A log file is a data file created by a computer that contains information about events that occur while an operating system or other software is running. In the context of this thesis, alert log files contain information from multiple monitors of SAP infrastructure. Upon analyzing these alert logs it is possible to determine the infrastructure's health and current state, helping to identify and troubleshoot problems.

The goal of this thesis is to enhance major incident management by using machine learning techniques to create an early recognition system for these incidents.

2. Problem Specification

Currently the classification of major incidents is done in a hard-coded way. The monitoring tool gets alerts from several monitoring tools and creates indexes with all the information about those alerts. If the alerts hit a threshold, it will create a notification alert. A person will review the notification alert, and if a MI is detected, the responsible engineers are addressed.

In the scope of this thesis, the type of MI in focus was HTTP endpoints experiencing problems. The goal was to analyze response-time from Pingdom checks in order to do proactive triggering under degradation situations or before the connection goes down. Pingdom is a software as a service solution, which is configured to watch HTTP endpoints every minute. Pingdom assists in providing third-party evidence of meeting key service objectives and service-level agreements (SLAs). The data collected contains information about each of the measures it checks, for example, if a website hosted by your service is down, slow, or not working as it should [1].

Based on Pingdom check alerts, an artificial intelligence (AI) model automatically decides if a MI (HTTP endpoint issue) is happening or not, and if necessary, handles the incident similarly as mentioned above.

The dataset that was used contains, among other information, URL response-times, event status and event severity (more details about the dataset are described in the section 6.1.). As there is no label pointing to an anomalous situation in the dataset, the approach was to use Unsupervised Learning to label the data as ‘outlier’ or ‘inlier’. Since historical alerts were used, i.e., there was no constant stream of data entering the system, batch learning technique was the choice. In this approach, the model is trained using all the available data and after it is launched it runs without further learning.

3. Literature Review

3.1. Unsupervised Learning

The training data is unlabeled in unsupervised learning. The system attempts to learn without the assistance of a teacher. Unsupervised learning algorithms are used for a variety of purposes, such as Clustering, Visualization and dimensionality reduction, and Association rule learning [2].

In dimensionality reduction the goal is to simplify the data while retaining as much information as possible. One method is to combine several correlated features into a single one [2].

Anomaly detection is another important unsupervised task. The system is trained on normal instances, and when it encounters a new one, it can determine whether it appears to be normal or if it is most likely an anomaly [2]. The algorithm used for Anomaly Detection in this thesis was Isolation Forest and it will be discussed in detail in the following section.

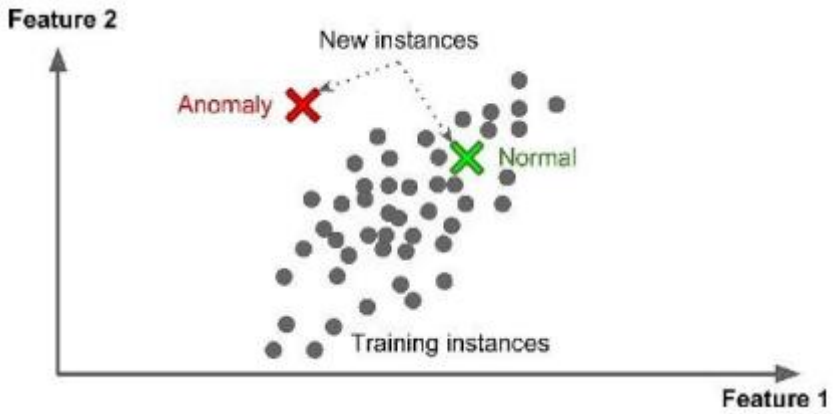

Figure 3.1: Anomaly detection [1, p. 29]

3.2. Isolation Forest

As already mentioned before, anomaly detection is the action of detecting noise or deviations from the expected pattern of a particular dataset. It is an important application in machine learning, considering that such anomalies in a dataset typically translates to problem like an unusual network traffic, fraud, a failing machine in a server, a cyber-attack, event detection in sensor networks, or simply identify data for cleaning before analysis [4].

Depending on the condition of the dataset, the anomaly detection can be done in supervised, semi-supervised ("clean"), or unsupervised manner, as it is illustrated in Figure 3.2. In the supervised case, the anomaly detection is based on labeled data (the training data) that is then used to forecast abnormal events (the testing data) [5]. As for the "clean" case, the training data is presumed to be composed of only nominal data points, then, during testing, anomalies might be identified from inside the dataset[4]. Finally, the unsupervised case, is when the dataset does not have any label, also there is no distinction between a training and a test dataset. To detect the anomalies, usually, distances or densities methods are used [6].

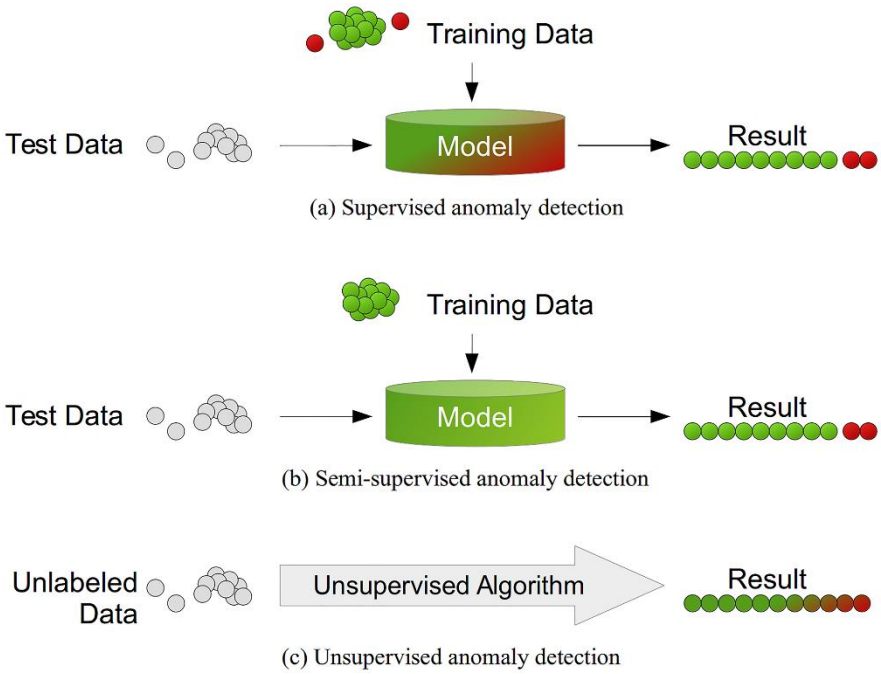

Figure 3.2: Different anomaly detection modes depending on the availability of labels in the dataset [7].

One of the most efficient learning algorithms for unsupervised anomaly detection is called Isolation Forest, and it was proposed by Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. This algorithm is based on the fact that anomalies are in less number and different compared to nominal data, making them susceptible to isolation [8]. Still according to the paper published, the Isolation Forest is built by an ensemble of Isolation Trees (binary trees) for the given dataset, and the anomalies are more likely to be isolated near the root of an Isolation Tree while normal points are more likely to be isolated farther from the root, therefore anomalies are those instances which have short average path lengths on these trees [8].

The following steps are applied to a dataset to perform this algorithm. This process

is continued recursively until each observation is isolated or until the defined maximum depth is reached:

1. Choose a feature at random (from the set of all N features).
2. Randomly define a threshold (any value between the minimum and maximum values of the previously chosen feature). If the value of a data point is less than the elected threshold, it is routed to the left branch; otherwise, it is routed to the right.

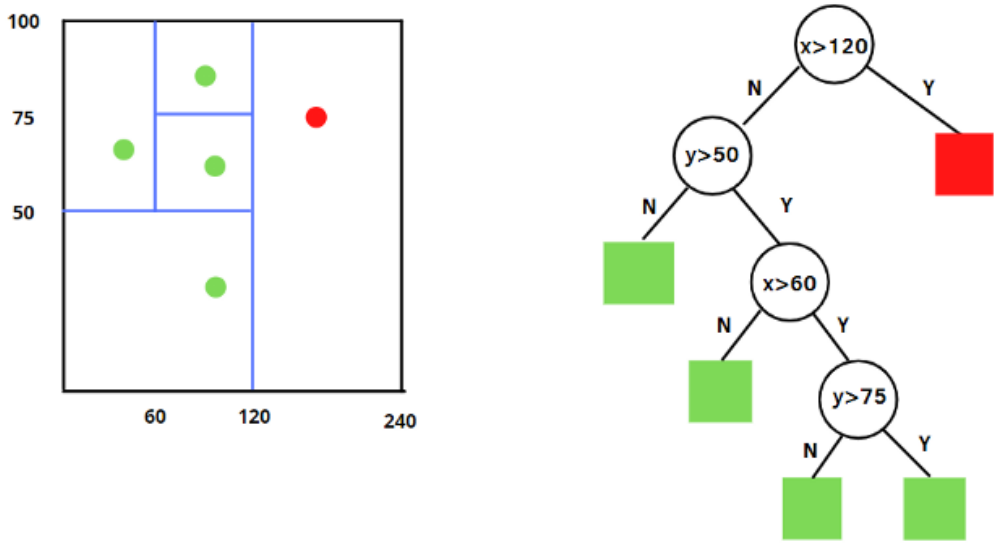

Figure 3.3: How Isolation Forest works [7].

Afterwards, an Anomaly Score is assigned to each observation, and it is used to define how anomalous the data point is, defined as:

$$s(x,n) = 2^{-\frac{E(h(x))}{c(n)}}$$

Where *E(h(x))* is the average of *h(x)* from a collection of Isolation Trees, *h(x)* is the path length from the root node to the external node *x*, and *c(n)* is the average of *h(x)* given *n* and is used to normalize *h(x)*[8].

Based on this score, there are three possible conditions as E. Anello proposes:

- When the score is near to 1, the path length is very small, and it indicates anomaly.
- When the score is much smaller than 0.5, the path length is large, and it indicates normal data point.
- If all scores are near to 0.5 then the whole sample might not have clearly separate anomalies[7].

However, in the implementation of sklearn version 0.24.2, used in this thesis, outliers are represented by -1 and inliers by 1.

3.3. Data Preprocessing

The goal of data preprocessing is to make the raw data more machine learning algorithm's friendly. One of the main challenges when we are working with machine learning models is to be able to provide the right data for our model, so that the quality of the results will be as desired. Poor quality data which is erroneous or inconsistent as well as irrelevant features can lead to inaccurate or false results. That is why it is highly important to correct or remove the inadequate data from our dataset first before we can start feeding it into our machine learning model. This process of preparing the data is called data cleaning.

The data should have the following characteristics to make learning easier:

- Take small values - Usually, most values should fall between 0 and 1.
- Be homogenous - In other words, all features should have values in approximately the same range [3].

3.3.1. Handling Missing Values

As a general rule, it is safe to use 0 representing missing values with neural networks, provided 0 is not already a meaningful value. Following data exposure, the network will learn that the value 0 represents missing data and will begin to ignore the value [3]. Other solutions are deleting the rows with missing value or filling in the missing values (imputation). At the same time, it is important to be aware that deleting lines of data can lead to bias, and if that is the case, imputation is a better solution.

3.3.2. Feature Extraction

Thousands, or even millions, of features are involved in many Machine Learning problems. This not only heavily slows down training, but it also makes finding a good solution much more difficult. The curse of dimensionality is a term used to describe this problem [2].

In real-world problems it is frequently possible to significantly reduce the number of features, transforming an intractable problem into a tractable one. One way to do it is by using Feature Extraction. It is the process of using your own knowledge of the data and the used machine learning algorithm to make the algorithm work better by applying hardcoded (non-learned) transformations to the data before it is fed into the model. In many cases, expecting a machine learning model to learn from completely arbitrary data

is unrealistic. The data must be presented to the model in a way that facilitates the model's work [3].

By reducing the number of dimensions down to two (or three) it is possible to plot a high-dimensional training set on a graph and visually detect patterns to gain valuable insights [2].

3.3.3. Label Encoding

It is common to datasets to contain text or categorical values. For example, features as color, gender, month etc. However, the majority of algorithms rely on numerical values to produce state-of-the-art results, thus it is important to transform categorial values into numerical values.

As explained by Yadava [9], there are many ways to accomplish this task, for example: One-Hot-Encoding and Label-encoder. The SciKit-learn, a Python library, includes both encoders.

Label encoding approach consists of converting each value in a column into a number. As a result, depending on the values and type of the data, label encoding brings a new problem since it uses number sequencing. The problem with using numbers is that they introduce comparison/relationship between them. The algorithm may incorrectly assume that the data have some sort of hierarchical structure or order and assign them different weights.

4. Similar Methods

In the context of this project, alert logs hold chronological information about issues happening on the infrastructure of datacenters and on the application side. They help to monitor the health of the infrastructure, troubleshoot incidents, checks if application is overloaded and so on.

In two of the following studies presented here, the data to be processed are system logs. The primary aim of a system log is to capture system states and noteworthy events at various critical points in order to aid in the debugging of system failures and root cause investigation [10]. Thus, the approach to process system logs and alerts logs are related.

The third study focus on anomaly detection in cloud data centers, bringing a new approach to process the high volume and very dynamic information of infrastructure monitoring.

4.1. Studies

DeepLog is a general-purpose framework that uses a deep neural network-based technique for online log anomaly detection and diagnosis. It uses a Long Short-Term Memory (LSTM) to model sequences of log entries, allowing it to automatically learn a model of log patterns from regular execution and flagging deviations as anomalies.

First, unstructured, free-text log entries are parsed and converted into a structured representation, making possible to learn a sequential model over this structured data. Extracting a "log key" from each log entry is an effective method. The log key of a log entry *e* denotes the string constant *k* from the print statement in the source code that printed *e* during the execution of the code. DeepLog keeps into a vector $\vec{v}$e the parameter values for each log entry *e*, as well as the time elapsed between *e* and its predecessor. DeepLog uses this vector in addition to the log key [10].

The log key anomaly detection model, the parameter value anomaly detection model, and the workflow model to diagnose discovered anomalies are the three core components of DeepLog's architecture [10].

Another relevant study explored the viability of using Isolation Forest, also referred to as iForest, an anomaly detection method based on isolation, to discover anomalies in large-scale cloud data centers. Isolation-based approaches have the benefit of being simple to train and detect, as well as being optimized for detecting failures [11].

Isolation Forest explores the fact that anomalies are data points that are far from the dataset's usual positions. Because of this trait, random partitions in the attributes space

are more likely to separate these points from regular points. It means that isolating an anomaly requires fewer random partitions in the attributes space than isolating a regular point [11].

The study concludes that one of the benefits of iForest over its competitors is the method's ability to detect abnormalities quickly, which makes it suited for real-time anomaly detection.

4.2. Tools

Besides the researches mentioned previously, there are already available in the market solutions for major incident detection.

Predictive Intelligence by ServiceNow [12] flags similarity across issues using machine learning. It compares the similarity of text within open incidents across the enterprise so IT workers can quickly identify and then propose new major incidents. After identifying links between alerts, incidents and knowledge articles, the tool may propose material to agents.

The tool uses natural language processing (NLP), machine learning, and deep learning techniques to quickly analyze and compare records across all ServiceNow applications

Another solution is Splunk IT Service Intelligence (ITSI) [13], a monitoring and analytics tool for IT Operations (AIOps). It gives an insight into the health of critical IT and business services, as well as their infrastructure. The tool uses ML to establish a baseline of normal behavior, receive alerts when abnormal conditions occur, and dynamically adjust thresholds in real-time.

Elasticsearch offers a X-Pack that includes anomaly detection for analyzing time series data by establishing precise baselines of normal behavior and detecting abnormal patterns in the dataset. This solution uses a combination of different approaches such as Bayesian distribution modeling, clustering, various types of time series decomposition, and correlation analysis [14].

5. Description of the existing environment

To set up a deep-learning workstation Windows Subsystem was chosen for Linux 1, as it is recommended to work in a Unix environment and the available machine contains Windows 10 Enterprise. There are two versions of the WSL, the version 1 was used as the second one could not work properly due to firewall and VPN settings configured in the computer.

For developing in Python, the IDE used was Visual Studio Code with Visual Studio Code Remote - WSL extension which allows the development in a Linux-based environment, with Linux-specific toolchains and utilities, and enables running and debugging Linux-based applications from Windows [15].

It was also utilized the Jupyter Notebook, which is a web application for writing and sharing documents that contains code, visualizations, and text-based documents [16]. It is especially helpful its cell-based approach, which in the case of working with data frames, allowing to import new data without the need to reload the previous imported ones.

Inside the WSL, the following Python libraries for data science were installed [17]:

- NumPy: it is used to process arrays and matrices. It provides high-performance multidimensional array objects and support to work with them.
- Pandas: it provides fast, flexible and easy-to-use data structures, data analysis and manipulation for numerical tables and timeseries.
- Scikit Learn: it integrates many state-of-the-art machine learning algorithms for supervised and unsupervised tasks. Some examples of algorithms are random forests, k-means clustering, decision trees, naïve bayes, mean shift, cross-validation, isolation forest etc.
- Matplotlib: it is serves as a visualization utility through plotting and provides an API for embedding plots into applications.
- Seaborn: it is based on Matplotlib, used for data visualization with a high-level interface for drawing aesthetic and useful statistical graphics.

6. Design of the System

6.1. The Dataset

The data used in this thesis is a dataset obtained from an internal SAP monitoring tool. The data was collected by the data centers' monitoring software and aggregated inside one single tool. The tool receives alert logs constantly, 24/7, for this reason there is a huge amount of information. The dataset contains monitoring information from websites, as a result of Pingdom checks.

The historical data used was from about two and half months of monitoring. The alerts have eighty-three attributes altogether, but not all attributes were used. Among the set of attributes, there are status, time_response, timestamp and severity (important to note that in this thesis and in the code, the attribute names in some places are referred to as event_status, last_time_response, event_timestamp_updated and event_severity, respectively). The original information about the other features cannot be displayed and explained in detail here due to confidentiality issues.

The records were stored in Elasticsearch and accessed using a Python script, described in the section 7.1, then they were preprocessed with the help of Pandas and NumPy.

The existing attribute types are timestamp, numerical and text. From those attributes, some were discarded as they were not relevant for the problem and others were transformed to give more meaningful information as well as to be more amenable to algorithms. This includes handling missing values, feature extraction, label encoding, and other steps described in the section 7.3.

In summary, the characteristics of the dataset are:

- 110681 logs, which after cleaning decreases to 27597.
- Two and half months of historical data.
- The dataset describes Pingdom checks, which measures the latency of the websites it monitors. The logs contain, among other information, URL response times and event statuses.
- Each record (row) in the data frame corresponds to a website’s availability in every minute.
- The data is saved in .pkl format.

6.2. The Model

The decision to use Isolation Forest Algorithm is based in the following logic:

a) The dataset is very imbalanced, so it could be treated as anomaly detection problem.
b) Isolation forest was proven to be very efficient for this type of problem, as described in section 3.2.
c) The data in hands are not labelled, thus it was necessary to use an unsupervised learning algorithm.
d) This algorithm can be used for small datasets.
e) Small memory requirement, which is an important characteristic of the environment that was available.
f) The algorithm does not require a "normal behavior" dataset.

The model is integrated on Scikit-learn, a general-purpose high-level language that focuses on bringing machine learning to non-specialists. Isolation Forest Algorithm is under the ensemble module that includes ensemble-based methods for classification, regression, and anomaly detection.

The algorithm returns the anomaly score of each sample, 1 for inliers and -1 for outliers. As declared in the official scikit-learn website, the parameters are:

- n_estimators: The number of base estimators in the ensemble.
- max_samples: The number of samples to draw from X to train each base estimator.
- contamination: The amount of contamination of the data set, i.e., the proportion of outliers in the data set.
- max_features: The number of features to draw from X to train each base estimator.
- bootstrap: If True, individual trees are fit on random subsets of the training data sampled with replacement. If False, sampling without replacement is performed.
- n_jobs: The number of jobs to run in parallel for both fit and predict.
- random_state: If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by np.random.
- verbose: Controls the verbosity of the tree building process. [18]

The idea for this thesis was to train the model with different combinations of parameters.

First scenario used a “random” setting – based on the exploratory analysis of the data, as for example, to have an idea of the contamination of the dataset. Although, this way of determining contamination is not accurate, as the data do not have any label.

The second scenario used the previous result’s inlier/outlier to feed a parameter tuning algorithm, and then trained again the model and checked if there was any improvement in performance.

The hyper-parameter tuning used was GridSearchCV, also provided in scikit-learn, under sklearn.model_selection.GridSearchCV. It does a comprehensive search for an estimator using the given parameter values. The parameters of the estimator used to apply these methods are optimized by cross-validated grid-search over a parameter grid [19]. In order to use GridSearchCV, it is necessary to split the data into trained and test data. Then using sklearn preprocessing StandardScaler [20] to standardize features.

To check the validity of the trained model with new data, a twin-sample dataset was used. This dataset was from a different device/website, and it has 71023 entries. It holds the same characteristics as the original dataset, but it was sliced to have a different timeline.

The third scenario adds a new feature for the data: mean response time from the previous 24h. Then, together with the other features, it was used to fit the best model from the previous scenarios.

7. Implementation

This chapter will discuss the implementation of all parts of the project: Firstly, how the data was fetched, then the steps to clean the data and prepare for Machine Learning and what are the experiment scenarios for the model.

7.1. Fetch the Data

The first step of the project implementation was to fetch the data for further processing. The data was stored in Elasticsearch as mentioned in the section 6.1.

Deciding what data could be useful for Machine Learning was a very challenging step, as there are many types of data being stored from the monitoring tool, covering different areas of the datacenters. This step of fetching and analyzing data was done several times. As already described in detail in the section 6.1., at the end it was decided to use data from Pingdom checks.

Another challenge was computational power, since at SAP we are dealing with massive amounts of data (millions of logs). For this reason, the conventional querying methods are discouraged to be used and a solution was to perform asynchronous processing.

The libraries used in the fetching process are:

```
import pandas
import asyncio
from elasticsearch import AsyncElasticsearch
from elasticsearch.helpers import async_scan
from dottedDict import DottedDict
```

The first import includes the Pandas library, which according to their home page "is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool" [21]. It offers various functions from general evaluation to more complex data manipulation technics. Pandas was used in other parts of the implementation as well.

Asyncio was necessary to ensure that the code can work on several distinct parts simultaneously. With that, it is possible to use event loops, which is a key component when we want to speed up things.

Elastic Search is a distributed analytic engine, created for massive amounts of data. It uses a JSON format-based storage mechanism, with inverted indexes. This is

especially useful for fast querying. AsyncElasticsearch lets us define a client – in our case with a URL -, and then the aync_scan will provide an API like interface for the data.

The last component is dottedDict, that can be useful for dictionaries with deep path, with lists in lists and dictionaries.

It is important to mention that the source-code for this section itself will not be available, as it deals with confidential information, for this reason it will only be described hereafter.

The first step taken was to establish an async client session with Elasticsearch. Inside Elasticsearch, data is stored in indexes. Each document is a collection of fields, which are the key-value pairs that comprise the data, and an index may be thought of as an optimized collection of documents [22]. Based on that, the name of the index where the data is stored in was defined and a dictionary field is declared to store the answer from async_scan.

Async_scan requires a client, a query and an index. The query was used to select, roughly, the fields that could be meaningful to analyze, this way it was possible to reduce the fields from more than 70 to only 12. For this, it was entered a list of field names stored on _source field, which contains the original JSON document body that was passed at index time and returned when executing fetch requests. Also, it was possible to define a range of time for the data returned.

The "async for" creates a parallel loop to iterate through every doc that comes back from the async_scan. Inside this loop, there is another "for" loop to go through _source data and append keys on fields dictionary. When all the data has been read, with the pandas library's DateFrame.from_dict() function, a DateFrame object is constructed from the fields dictionary. Indexes can be swapped back to be normal with the transpose function, and for a given key, the data can be serialized with the to_pickle() method call.

7.2. Exploratory Data Analysis

Exploratory Data Analysis, as the name implies, uses a variety of techniques to gain perception and draw conclusions about the data in hands.

7.2.1. Variable Identification

Each variable is determined by identifying its type.

```
dataset.dtypes
```

```
V1              object
V3              object
V2              object
V4              object
V5              object
severity        object
response_time   object
V6              object
timestamp       object
event_status    object
V7              object
V8              object
```

Figure 7.1: Types of data after import.

As the data was saved in .pkl format, after import all the types are object, which in Pandas dtype means string or mixed types. It is important to have the correct dtype to avoid unexpected results and the program know how to store and manipulate data, so converting the data to its correct type was one of the steps during data preparation.

7.2.2. Univariate Analysis

For continuous variable, some graphical techniques used for analysis are histogram, KDE and boxplot.

A box plot (also known as a box-and-whisker plot) illustrates the distribution of quantitative data. Except for points that are determined to be "outliers" using a method that is a function of the inter-quartile range, the box shows the dataset's quartiles, while the whiskers extend to represent the rest of the distribution. [23].

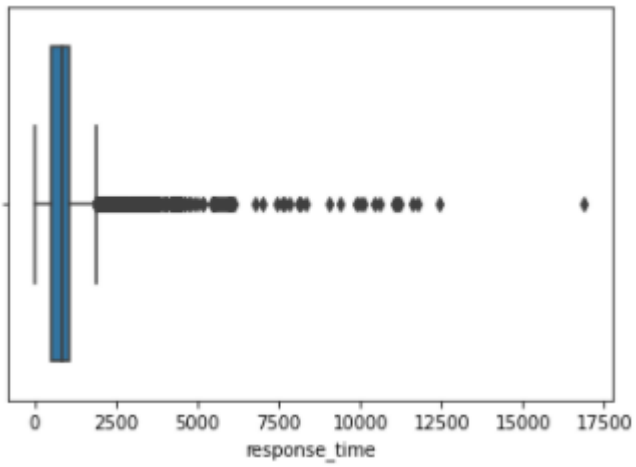

Figure 7.2: Boxplot to show distribution of response_time.

The feature response_time was used as input of the plot in Figure 7.2. The straight lines (also called as whiskers) represent the maximum (around 1900) and minimum (0) response_time. The outliers can be deduced from the points outside of whiskers. The plot also gives a representation of 25th, 50th, 75th percentiles. The value at which 25% of the data values are below this number is known as the 25th percentile, the same applies for the 50th and 75th percentiles. The median value is represented as a line drawn inside the box (around 900). It's also possible to see the Inter-quartile range (IQR), the distance between the upper (75th percentile) and lower (25th percentile) lines of the box (from around 500 to 1100) containing the most data information.

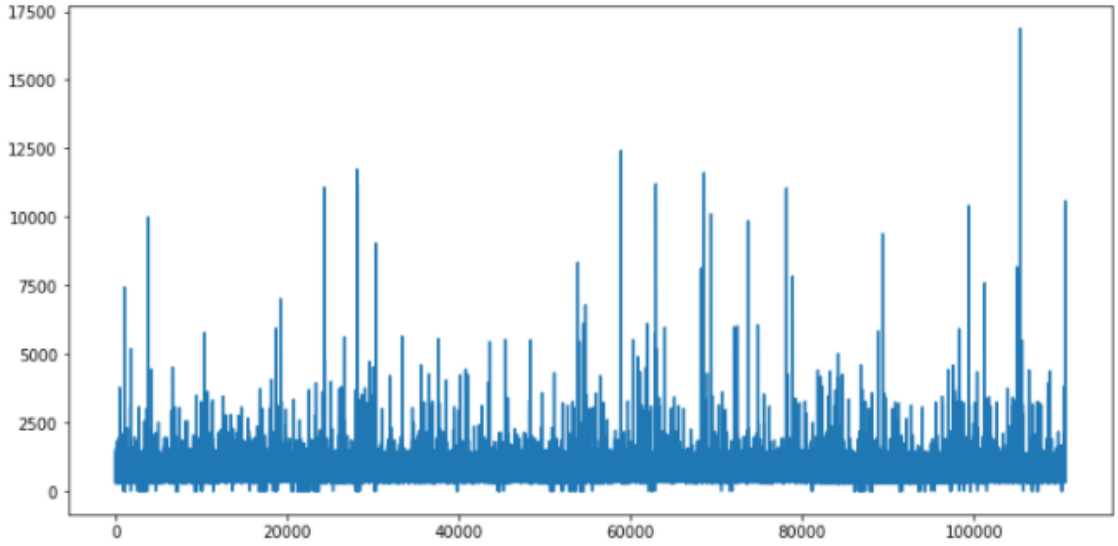

Figure 7.3: Simple plot for response_time.

The plot in Figure 7.3 uses a function plot() from Pandas data frame. This type of plot was helpful to visualize if there is any break in response_time during a timeline. It was used in other parts of the development as well.

7.2.3. Bivariate Analysis

It is possible to investigate the association between any two variables (which can be categorical-continuous, categorical-categorical, or continuous-continuous) by using Bivariate Analysis.

Heatmaps shows the correlation between attributes. The intensity of colors and range of grades gives information of how much these attributes are related: 0 means no correlation, 1 means the data moves in the same direction (i.e., if A increases, B also increases) and -1 means the data moves in opposite directions. The diagonal is 1 because,

of course, a feature is totally correlated with itself, and the map is mirrored from this diagonal.

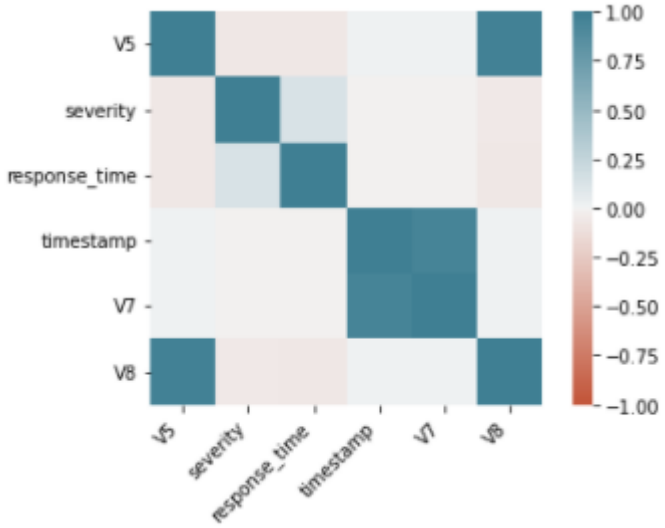

Figure 7.4: Heatmap of correlation between features.

From Figure 7.4 and other maps created after data cleaning, there were not many correlations between the features, this helped to decide which features to drop that possibly would not make difference for the model performance.

7.3. Data Preparation

The preparation of data for training the machine learning model is a fundamental step, as the quality of the results heavily depends on the quality of the input values.

7.3.1. Data Cleaning

After insights about the data were drawn and had checked the type of problems the dataset was presenting, it was time to clean the data and make it ready for machine learning algorithms. It is a time-consuming but crucial step to this work because of the above-mentioned reasons. Problems with the data includes too much data, non-representative data, duplicate data, and missing values.

- **Too Much Data Problem**

One of the problems that can arise when dealing with data is called the curse of dimensionality, meaning that we have a high-dimensional feature space - or to put it more simply, too many columns in the dataset. Because each feature can hold a range of possible values, it would be necessary a massive quantity of training data to ensure several samples with every potential value combination [24].

The solution to this problem is to select only the features that are the most relevant to the situation. Only if the training data contains enough relevant features and not too many irrelevant ones will the system be able to learn.

The first dataset used to work on had more than 80 columns stored in Elasticsearch, then in the fetch process 12 columns were selected to begin with, and from these the 9 most relevant ones were selected using pandas.DataFrame.info() to get insight, and then the others were removed with the pandas.DataFrame.drop().

- **Missing Values**

Another typical problem that comes up while dealing with data is having rows with missing values (features). If that is the case, we can either choose to delete these rows or fill in the missing values (imputation).

#	Column	Non-Null Count
0	V1	110681 non-null
1	V3	110681 non-null
2	V2	110681 non-null
3	V4	110671 non-null
4	V5	110681 non-null
5	severity	110681 non-null
6	response_time	110681 non-null
7	V6	110681 non-null
8	timestamp	72243 non-null
9	event_status	110681 non-null
10	V7	72243 non-null
11	V8	110681 non-null

Figure 7.5: Details about missing values in the dataset.

Since most of the rows - 72243 out of 110681 entries - did not miss any information, it was decided to abandon the ones that had holes in them by using:

newDF = df.dropna(subset=[‘timestamp’])

Another type of “missing values” faced in the dataset was that for a certain period of time the dataset does not have logs. This type of missing information could only be identified through plotting, as we can check in Figure 7.6:

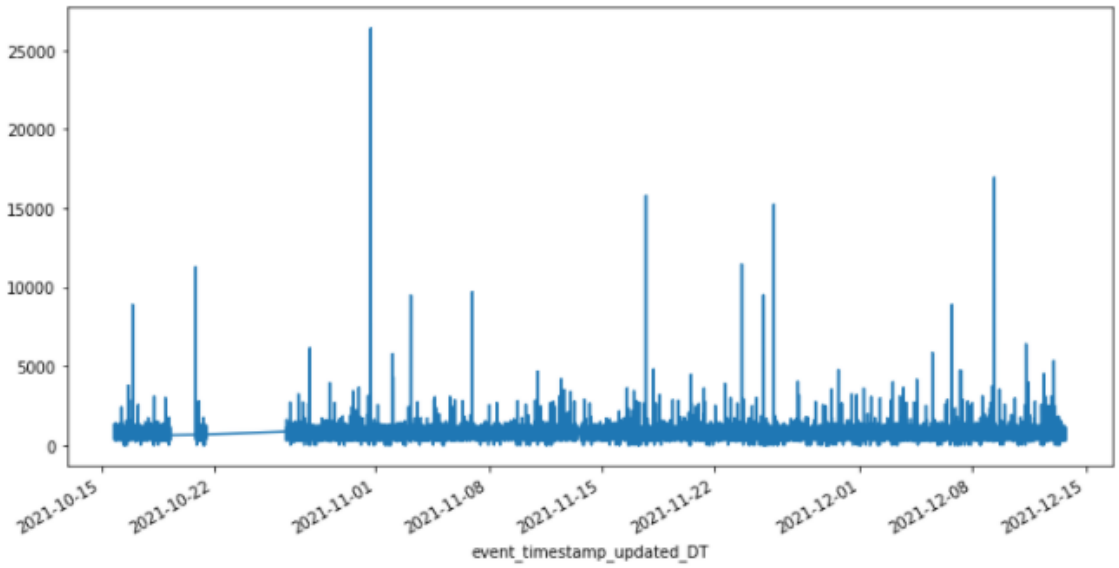

Figure 7.6: Original timeline of the dataset.

In the figure 7.6 we can see that during 22-10-2021 until around 26-10-2021 there is no information for time response. Based on this, it was decided to slice the data frame to have entries only from after 26.10.2021 to avoid misinterpretation from the anomaly detection system.

- **Duplicated data**

Removing duplicated data is another basic data cleaning routine since duplicates are not only unnecessary but they also distort the model. This applies to both duplicate features and rows. To find duplicate features the columns with very similar names were compared to check if they were identical or not, as it is possible to see in the example below:

df['event.status'].equals(df['event_status’])

If it turned out that the two columns would contain the exact same data, one of them was removed from the dataset.

- **Outliers**

As mentioned in the section 3.2, outliers are data points which differs significantly from the same dataset’s additional data points. Usually, it is recommended to remove outliers from the dataset, but in the case of this thesis outliers, or anomalies, is exactly what we are trying to predict by using Isolation Forest.

7.3.2. Data Transformation

The other step before being able to use the data in the model is transforming it, if necessary, to the correct data type or format. It is important to have the correct data type to avoid unexpected results and for the program to know how to store and manipulate the data.

As explained in the section 7.2.1, pandas.DataFrame.dtypes() was used to check the types of the columns, which initially were all object dtype. It was also used pandas.DataFrame.describe() to get insight about the types of transformations that could be done on them.

One of the transformations applied was to convert the epoch values to datetime, for this, firstly, it was necessary to convert from object dtype to integer and then, a function was implemented and applied for every row of the specific feature:

```
"""CONVERT COLUMNS FROM OBJECT DTYPE (IN REALITY, EPOCH) TO INT64 """
df[['timestamp', 'V2']] = df[['timestamp', 'V2']].apply(pd.to_numer
ic, axis = 1)
```

```
"""CREATE A FUNCTION TO APPLY TO EACH ROW OF THE DATA FRAME"""

def epochToDatetime(value):
      """CONVERTS EPOCH VALUE TO DATETIME"""
      return dt.fromtimestamp(value)

"""APPLY THAT FUNCTION TO EVERY ROW OF THE COLUMN"""""
df["timestamp_DT"] = df["timestamp"].apply(epochToDatetime)
```

7.3.3. Data Encoding

Label encoding is a process that aims to convert different labels into numeric form, making it more readable for the model. In the data there are four types of status, which first were converted to category and then encoded:

```
"""CONVERT THESE COLUMNS FROM OBJECT DTYPE (IN REALITY, STRING) TO
CATEGORY"""
df['event_status'] = df['event_status'].astype('category')

""" LABEL ENCODING - ASSIGN THE ENCODED VARIABLES TO A NEW COLUMN"""
df["event_status_Categ"] = df["event_status"].cat.codes
```

The resulting categories are:

0 – down
1 – unconfirmed_down
2 – unknown
3 – up

Another type of label that had to be dealt with, in a different way, was the event severity. The original labels were from high to low in this order: 1, 3, 4, 5.

Taking into consideration the problem explained in the section 3.3.3, that using numbers can introduce relation/comparison between them - and in this situation that is what was intended – it was the case to change the order of the labels, from low to high: 1, 2, 3, 4. This way the number 4 could get a higher weight and thus not be a problem.

```
"""CHANGE event_severity PATTERN”””

unpiclked_df["event_severity_1to4"]=unpiclked_df["event_severity"].replace({1: 4, 3:
3,4: 2,5:1})
```

7.4. Model

When starting a project and trying to have an understanding about the problem, establishing a measure of baseline performance helps us to get some information on how to tackle the issue and define what is the best approach in the next steps.

A baseline is the outcome of a solution or model that is both easy to set up and has a good possibility of producing acceptable results. It is advised to first start with a baseline and then make more complex solutions to get better results.

Two approaches for Anomaly Detection were used. In each of them the baseline was calculated first, and then the Isolation Forest was trained to provide better results. In the first approach, the model was trained using the original features of the data, and the training was repeated using different combination of parameters. In the second approach it was defined a new feature to check how it would impact the prediction capability of the

model.

7.4.1. Training the Model – First Approach

First of all, to find a measure of baseline performance boxplot was used to gather information about the dataset. As described in the section 7.2.2, the boxplot is from seaborn.

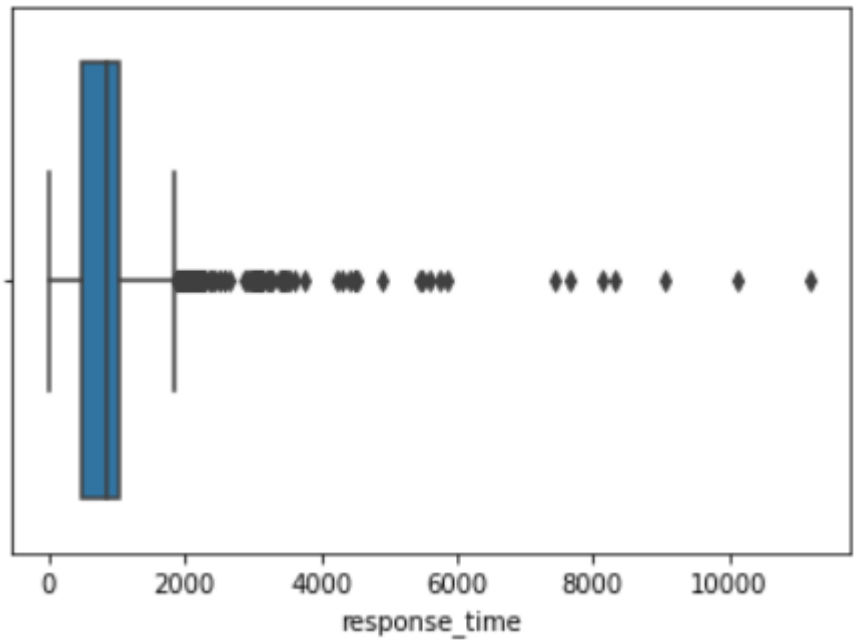

Figure 7.7: Boxplot to inference outliers from response_time.

In the plot shown above (Figure7.7), the points that are found outside the lines (whiskers) are the outliers, while the blue box represents the dataset's quartiles and the extension of whiskers shows the remainder of the distribution [23]. From the plot it is possible to identify that the outliers, defined by the inter-quartile function, starts from around 2000ms.

Based on this information, a function was implemented to flag as anomaly the data points that have response_time above 2000ms. The method receives response_time as parameter and it returns a True/False based on the condition.

To apply a function along an axis, in this case the column response_time, Pandas have a function apply() and the result of calculation is a new column with Boolean values to flat outliers. The result of the function flagged 27493 normal points and 104 outliers and the source-code for the implementations is shown below.

```
"""FUNCTION TO APPLY TO EACH ROW AND CALCULATE IF OUTLIER OR NOT"""

def checkOutlier(responseTime):
   outlier = False
   if responseTime > 2000:
      outlier = True
   return outlier

df1["baseline_outlier"] = df1["response_time"].apply(checkOutlier)
```

The second step taken was to build and train a simple IsolationForest model. Usually, it is extremally recommended to split the data between train (practice), test and validation to prevent the model to overfit and accurately evaluate the model. But as the dataset used was not labeled, there was no means to evaluate the model in the usually recommended way (comparing the actual class with the predicted class). Also, the size of the dataset was small, for this reason it would not be viable to split the dataset.

To build the model, initially it was created a clf1 (which stands for classifier 1) variable and the IsolationForest class was instantiated. The values for all parameters were passed to the IsolationForest method. Note that not all parameters are obligatory. As the parameters were already described on the section 6.2, here I show only the values used for the parameters:

- n_estimators=100, the default value
- max_samples='auto'
- contamination=float(.03) – the algorithm is quite sensitive to this value. I decide to use 0,03 based on the result of the baseline approach, which detected 3,8% of outliers
- max_features=1.0, the default value
- bootstrap=False, the default value
- n_jobs=-1, means using all processors
- random_state=42, the value a set for my seed
- verbose=0, the default value

After building the model, the fit() method was used to train the model using the dataset. To find the values for the anomaly column, the method predict() was used, where it returns -1 for outliers and 1 for inliers. The result was then inserted into the column ‘anomaly’.

1st Experiment	
Features:	response_time, severity, status
Split ratio:	-
Anomalies:	3.07%

Table 7.1: Summary of the 1st experiment.

For the second experiment, in order to be able to use GridsearchCV to tune the model parameters afterwards, the dataset was split into input variables (X), which are: response_time, severity and status, and into an output variable (y) which is the anomalous data column, resulting from the 1st experiment.

Then, the method train_test_split() from sklearn.model_selection was used to, as the name suggests, divide the data into random train and test subsets. The method receives the input variables (X), the output variable (y), the quantity of the dataset to incorporate in the test split (0.2), and the random_state that rules the shuffling applied to the data before splitting - setting it to an integer allows reproducible output across multiple functions calls. The method returns a list containing train-test split of inputs (X_train, X_test, y_train, y_test).

The next step taken was to define an instance of StandardScaler with default hyperparameters. Once defined, the fit_transform() function was called and the X_train dataset was passed into it to create a transformed version of the dataset and then use transform() for the test set.

What StandardScaler does is to standardize the dataset as the input variables have different scales: response_time in milliseconds, severity in scale 1 to 4, status in scale 0 to 3. Differences in the scales across input variables may lead to poor performance during learning.

```
"""SPLIT TRAIN AND TEST DATA"""

X = df1 (['response_time', 'severity', 'status')
y = df1['anomaly']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Now, with X_train and X_test standardized it was possible to use them into the GridSearchCv algorithm. The GridSearchCv is an algorithm under sklearn.model_selection that does extensive search across the supplied parameter values for an estimator, i.e. it tests a variety of distinct hyperparameters to find the optimal

settings for a model – in the case of this project, an Isolation Forest.

The model was refitted to the training data with a fresh set of parameters on each iteration of the grid search, and the mean squared error was recorded. The best set of model parameters was returned once all of the permutations have been tried [25].

First, a variable model was created and instantiated an IsolationForest. Then, the set of parameters were defined with its respective ranges to be used in the search.

A variable grid_search was created, and the model instantiated. The parameters it receives are:

a) the model defined earlier (IsolationForest),
b) the set of parameters and its values,
c) the scoring - which is the strategy to evaluate the performance of the cross-validated model on the test set, in this case a neg_mean_squared_error (the negated value of the metric),
d) the refit value set to True to refit estimator using the best-found parameters on the whole dataset,
e) the cv value to determine the cross-validation splitting strategy and
f) the return_train_score set to True, so computing training scores is used to get insights on how different parameter settings impact the overfitting/underfitting trade-off. They can be accessed via cv_results_ attribute [19].

Finally, the X_train, y_train values were given to method fit() to run fit with all sets of parameters.

```
"""USE GRIDSEARCHCV TO TUNE DE MODEL BY FINDING THE BEST COMBINATION
FOR THE PARAMETERS"""

model = IsolationForest(random_state=47)

param_grid = {'n_estimators': [100,1000, 1500],
'max_samples': [10],
'contamination': ['auto', 0.003, 0.04],
'max_features': [0, 3],
'bootstrap': [True,False],
'n_jobs': [-1]}

grid_search = model_selection.GridSearchCV(model,
param_grid,
scoring="neg_mean_squared_error",
refit=True,
cv=10,
return_train_score=True)

grid_search.fit(X_train, y_train)

best_model = grid_search.fit(X_train, y_train)
```

The result for optimum parameters: {'bootstrap': True, 'contamination': 0.04, 'max_features': 3, 'max_samples': 10, 'n_estimators': 100, 'n_jobs': -1}.

The parameters of Isolation Forest were set to these results and then trained and fit the model again. The result was 1046 detected outliers.

7.4.2. Training the Model – Second Approach

A second approach taken was to inference a new feature that could be meaningful for the prediction problem, differentiating from the first approach that had only original features. A function was created to compute the mean response_time from the last 24h for each entry.

```
"""FUNCTION TO APPLY TO EACH ROW AND CALCULATE MEAN OF
LAST_RESPONSE_TIME FROM LAST 24H"""
def getMean(currentTime,responseTime):
 """GET ALL RECORDS FROM LAST 24H """
   last24h = dataForPlot.loc[(dataForPlot['event_timestamp_updated_DT'] >=
(currentTime - timedelta(hours=24))) & (dataForPlot['event_timestamp_updated_DT']
<= currentTime)]
   return last24h['last_response_time'].mean()

"""SELECT THE COLUMNS THAT ARE INVOLVED IN THE CALCULATION AS A SUBSET
OF THE ORIGINAL DATA FRAME, AND USE THE APPLY FUNCTION TO IT """

dataForPlot["mean_responseTime24h"] =
dataForPlot[["event_timestamp_updated_DT", "last_response_time"]].apply(lambda x :
getMean(*x), axis=1)
```

2nd Experiment	
Features:	response_time, mean_resp_24h
Split ratio:	-
Anomalies:	3.07%

Table 7.2: Summary of the 2nd experiment.

The parameters used for the model were:

clf2=IsolationForest(n_estimators=100, max_samples=10, contamination=float(.05), max_features=2, bootstrap=True, n_jobs=-1, random_state=42, verbose=0).

Also, again the boxplot was used to visualize and measure a baseline performance:

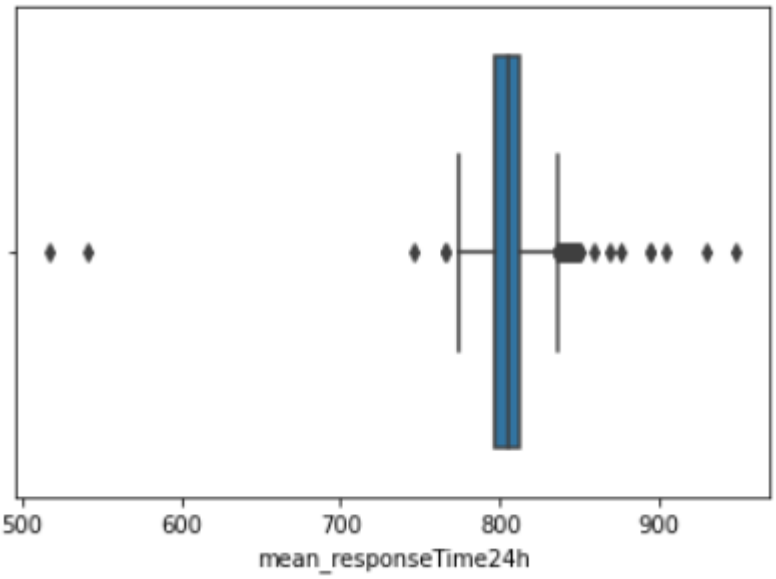

Figure 7.8: Boxplot to inference outliers from mean response_time.

From the plot shown above (Figure7.8) it is possible to identify that the outliers, defined by the inter-quartile function, are the points which the response time are below 775ms and above around 840ms, based on the mean response of the last 24h.

8. Evaluation and Discussion

Evaluation is carried out for each model experiment. As in this scenario there is no ground truth, i.e., ideal expected result- because of lacking a set of examples with output labels- the results were evaluated using plots to have an inference about the overall data points and the outliers.

8.1. Evaluation Scenario 1

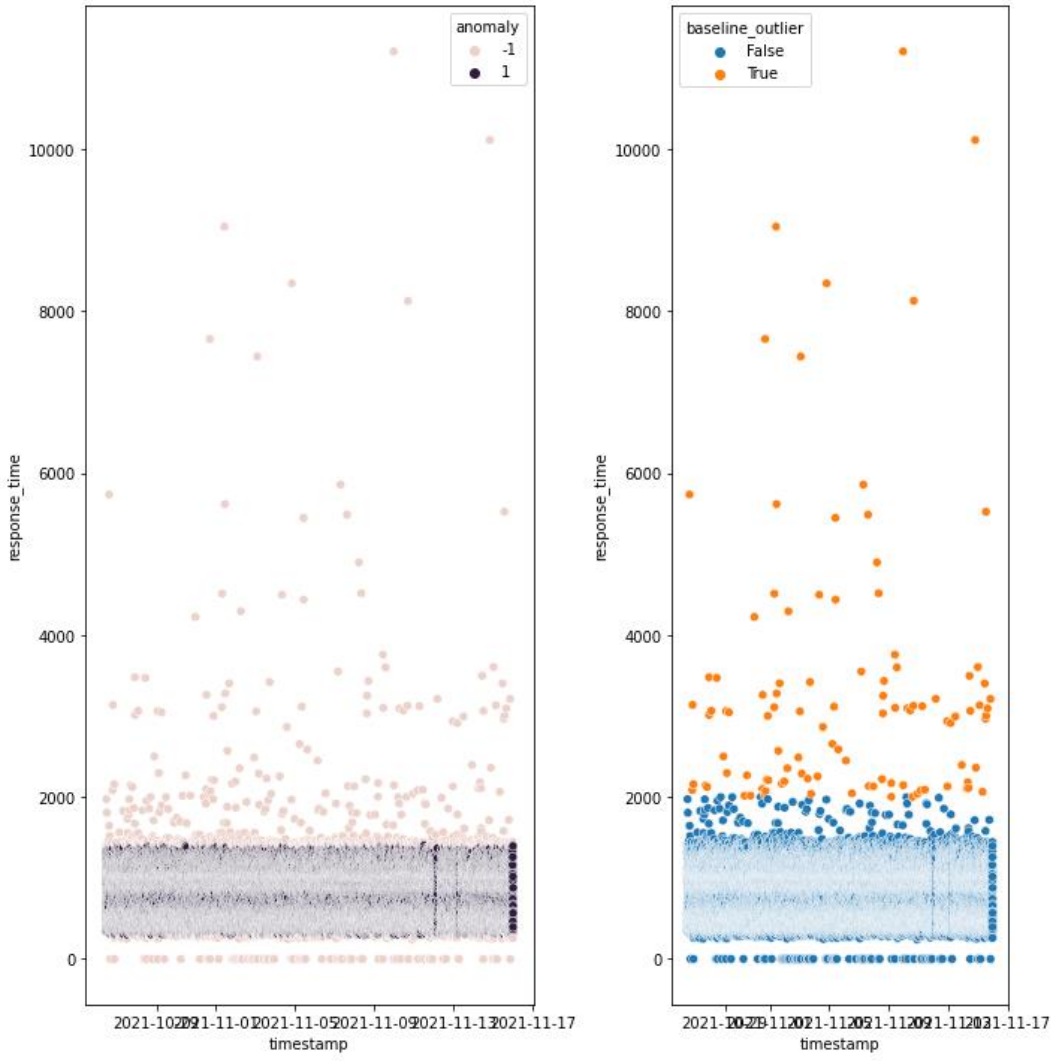

Figure 8.1: Plot comparing model 1 results (left) and baseline performance (right)

Comparing these plots, it is possible to see that the two results are very similar – differentiating only by the limit response_time to be considered anomalous. While Isolation Forest classify the data points (3,07%) in the range 0-300 and above 1400 as anomalous, the baseline – as configured from the boxplot – only classify data points with a response_time over 2000 as outliers. Also, it is possible to see that Isolation Forest was a bit more flexible when analyzing the events on the edges, while the other has a very clear limit.

8.2. Evaluation Scenario 2

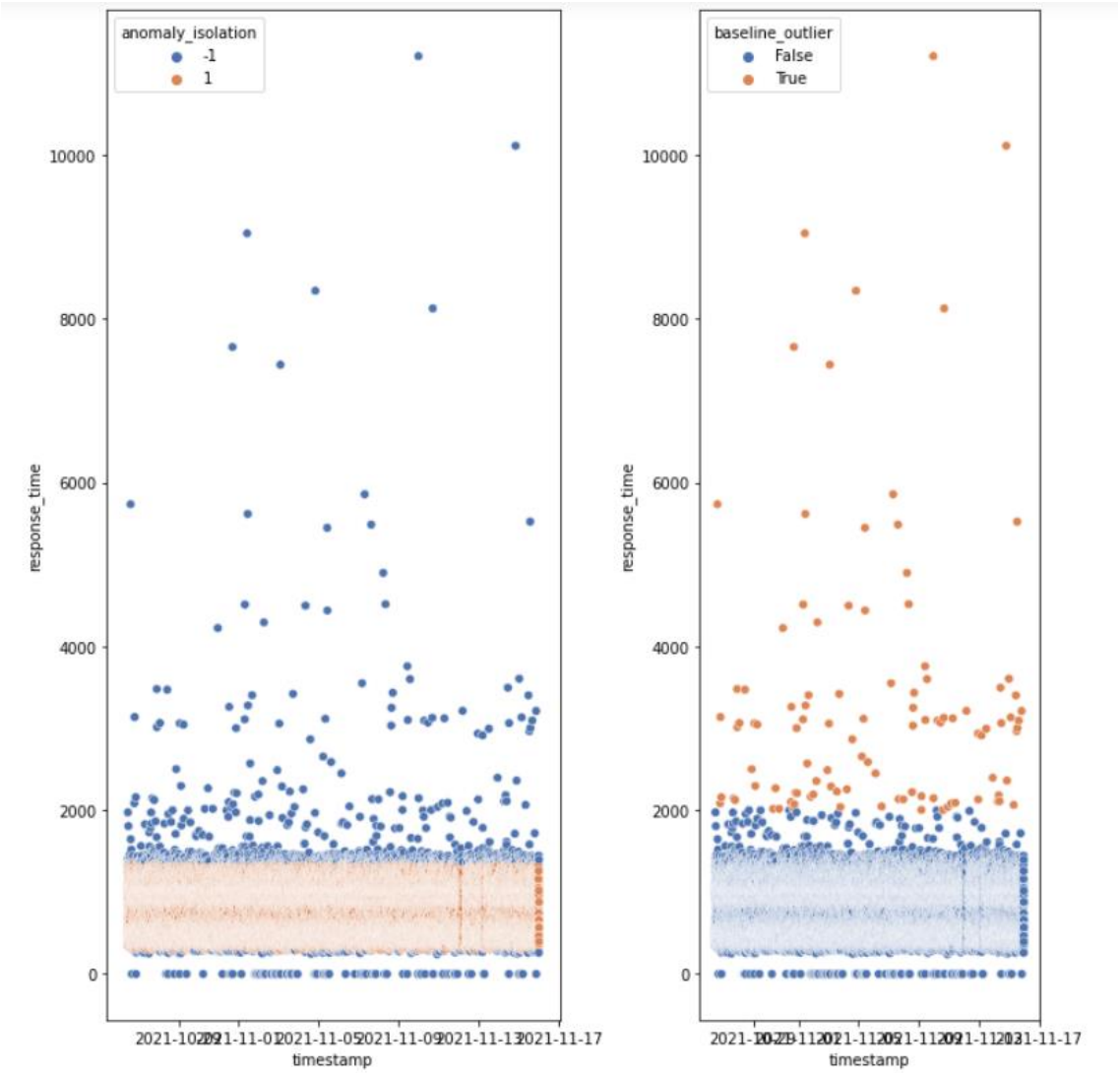

Figure 8.2: Plot from the second experiment: Isolation Forest (left) vs Baseline performance (right)

The results and the graph are very similar to the result of the first scenario, Isolation Forest tagged 3,94% of the data as anomalous. With this result it was possible to conclude that using the result of the previous training as label for training GridSearchCV did not show more sophisticated detection of outliers.

To test the performance with new data, and because there were no older or newer data from this URL, information from another URL was fetched, with the same features and similar behavior, but with a different timeline. The figure below (8.3) shows the result of a prediction for the new data, using the model trained before:

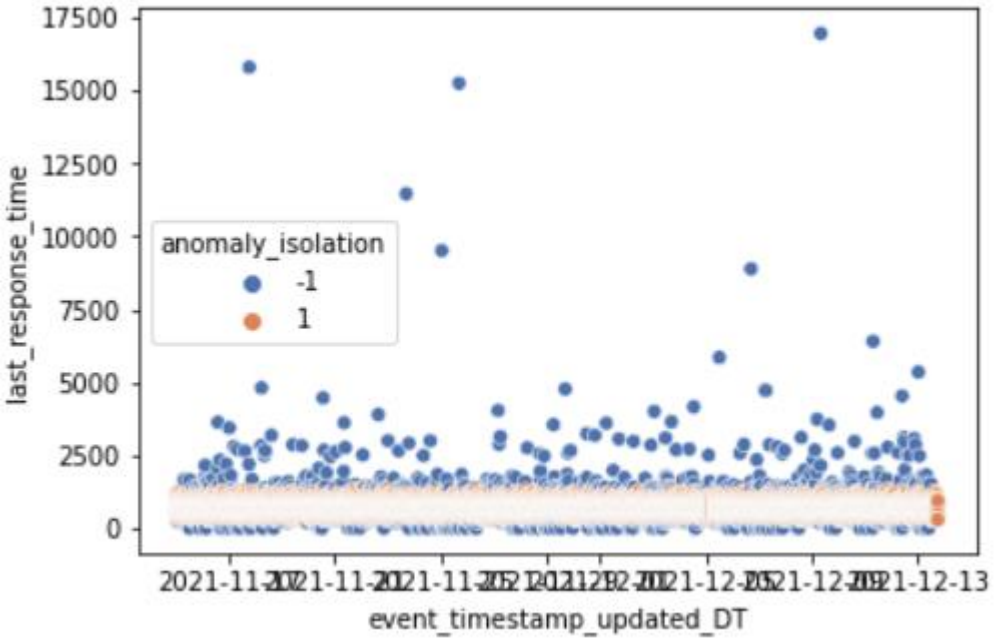

Figure 8.3: Plot utilizing the new twin-sample of data.

The model predicted 1,53% of anomalous data points. Seeing the graph, it is possible to notice that the way the prediction works is not so different from the baseline, therefore a new approach was tried, and it is described below.

8.3. Evaluation Scenario 3

Since in scenario 3 different features were used to train the model, the way the points were flagged anomalous, and consequently its resulting plot, was totally different from the previous approach:

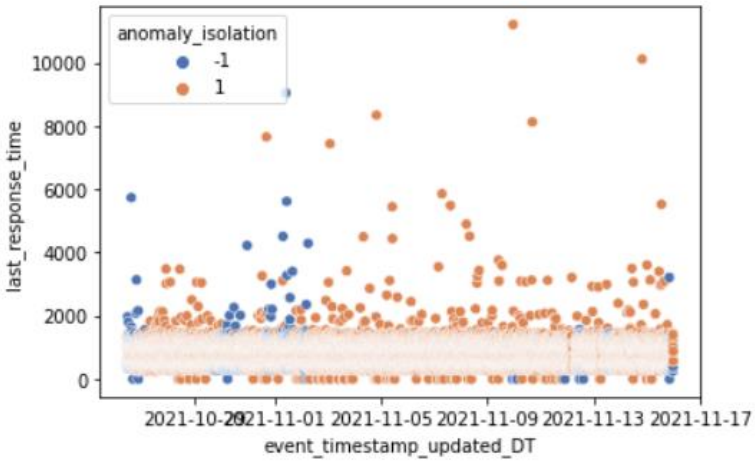

Figure 8.4: Results for mean response time approach. Response_time vs timestamp.

Utilizing the same types of parameters as in the previous approach, for plotting the results of the new approach, Figure 8.4, does not show a pattern in the results. In addition, the fact that the plot shows the whole data from 20 days makes it difficult to have a clue about how the data points were classified as anomaly. Because of these reasons, the data was sliced to be able to be seen in more details from a 4-day long period and compared mean_time_response with the timestamp in order to see the behavior during a specific period.

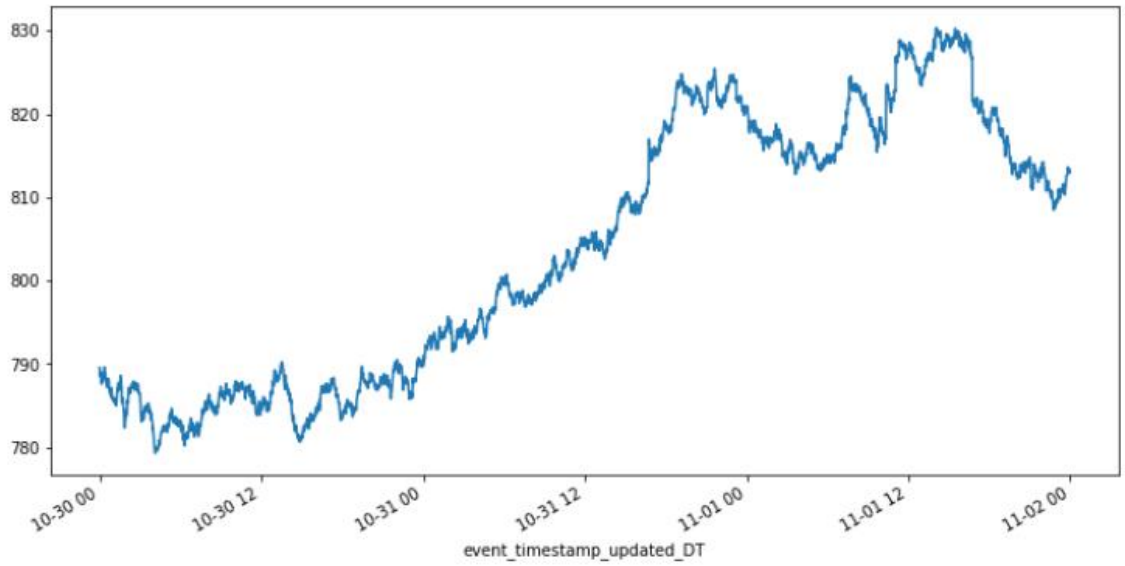

Figure 8.5: Plot utilizing mean response_time vs timestamp from 4-days period.

With this new graph on Figure 8.5 it is possible to see that in one of the time-periods detected as anomalous, the response_time has a raising pattern, which justifies the anomaly classification.

Based on the flagged data, the dataset was divided into normal and anomalous observations. To plot a graph with more sophisticated information, the following code was used:

```
      # PLOT THE LINE, THE SAMPLES, AND THE NEAREST VECTORS TO THE
PLANE

xx, yy = np.meshgrid(np.linspace(0, 20000, 50), np.linspace(0, 20000, 50))
Z = clf2.decision_function(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.title("IsolationForest")
plt.contourf(xx, yy, Z, cmap=plt.cm.Blues_r)
b1 = plt.scatter(normalValues.iloc[:,0], normalValues.iloc[:, 1], c="green", s=20,
edgecolor="k")
b2 = plt.scatter(Outliers.iloc[:,0], Outliers.iloc[:, 1], c="red", s=20, edgecolor="k")
plt.axis("tight")
plt.xlim((200, 2000))
plt.ylim((500, 1000))
plt.legend(
   [b1, b2],
   ["normal observations", "anomalies"],
   loc="upper left",)
```

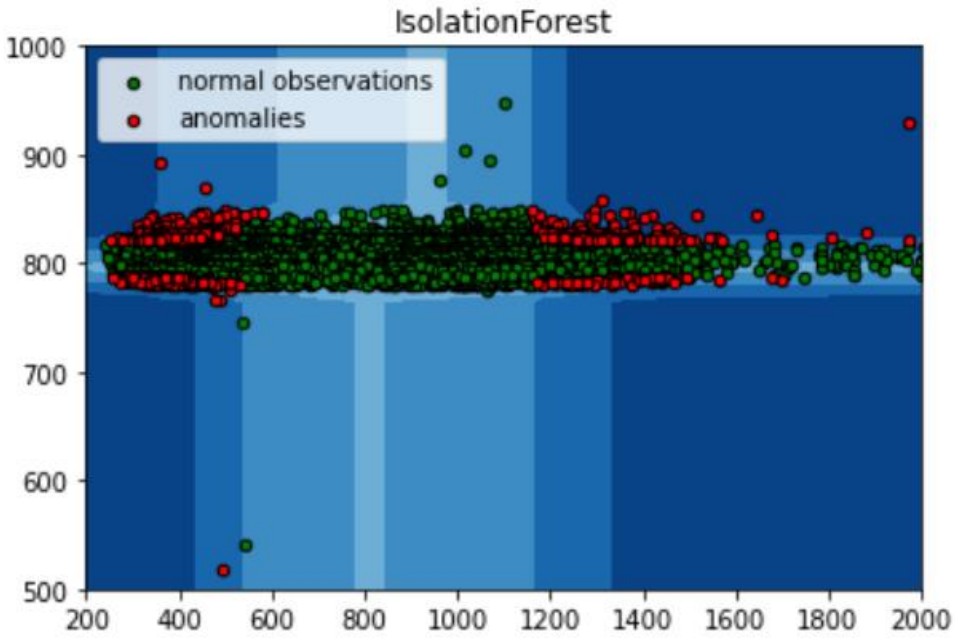

Figure 8.6: Distribution of anomalies – darker blue region means anomaly.

In the Figure 8.6 it is possible to see in a better way how the anomalous and normal observations interact with each other in the space. The data points that fall in the darker blue region are considered anomalies.

Based on these evaluations, it is possible to conclude that the data would need to have more enriched features in order to be able to have a better prediction. Only time_response by itself is not good descriptive enough of the system and its behavior to make predictions that are much better than a simple baseline rule. It is not possible to say with confidence if the anomaly detections are valid, or how much of it is valid. As the only way to evaluate the predictions was through graphic visualization, and within the graphs generated the normal and anomalous data points were not totally isolated from each other is space.

9. Further Development

For future work there are some possibilities that could be tried. Firstly, investing some time to label the data with the help of domain specialists. With an adequate amount of labeled data, it would be possible to use semi-supervised learning to increase the amount of data labeled.

Using labeled data, then it would be possible to check the proportion of outliers amongst the logs too and thus, be able to set the contamination parameter in the Isolation Forest in an accurate way. Also, by having labels it would be possible to fit hyper-parameter tuning algorithms to find out, in an accurate way, what are the best combination of parameters.

Many solutions nowadays, like the ones mentioned in the Literature Review, use normal behavior metrics to train the model and further to detect anomalies. In this way, it would be worth to try to gather metric's normal behavior to use it to train models.

Another possibility is that it would be interesting to combine univariate and multivariate anomaly detection, as approached by Anodot [26]. A second layer of machine learning takes over after an anomaly detection in individual metrics and aggregates anomalies from related metrics together. This gathering of correlated anomalies compacts the original flood of individual alerts into a smaller, more convenient number of underlying incidents. Multivariate anomaly detection learns a single model for all the metrics in the system, whereas univariate anomaly detection looks for anomalies in each individual metric.

Finally, after having quality data to train a model and validate its prediction through F1 scores and other metrics, it would be interesting to deploy the model for online learning too, this way it would keep learning with the new data coming from Pingdom.

10. User Guide

The full code of this project can be found in the GitHub repository (https://github.com/laryssaduarte/Thesis) created for it.

Due to confidentiality of other code dependencies from SAP, this project is not possible to run with the same results without employee access to those dependencies.

To build the project is necessary to clone the project repository

Git clone https://github.com/laryssaduarte/Thesis

Install:

- pip3
- python3
- pandas
- numpy
- Scikit Learn
- Matplotlib
- Seaborn

11. Conclusion

Machine Learning is a quickly growing field which gets a lot of attention nowadays. One of its applications is Anomaly Detection, which can be used to monitor systems to proactively detect and prevent failure cases.

To solve the proposed task is necessary to understand better the root causes of HTTP endpoint issues, that was the main challenge faced during the development of this thesis, as it requires a lot of domain knowledge and as we can only automate well understood processes.

To implement an accurate model for major incidents alert system, it would be necessary to define features with high prediction capability. Time response alone is not descriptive enough of the system and its behavior to make predictions that are much better than a simple baseline rule. The challenge lies on the size of system, there are many metrics to be evaluated and it would be necessary to collect issue-relate data.

It is not possible to say in a confident level if the anomaly detections performed in the dataset are valid, or how much of it is valid, since the only way to evaluate the predictions was through graphical visualization, and in the graphs generated the normal and anomalous data points are not totally isolated from each other is space.

References

[1] ‘Website Performance and Availability Monitoring | Pingdom’, *pingdom.com*. https://www.pingdom.com/ (accessed Dec. 15, 2021).

[2] A. Géron, ‘Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems’, p. 718.

[3] F. Chollet, *Deep learning with Python*. Shelter Island, New York: Manning Publications Co, 2018.

[4] J. Johnson, ‘Anomaly Detection with Machine Learning: An Introduction’, *BMC Blogs*. https://www.bmc.com/blogs/machine-learning-anomaly-detection/ (accessed Dec. 13, 2021).

[5] X. Hang and H. Dai, ‘Applying both positive and negative selection to supervised learning for anomaly detection’, in *Proceedings of the 2005 conference on Genetic and evolutionary computation - GECCO ’05*, Washington DC, USA, 2005, p. 345. doi: 10.1145/1068009.1068064.

[6] M. Goldstein and S. Uchida, ‘A Comparative Evaluation of Unsupervised Anomaly Detection Algorithms for Multivariate Data’, *PLoS ONE*, vol. 11, no. 4, p. e0152173, Apr. 2016, doi: 10.1371/journal.pone.0152173.

[7] E. Anello, ‘Anomaly Detection With Isolation Forest’, *Medium*, Nov. 02, 2021. https://betterprogramming.pub/anomaly-detection-with-isolation-forest-e41f1f55cc6 (accessed Dec. 13, 2021).

[8] F. T. Liu, K. M. Ting, and Z.-H. Zhou, ‘Isolation-Based Anomaly Detection’, *ACM Trans. Knowl. Discov. Data*, vol. 6, no. 1, pp. 1–39, Mar. 2012, doi: 10.1145/2133360.2133363.

[9] D. Yadav, ‘Categorical encoding using Label-Encoding and One-Hot-Encoder’, *Medium*, Dec. 09, 2019. https://towardsdatascience.com/categorical-encoding-using-label-encoding-and-one-hot-encoder-911ef77fb5bd (accessed Dec. 15, 2021).

[10] M. Du, F. Li, G. Zheng, and V. Srikumar, ‘DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning’, in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, New York, NY, USA, Oct. 2017, pp. 1285–1298. doi: 10.1145/3133956.3134015.

[11] R. N. Calheiros, K. Ramamohanarao, R. Buyya, C. Leckie, and S. Versteeg, ‘On the effectiveness of isolation-based anomaly detection in cloud data centers’, *Concurrency and Computation: Practice and Experience*, vol. 29, no. 18, p. e4169, 2017, doi: https://doi.org/10.1002/cpe.4169.

[12] ‘Predictive Intelligence’. https://www.servicenow.com/products/predictive-intelligence.html (accessed May 10, 2021).

[13] ‘Splunk IT Service Intelligence’. https://splunkbase.splunk.com/app/1841/ (accessed May 10, 2021).

[14] ‘Overview | Machine Learning in the Elastic Stack [7.12] | Elastic’. https://www.elastic.co/guide/en/machine-learning/current/ml-overview.html (accessed May 13, 2021).

[15] ‘Developing in the Windows Subsystem for Linux with Visual Studio Code’. https://code.visualstudio.com/docs/remote/wsl (accessed May 10, 2021).

[16] ‘Project Jupyter’. https://www.jupyter.org (accessed Dec. 15, 2021).

[17] R. Desai, ‘Top 10 Python Libraries for Data Science’, *Medium*, Dec. 20, 2019. https://towardsdatascience.com/top-10-python-libraries-for-data-science-

cd82294ec266 (accessed Dec. 15, 2021).
[18] ‘sklearn.ensemble.IsolationForest — scikit-learn 0.20.4 documentation’. https://scikit-learn.org/0.20/modules/generated/sklearn.ensemble.IsolationForest.html#sklearn.ensemble.IsolationForest (accessed Dec. 15, 2021).
[19] ‘sklearn.model_selection.GridSearchCV’, *scikit-learn*. https://scikit-learn/stable/modules/generated/sklearn.model_selection.GridSearchCV.html (accessed Dec. 15, 2021).
[20] ‘sklearn.preprocessing.StandardScaler’, *scikit-learn*. https://scikit-learn/stable/modules/generated/sklearn.preprocessing.StandardScaler.html (accessed Dec. 15, 2021).
[21] ‘pandas - Python Data Analysis Library’. https://pandas.pydata.org/ (accessed Dec. 11, 2021).
[22] ‘Data in: documents and indices | Elasticsearch Guide [7.16] | Elastic’. https://www.elastic.co/guide/en/elasticsearch/reference/current/documents-indices.html (accessed Dec. 11, 2021).
[23] ‘seaborn.boxplot — seaborn 0.11.2 documentation’. https://seaborn.pydata.org/generated/seaborn.boxplot.html (accessed Dec. 12, 2021).
[24] ‘Curse of dimensionality’, *Wikipedia*. Dec. 09, 2021. Accessed: Dec. 13, 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Curse_of_dimensionality&oldid=1059361687
[25] ‘How to use the Isolation Forest model for outlier detection’. https://practicaldatascience.co.uk/machine-learning/how-to-use-the-isolation-forest-model-for-outlier-detection (accessed Dec. 15, 2021).
[26] ‘Benefit from Multivariate and Univariate Anomaly Detection Techniques’, *Anodot*, Oct. 17, 2019. https://www.anodot.com/blog/anomaly-detection-techniques-focus-multivariate-univariate/ (accessed Dec. 15, 2021).