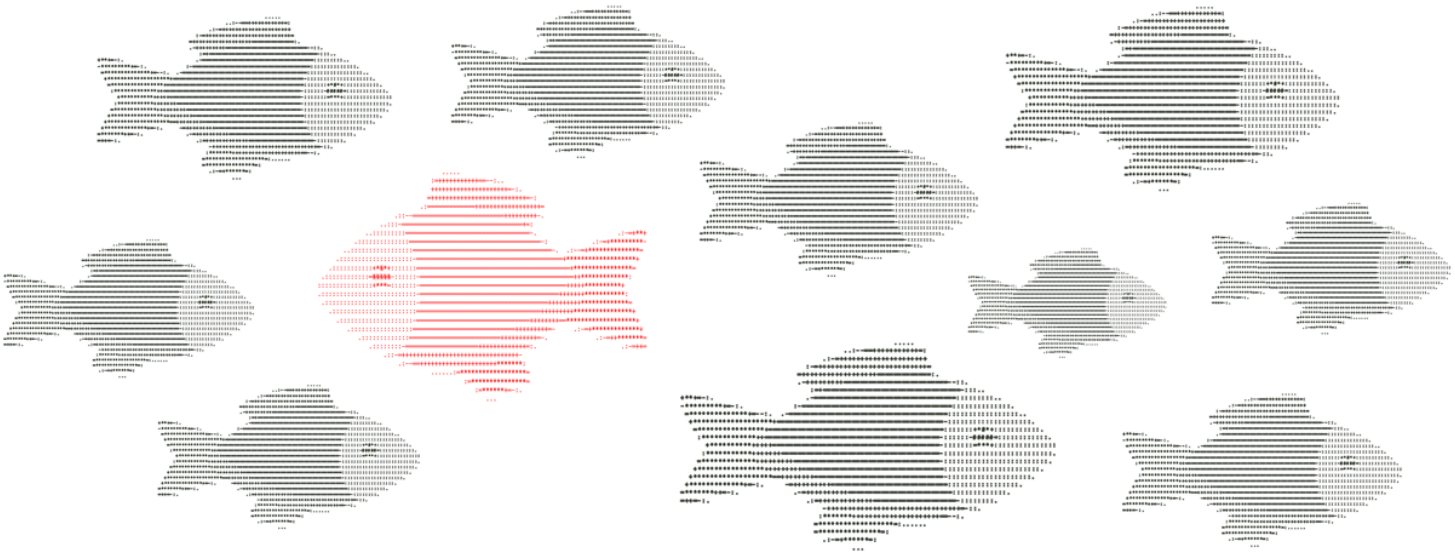


Uncovering Anomalies using Isolation Forest
–A Machine Learning Approach for Request Analysis

Degree Project in Computer Engineering

Viktoria Hagenbo
Lovisa Rosin

Department of Computer Science and Engineering

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2023
www.chalmers.se

Degree Project 2023

Uncovering Anomalies using Isolation Forest
A Machine Learning Approach for Request Analysis

Viktoria Hagenbo & Lovisa Rosin

CHALMERS

Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden 2023

Uncovering Anomalies using Isolation Forest
- A Machine Learning Approach for Request Analysis
Viktoria Hagenbo, Lovisa Rosin

Supervisor: Peter Moberg
Supervisor: Neethu Bal Mallya, Department of Computer Science and Engineering
Examiner: Lars Svensson, Department of Computer Science and Engineering

Degree Project 2023
Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: The cover has been designed using an image from Flaticon.com.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2023

Uncovering Anomalies using Isolation Forest
- A Machine Learning Approach for Request Analysis

Viktoria Hagenbo, Lovisa Rosin
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

In an increasingly digital era, the prevalence of misconduct increases as online social networks enable the creation of bots posing as normal users. This type of misconduct can appear in various forms, for example, emails containing unwanted advertisements, attempts of malware distribution, or simply collecting user-sensitive information. To detect this behaviour, using machine learning is well-considered and researched, especially regarding the analysis of the content of messages and online posts. This project explores the approach to analyze metadata from HTTP requests to find patterns for anomalous behavior, with the end goal being a machine learning module that can be integrated into a larger system for request analysis.

After reviewing different approaches suggested by previous research and theoretical reasoning, the proposed system has been designed and implemented using the Isolation Forest model. Feature engineering has been utilized to extract information from sequences of input requests. The system consists of two different model instances which operate on different sequence length intervals. The conclusion to use the selected models has been obtained when evaluating differently trained Isolation Forest instances using precision, recall, and the F1 score as metrics.

Keywords: Machine learning, Isolation Forest, Unsupervised learning, Request analysis, Anomaly detection, Feature engineering.

Acknowledgements

We would like to acknowledge and express our gratitude to those that have helped us during this project. In particular, our technical supervisor Peter Moberg who offered an interesting outline for us to expand upon, as well as being a source of guidance for the duration of this project. We would also like to express our warmest gratitude to our supervisor Neethu Bal Mallya at Chalmers for her invaluable feedback and advise when writing this report.

Viktoria Hagenbo, Lovisa Rosin, Gothenburg, June 2023

List of Acronyms

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

ML	Machine Learning
GRA	Gateway Request Analyzer
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
API	Application Program Interface
URL	Uniform Resource Locator
OSN	Online Social Networks
IF	Isolation Forest
RNN	Recurrent Neural Network
LSTM	Long-short-term-memory
NN	Neural Network
FFNN	Feedforward Neural Network
CNN	Convolutional Neural Network
BPTT	Backpropagation Through Time
BoW	Bag-of-words
TF-IDF	Term Frequency Inverse Document Frequency
TF	Term Frequency
IDF	Inverse Document Frequency
TP	True Positive
TN	True Negative
FN	False Negative
FP	False Positive

Contents

List of Figures

List of Tables

1
Introduction

1.1 Background

The prevalence of spam is a common issue in all kinds of online communication. It may include unsolicited messages/emails containing advertisements for miscellaneous products, attempts of malware distribution and identity theft [1], and fictitious reviews to sell goods or services [2]. Detecting and preventing spam or other anomalous behaviour using *machine learning (ML)* is a well-documented issue. The most common approach includes implementing a classifier that makes the distinction based on the content of the message/email or post. However, spam detection on social networks is a more complicated issue than classifying an email as spam or "ham". This is because messages on social media tend to be shorter and noisier than ordinary emails [3].

In November 2022, a *Gateway Request Analyzer (GRA)* [4] was created as part of the course DAT067 and in collaboration with an American company that offers solutions for live streaming and owns various applications. The purpose of the system was to analyze incoming requests in the company's gateway and detect spam or other unduly behavior by analysing metadata from HTTP requests. A basic rate limiter was put in place that could help limit the number of requests from a user or IP address. This implementation did not help with more sophisticated efforts from spammers, such as cases where spam can not be detected by solely examining numbers of request. This degree project builds upon the DAT067 course project with the end goal being an ML system that could be used as an additional tool in detecting anomalous behaviour on social media networks and other websites.

1.2 Objective

Within the scope of DAT067, a communication channel between a client and server has been implemented as well as an algorithm for rate limiting for simpler analysis purposes. This means that the system solely examines the volume of requests without making any further intelligent analyses, which does not offer protection against more sophisticated attacks or different kinds of misuse. To remedy this, an ML model can be a useful tool for additional analysis which will be the focus within the scope of this project to implement.

The aim is to create a system that can differentiate between normal and abnormal

user behavior when using an application or service. The general idea is to examine several requests made by a user and detect patterns with regard to normal versus abnormal behavior from the requests made. The result of the analysis should form a basis for an appropriate action towards the user.

1.3 Limitations

The data used for this project is not labeled; hence, the project will only include unsupervised ML models. The implemented system will conduct its analysis by examining API calls, i.e., it will not acknowledge requests done on other units outside the application. Therefore, detecting anomalous behaviour conducted by users in a more sophisticated way will not be taken into consideration while constructing the model.

Spam detection on social networks tends to be more complicated than email due to the short and noisy nature of social media posts or messages [3]. There are techniques to overcome this, but for this project the data used to train and test the ML model will consist of the metadata extracted from user requests. For example, user session, URL, and timestamps will be considered. As such the analysis will be made by examining user behaviour on a more general basis rather than the content of their posts or the messages they send.

For the initial scope of the project the ambition was to train and test the model on real-life data provided by the company. However, due to changes within the company, access to real data was no longer available. Consequently, the result of this project will be based solely on synthetic data.

During implementation, the possibility of comparing and exploring the performance of different models was raised due to finding new possible solutions. The idea was to select and incorporate the model demonstrating optimal performance on real-life data in the final implementation. However, when it became known that any real data would no longer be imminent, new challenges materialized that had to be addressed and prioritized. Significant time was dedicated to researching open-source data that could be incorporated into our existing module.

However, no available data sets were found containing the information required to match the implementation already in place; otherwise, extensive modifications would have to be made to ensure compatibility. In other words, utilizing another data set would entail an entirely new project, which would not align with the intended objective. This led to the decision that the initial synthetic data would have to suffice and that no comparison between models would be made. This decision was based on the recognition that relying solely on synthetic data would not provide a robust enough foundation to draw meaningful conclusions from such a comparison. This is a limitation that will resurface and be discussed throughout the report.

1.4 Social, Ethical and Ecological Aspects

As mentioned in Section 1.2, this project aims to create an ML system that can recognize normal and abnormal user behaviour. Today, many people use social networks as their main source of information about current world events, which can be considered beneficial from a transparency perspective. However, this also enables social media to be used as an effective conduit for misinformation for those wanting to manipulate public opinion in any shape or form, for which bots are a commonly used tool [5]. As such, differentiating between illegitimate and legitimate users would increase the ability to limit the number of bots operating on an online service, which not only is a benefit regarding the overall user experience but also can serve as a protection for the ordinary user.

If real-life data were to be used for this project, great care would be required regarding the responsible handling of the data provided. Regardless of content, it is of great importance that no personal information is leaked, and it is upon those with access to this information to prevent that from happening. In the case of this project, no real-life data is used, which means that this aspect will not need to be considered. Instead, we will need to consider the result yielded skeptically and be mindful of the conclusions drawn.

The project does not incur any ecological consequences. Thus, the ecological aspect will not be taken into consideration for the duration of this project.

1.5 Organization of the Report

The remainder of the report is structured as follows:

- Chapter 2 (*Method*) provides an overview of how the project will be carried out.
- Chapter 3 (*Theory*) provides a comprehensive overview of the relevant concepts, theories, and models, providing a platform for understanding our project and the subject.
- Chapter 4 (*System Design and Development*) discusses the process of establishing a system design and illustrates the overall system architecture.
- Chapter 5 (*Implementation*) describes the tools and frameworks used in the project and the details of integration with the GRA application.
- Chapter 6 (*Experimental Methodology*) discusses our experimental methodology in setting up the test environment and the metrics considered in the evaluation.
- Chapter 7 (*Evaluation*) presents the results with a discussion of our observations and interpretation of their implications.
- Chapter 8 (*Conclusion*) summarizes our conclusions along with our ideas for future work.

2
Method

The following sections outline the different phases of the project.

2.1 Research

There will be a strong emphasis on researching suitable solutions for the problem at hand. This phase includes searching for, reading, and summarizing findings on relevant literature and articles. All previous research is found by browsing different databases such as Scopus and IEEE Xplore. Since the task is to find a solution for finding abnormal patterns in user behaviour using unsupervised learning, the keywords used for finding reliable sources will reflect this and be dependent on the stage of the project.

2.2 Main phase

The second phase encompasses all the steps leading up to the main contributions of this project.

2.2.1 System Design and Development

This project will encompass considerations regarding which ML model to be used, as well as the overall architecture of the system. The data for training the ML model will be artificially generated and will require pre-processing before it can be analyzed by the model. These steps will include feature engineering and dividing the data into training and testing sets.

2.2.2 Implementation

In order to take advantage of existing libraries and tools regarding ML, the implementation will be written in Python 3.11. Depending on what models are chosen to examine more closely, libraries such as SciKit-Learn, TensorFlow, and Keras are expected to be used. The model will use an unsupervised ML method due to the unlabeled data.

2.2.3 Evaluation of Results

The performance of the system will be evaluated throughout the course of the project, as the implementation will be adjusted in accordance with the results yielded during the evaluation. Several scores based on the confusion matrix will be used in the evaluation.

2.3 Integration

The final step of the project will be to integrate the proposed ML module with the existing GRA system.

2.4 Documentation and Final Report

All the essential information related to the project will be documented throughout the different steps. A final report and a presentation will be made available towards the end of the project.

3
Theory

This chapter includes theoretical background to different types and approaches in ML, followed by a discussion on relevant models.

3.1 Spam Detection on Social Networks

There are a few different approaches for detecting spam on *online social networks (OSN)*, based on either content- or user-related features. For content-based features, the posts themselves are analyzed for obtaining information about the location, number of likes, the actual content, and more. For account-based features, information regarding the user is analyzed for detecting anomalies [6]. In addition, some studies suggest taking the relationship between users and overall social context into account when detecting spam messages and other anomalies [7] as well as performing a topic-based analysis on the content posted by a user [8].

3.2 Machine Learning Approaches

Regarding using ML to detect spam on OSNs, several approaches have been used depending on the nature of the data. This section explains the foundational differences between these approaches and how each of them has been utilized in previous related work.

3.2.1 Supervised Learning

Supervised learning includes all ML models that are trained using pre-labeled data and can be used for *classification* tasks as well as *regression* tasks. This means that each training instance x has a number of features describing its characteristics, as well as a label or a target value y. By examining the features, these models learn to find the patterns and characteristics for each training instance $\{(x_i, y_i), (x_j, y_j), ...\} \in (X, Y)$ such that it can accurately predict the output Y_{new} for new unseen input X_{new}. [9]

Supervised learning can be divided into classification tasks and regression tasks. Classification consists of categorizing data into a number of pre-set classes given the input. In contrast, regression aims to predict a numerical value as close to the target value as possible.

There are several studies regarding spam detection on social networks using a supervised approach. Al-Thelaya et. al. [10] suggest two methods for detecting spam on social media, using graph-based features and sequences of interactions between users. The graph-based features were divided into demographic, general, and graph features and served to measure the similarities between two users with any kind of relation. The analysis served to examine sequences of actions between users. The interaction between normal users expects a specific pattern, and thus a user with repeated abnormal interactions can be identified as a spammer. For this model, two deep-learning techniques were used to measure the performance.

The availability of pre-labeled data is a good asset when creating accurate ML models. However, such data is not always possible and/or very costly to obtain. In these cases, other techniques need to be utilized instead.

3.2.2 Unsupervised Learning and Clustering

Unsupervised learning consists of methods that aim to find structure and information about data without using pre-established labels [11]. The most commonly used unsupervised learning method is *Clustering*, which involves grouping similar data points into clusters. Clustering makes it possible to distinguish groups that share key features and label them depending on how compact and well-separated the different clusters are. This also enables the detection of outliers within the data set, i.e., data points that do not, or only to a small extent, share similarities with other data points. Detecting the outliers and excluding them from the data set can effectively and positively affect the accuracy of the system. The main issue with clustering is to properly assign clusters that are well-defined enough to provide meaningful information [12]. Thus, it is important to carefully define how many clusters are identified within the data so that the model does not become under- or overfitted.

3.2.3 Semi-supervised learning and other approaches

In some cases, the data set to be used by a model consist of a small subset of labeled data, while the rest is unlabeled. When this is the case, a semi-supervised approach can be used, in which conclusions can be drawn from the unlabeled data by comparing its properties with the labeled data. The advantage of a semi-supervised approach is that fewer resources will be required for annotating an unlabeled data set, while still obtaining a good accuracy [9].

It is also possible to combine an unsupervised and a supervised approach for detecting spam on OSNs. In an aim to create an unsupervised model to detect spam tweets, Washa et. al. [13] constructed a system utilizing two modules. The first module took advantage of unsupervised learning and created annotated data sets based on the result. These data sets were then used for training the other module which relied on supervised learning for detecting spam. This resulted in an average recall value of 91% and thus showed promise since the model would not require efforts and resources to manually annotate the data.

3.3 ML Models for Anomaly Detection

In this section, an overview of a few examined ML models geared toward anomaly detection will be presented. First, the tree-based ensemble model *Isolation Forest (IF)* will be covered. This is followed by *Recurrent Neural Network (RNN)* and a version of RNN called *Long short-term memory (LSTM)* model combined with the *Autoencoder*.

3.3.1 Isolation Forest

The IF [14] model initially developed in 2008 by Fei Tony Liu and Zhi-Hua Zhou, presents a fundamentally different anomaly detection approach with respect to the already existing ones. Liu and Zhi-Hua suggest the contrast in characteristics between normal and anomalous data patterns, and two major flaws in the already established anomaly detection methods (such as statistical, clustering-based, and classification-based methods). These methods are built on the profiles of normal instances and the separation of those data points that do not conform to these profiles, thus identifying them as anomalies. Due to this 'backwards design', the optimization is carried out on models profiling normal behaviour instances, resulting in the detection of anomalies not being as efficient and providing more false positives.

Liu and Zhi-Hua [14] instead propose the decision-tree-based model IF as an alternative solution, where an ensemble of *Isolation Trees* is designed to isolate the anomalies in contrary to profile normal instances. The method is based on the conception that anomaly data points are few and different, and these properties regarding feature values and quantity are taken advantage of for constructing the model. It is shown that every instance of anomaly will be isolated due to the tree structure design where anomalies are located and isolated closer to the root of the tree. Based on the *Decision Tree* algorithm, a random feature is selected amongst all features, and a random split value for that feature is selected.

For a normal data point x_n, where normal indicates instances that are similar to the rest of the data set, it generally requires more partitions to isolate it. This is due to the instances being more alike each other since the feature values are more similar. For an anomaly point x_a, the opposite is more accurate since they significantly differ from normal data points, which will result in the instance being isolated in fewer partitions. The recursive partitioning will result in a deeper tree and a longer path from the root node to the terminating node for normal instances, and a shorter path for anomaly ones. Figure 3.1 depicts an arbitrary isolation tree with conditions $c \in C$ and shows how the isolation of anomaly instance x_a is done in fewer partitions than normal instance x_n.

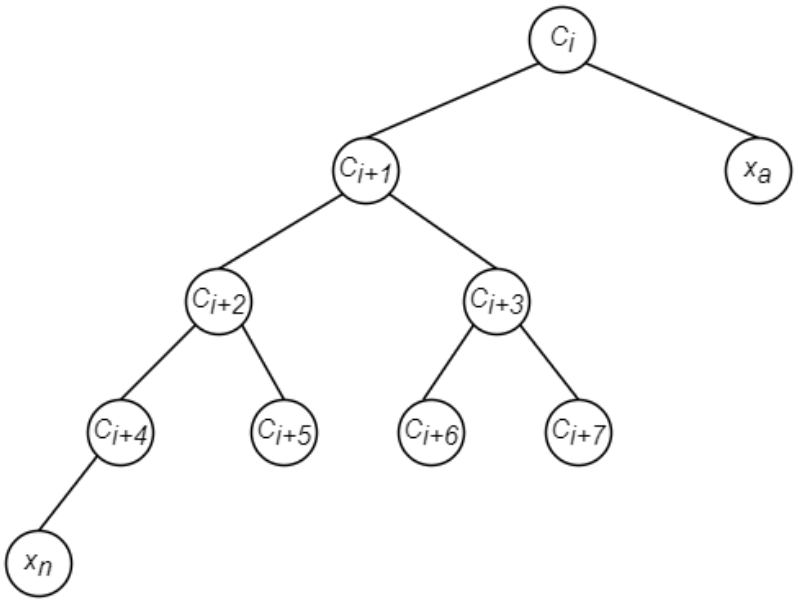


Figure 3.1: The traversal of an Isolation tree in an Isolation Forest. The tree detects anomaly instance x_a in less partitions than normal instance x_n based on conditions $c \in C$.

An anomaly score s of n instances is derived from path length $h(x)$ where x is the number of edges traversed from the root node to the terminating node as seen in the equation below.

$$s(x, n) = 2^z$$

where

$$z = -\frac{E(h(x))}{c(n)}$$

where $E(h(x))$ is the average of $h(x)$ from a collection of isolation trees and $c(n)$ is

$$c(n) = 2H(n-1) - \frac{2(n-1)}{n}$$

where $H(i)$ is the harmonic number and it can be estimated by $ln(i)+0.5772156649$.

The IF model is shown to work well on high dimensional data and improves the detection performance as well as providing a low processing time [14]. The model also performs well when trained on data sets where no anomalies are present.

An example of how the IF model can be utilized for anomaly detection was proposed by Ding and Fei [15]. They introduced an IF model using the frame of sliding windows and thereby taking into account a sequential drift. This solution demonstrated that the algorithm can show effective results for detecting anomalous instances on streaming data.

3.3.2 Recurrent Neural Network

An RNN is a *neural network (NN)* which uses time series data or sequential data, thus introducing the notion of time and temporal information into the model [16]. RNNs, alongside *feedforward neural networks (FFNN)* and *convolutional neural networks (CNN)*, utilize their training data in order to learn. However, unlike FFNNs and CNNs, RNNs keep an internal state which can represent information from arbitrary previous inputs, thus preserving a 'memory'. This previous information in the RNN can be used as input to modify the current input and output. In other words, the RNN works as an FFNN but also takes previous information into consideration in order to place things into a context.

While a traditional deep NN works under the assumption that the input and output are independent of each other, an RNN's output is influenced and dependent on the previous elements of the sequence [17]. Another distinguishing feature of an RNN is its usage of the *backpropagation through time (BPTT)* algorithm which slightly differs from the traditional backpropagation. The BPTT algorithm, as the traditional backpropagation, trains by calculating the errors between its output and input layer which are then used to adjust the model's parameters. The BPTT algorithm however sums the errors between each time step due to the fact that the RNN shares parameters across the layers of the network [17].

With the hidden current state h_t which saves historical information about the sequence at time step t, the RNN can be expressed as the following [16]:

$$h_t = f(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$

Here, the x_t represents the input and output at time step t. W_{hh}, W_{xh}, and W_{hy} represent the weight matrices between the layers at time step t. The hidden state h_t at x_t is computed by taking the sum of the previous hidden state h_{t-1}, the current input x_t, and a bias term b_h. The sum is then passed through an activation function f to result in an updated hidden state h_t. This activation function can be either the *ReLU*, *sigmoid*, or *tanh* function. The output y_t can be expressed as the following:

$$y_t = g(W_{hy}h_t + b_y)$$

The output y_t at time step t is computed by taking the hidden nodes' values h_t and the bias term b_y. This weighted sum is passed through another activation function g to result in the output. Here, the activation function can be either a *softmax*, *sigmoid*, or *linear* function [16].

Figure 3.2 depicts the internal structure of an RNN. The figure to the left of the equals sign is the representation of the RNN. Input x_t passes through layers A and outputs h_t. The information is then recurrently passed from one step of the network to the next. On the right side of the equation is an unrolled representation of the RNN, showcasing the passed-on information sequence through time. Current input x_i together with previous output h_{i-1} is passed through the network resulting in the current output h_t. h_t is then passed to input x_{i+1} and the sequence continues, thus passing along previous information.

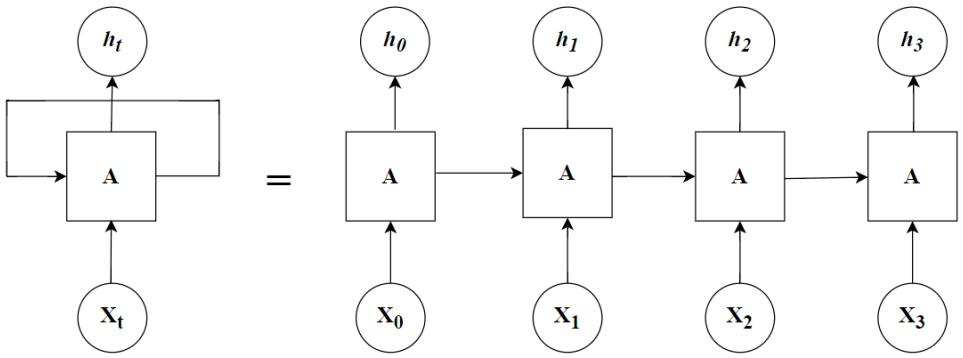


Figure 3.2: The internal structure of the RNN model and the unrolled version (right of equals sign).

3.3.2.1 Limitations in a Recurrent Neural Network

As mentioned, in the training of an RNN using BPTT, the gradients of the loss function are computed through backpropagation not only for the current time step t, but also for all previous time steps. However, a problem can occur when the gradients become too small as they are multiplied repeatedly by the weights of the network during backpropagation [18].

This problem is known as the *vanishing gradient problem* [16] and can be particularly challenging for RNNs with multiple hidden layers that use activation functions like the *sigmoid*. In such cases, the small gradients can get multiplied repeatedly as they propagate through multiple layers, causing the gradient to decrease exponentially as it moves backward in time [16]. As a result, the weights in the initial layers of the network may not be updated correctly, leading to inaccuracies in the entire network. This can be especially problematic for RNNs that need to learn long-term dependencies, as the small gradients can prevent the network from retaining information from earlier time steps.

3.3.3 The LSTM Model and the Autoencoder

The problem of the vanishing gradient was remedied by S. Hochreiter and J. Schmidhuber when they presented the LSTM model in 1997 [19]. At the heart of the LSTM model is the *memory cell unit* which replaces the nodes in the hidden layers of traditional NNs. The memory cell allows the NN to selectively choose what information to absorb or forget over time. A distinctive feature of the LSTM model is the sigmoidal units *input gate*, *forget gate*, and the *output gate* which lies in each memory cell and control and filter the information through the chain. In the memory cell, there is also the input node and the internal state.

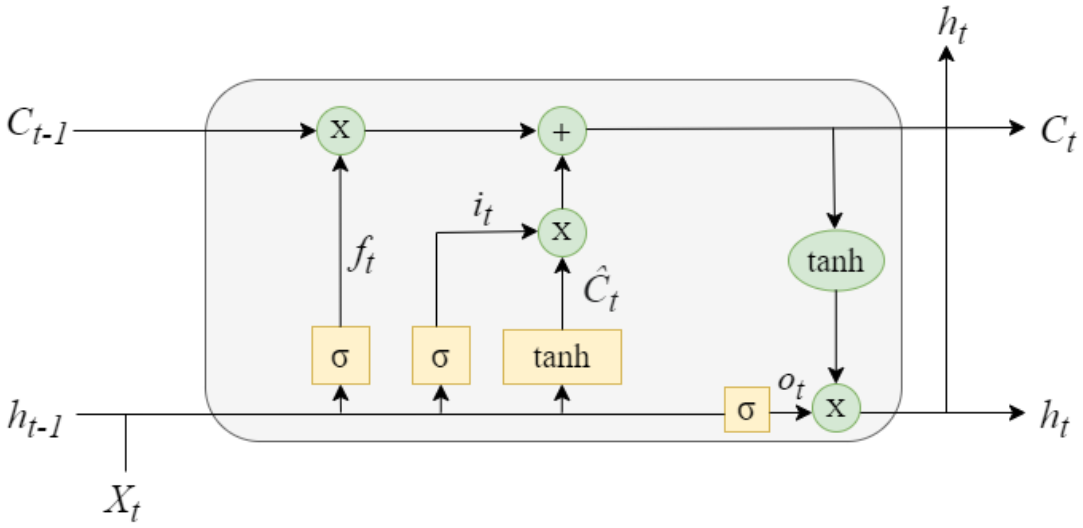


Figure 3.3: The LSTM cell architecture at time step t. Here X_t: input, h_t: output, C_t: cell state, f_t: forget gate, i_t: input gate, o_t: output gate, $\hat{C}_t$: internal cell state.

Figure 3.3 shows the internals of the LSTM cell *C's* architecture. The input gate selects what information is passed through into the memory cell at time step t. The input consists of the current input x_t and the previous hidden state h_{t-1}. The tanh activation function results in a value between 0 and 1 and indicated how much flow from the other node is cut off or not [16]. The forget gate determines how much content is in the internal state to discard at time step t. Producing a result between 0 and 1, the forget gate decides what should be retained for the next time step x_{t+1}. Finally, the output gate decides what information the memory cell should output at each time step, which is a combination of the input along with the current memory cell.

Reflected by the name, the LSTM model's purpose is to remember information over a long time and forget when it is no longer relevant. The three gates establish the opportunity for the LSTM model to do so, and as a result, combats the problem of the vanishing gradient [16].

The *Autoencoder* [20] was introduced as a NN and operates in an unsupervised manner by training to reconstruct the given input by encoding it and then decoding it. Figure 3.4 illustrates an arbitrary LSTM Autoencoder. The input X_1, X_2, ..., X_t is compressed into a lower dimensional representation through the encoder which learns the underlying features and produces the encoded feature vector z. This "middle section" is a hidden layer called the *bottleneck* and acts as a feature extraction for the decoder that follows. The Autoencoder can have more than 1 hidden layer, resulting in a *stacked autoencoder* or *deep autoencoder*.

As mentioned, the evaluation of the Autoencoder's performance is based on the ability to reconstruct the input, which is measured with the *Reconstruction Loss*

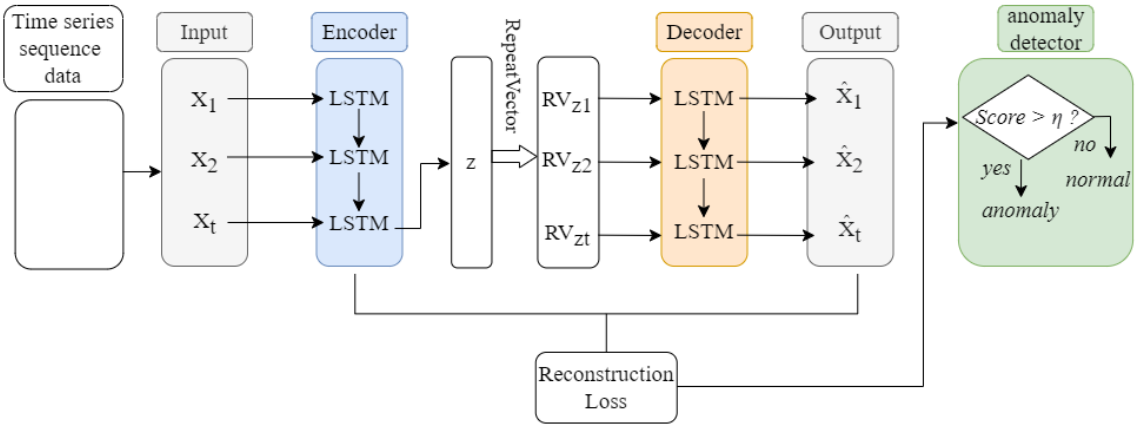


Figure 3.4: An arbitrary Autoencoder with LSTM encoder and decoder layers. Here, the threshold is represented by η.

(L) [21]. An anomaly can be detected by diverging from the rest of the data, and a threshold η can be set to decide the size of how much the instance can deviate. The Reconstruction Loss is calculated to minimize the difference between the output and the input as following:

$$L(x - \hat{x}) = \frac{1}{n} \sum_{i=1}^{n} = |\hat{x}_t - x_t|$$

Here, x represents the input data, $\hat{x}$ indicates the output data and n is the number of samples in the training dataset [21]. In order to work with sequential data, the autoencoder can be based on LSTM layers as depicted by Figure 3.4.

Using an LSTM model on sequential data has for example been covered by Wei et. al. [21] who proposed using an LSTM model on time series data regarding indoor air quality. This showed promising results for detecting anomalies and outperforming similar models.

There are also examples of using a combination of the LSTM model and the IF. Priyanto et. al. [22] presented a system where IF is used for labeling the data and then applying an LSTM Autoencoder for detecting anomalies related to agriculture. This combination demonstrated assuring results for the IF to label data, and the LSTM Autoencoder to generate satisfying results.

4
System Design and Development

This chapter will present all relevant design choices made when creating the ML module. This includes consideration regarding which model to implement, the execution of pre-processing the data, and finally the overall architecture of the developed system.

4.1 Choice of Model

Throughout the project, considerable time has been invested in researching and understanding different solutions for an ML model fit for anomaly detection. In particular, the approaches to use the IF or the LSTM Autoencoder model (See Chapter 3) stood out as advantageous choices for implementing the ML module. There was an ambition to implement both models and compare their performance. However, due to the emerged data limitations, a decision was made to only focus on one approach. Therefore, a decision between these models had to be made.

Being an ensemble algorithm consisting of several estimators (trees), the IF model is robust and capable of handling multidimensional data, both of which are great advantages. In addition, it is somewhat easy to interpret, making it possible to draw conclusions regarding what features contribute to the biggest difference between normal and anomalous instances. However, since anomalies are expected to be found when reviewing sequences of requests as opposed to each request in isolation, this approach would require that the information in each sequence is interpreted and compounded into one single data point before it is reviewed by the algorithm. This implies that the data need to be processed before reaching the ML model, which can be considered problematic in a system that expects a large amount of traffic.

In comparison, the LSTM model is made to handle sequential data. Using this approach would mean that lesser computational power would need to be used to process the data before getting analyzed. Since some of the data used in this project is non-numerical, some preparation of the data would still need to be made by interpreting the non-numerical values to suitable numerical ones. Nevertheless, the LSTM model has advantages in cases where anomalies follow less obvious patterns whereas IF is a good choice for tasks where anomalies are few and easily separable from normal data points. Being able to detect anomalies from complex patterns might be preferable in real-life applications. However, by being a NN the LSTM requires a large amount of data to perform at its peak, which was not available for this project.

After weighing the options, and considering the advantages and disadvantages above, the final decision was made to implement a system using the IF model. In particular the favorable robustness and interpretability of IF along with its capacity to handle multidimensional data influenced this decision.

4.2 Pre-processing of Data

The data that will be used in this project will consist of entirely synthetic data where each data point will include metadata for one request. Table 4.1 shows examples of the synthetic data instances that was provided for this project. The parameters included are:

- *timestamp* represents the UNIX timestamp when the request was made.
- *userID* identifies the user making the request.
- *sessionID* is the unique token for the session in which the request was made.
- *expiring* represents the UNIX timestamp when the request expires.
- *URL* depicts the URL path for the request and thus gives insight into the kind of request made.

Table 4.1: Examples of data instances

'timestamp'	'userID'	'sessionID'	'expiring'	'URL'
1676373343936	user-1003	c85cbd00-d2fc-4cda-...	1676373495816	/chat/send/442
1676373344344	user-1005	a0038f81-b46a-4f26-...	1676373476316	/profile/407
1676373344456	user-1003	c85cbd00-d2fc-4cda-...	1676373495816	/chat/send/969
1676373344912	user-65	fd48a9f9-67f4-4185-...	1676373604676	/searchUsers?page=0
1676373345020	user-1005	a0038f81-b46a-4f26-...	1676373476316	/profile/148

Since the purpose of this project is to detect anomalous behavior, a large amount of information would get lost by examining each request in isolation. For this reason the data used as input will be based on information procured from a sequence of requests rather than individual requests. Due to the importance of the model being able to analyze a sequence of requests as fast as possible, a few decisions are made to achieve this:

- Limit the number of requests allowed in each sequence.
- Summarize and interpret the information in each sequence into one vector consisting of new informative features (Section 4.2.1).

By limiting the number of sequences allowed in each sequence, the number n of request sequences for each user u in the data set will be derived by:

$$n = \frac{\text{Requests made by user } u}{\text{Maximum sequence length}}$$

4.2.1 Feature Engineering

In this section, each step in the process of extracting features from the data will be presented.

4.2.1.1 Bag-of-Words Representation and Vectorization

The URL path of a request can be examined to extract information regarding what type of request a user has made. However, the URL path is represented by a string which is a format not recognized by the model. In order for an ML model to interpret and find patterns within the data used to train it, the data itself must consist of numerical values.

In *natural language processing*, a *Bag-of-Words (BoW)* technique can be used to create a numerical representation of each word within a string of text. This is done by converting the string into unique tokens (words) and then counting the occurrence of each token within the text. This technique disregards the grammar and order of the tokens and simply keeps track of how many times they occur within a text document [23]. Table 4.2 shows three examples of this technique.

Table 4.2: Bag-of-Words representation for three sentences

String	'spam'	'is'	'annoying'	'you'	'got'	'rhymes'	'with'	'ham'
"Spam is annoying"	1	1	1	0	0	0	0	0
"You got spam"	1	0	0	1	1	0	0	0
"Spam rhymes with ham"	1	0	0	0	0	1	1	1

This BoW technique is the fundamental component in the implementation of different vectorizers, that serve the purpose to interpret non-numerical data. For this thesis, the string that will be vectorized consists of all the URL paths within a sequence of requests made by a user. Table 4.3 shows the concatenated URL strings for each user in the dataset shown in Table 4.1.

Table 4.3: Examples of concatenated URL strings

user-1003	/chat/send/442,/chat/send/969
user-1005	/profile/407,/profile/148
user-65	/searchUsers?page=0

In order to vectorize these URL strings, three different vectorizers have been examined. The result from the vectorization will be used when extracting features from each request sequence.

- **CountVectorizer**: A *CountVectorizer* is the direct implementation of the BoW technique and a common tool for representing textual data into numerical values. The vectorizer both carries out the tokenization and counts all the occurrences of each word [24]. Table 4.4 shows the results of running the strings in Table 4.3 through a CountVectorizer.

Table 4.4: Result of the CountVectorizer on the URL-strings

user	'chat'	'send'	'profile'	'searchUsers'	'page'
user-1003	2	2	0	0	0
user-1005	0	0	2	0	0
user-65	0	0	0	1	1

- **TfidfVectorizer**: In some applications a few specific words might have a high occurrence rate throughout the entire corpus of text, while not offering much information about the content. In these cases a *Term Frequency Inverse Document Frequency (TF-IDF)* vectorizer [25] can be used to produce an importance score for each word within a text. This importance score is based on how many times a word occurs within a specific text in comparison to how often it occurs within the entire corpus. The *Term Frequency (TF)* calculates the frequency of a word t in document d, and the frequency serves as a metric in order to define the number of occurrences. The TF is derived as follows:

$$TF(t,d) = \frac{\text{counts of word } t \text{ in document } d}{\text{number of words in document } d}$$

  The *Inverse Document Frequency (IDF)* parameter calculates the relevancy of a word where words deemed to be more significant will retrieve a higher frequency score. The IDF is derived as follows:

$$IDF(t) = log(\frac{\text{number of documents in corpus } D}{\text{number of documents containin word } t})$$

  The importance score is derived by calculating $TF \times IDF$, which results in a value between 0-1 where tokens weighted close to 0 are considered less important. This establishes a relationship between the words and the documents in which they appear and a computer can draw more firm conclusions about a text and how it relates to other texts within the corpus.

- **HashingVectorizer**: Another alternative for vectorizing data is to use a HashingVectorizer [26]. Similarly to the CountVectorizer, the HashingVectorizer tokenizes the input and counts the occurence of each token, and stores these in a sparse matrix, with each token being mapped to an index using a hash function on each token's string name. The advantage and drawback of this method is that the vectorizer does not store a vocabulary dictionary, which makes it ideal for applications processing vast amounts of data. However, this also means that it is not possible to extract and compare each token in regards to occurrence rate or importance score, which makes it unsuitable for applications where this is a requirement. Additionally, the size of the matrix needs to be defined upon the creation of the vectorizer. This opens up the risk of hash collisions since similar words might be mapped to the same index in case the matrix size is too small.

4.2.1.2 Features

Each sequence of requests will be summarized and interpreted into one single feature vector. These features were chosen based on previous domain knowledge obtained during the research phase and guidance from the project's technical supervisor.

Variance Score: The essence of behaviour classified as spam is the repetitive nature of the requests. For example, a spammer conducting a phishing scam will send messages to a sizeable amount of users in the hope of someone falling for the bait [27]. This will result in very little variation regarding what kind of requests are made, and the messages will most likely follow a similar structure. This thesis will be limited to analyzing the metadata only, which means that analysis regarding the content of messages will not be considered. Instead, the information stored as the *URL* attribute will offer insight into what kind of requests the sequence consists of. For example, a user or a bot indulging in sending spam might have a significantly larger amount of requests related to searching for profiles, sending messages, and repeating this process; A normal user might have a more widespread distribution.

Based on the values from the vectorizer chosen, the variance score will be calculated and extracted as a feature called *var_score* in order to detect repetitions in behaviour. The variance σ^2 calculates the distance from each element X_i to the mean value of the data set μ and squares it. This statistical measurement depicts the average degree to which each point differs from the mean. The formula to calculate the variance is the following:

$$\sigma^2 = \sum_{i=1}^{n} \frac{(X_i - \mu)^2}{n}$$

Table 4.5: Variance score (σ^2) comparison on synthetic data for the CountVectorizer and TfidfVectorizer between a normal user and a spammer.

	CountVectorizer		**TfidfVectorizer**	
	Normal user	**Spammer**	**Normal user**	**Spammer**
conversation	16	0	0.588897	0.000000
profile	14	0	0.514772	0.000000
page	10	15	0.366963	0.248065
markasread	8	0	0.294448	0.000000
searchusers	6	15	0.220178	0.248065
chat	5	40	0.183664	0.662166
send	5	40	0.183664	0.662166
inbox	4	0	0.159757	0.000000
login	4	0	0.146785	0.000000
block	1	0	0.040787	0.000000
report	1	0	0.046614	0.000000
config	0	0	0.000000	0.000000
like	0	0	0.000000	0.000000
photos	0	0	0.000000	0.000000
userconfig	0	0	0.000000	0.000000
σ^2	26.5	203.1	0.0355	0.0556

Table 4.5 shows the result when applying the CountVectorizer and TfidfVectorizer to a request sequence made by a normal user and a spammer, as well as the variance score σ^2 for each. We observe that the variance score based on the result from the CountVectorizer seems to offer a higher disparity between a normal user and a spammer compared to the TfidfVectorizer. This is due to the fact that the requests appearing frequently in the sequence made by the spammer also appear frequently in the entire corpus. As such there are no requests that are more tightly associated with spam behaviour in comparison to the rest.

For the above reasons, the decision was made that the variance score will be calculated based on occurrence count rather than importance score, which also can be considered the more intuitive choice. Thus, during the initial stages of implementation, the CountVectorizer was used. This was however later changed and the HashingVectorizer was used instead. This change was made due to the benefit of not being restricted to a fixed, finite vocabulary while still keeping the occurrence count aspect.

Longest Consecutive: In addition to detecting repetitive behaviour by calculating the variance score for each request sequence, the longest chain of consecutive identical requests will also be counted for each sequence and extracted as the feature *longest_consec*. A higher score for this attribute will indicate a more repetitive behaviour with less distribution amongst the type of requests.

SessionID Update Frequency: The *sessionID* parameter represents an identifier for the session in which a request was made. For a legal session, a session ID is obtained when the requesting user initiates has passed authentication, for example, login, and stays valid for a finite period of time. When the session token has expired, the user needs to pass authentication again [28].

For this thesis, the sessionID update frequency was examined. This was done by calculating the number of unique sessionIDs active during five-minute windows for each user and storing the result as *avg_tokens*. The purpose of this was to examine whether there was a difference between normal users and abnormal users regarding how often they initiate new sessions.

Request Frequency: Another feature to be examined is regarding whether the request frequency can indicate that a user shows anomalous behaviour. The request frequency will be derived by calculating the average time difference between each request in the sequence and is represented as the feature *request_freq*.

Sequence Time: In addition to examining the average time between each request in a sequence, the total time between the first and the last request will be examined as the feature *sequence_time*.

4.3 The Anomaly Detection System Architecture

Figure 4.1 illustrates an overview of the ML module that has been developed during the project. It includes every step from the module obtaining data from the GRA as an API call, to it sending back a response with the suggested action or an error code.

- All data is transferred to the module as a JSON file containing a number of requests, i.e. a sequence.
- After the data has been unpacked, it goes through a number of pre-processing steps which are represented by the *feature extraction* box.
- Depending on how many requests were sent to the module, two different models are utilized; one trained on sequences of 30 requests and the other on sequences of 60 requests.
- After the model has evaluated the data, action is suggested accordingly and sent back as a response to the GRA. The possible actions include *pass*, *block*, and *verify*. The suggested action *verify* prompts the server to make the API call that the user in question should pass a verification test before being allowed further usage of the application. What kind of verification will be used is decided by the service provider and is not considered in this project.

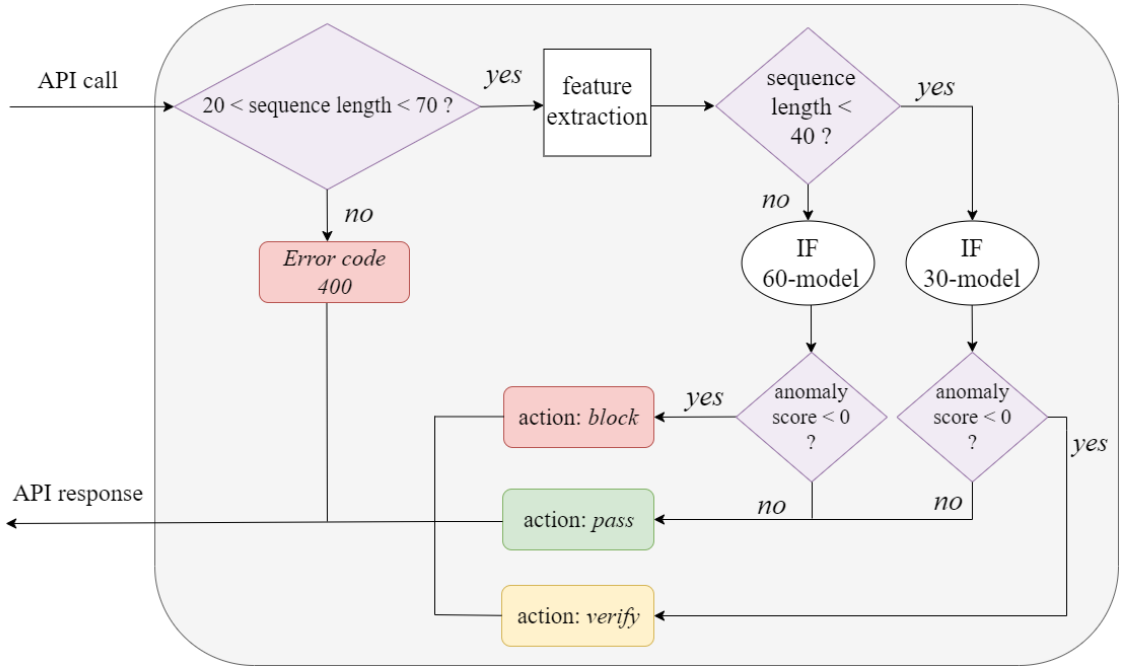


Figure 4.1: System architecture.

5
Implementation

The thesis mainly involved software implementation of the proposed ML system; no hardware setup was required. This chapter discusses the tools and frameworks used to implement the ML model described in Chapter 4. The developed model was finally integrated and tested with the GRA) application.

5.1 Tools and Frameworks

The software and tools utilized in the implementation of the thesis are presented below:

- **Python**: Python [29] is a high-level open-source programming language that is widely used for several kinds of applications. For scientific and ML purposes, it is particularly useful since a lot of libraries and frameworks are implemented in Python. In order to take advantage of existing libraries and tools regarding ML, the implementation for the most part will be written in Python 3.11.
- **SciKit-learn**: For this project, the *Scikit-learn* version 1.2.2 [30] API has been the main tool for creating the ML system. Scikit-learn is a Python module featuring various ML algorithms for both supervised and unsupervised purposes.
- **NumPy**: Scikit-learn uses *NumPy* [31] as an underlying technology. NumPy is a Python library that uses multidimensional arrays (*ndarrays*) at its core. The library consists of various operations on these arrays such as mathematical, logical, shape manipulation, and more. The algorithms found in Scikit-learn use these *ndarrays* as input, which is why a large part of the computations on the data is used using tools from NumPy 1.24.2.
- **pandas**: The data used for training and testing the model will be stored in DataFrames. *pandas* [32] is an open-source project that offers solutions for reading and writing data to and from files of various formats and for creating DataFrame objects that can easily and efficiently be reshaped and manipulated. The pandas version 2.0 will be used.
- **Flask**: *Flask* (version 2.2.3) [33] is a Python web application framework and will be used for creating the API. Flask is based on Werkzeug and Jinja, which is a library for WSGI (Web Server Gateway Interface) applications and a template engine for Python, respectively.

- **Docker**: *Docker* (version 1.25.1) [34] is a development tool with the purpose of packaging software into lightweight standardized units. This is done by creating Docker images, which contain all necessary components for running and executing the application including all necessary dependencies. The docker image works as a blueprint for the application and creates a standalone container for the application during runtime. A benefit of using containers is that several containers can efficiently share the same operating system kernel, running as isolated processes. Since the ML module that will be developed during this project will be a puzzle piece of a larger request analyzing system, this will be an important part of the integration step.

5.2 Integration

The developed ML module will be integrated into the GRA system by deploying it as an as an API. This will make it possible to access the module as long as the correct endpoint is provided. The ML-API will be containerized and run together with the GRA system using Docker.

6
Experimental Methodology

This chapter will present the methodology carried out to set up the testing environment and the evaluation metrics considered.

6.1 Simulating Data Sequences from the GRA

Since the system will only operate on metadata, no further information will be available for the analysis, for example, the content of a message sent from a user. As already discussed, this also implies that no informative analysis can be made by examining each request in isolation since the metadata alone will only provide a snapshot without any context regarding previous and future behaviour. For this reason, the requests need to be simulated and investigated as a sequence in order to put the metadata into a perspective of normal versus abnormal conduct.

Sequence Length Considerations: To simulate how requests will be sent to the ML module in sequential order, the data provided was divided into several data chunks for each user within the data set. Each data chunk represents one request sequence. An important aspect emerged regarding the length of each sequence, which needs to be a trade-off between accuracy and efficiency. Due to the requirements to detect abnormal users while also maintaining an acceptable runtime, the length of the sequence needs to be long enough to provide context while also not being unnecessarily long. For this reason, the analysis will be conducted on sequences of 10, 20, 30, 40, 50, 60, 70, and 80 requests.

Model Considerations: For this thesis, the decision was made to try out several instances of the IF model. The difference is the sequence length of the training data, which corresponds to the established lengths of the testing sequences. These models will be referenced as the 10- to 80-model, which leads to 8 models where the number signifies the sequence length of the training data.

6.2 Feature Engineering on a Sequence

After establishing that the data will be divided into chunks to simulate how the system will operate, sequences of data points were interpreted into feature vectors in accordance with the feature engineering. Table 6.1 and Table 6.2 showcase an example of the contrast between two instances of feature vectors (normal and ab-

normal) after all pre-processing steps have been made. These vectors are derived from sequences consisting of 30 and 60 requests respectively.

Table 6.1: Examples of feature vectors after extracting information on 30 requests for an abnormal user (user-1003) and normal user (user-72).

user	*request_freq*	*avg_tokens*	*longest_consec*	*var_score*	*sequence_time*
user-1003	2.0884	1.00	11.00	0.0005	60.5650
user-72	872.0159	0.0526	5.00	0.0001	22672.4130

Table 6.2: Examples of feature vectors after extracting information on 60 requests for an abnormal user (user-1003) and normal user (user-72).

user	*request_freq*	*avg_tokens*	*longest_consec*	*var_score*	*sequence_time*
user-1003	2.1348	1.00	15.00	0.0020	125.9540
user-72	846.6023	0.0450	5.00	0.0003	33017.4900

From these tables, possible differences between the behaviour of normal and abnormal users can be observed. For this example, the normal user has significantly more time between requests in comparison to the abnormal user, which can be deduced from the numbers in the *request_frequency* and the *sequence_time* columns. In addition, all other features also indicate differences between normal and abnormal user behaviour, but not as clear as the ones previously mentioned. When comparing the two tables, it is also possible to observe that some of the features start to diverge even more as sequence length increases. This is especially true for *longest_consec*, *var_score*, and *sequence_time*.

6.3 Partitioning Training and Testing Data

The synthetic data used for this thesis is generated to mimic normal as well as abnormal behaviour. The set includes metadata from 263211 different requests made by 1004 simulated users, of which there are two users mimicking typical spam behaviour and one mimicking web scraper behaviour. These three abnormal users were responsible for 134020 requests, which is more the half of the total number of requests and does not necessarily reflect the real-life distribution. For this reason, only a small subset (0.2%) of the data points from abnormal users were included in the training data set, whereas 80% of the normal instances were included. This resulted in a distribution of approximately 99.8% normal and 0.2% abnormal instances in the training data sets and 17% normal and 83% abnormal in the testing data sets.

In order to optimize and evaluate the performance of the models, a number of tests will be conducted (see Chapter 7). For these tests, all scores for the confusion matrices will be obtained by training and testing each model for 100 iterations. These iterations will have different randomly assembled training and testing data sets. The final scores will be represented by the average of all iterations. For every iteration, the IF model will be implemented with a contamination score of 0.002, which corresponds to the number of anomalies in the training data sets.

6.4 Evaluation metrics

This section discusses the performance metrics used in the evaluation of the system. A confusion matrix and several other calculations that can be derived from the matrix are used to analyze the performance of the classification algorithm.

6.4.1 The Confusion Matrix

A confusion matrix is a popular method for evaluating classification models, which shows how many predictions are correct and incorrect for each class. From this matrix, calculations can be conducted on the number of predictions where the predicted value equals the true value, i.e., where $\hat{y}(x) = y(x)$, and also on the instances which were classified incorrectly [35].

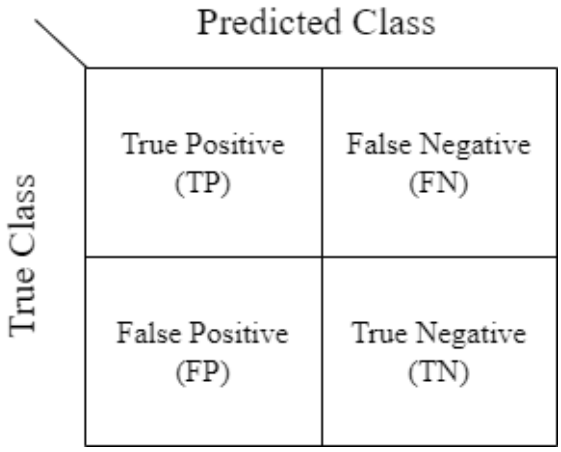


Figure 6.1: The confusion matrix

Figure 6.1 summarizes the results of the predictions. In an analogy of anomaly detection, the possible cases include –

- *True Positive (TP)* - users who have been correctly classified as abnormal.
- *True Negative (TN)* - users who have been correctly classified as normal.
- *False Negative (FN)* - users who have been misclassified as normal while actually conducting abnormal behaviour.
- *False Positive (FP)* - users who have been misclassified as abnormal while actually conducting normal behaviour.

In summary, the confusion matrix provides insight into how correct the model's predictions were and how well they hold up against the actual values. From these values, the commonly used evaluation metrics *Precision*, *Recall*, *F1* and *Accuracy* can be derived.

6.4.2 Precision

The *precision* of an anomaly detection model's predictions is the ratio of users being correctly classified as abnormal (TP) out of all the users predicted to be abnormal (TP + FP) [35].

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

A high precision score, e.g. 0.93, would mean that when finding abnormal users, the model is correct 93% of the time. A high precision score is desired in applications where it is important to correctly detect relevant data points and not wrongfully classify negative instances as positive. In the case of this project, a high precision score would indicate that the users flagged as suspicious has a low probability of being misclassified as normal user.

6.4.3 Recall

The *recall* is a metric of the ratio of correctly classified abnormal users (TP) divided by all the users with abnormal behaviour (TP + FN).

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

The recall, or *sensitivity*, measures how efficient the model was at identifying as many true positives as possible [35]. In the case of anomaly detection, a high recall is wanted since it will depict how well the model identified the relevant anomaly instances. A low recall would indicate that a significant amount of abnormal users were not identified and thus slipped under the radar when they instead should have been recognized.

6.4.4 F1 Score

Achieving a high precision and a high recall at the same time is in most real-life situations not achievable [36], and a trade-off between them might have to be made regarding which of these metrics is the most important. However, in cases where precision and recall are considered equally important, the *F1* score can be used to measure the performance, as it is the harmonic mean between the two.

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

For ML models performing an anomaly detection task, it could be important to not only receive a high precision score in order to avoid misclassifying normal users but also to detect all the cases of actual abnormal behavior. A high F1 score could indicate that the model has an acceptable balance between precision and recall.

6.4.5 Accuracy

A model's accuracy is a simple metric of the ratio between correct predictions and total predictions.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

6.4.6 Summary

Despite the intuition of an accuracy score, the recall and precision offer additional insight and clarity beyond this single value [36]. Additionally, a high accuracy does not guarantee a high recall or precision score. Meaning, a model that produces predictions with a high accuracy score could at the same time have a low recall. The same could be true regarding the precision score. This is especially the case when the data is very unbalanced, which is common for anomaly detection. Due to the misleading interpretation that might occur when looking at an accuracy score, this metric will not be used or analyzed further in the evaluation. For this purpose, the precision and recall along with the F1 score will be a better indicator of the model's performance.

7
Evaluation

Several experiments were done to study the performance of the implemented ML system. This chapter discusses our results and observations on the experiments conducted.

7.1 Initial Comparison of Models

The first experiment serves to examine the performance of the models when evaluating data from sequences of the same length as those it was trained on, and comparing the result between each model. Figure 7.1 shows the performance scores from this experiment.

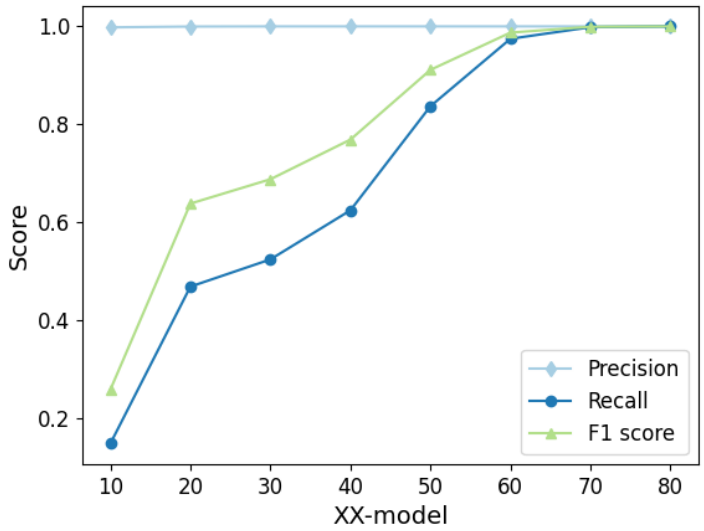


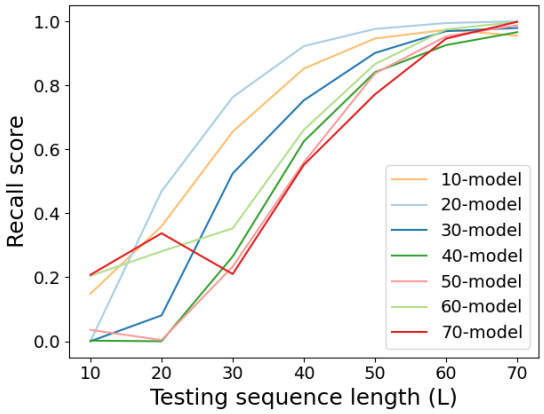
Figure 7.1: Performance scores of models (*XX-model*) trained and tested with the same sequence length (*XX*).

It can be observed that the precision score is constant around 100% regardless of sequence length. For recall and F1 score, both metrics increased along with the sequence length. A conclusion can thus be drawn that when the model is trained and tested on sequences of equal length, the performance increased as the sequences grew longer. This could be due to the difference between normal and abnormal behaviour being amplified and more detectable as more data is taken into consideration.

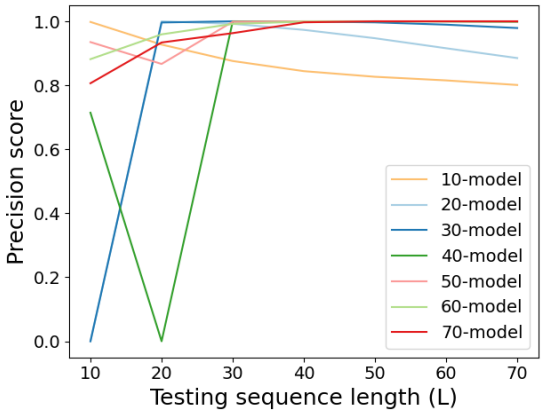
Going forward with the tests, the 80-model will no longer be considered due to the aspiration to keep the sequences short. Additionally, this model is not expected to perform significantly better than the 70-model since precision and recall both already converge towards 100% for the 70-model. A testing sequence length of 80 will also not be considered.

7.2 Decoupling training and testing sequence lengths

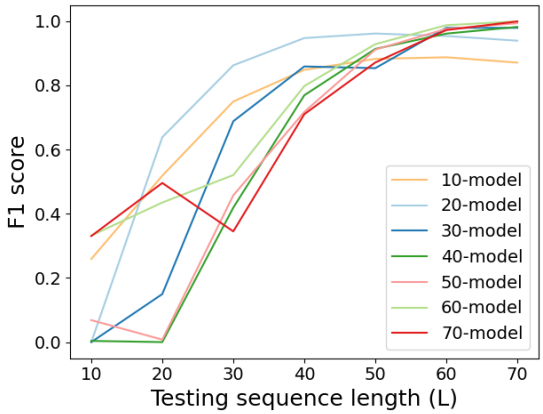
To examine the trends when decoupling training and testing sequence length, experiments were made on models trained on 10, 20, 30, 40, 50, 60, and 70 requests. These models were tested on sequences with lengths ranging between 10 and 70 requests. Figure 7.2 shows the resulting recall, precision, and f1 scores for the different models on testing sequence lengths *(L)* varying from 10 to 70.



(a) Recall score



(b) Precision score



(c) F1 score

Figure 7.2: Recall, Precision and F1 scores for different models on varying the testing sequence lengths (L).

The following observations were made from the scores:

- 10- and 20- models generally receive the best recall scores on L longer than 20 requests, followed by the 30-model and 60-model as the 3rd and 4th highest scoring. This leads to a general conclusion that longer sequences of requests are easier to classify correctly for these models. Meanwhile, the 10- and 20-model tend to become less precise, showing their precision score decreasing as the L increases.

- For $L = 10$ and $L = 20$, all the models demonstrate relatively low precision and recall scores. However, these scores show an increasing trend as L increases. For example, for $L = 20$, the 20-model exhibits the highest recall score of 45% meaning more than half of the anomalies go unnoticed. For $L = 10$, none of the models achieve a score higher than 25%. At $L = 20$, the 30-model obtains the highest precision score of approximately 100%. However, at $L = 10$ it holds a score of 0%.

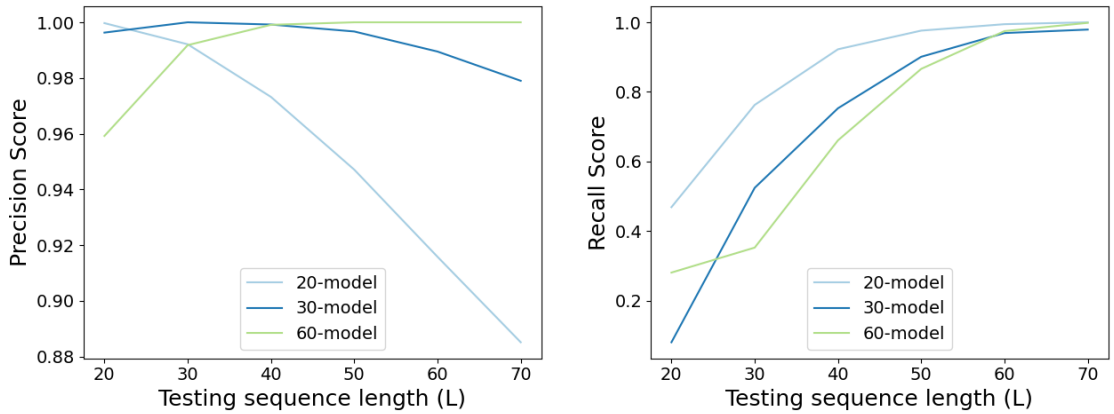
- 40-, 50-, and 70- models exhibit generally poor recall performance compared to the other remaining models for various lengths of L. Similarly, F1 scores for these models demonstrate the general unpromising outcomes.

These tests leave us with two conclusions: Firstly, the majority of the models exhibit their poorest performance where L ranges from 10 to 20. As a consequence of these findings, a sequence length below 20 will not be taken into consideration for any of the models. Secondly, due to the nature of the system of which the ML module will be a part of, a high precision score is preferable to not cause disruption for normal users by misclassifying them as abnormal. A diminishing precision score, as observed for the 10- and 20-model in Figure 7.2b, suggests that these models are not suitable for the application. However, the 20-model stands out in terms of recall and F1 score and has an acceptable precision score up until sequences of 30 requests when it starts to decrease. This makes it unsuitable for analyzing longer sequences but could be used for shorter ones, which is a reason for further examination.

Going forward, the models left that merit further consideration are the 20-model, the 30-model, and the 60-model. This is due to their demonstration of high and consistent recall scores and generally high precision scores, with no significant regressions observed as the test sequence length grows. In terms of the testing sequence length, the range of examination has been narrowed down to a minimum of 20 requests and a maximum of 70.

7.3 Examination of the 20-, 30- and 60-models

To conduct a closer comparison between the 20-, 30- and 60- models, the third set of experiments will look closer into the recall and precision scores of the selected models. L will again range from 20 to 70 as disclosed in the first test. The results are shown in Figure 7.3.



(a) Precision score **(b)** Recall score

Figure 7.3: Precision and Recall score comparison between the 20-, 30- and 60-model.

The following observations were made from the scores:

- In Figure 7.3a, the trends regarding precision scores can be observed. The most notable being the downward trajectory of the 20-model as well as the almost inverse relationship between the precision scores of the 30 and 60-models. This clearly illustrates within what sequence length windows these two models excel and regress.
- As for recall in Figure 7.3b, the 20-model excels with the highest recall score throughout the entire interval L. This in combination with the decreasing precision scores indicates that at a certain point, the 20-model becomes less proficient at differentiating between spam and normal behavior, but rather between short and long sequences.
- Similarly to the 20-model, the 30-model performs best when evaluating shorter sequences. In terms of precision, it receives approximately 99.6%, 100.0%, and 99.8% for $L = 20$, 30, and 40 respectively. At $L = 40$, the 30-model starts to recede this score as the 60-model starts to perform better and converges to 100% as L increases.
- Both the 30- and the 60-model show an increase in recall, but with a less harsh decline in precision compared to the 20-model. This indicates that they have a greater capability to notice relevant patterns in the data. The 30-model overall has a slightly better recall performance with the 60-model closing the gap at sequences of 60 requests.

These results confirm that the 20-model performs the best when evaluating sequences shorter than 30 requests, and does manage to early on find a fairly large share of anomalies within this interval. After $L = 30$, the result can no longer be trusted due to decreasing precision score.

Regardless of how many requests have been made by a user, being wrongfully flagged as an anomaly poses the risk of being disruptive. For longer sequences, there is an expectation that a more precise evaluation should be made due to access to more information. However, the lack of information for shorter sequences is a good reason for extra caution. This is why for this application, a precision score as close as possible to 100% for the shorter sequences is preferable to a high recall.

Because of the above reasons and the fact that the 30- and 60-models on different intervals both receive a higher precision score than the 20-model, only these two will be taken into account for further examination. This decision is made with the understanding that the 20-model could be a viable option depending on considerations made regarding the trade-off between precision and recall.

7.3.1 Confusion Matrix Analysis

To achieve clearer insight into how the 30- and 60-models behave throughout L, a comparison of the models' confusion matrix values (TP, FP, TN, FN) is done. The result for different L on the two models is presented in Figures 7.4 and 7.5.

The following observations were made from the scores:

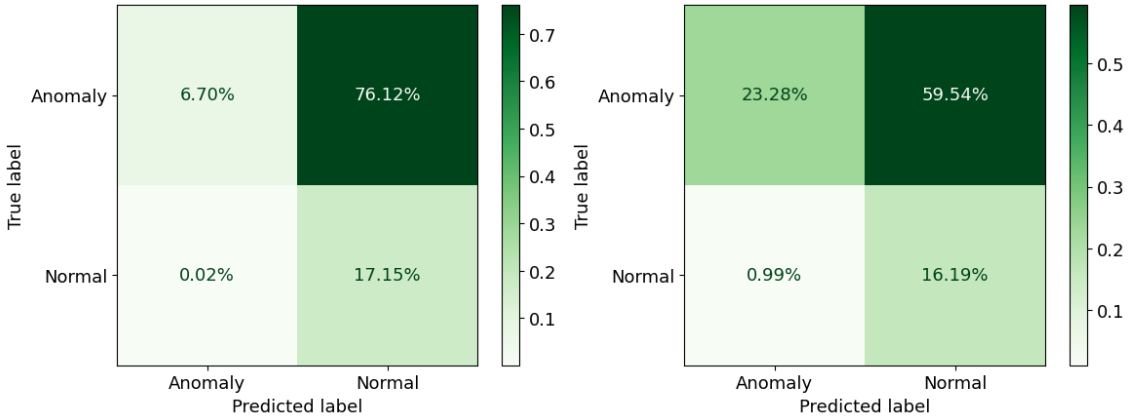
- The confusion matrices (a) and (b) in Figure 7.4 illustrates how for $L = 20$, both models have poor performance regarding the TP score. The 60-model did receive a higher score of 23.28% compared to the 30-model's 6.70%, however, the 30-model did receive a lower FP score of 0.02%.

- At $L = 30$ as illustrated in the (c) and (d) matrices in 7.4, the 30-model shows a considerable increase in its TP score of 43.19%, as the FP lands on 0.00%. The 60-model increases slightly in its TP score, and the FP score settles at 0.24%.

- At $L = 40$ depicted in the matrices (e) and (f) in Figure 7.4, the two models both hold an FP score of 0.05%, while the TP is slightly higher for the 30-model. This result aligns with Figure 7.3b where the two models' precision score intersects and the recall scores are similar.

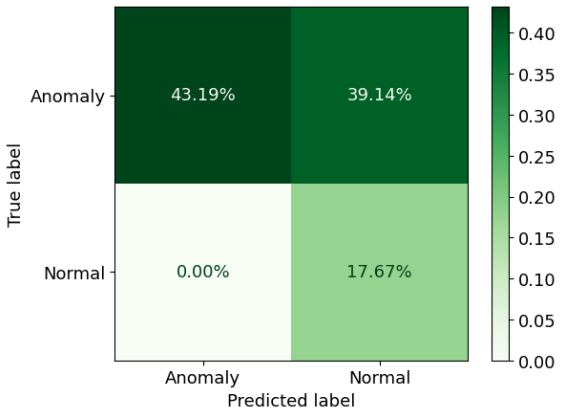
- For $L = 50$ shown in Figure 7.5 matrices (a) and (b), the 60-model has started to outperform the 30-model regarding occurrences of FP. The TP rate for both models also increased.

- As L increases up to 60 and 70 in Figure 7.5 (c - f), the 60-model keeps its FP rate at 0.00% as the TP increases. For the 30-model, a similar increase in TP occurs. However, the number of FP keeps growing as well, which illustrates the 30-model emerging tendency to misclassify normal users as abnormal with growing sequence length.

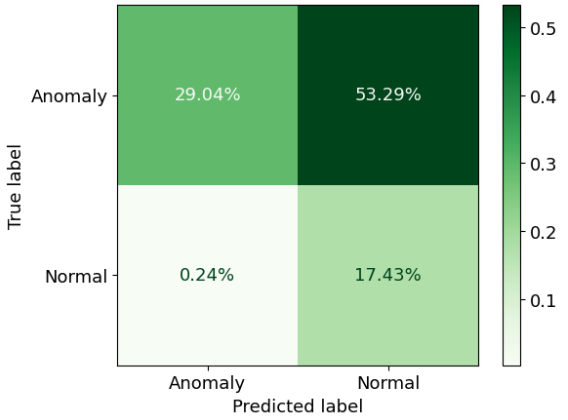


(a) 30-model, $L = 20$

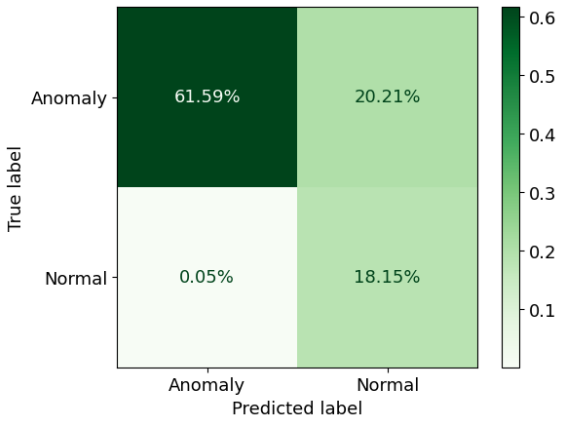
(b) 60-model, $L = 20$



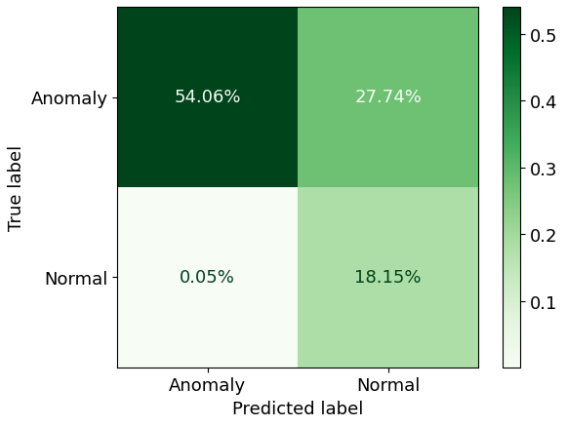
(c) 30-model, $L = 30$



(d) 60-model, $L = 30$

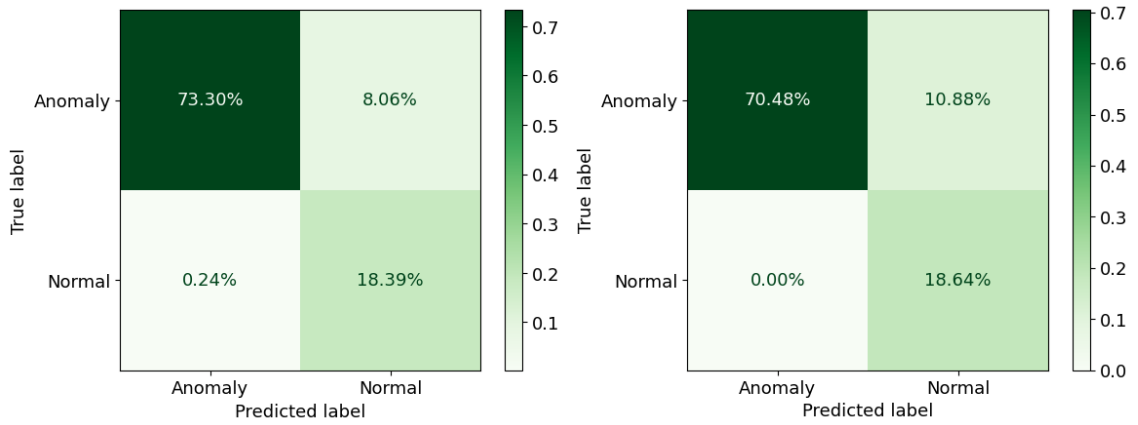


(e) 30-model, $L = 40$



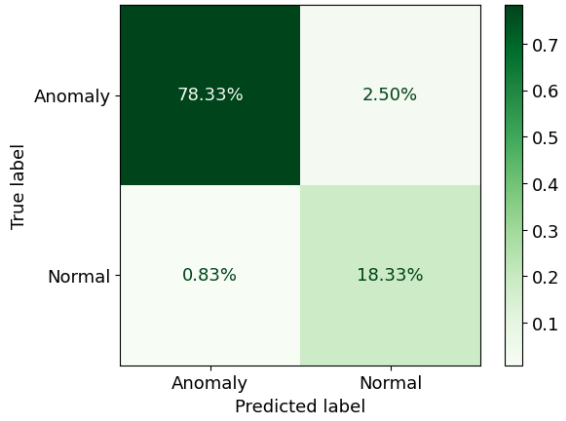
(f) 60-model, $L = 40$

Figure 7.4: 30-model and 60-model confusion matrices for $L = 20$, 30, and 40.

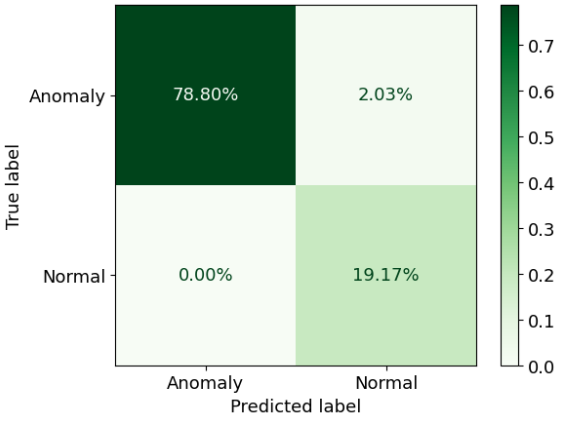


(a) 30-model, $L = 50$

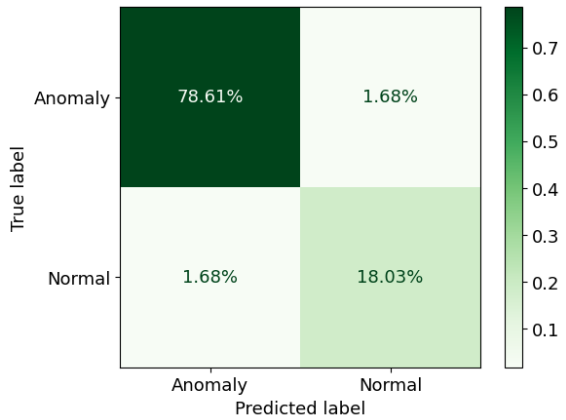
(b) 60-model, $L = 50$



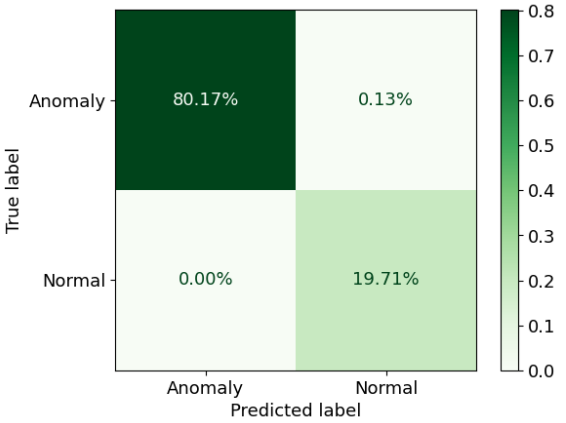
(c) 30-model, $L = 60$



(d) 60-model, $L = 60$



(e) 30-model, $L = 70$



(f) 60-model, $L = 70$

Figure 7.5: 30-model and 60-model confusion matrices for $L = 50$, 60, and 70.

These observations further emphasise and explore the 30-model's ability to be more precise on shorter sequences, as it receives higher TP scores and lower FP scores as L grows towards 40. However, around $L = 50$ the 60-model starts outperforming the 30-model as it receives lower FP scores. After this point the 60-model also yields slightly better TP scores. In conclusion, the 30-model performs better regarding the precision on an L from 20 to 40, and the 60-model for L from 50 to 70.

7.4 Feature Importance Analysis

As a final test to examine the 30- and 60-model, each feature will be analysed to get further insight into how they contribute to the performance. In order to show the correlation between all features and how they affect performance, the resulting clusters produced with $L = 60$ on the 60-model are visualized in Figure 7.6. The setup to use the 60-model with $L = 60$ was chosen due to it being the one yielding the overall best result, which makes it easier to identify well-defined clusters.

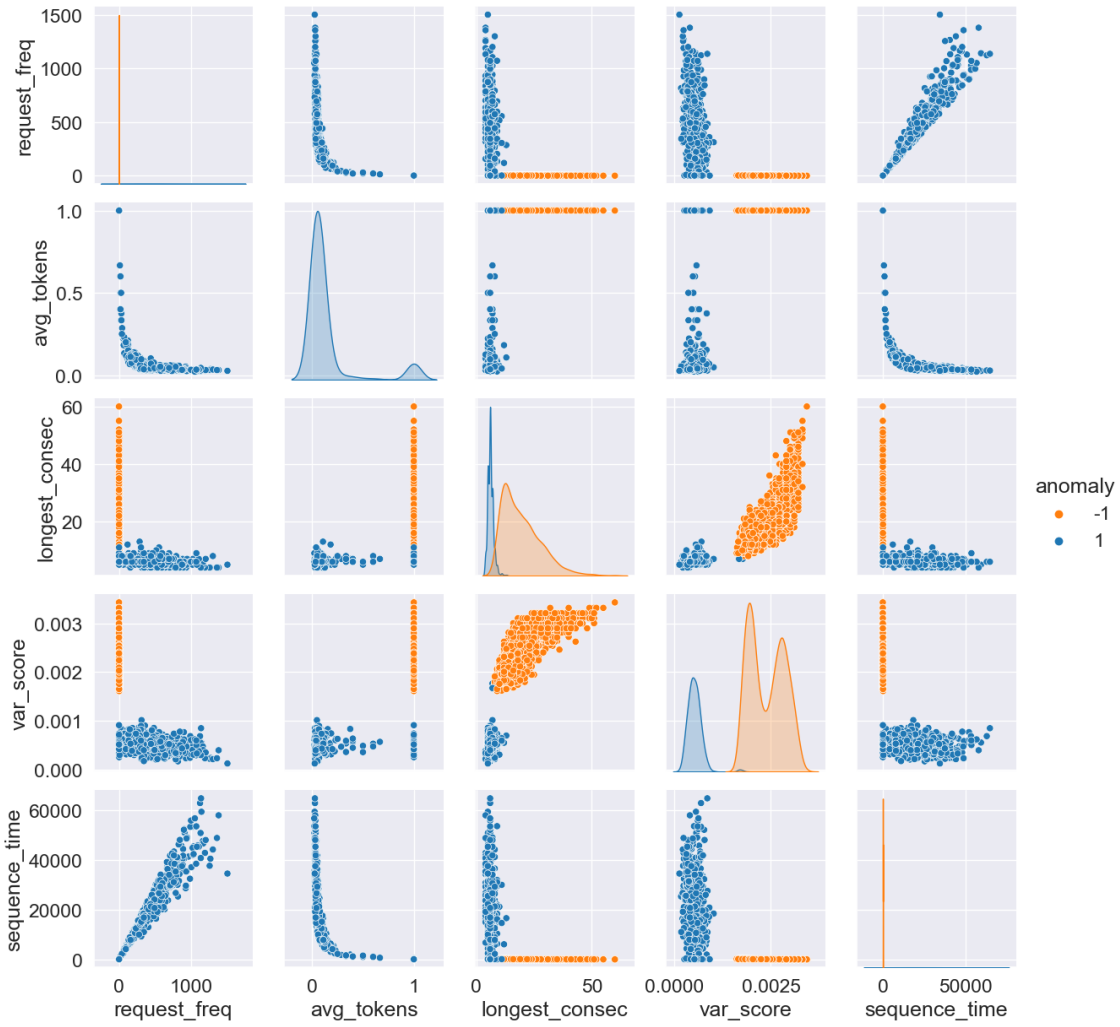
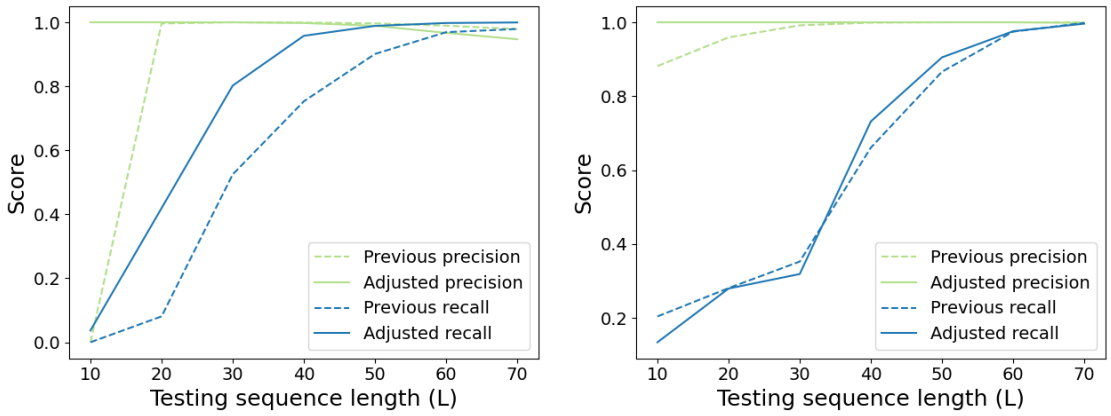


Figure 7.6: The resulting clusters from the 60-model with $L = 60$ for all features.

It is clearly evident from the visualization in Figure 7.6 that the features creating the biggest divide between normal and abnormal users are *longest_consec* and *var_score*. From this information, the 30- and 60- models were retrained and tested on data that includes only these two features in order to examine how this would affect the performance. Figure 7.7 shows the performance scores from this experiment.



(a) 30-model **(b)** 60-model

Figure 7.7: A comparison of the performance of the 30- and 60-model with adjusted features (*longest_consec* and *var_score*).

When including only the features *longest_consec* and *var_score*, it is observed that both the 30-model and 60-model receive a precision score close to 100% in the sequence length interval from 10 to 40, which is an overall improvement. Additionally, a significant improvement in the recall is observed for the 30-model. However, after a sequence length of 40 the adjusted 30-model's precision started to decrease at a higher rate than previously. For the 60-model, a slight improvement in the recall can be observed from sequences of 30 requests and longer. However, this improvement is not as significant as for the 30-model.

A conclusion can be made that when the system operates on sequences between 10 and 40, the 30-model produces enough satisfying precision and recall scores that it could be sufficient enough on its own.

7.5 Summary

The aim of this project has been to create an ML module that will be used as a component in a larger system to detect anomalies. One of the key requirements for the ML system is that the model should be stable and consistent regardless of sequence length. This includes being able to detect anomalies with as high precision as possible throughout the entire range of sequence length in order to not cause disruption for normal users. Although a high recall score is important to achieve, it should not be done at the expense of a good precision score.

In this chapter, an extensive comparison between models trained on different testing sequence lengths *(L)* has been presented. This resulted in a conclusion that a model trained on sequences of 30 requests offered the best result regarding precision when evaluating shorter sequences, while a model trained on sequences of 60 requests had the best precision performance on longer sequences. This indicates that based on the data available, the best overall performance regarding precision would be achieved by using two different models depending on the length of the incoming sequence.

However, as illustrated in Figure 7.7, another solution emerges when adjusting the features to only include *longest_consec* and *var_score*. This graph suggests that for sequences between 10 and 40, the 30-model is sufficient enough to provide both great precision and a satisfying recall. This could be a preferred approach since there is great value in the ability to detect abnormal behaviour early after relatively few requests have been made. Both for the purpose of blocking unwanted users early as well as making each sequence less computationally heavy to analyze. For the combined 30- and 60-model approach, the interval between 10 and 20 requests is as stated not taken into consideration due to the low performance, but instead, the interval stretches to 70 requests.

We can conclude that using an IF model and the features extracted from the available synthetic data can be a promising approach regarding anomaly detection. A fact that is important to note however is that since the data used for this project have been completely synthetic, it is possible that for real-life users the difference between normal and anomalous behaviour is smaller, that the most important differences lie in different areas, or that the features do not provide enough information to make a definitive prediction. As such, any model trained on real-life data might yield a different result regarding what features are the most important which in turn affect what to include as input. For this reason, the decision was made to not exclude any of the features for the final implementation, and instead use the solution utilizing two different models since that would make the solution more general and less tailored to the available data.

From the tests conducted it was found that although a satisfactory precision score was achieved for the entire interval using two models, it was slightly lower for shorter sequences using the 30-model. This means this approach still has some issues with precision for the shorter sequences. For this reason the decision was made that user flagged as anomalies by the 30-model would not be blocked immediately, but instead have to pass a verification test as shown in Figure 4.1.

8
Conclusion

The aim of this project has been to create an ML system that could differentiate between normal and abnormal user behavior through request analysis when using an application or service. The system has been implemented with the Isolation Forest model, which operates by separating outliers in the data. Feature engineering has been implemented and applied to the data in order to extract information regarding user requests.

The evaluation has been made using the metrics *Precision*, *Recall*, and *F1 score*. The result has suggested the implementation of a system that combines two models, the 30-model and the 60-model, which operate on different sequence lengths: 20 to 40 and 40 to 70. This solution offered the best result regarding precision score while also offering an acceptable recall when involving all features extracted in the pre-processing step.

Using this solution for analyzing sequences from 20 to 70 requests, we have achieved a precision score close to 100% for the entire interval. Regarding recall, the 30-model has obtained a score between 8% and 75% whereas the 60-model obtained a score ranging between 66% and 100%. Both models have achieved a better result as the sequence length increases as long as they stay within their respective intervals.

The project has progressed more or less as described in Chapter 2. Some adjustments have been made during implementation due to added limitations, but apart from that the overall objective (Section 1.2) has been achieved. Throughout the project, the outlined timeline has been kept consistent.

8.1 Future Work

Previous works [15, 21, 22] show a wide range of orthogonal solutions regarding anomaly detection. The examples referred to in Chapter 3 along with multiple other research and articles that we have come across, demonstrate the wide range of solutions that can be applied to this kind of problem.

Finding the optimal solution for this specific application would require more time and resources than what has been available during this project. For this reason, future work could involve a comparison with other ML models and approaches. In particular, an approach using LSTM Autoencoder since research shows promising

results regarding this model's performance for anomaly detection purposes. Another comparison aspect could involve an analysis of the time complexity and overall efficiency between different approaches.

Since this project is only based on the metadata from user requests, another potential direction could be a more comprehensive implementation that includes an analysis of the contents of a post or message. Finally, an evaluation of the current approach using real-life data could establish more legitimate results. This could follow in a more sophisticated analysis and perhaps support the ML model for quicker and easier spam detection.

Bibliography

[1] Dave DeBarr, Harry Wechsler, "Spam detection using Random Boost", *Pattern Recognition Letters*, Volume 33, Issue 10, 2012, Pages 1237-1244, ISSN 0167-8655, https://doi.org/10.1016/j.patrec.2012.03.012.

[2] Maurya, S.K., Singh, D. & Maurya, "A.K. Deceptive opinion spam detection approaches: a literature survey". Appl Intell 53, 2189–2234 (2023). https://doi.org/10.1007/s10489-022-03427-1

[3] Ghanem, R., Erbay, H. "Spam detection on social networks using deep contextualized word representation." Multimed Tools Appl 82, 3697–3712 (2023). https://doi.org/10.1007/s11042-022-13397-8

[4] V. Hagenbo, L. Rosin, L. Bagiu, E. Berzelius, C. von Schenck, E. Näslund, "Gateway Request Analyzer", Chalmers University of Technology, Göteborg, Sweden, Jan. 5, 2023. Accessed: Jan. 16, 2023.

[5] M. Kolomeets and A. Chechulin, "Analysis of the Malicious Bots Market", 2021 29th Conference of Open Innovations Association (FRUCT), Tampere, Finland, 2021, pp. 199-205, doi: 10.23919/FRUCT52173.2021.9435421.

[6] L. Das, L. Ahuja and A. Pandey, "Analysis of Twitter Spam Detection Using Machine Learning Approach", 2022 3rd International Conference on Intelligent Engineering and Management (ICIEM), London, United Kingdom, 2022, pp. 764-769, doi: 10.1109/ICIEM54221.2022.9853100.

[7] Kiliroor, C.C., Valliyammai, C. (2019). "Social Context Based Naive Bayes Filtering of Spam Messages from Online Social Networks." In: Nayak, J., Abraham, A., Krishna, B., Chandra Sekhar, G., Das, A. (eds) Soft Computing in Data Analytics . Advances in Intelligent Systems and Computing, vol 758. Springer, Singapore. https://doi.org/10.1007/978-981-13-0514-6_66//

[8] Abu-Salih, B., Qudah, D. A., Al-Hassan, M., Ghafari, S. M., Issa, T., Aljarah, I., Beheshti, A., & Alqahtani, S. (2022). An intelligent system for multi-topic

social spam detection in microblogging. Journal of Information Science, 0(0). https://doi.org/10.1177/01655515221124062

[9] Machine Learning, A First Course for Engineers and Scientists (Applied Maskininlärningsboken)

[10] K. A. Al-Thelaya, T. S. Al-Nethary and E. Y. Ramadan, "Social Networks Spam Detection Using Graph-Based Features Analysis and Sequence of Interactions Between Users", 2020 IEEE International Conference on Informatics, IoT, and Enabling Technologies (ICIoT), Doha, Qatar, 2020, pp. 206-211, doi: 10.1109/ICIoT48696.2020.9089509.

[11] Skiena, S.S. (2017). Machine Learning. In: *The Data Science Design Manual. Texts in Computer Science*. Springer, Cham. https://doi.org/10.1007/978-3-319-55444-0_11

[12] Skiena, S.S. (2017). Distance and Network Methods. In: *The Data Science Design Manual. Texts in Computer Science*. Springer, Cham. https://doi.org/10.1007/978-3-319-55444-0_10

[13] M. Washha, A. Qaroush, M. Mezghani, F. Sedes, Unsupervised collective-based framework for dynamic retraining of supervised real-time spam tweets detection model, Expert Systems with Applications, Volume 135, 2019, Pages 129-152, ISSN 0957-4174, https://doi.org/10.1016/j.eswa.2019.05.052.

[14] F.T. Liu, K.M. Ting, and Z. Zhuo, "Isolation forest", presented at the IEEE International Conference on Data Mining 2008 (ICDM 08), Pisa, Italy, Dec. 15-19, 2008.

[15] Z. Ding, M. Fei, "An Anomaly Detection Approach Based on Isolation Forest Algorithm for Streaming Data using Sliding Window", presented at 3rd IFAC International Conference on Intelligent Control and Automation Science, Chengdu, China, Sept. 2-4, 2013, doi:https://doi.org/10.3182/20130902-3-CN-3020.00044.

[16] Z.C. Lipton, J. Berkowitz, C. Elkan, "A Critical Review of Recurrent Neural Networks for Sequence Learning", University of California San Diego, San Diego, California, USA, Jun. 5, 2015. Accessed: apr. 18, 2023. [Online]. Available: https://arxiv.org/pdf/1506.00019v4.pdf

[17] IBM, “Recurrent Neural Networks”, IBM, 2018. Accessed: Apr. 18, 2023. [Online]. Available: https://www.ibm.com/topics/recurrent-neural-networks

[18] C. Wang, “The Vanishing Gradient Problem The Problem, Its Causes, Its Significance, and Its Solutions”, Towards Data Science, Jan. 8, 2019. Acessed: Apr. 18, 2023. [Online]. Available: https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484

[19] S. Hochreiter, J. Schmidhuber. “Long short-term memory”. Neural Computation, 9(8):1735–1780, 1997. Accessed: May. 30, 2023. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[20] A. Gensler, J.Henze, B. Sick, N.Raabe, “Deep Learning for Solar Power Forecasting – An Approach Using Autoencoder and LSTM Neural Networks”, 2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC 2016), Budapest, Hungary, Oct. 9-12, 2016.

[21] Y. Wei, J. Jang-Jaccard, W. Xu, S. Fariza, S. Camtepe, M. Boulic, “LSTM-Autoencoder based Anomaly Detection for Indoor Air Quality Time Series Data”, Massey University, Auckland, New Zealand, Central Queensland University, Sydney, Australia, Apr. 14, 2022. Accessed: Apr. 27, 2023. [Online]. Available: https://arxiv.org/pdf/2204.06701.pdf

[22] C. Y. Priyanto, H. D. Purnomo, Hendry Faculty of Information and Technology, “Combination of Isolation Forest and LSTM Autoencoder for Anomaly Detection”, presented at 2nd ICITech, Salatiga, Indonesia, Sept. 23-25, 2021, doi:10.1109/ICITech50181.2021.9590143.

[23] Scikit-learn, “The Bag of Words representation”. Accessed: 2023-04-26. [Online]. Available: https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction

[24] Scikit-learn, “Common Vectorizer usage”. Accessed: 2023-04-26. [Online]. Available: https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction

[25] Scikit-learn, “Tf-idf term weighting”. Accessed: 2023-04-26. [Online]. Available: https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction

[26] Scikit-learn, “Vectorizing a large text corpus with the hashing trick”. Accessed: 2023-04-26. [Online]. Available: https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction

[27] Gao, H., Hu, J. Wilson, C. Li, Z. Chen, Y. Zhao, B.Y. “Detecting and Characterizing Social Spam Campaigns”. Northwestern University, Evanston, IL, USA, U.C. Santa Barbara, Santa Barbara, CA, USA, Huazhong Univ. of Sci. & Tech. 2010. [Online]. Available: https://people.cs.uchicago.edu/ ravenben/publications/pdf/fbspam-imc10.pdf [Accessed: Mar. 31, 2023]

[28] L. Chen, D. Feng, Z. Shi and F. Zhou, “Using Session Identifiers as Authentication Tokens”, 2009 IEEE International Conference on Communications, Dresden, Germany, 2009, pp. 1-5, doi: 10.1109/ICC.2009.5199560. 2023-03-30

[29] Python “General Python FAQ”, Accessed: 2023-03-29. [Online]. Available: https://docs.python.org/3/faq/general.html#what-is-python, 2023-03-29

[30] Pedregosa et al., “Scikit-learn: Machine Learning in Python”, JMLR 12, pp. 2825-2830, 2011. [Online]. Available: https://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html. 2023-03-29

[31] Numpy, “What is NumPy?”, [Online]. 2023-03-29. Available: https://numpy.org/doc/stable/user/whatisnumpy.html.

[32] McKinney, “Data structures for statistical computing in python”, Proceedings of the 9th Python in Science Conference, Volume 445, 2010. [Online]. Available: https://conference.scipy.org/proceedings/scipy2010/pdfs/mckinney.pdf

[33] The Pallets Projects, “Flask - web development, one drop at a time”. [Online]. Available: https://flask.palletsprojects.com/en/2.2.x/. Accessed: 2023-04-12

[34] docker, “What is a Container?”. Accessed: 2023-05-30. [Online]. Available: https://www.docker.com/resources/what-container/

[35] A. Lindholm, N. Wahlström, F. Lindsten, and T. B. Schön, “MACHINE LEARNING: A First Course for Engineers and Scientists”, Cambridge University Press, 2018. Accessed: Apr.10, 2023. [Online]. Available: http://smlbook.org/

[36] A. Singh, “Precision-Recall in Machine Learning”, Analytics Vidhya, Sep. 2020. [Online]. Available: https://www.analyticsvidhya.com/blog/2020/09/precision-recall-machine-learning/. [Accessed: Apr. 13, 2023]. https://www.analyticsvidhya.com/blog/2020/09/precision-recall-machine-learning/

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS