DEGREE PROJECT IN TECHNOLOGY,
FIRST CYCLE, 15 CREDITS
STOCKHOLM, SWEDEN 2022

Scraping bot detection using machine learning

Hamta Dezfoli and Joseph Newman

KTH ROYAL INSTITUTE OF TECHNOLOGY
ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Title

English: Scraping bot detection using machine learning
Svenska: Botdetektering med hjälp av maskininlärning

Authors

Hamta Dezfoli <hamta@kth.se>
Joseph Newman <jhnewman@kth.se>
Information and Communication Technology
KTH Royal Institute of Technology

Place for Project

Stockholm, Sweden

Examiner

Matthias Becker
KTH Royal Institute of Technology

Supervisor

Fadil Galjic
KTH Royal Institute of Technology

János Tóth-Égetö
The Mobile Life

Host organisation

The Mobile Life AB

Abstract

Illegitimate acquisition and use of data is a problematic issue faced by many organizations operating web servers on the internet today. Despite frameworks of rules to prevent ”scraping bots” from carrying out this action, they have developed advanced methods to continue taking data. Following research into what the problem is and how it can be handled, this report identifies and evaluates how machine learning can be used to detect bots. Since developing and testing a machine learning solution proved difficult, an alternative solution was also developed aiming to polarize (separate) bot and human traffic through behavioral analysis. This particular solution to optimize traffic session classification is presented and discussed, as well as, other key findings which can help in detecting and preventing these unwanted visitors.

Keywords

Artificial agents, Bot detection, Machine learning, Data analysis, HTTP requests, ReCaptcha

Abstrakt

Olaglig insamling och användning av data är problematiskt för många organisationer som idag använder sig av webbservrar på internet. Trots ramar av regler för att förhindra ”scraping bots” så har de utvecklat avancerade sätt att komma åt data. Efter forskning om vad problemet är och hur det kan hanteras, identifierar och evaluerar denna rapport hur maskininlärning kan användas för att detektera bottar. Då utvecklingen och testningen av en lösning med hjälp av maskininlärning visade sig bli svårt, utvecklades en alternativ lösning med målet att polarisera (separera) bottrafik och legitim trafik. Denna lösning presenteras och diskuteras i rapporten tillsammans med andra nyckelresultat som kan hjälpa till att upptäcka och förhindra dessa oönskade besökare.

Nyckelord

Artificiella agenter, Detektering av bottar, Maskininlärning, Dataanalys, HTTP förfrågningar, ReCaptcha

Acknowledgements

We would like to thank our advisors at KTH Royal Institute of Technology, our supervisor Fadil Galjic and examiner Matthias Becker for providing us with valuable insights and feedback through the degree project. We would also like to thank our company supervisor at The Mobile Life, János Tóth-Égetö for his guidance and help during the project.

List of acronyms and abbreviations

ML Machine Learning

TML The Mobile Life

API Application Programming Interface

WAF Web Application Firewall

HTTP Hypertext Transfer Protocol

IP Internet Protocol Address

AI Artificial Intelligence

URL Uniform Resource Locator

BSD Berkeley Software Distribution

ISP Internet Service Provider

VPN Virtual Private Network

Contents

1 Introduction

This report analyzes different aspects of machine learning algorithms within the area of web traffic analysis based on artificial agents. Within the field of machine learning, there is a topic of categorizing a given set of data into classes. This classification can be performed on both structured or unstructured data. An example is classification of email-messages. A message can be mapped in spam or non-spam. In this research, the work is intended to solve a similar problem with detecting human or non-human actors.

1.1 Background

The following background information was provided by The Mobile Life (TML) in the thesis problem description and through discussions with the project supervisor at the company.

TML builds mobile applications and platforms for a variety of customers. One of the largest business areas is mobile applications and middleware Application Programming Interfaces (APIs) for the airline industry, which support, among other things, booking systems. The security aspect is important in this field, since web traffic is increasingly dominated by bots, which is another name for artificial agents. These artificial agents might pose threats to the industry in terms of security, privacy and performance, amongst other things.

The mentioned APIs serve as an interface between customers using an application and a large industry-wide shared database, which holds information about flight availability, prices and tickets, amongst other things. Using an application, a customer may book tickets or check in for a flight, as an example. A key problem which has arisen in connection with this is that the APIs are frequently targeted by bad players for various illegitimate purposes. Here are some examples:

- Access to information without partnership and authorization
- Presentation on third party websites

- API used in order to conduct fraud, e.g. book flights at a reduced price by exploiting the API and using it in a malicious way

Due to misuse, there is an urgent need to detect and subsequently block malicious sources, which are costly and might pose a security threat. A common method of protecting an API and blocking unwanted traffic, is to use Web Application Firewall (WAF) rules, which can be attained by blacklisting or whitelisting Internet Protocol Address (IP), applying rate limiting and blocking suspicious user-agents, amongst other things. The main issue with this approach is that bots could get around static rules by changing Hypertext Transfer Protocol (HTTP) headings or spreading the traffic on many IPs, amongst other things.

Another issue is that systems must be monitored by humans, which is not effective in terms of automation. Therefore, TML suggests using Machine Learning (ML) in order to analyse incoming requests to categorize the source. A given request will be labelled as bot or human. The solution should take in various inputs about an incoming request and then output a prediction of its source, such as 70% likely human and in that case 30% likely to be from a bot source. This output can in turn be used to determine how the request is handled by the server.

An example to illustrate the problem and how it can be detected through http-request analysis is as follows. Two common request types can be categorized as availability and price calls. Human users typically request these in a 50/50 ratio, however with bot traffic 99.5% of calls are availability. Knowledge of this information can thus be used in a situation when a given source is requesting only one type of data, then there is a likelihood this is coming from a bot.

TML is also aware that the API that will be developed gets at least 1.5M calls, from a total traffic of 2.5M calls on a daily basis from confirmed bots, approximately 60%. This research aims to find effective methods to help in detecting and preventing these unwanted requests and to prevent accessing information.

1.2 Problem

Scraping technologies intend to take valuable data from websites and databases without permission. Since this information can be subsequently misused in a variety of ways, organisations seek ways of detecting and blocking bots and aim to only answer requests from legitimate human users. This thesis is focused on investigation of web scraping bots and how to effectively detect them.

The research will be focused on ML techniques, asking if these can be applied effectively in the bot detection process and make a significant contribution to protecting data from unauthorized traffic.

Research question

The central research question of this report is as follows:

Can scraping bot traffic be detected and blocked with the help of machine learning?

In order to address the central question, this thesis will attempt to answer the following sub-questions:

- Can bot traffic be detected effectively?
- If so how can it be detected?
- Which solution(s) is/are suitable to implement or focus future work on at TML in handling issues with bots?

1.3 Purpose

The purpose is to facilitate automation of traffic classification by creating an implementation that can be used by TML to avoid fraud by distinguishing bots from humans. Actors whom might find this thesis useful are TML and Radware, where the latter is a provider of cybersecurity and application delivery products for software-defined datacenters. Radware might find the study useful, since they have an ongoing work of implementation in the field.

1.4 Goal

The goal is to research and evaluate ML-methods for bot detection in line with the central research question, and its sub-questions. The aim is to research solution options and furthermore to develop, implement and test a model, which analyses real time incoming traffic to a server and outputs the probability of requests as being from a bot or human source.

The ideal solution can be automated and self updating, i.e. not a static framework of rules which bots are developed to find ways around. The aim is to find an automated solution to detect and block bot traffic, that adapts to the traffic at the same speed as the bots themselves change behavior.

1.5 Methodology

Literature study is the first approach in this research in order to understand how to analyze and answer the main research question. The purpose of literature study will be to analyze and compare prior studies and theoretical articles within the given area, which may be helpful in solving the given problem.

The second approach in the research is the usage of quantitative experimental studies in order to test the data collected from The Mobile Life and use of internal tools in order to compare, find similarities and conclude with a solution.

Some important parts of how the methodology was carried out practically are listed below.

- Understanding the problem and the specific data types to be processed, including getting data from the company, querying what the problem is and how it can be solved.
- Data analysis to get a clear picture of how the problem can be solved.
- Identify possible solutions, different algorithms in machine learning and research which solutions exist for similar problems. Finally, identify the method or methods to be used.

- Implement some basic models and test the effectiveness of each approach.
- Carry out tests and, if necessary improve the solution, or solve additional interconnected problems if these exist.

1.6 Delimitations

Problems in connection with scraping bots is a broad issue in terms of applications that can be attacked, and also advanced in terms of the resources spent developing effective bots. Aspects of general theory related to bot management, with solutions presented that are limited to specific cases which the company TML are facing, will be presented in this theses. Two important delimiting factors of this project are presented below.

1.6.1 Long-term effectiveness of solutions

In this report, experiments and models will be based on using ML and analysing if it can be an effective tool. Long-term effectiveness of any suggested solution cannot be guaranteed. Detecting scraping bots is inherently a continuous and ongoing problem, due to adaptations from organizations to prevent them and find ways around adaptions made by bots, it is by nature a game with moving goalposts. As such, whilst the aim is to make a quantifiable improvement in the detection, it is impossible to know how long such a solution will remain relevant or effective, or if adaptations can be made to overcome any implementation.

The intention is to find, through theory and experimentation, and to subsequently implement, a working model with which a measurable and significant proportion of bots will be detected. Since it is impossible to know what may be developed in the future, focus will be on the current state of the problem.

1.6.2 Sensitivity in handling HTTP requests

Handling requests is a further delimiting factor which requires great care, since any request can be from a genuine human user. If an algorithm makes an incorrect prediction and handles a person as a bot, it may lead to a very poor user experience. This is a particular concern for the companies which use this

software, since mistakes like this can lead to damage to their brand reputation and fewer customers. Therefore, a high level of certainty is vital in the prediction that something being detected as a bot really is one. Some less certain cases may be needed to give benefit of the doubt that traffic is from a real user, even if suspicious. In connection with this, there may be a limit in the ability to test bots, which is one commonly used approach to deal with this problem.

1.6.3 Dataset limitations

The type of dataset which can be obtained for the project will likely have a large effect on possible experiments and outcomes. In order to produce an useful solution, a large and processed dataset is required. Results will be produced by using generic data sets if a processed dataset is not provided. The outcome will in this case be theoretically useful within the field.

1.7 Ethics and Sustainability

Artificial Intelligence (AI) is important for both business and society. Economic benefits, higher levels of efficiency and productivity are important for many industries. Many challenges can be solved by AI, which means that morally and ethically challenges also might arise since privacy and data protection are important for a secure solution. Since it is intended that this thesis is based upon real data, it should be collected and maintained in a secure way.

Questions such as how data is collected, stored, used, encrypted and protected must be asked and addressed at every step where appropriate. As mentioned in the workshop report by The International Risk Governance Center [3], since AI has the ability to analyse quantities and large sources of data, far beyond what humans are capable of, means that doing so must be done responsibly.

AI can link data, find patterns and is also more consistent than humans, and have the possibility to change inputs and adapt effective solutions faster compared to human actors based on various inputs. Therefore, it is beneficial due to the fact that one can reduce complex problems with large data sets, or tasks that might be

repetitive by using AI. As an example, one can reduce commuting times or increase effectiveness of email spam filters which can make life a lot more easier.

1.8 Outline

The thesis has the following outline.

Chapter 2 presents the theoretical background. Chapter 3 presents the methods and research approach for the thesis. Chapter 4 covers the practical description of how the different methods were applied. Chapter 5 presents the results of the degree project. Chapter 6 presents the conclusions, effects, evaluation of the results and recommended focus for future work.

2 Theoretical Background

This chapter introduces relevant theory that might be helpful in order to understand how to answer the research question. Areas such as supervised and unsupervised learning, and ML from a mathematical point of view are presented in this chapter. Related work is also presented in order to get a broad insight of the theory in the research field.

2.1 Automation of web extraction

This section describes automation of web extraction, i.e. utilizing of bots in order to automate the process of web extraction from a selected websites, and storing the extraction for further usage.

2.1.1 Web scraping

Web scraping is a method used in order to extract information from a web page, with other words web data extraction. Web scraping is used in many cases, such as price monitoring, news monitoring and market research, amongst other things [31]. The benefit of using web scraping is that it gives structured web data when extracting content from any public website. The figure below shows the architecture of web scraping.

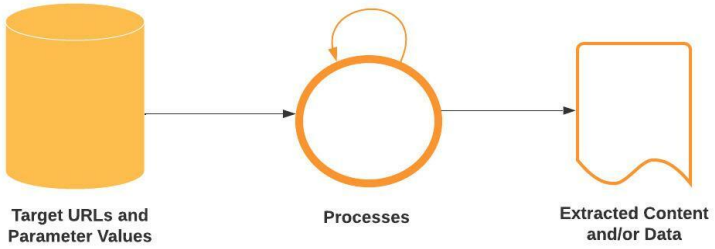

Figure 2.1: Web scraping architecture.

The target Uniform Resource Locators (URLs) and parameter values found in websites get extracted with a web scraping tool. This process is done in order to gather data in a systematic manner from servers. Once taken, the data is transformed into structured data. This can be used for any number of subsequent purposes including fraud or being integrated into business processes to analyze

and business decisions exploiting the illegitimately gathered information [31]. Examples would be to track prices of competing companies within a certain market, such as a clothing web shop, or to collect real estate data.

2.1.2 Web crawling

A web crawler, also called a search engine bot, is a bot which uses automated information gathering to map web page structures on the Internet. Web crawling is therefore the process where web pages are catalogued from the internet. A key difference from scraping is that the data itself is not copied, instead crawlers extract hyperlinks in these web pages by downloading and parsing. This process maps out which resources are available and where they on a server. Web pages are associated with a certain URL [6].

These URLs are sent to a queue and scheduled in order to be indexed. Once the indexation is done, they are sent to a database or some form of file storage, which makes it possible for search engines such as Google to list associated web pages [6]. The figure below shows the architecture of how a web crawler operates.

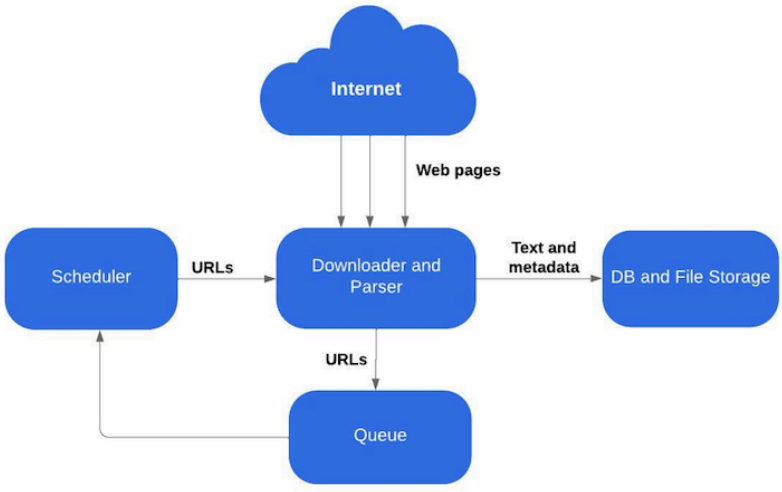

Figure 2.2: Architecture of a web crawler.

A web crawler performs these actions without any human intervention. According to a report regarding 2021 made by Imperva, approximately 41% of the internet traffic was not human, as human traffic decreased with almost 6% [14].

2.1.3 Bot mitigation techniques

There are different approaches in order to mitigate bots. These approaches could be combined in order to have a more secure and a separated traffic between bots and human actors. One common approach used in order to obtain protection against bots is the static approach. The static approach concerns static analysis and tools which can be used to identify web requests and header information of malicious bots and determine the identity of the bot, and perhaps blocking it if needed. Another approach is the challenge-based approach, where solutions such as CAPTCHA or reCAPTCHA are included.

A CAPTCHA is a web authentication test designed to proactively identify the source of incoming traffic, to see if the traffic originates from bots or human actors. When bots have developed better solutions regarding solving CAPTCHAs, Google developed a reCAPTCHA system using ML and AI in order to have a more secure and protected system against malicious bots [8].

Another bot mitigation technique is the behaviour-based approach. This approach concerns advanced bots which adapts to human mechanism and behaviour, such as using behavioral pattern of a human user. If a web user differs in some way from this pattern and behaviour, this user is flagged in order to check the reliability and accuracy of the user [9].

2.2 Machine learning

ML is a method of data analysis and a branch of artificial intelligence concerning systems that can learn from data, identify similar patterns between data and make decisions without any human actor involved or using minimal human intervention. The purpose is to train machines to recognize patterns in an automated way, without being explicitly programmed to do certain tasks.

ML is used in our daily lives, in areas such as self-driving cars, online recommendations received by sites such as Netflix or Amazon, based on our previous search history, and fraud detection such as financial law enforcement [26]. ML can be broken down into two key subjects, presented next.

Supervised learning

Supervised learning is where a set of input data and corresponding output results are analysed and used to predict output for future inputs [21]. A simple example of this can be looking at historic house prices depending on the square metre area. By processing these data a model can be made, and then used to predict a price given an input area. Within supervised learning, methods such as linear regression and logistic regression are included.

Linear regression

In a linear regression, a model is created to take an input and generate a continuous output value. The algorithm tries to find the best fitting model for the input data.

Linear regression aims to model the existing relationship between two variables, by attempting to find a suitable linear equation for the data which is under observation. The equation can be described as below, where the variable X represents an explanatory variable and Y is the dependent variable [24].

$$Y = a + bX$$

The figure 2.3 shows an example of different house prices in USD, dependent on the area which is shown per square metre. It can be seen how a relationship exists between the size of a property and its price. For a future estimation this model would, given an area input, output a price on the trend-line corresponding to that input value. Naturally, linear regression can be used to process more complex and multi variable data sets, but the process remains the same.

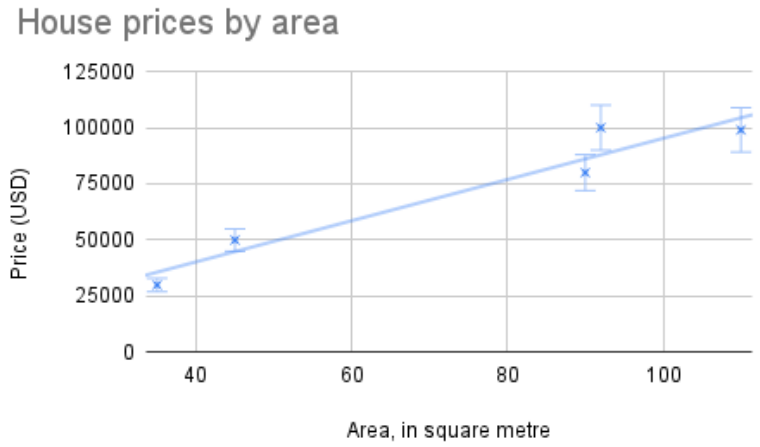

Figure 2.3: An example of house prices determined by area. The figure is made by the authors as an illustration of how ML cost calculation operates.

The model equation is assessed and seeks to minimize the ”cost” or difference between the model and the training set data. This cost is equivalent to the sum of the distances between each point on the graph and the model line. Making adjustments to such a model can be done using software or manually to help create robust and accurate predictions.

In ML this is achieved though incremental improvement during the training phase where data analysis is performed. To increase accuracy a variable may be used multiple times with different exponential powers and is factored or ’weighted’ to better fit the data and minimize the cost function [20]. In the equation below this would be achieved by adjusting the values a, b, c and so on, until the cost function is at its minimum [24].

$$Y = a + bX + cX^2 + dX^3 + ...$$

Logistic regression

Logistic regression is a ML-algorithm used for classification problems. It works similarly to linear regression, with the key difference that rather than calculating a continuous output, it outputs the probability of the output belonging to one of a discrete number of categories [18]. For example, the algorithm may output the likelihood of a patient having an illness or not, given a previous set of diagnoses corresponding to some input.

In the case of this thesis, that would be the probability of a HTTP-request coming from a human or a bot. Data is processed similarly here because the aim is also to minimize the cost function and find the best fitting model to the training set, in order to produce as accurate predictions for future inputs as possible. Models can contain one variable or multiple variables, with various powers/exponential values as a part of these terms, depending on the data and the problem being handled [7] and [18].

Unsupervised learning

Unsupervised learning models take a training set with only input data and through observation and analysis seek to find meaningful structures in the data [13]. These structures are commonly referred to as clusters. This differs from supervised machine learning, where training sets have both an input and output and where the aim is to predict future outputs for a given input.

A simple example of unsupervised learning could be taking a data set with the height and weight of a number of people and from this seeing if there exists a correlation, and if the data can be organised into groups according to some criteria. This may in turn be used to, for instance, recommend t-shirt sizes if a person in the data is organised into a given cluster.

Since unsupervised learning can be effective in recognizing patterns and similarities in data [13], it is applied in solutions such as recommending content to users having observed patterns of previous consumption and interest [26]. In this thesis, this can potentially be applied to observe if incoming HTTP-requests can be organized into various groupings, which can in turn be analyzed to see if there is correlation between a request coming from a specific cluster also being from a bot source. This combination approach, which initially runs a clustering algorithm and subsequently, uses clusters as an input for a classification algorithm, has been shown to produce highly accurate results in detecting bots [25].

2.3 Incoming requests analysis

This section seeks to systematically list and define various inputs extracted from HTTP traffic data and analysis methods, in order to determine if a requesting source is coming from a legitimate user or a potentially malicious source.

2.3.1 Basic inputs

Within a HTTP request header there exist various fields of information, which can be checked as a means of detection. The header contains a user agent, which is the browser or program supporting the user to send requests. It also contains an operating system of the client. More common contents in the header are the requested resource or page, the datatype which was requested or accepted by the client browser, timestamp, IP address and the connection or transfer protocols [29]. With this information various simple rules may be created to help detect bots. These can be the basic structure of web firewalls and will be used as a starting point in this project before improvements are made.

Examples of such rules are:

- The number of request calls by the client
- Frequency of the calls and possibilities of human actors
- Has the total number of calls or time since initial session call exceeded a boundary?
- Is the IP address trusted or blacklisted?

- Is the user agent trusted or blocked? Legitimate users are likely using one of a number of trusted browsers, such as Google Chrome, Safari or a trusted app interface. Bots however may use computer terminals for example.

2.3.2 Advanced input analysis

Over a period of time a database can be built up, storing information about incoming requests to a server and how they are answered. This allows for deeper analysis of client behavior, which can increase the effectiveness in detection. Advanced rules, which this project may evaluate and implement, include:

- Resource request patterns. As identified by Doran et al. [10], this can be an accurate and efficient method of identifying bot traffic. Legitimate users have roughly a 50/50 ratio of calls checking price vs availability, while bot traffic is known to consist of 99% availability requests. This, and similar analytical observations can be used to make a rule framework for bot detection. Another observed pattern is based on the size of requested resources, and the cumulative total data size requested during a session.
- Analysis of a client path into the server [2]. Here a comparison can be done using knowledge of how user interaction with the server should be. For instance, legitimate user should follow a path such as login/home/dates/specific request, whereas a bot with prior knowledge of the server structure may directly send specific requests for the resource which could lead to failure of such a test.

2.3.3 Automated honeypot

A honeypot works as a trap, in order to catch malicious or unauthorized actors. There are different types of honeypots, one suitable solution that could fit the purpose and requirements of TML, would be the spider honeypot. A spider honeypot has the intention of catching webcrawlers, in this case called spiders, by creating web pages and other links that are not accessible in the intended use of the server where links must be clicked on to navigate an application and access various resources. Thus, since only accessible to bots attempting to bypass the normal means of navigation a server, it can be assumed that any visitor to such

a resource is a bot. This might be an effective way of trapping and generating metrics to help detect malicious web traffic in the airline industry [16].

2.4 Related work

Prior to the thesis and within the field of bot detection, related works have been researched more thoroughly as a part of the literature study.

2.4.1 Bot recognition

Web traffic can reveal a tremendous amount of properties separating humans from bots, although some bots have developed sophistical methods in order to imitate legitimate users. There are relatively few studies in specifically classifying user sessions based on supervised learning. Most studies instead aim to investigate the partition of humans and bots into different clusters applying unsupervised learning. This cluster-based approach is documented in both Rovetta et al. [25], and Zabihimayvan et al. [19].

2.4.2 Contrasting web robot and human behaviours

Brown and Doran [2] describe an approach comparing the activity of humans and web crawlers with help of session graphs. Some of the conclusions of the differences between human and web robot traffic were the following:

- An observed transition in one direction, is likely not possible to observe the transition from the other direction, due to the hyperlink structure of the given website.
- The result shows that it is more common for bots to transit between resources, which are not directly connected by hyperlinks.

Simply put, the manner in which a user moves from one path in a server to another should follow predictable navigation through server paths determined by the links available in an application or on a website. Analysis of this movement can help determine if this is the case or not and subsequently conclusions can be drawn on the source of the session traffic.

2.4.3 Behavioral analysis

Other work has been carried out in the area of behavioral analysis. Pozzana and Ferrara [22] have performed research where metrics can be used to analyse behavior, and how to quantify and apply them. Their work centred on session analysis, in order to find a separation in patterns caused by bot automation in comparison with human behavior. Garcia [12] notes how static methods of detection and prevention such as static features, predefined rules and blacklists are outdated and no longer effective in handling modern complex bots. Garcia documents a number of technologies and alternative solutions which can be implemented to provide effective results.

3 Research Methodology

This chapter provides details of the methodological framework used to carry out work in order to answer the research question. An overview of methods is given and particular steps are described in detail. Each subsection in this chapter provides information related to some method, its structure and key activities.

3.1 Literature study

The first phase in the project was to research previous work related to the research area. This was in order to find existing solutions and ideas, which may help in producing useful results and helpful in order to answer the research question. Key sources of information were previous KTH thesis reports on related topics, research papers sourced online, produced at various universities and research bodies, and an online course in ML via education provider Coursera [30]. The results of literature study are found in the previous section.

3.2 Experimental development

Work on the research was performed in collaboration with TML. It was decided that the best approach to develop a program for detecting bot traffic was with a basic start, followed by incremental improvements. This approach enabled a good starting point and the opportunity to experiment, test and evaluate on an ongoing basis, making steady progress each step.

Testing and observations could be made from different set of rules, and by implementing and adding rules to the program, during each iteration. Iterative development was carried out during the second half of the thesis project and a total of five iterations were performed. The program was complex due to fact that the bot detection program should analyze real web traffic, with the aim that it could subsequently be deployed by TML upon successful completion.

This meant that the program was required to interact with company servers with the use of various Python libraries. As such, large amounts of the program were written by TML. The project contributed to the program and provided input through analysis, experimental ideas and theoretical ideas to implement.

3.3 Evaluation

Test data of real traffic to affected APIs were gathered to provide realistic and accurate testing. Three separate test sets would be created, since the bot detection program will be applied to three APIs, which TML has identified as being affected by bot attacks. Each test set would have two main subsets, one for verified bot traffic and one for verified real user traffic. The bot subset would also be divided into smaller subsets, depending on the category of bot attack.

Whilst using a limited set of data to perform testing is not representative of all traffic to the server, it provided a basis for evaluation with consistency, in a reasonable and appropriate way within the scope. It also allowed for comparison of results from iteration to iteration to track progress, and make solid conclusions and evaluation of results once research was completed.

3.4 Generating Results

Results will provide concrete data and information which will serve as a foundation for answers to the central research question. It is intended that results be generated through implementing solutions and subsequently performing tests to display the validity of such a solution and/or serve as a basis to make recommendations. Also, key findings from research will be provided, which might be helpful for future work. Results will be evaluated in the discussion section, and used to draw conclusions from the research.

3.5 Approach to bot detection

Much of the work of this project was centred on attempts to achieve effective bot detection using ML. The following model was formed a basis for our work in shaping the steps which were taken to work towards a solution.

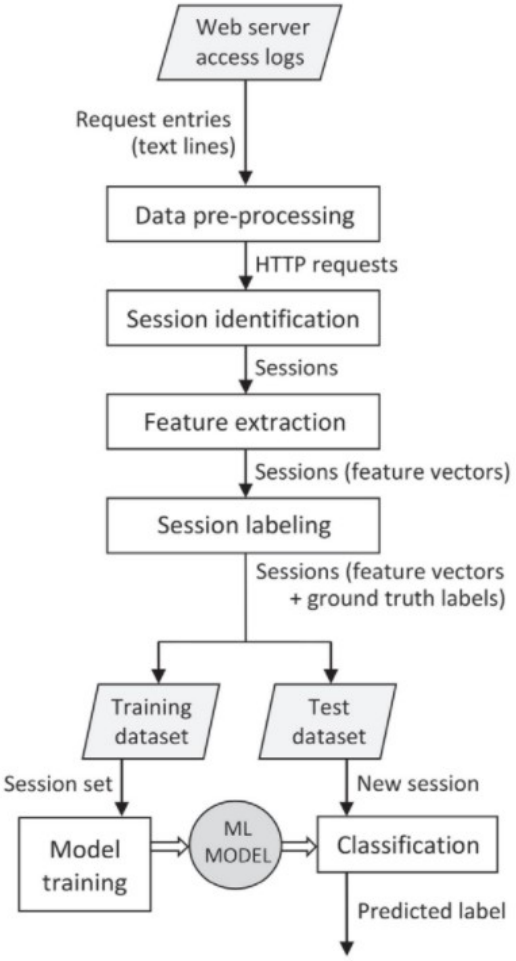

Figure 3.1: A model for the process of bot detection [25].

In line with the process of bot detection in figure 3.1, key steps of work on this project were as follows.

- The Kerberos service developed by TML handled the initial tasks of handling web server logs of HTTP-requests. This data was also processed by Kerberos, creating sessions by IP-address and then further processed to provide relevant features shaped by the session behavior such as counting the number of visits to given resource paths.
- Session labelling was a key task in attempts to develop an ML solution. This consisted of manual traffic analysis using the Instana tool to gather a number of bot and Legitimate IP addresses. With a list of IP addresses compiled, a test-set of traffic could be gathered and split appropriately into training and test sets for experimentation with an ML model.

- Following research and implementation of an appropriate ML program, experimentation could be done to assess the accuracy of the approach. Subsequent iterative development can be done to improve the solution either through adjustments to the ML solution or through development of additional features from the data.

Attempts to carry out the steps in this model formed the basis for much of the work described in the next chapter of this report.

3.6 Development tools

There were some essential tools required for development and experimentation in this project as presented in the next chapter.

- PyTorch which is an open source ML framework. PyTorch version 1.11.0 [23] was used in implementing a ML-solution to use as a basis for experimentation. The library has a number of highly optimized and well documented methods and a variety of tools.
- Jupyter, used mainly for plotting and visualization of the bot data as well as for a smooth programming experience with rapid feedback.
- Python NumPy library, offers mathematical functions based on linear algebra used in computing ML calculations.
- Anaconda simplifies package management and deployment, and is a distribution of the Python language. Anaconda version 3.10.4 [1] was used, largely since it was widely recommended online and allowed compilation and processing of the python tools in use. It is specifically optimized to handle, amongst other things, PyTorch and ML processing.
- Sklearn was a particularly useful resource for learning and experimenting with ML [28]. Due to it being pre-processed allowing much smoother implementation, the wine dataset was used for our experimentation in combination with PyTorch functions as a basis for generating results.
- Instana [15] is a web service tool, providing analytical information. Server traffic can be monitored in order to gain insight into patterns.

4 Bot detection

This section describes the work performed during the project to find and develop methods of bot detection. Key steps in this process included working in connection with development on a bot detection service called Kerberos. As part of this, efforts were made to develop an ML tool to optimize the accuracy of traffic evaluation. As a requirement for using machine learning algorithms, data sets are needed to train and test accuracy. Work was carried out to distinguish bot and legitimate traffic in order to gather data for experiments. This turned out to be a difficult process and alternative solutions were required to generate results. Finally, a particular solution to help optimize the bot detection process was also created during the project and is introduced.

4.1 Bot detection service Kerberos

This sub-chapter describes ongoing development on a bot detection service implemented by TML called Kerberos. Work on this project was closely connected with helping implement how bots could be detected and how clarity and certainty of detection could be maximized in order to take decisive actions to block bots from the server.

The initial task Kerberos performs is to provide an interface with the TML server which handles clients. It takes data which can be evaluated in steps toward bot detection. The second task is to provide a number of steps of data processing, performing tasks such as counting the number of requests from a given IP address to the server, subsequently counting visits to specific paths in the server and later even calculating ratios such as the proportion of requests to one path in comparison with another. These parameters are used as inputs into bot detection algorithms, which may be simple static rules, ML analysis or other alternatives. This sub-chapter also describes database design intended for managing bot traffic. The aim of the work during this project was to further develop methods of analysis with a focus on if machine learning could be applied. The following sub-chapters detail work on this process in attempts to optimize the accuracy of bot detection and develop strategies to distinguish behavioral differences in server traffic.

4.1.1 Initial setup with IP rules

As a starting point, following data gathering and processing, four static IP rules were implemented by TML. This was in order to evaluate and manage bots prior to implementing a machine learning solution in combination with the same rules or input data.

The four different rules are detailed in table 4.1 below. Rules include evaluating if any request path has a ratio above a set percentage of the total amount of calls, if the ratio between two resource paths is unbalanced in line with known behavioral patterns, and if the number of logins is too high.

Table 4.1: Description of static implemented rules for bot blocking.

Priority	Rule Name	Description
1	Overuse of path type	If more than 40% of all calls from this IP during the past 24 hours is to the same path -> set score to 0.0
2	Unbalanced ratio of availability and price calls	If the ratio between 'get availability' and 'get low fare trip availability' is lower than 0.45 or higher than 0.55 for the past 7 days -> set score to 0.0
3	Too many logins	If the ratio of login calls (/oauth/token) for the past 24 hours is higher than 10% of total calls -> set score to 0.0
4	Lack of variation - too few unique paths requested	If there are less than 6 different request paths for the past 24 hours -> set score to 0.0

Analysis of input parameters using the rule framework as detailed above allows quick and reasonably accurate evaluation of traffic. In the table above, a score of 0 indicates that the source is a bot while the implementation handles traffic so that it is scored as 1 initially and remains 1 if it passes the static rules above.

4.1.2 Client score database design

Each IP address accessing the server is viewed by Kerberos as a client and in line with a bot detection process is scored to help classify it as a bot or legitimate

user. The result of the scoring is stored in a Redis database. This enables the possibility to classify IP addresses as bot or legitimate, in order to subsequently take actions such as blocking and blacklisting suspicious traffic sources. Redis is a remote directory server where data can be read and written directly from the main computer memory [11]. It is helpful since the stored results can be used for fast lookup in the traffic flow.

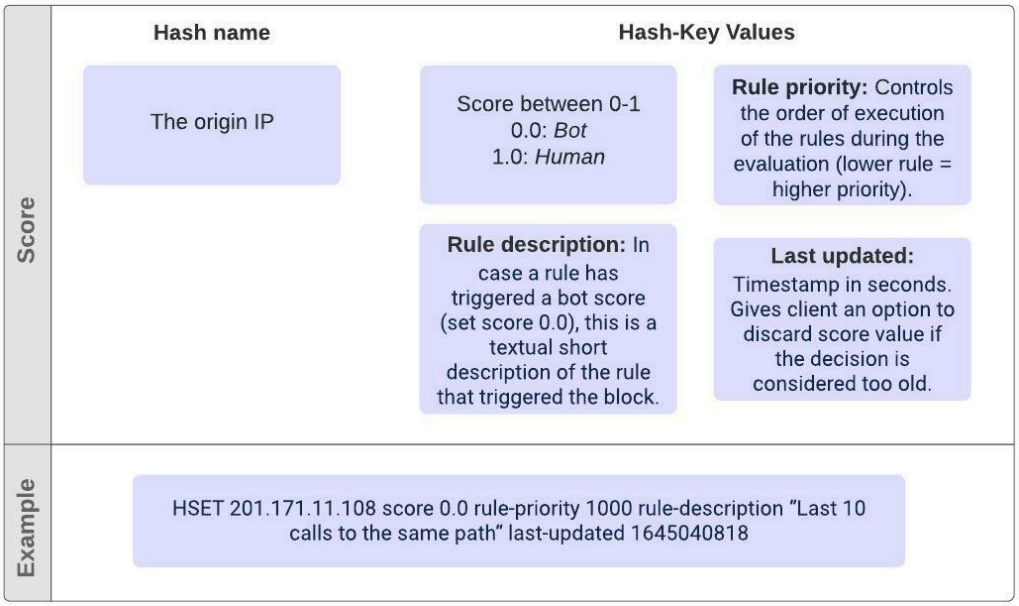

Figure 4.1: Database design of Redis.

As seen in figure 4.1, Redis enables storing the outcome of a bot detection scoring algorithm. A score per IP-address is constantly updated according to traffic analysis. The score can be accessed with a fast lookup upon a HTTP-request to the server, and to determine how a client is handled. This means that upon a certain score suspected bots can be blocked from accessing resources on the server.

4.1.3 Storing historic data with BigTable

BigTable [5], a Google Cloud service, was intended to be used as a database for the implementation of Kerberos to store historic data for analysis and processing. BigTable supports high read and write throughput at a low latency, in order to obtain a faster access to data. BigTable was set up locally, making it possible to retrieve call data from Kerberos, which is the traffic data to the TML server handling clients.

In order to retrieve call data, command-line tool was used with possibilities such as deleting or adding rows directly, among other options [4]. Ultimately, BigTable was used minimally in this thesis due to a delimiting factor with accessing relevant processed data.

During the project work, efforts were made to work with this data in order to develop a data-set for experiments and further development. However, issues with the data format were encountered and it was decided that it was not possible to resolve the issues within the scope and time-frame of the thesis. This led to the need to find and use alternative data, which was eventually sourced from Sklearn [28] and used in development of an ML plug-in for Kerberos.

4.2 Machine learning

In order to optimize bot detection, this project focused efforts on how ML could be used to further develop Kerberos. The aim was that a ML-feature could be developed and tested, which could be used to raise the accuracy of traffic analysis and detecting bot traffic.

Such a model should carry out the following steps.

1. The model takes a set of previously classified training data and subsequently learns which inputs lead to a given output.
2. Once learning is complete, a model (or equation) is established to process future inputs and classify the output into a given category, namely bot or legitimate traffic.
3. A portion of the training data is classified using the model and can be checked to decide if classifications made are correct or not, indicating the accuracy.
4. New inputs can be taken by the model and it will produce an output in the range [0, 1] indicating the likelihood of the input belonging to a given category. Here it is intended that values close to 0 should indicate bot traffic and values close to 1 legitimate. Placement in the interval indicates

the likelihood of each classification according to the model. This means an output of 0.01 would indicate 99% probability of traffic being a bot and 1% probability of it having a legitimate user according to the model's analysis.

5. In order to be useful in its intended environment, a model must be: fast and efficient, highly accurate and must produce very few (ideally 0) false positives, which in this case would be legitimate users being incorrectly classified as bot traffic.

These requirements shaped research work, testing and results produced. As well as having implications as to which type of machine learning is appropriate and how to implement it. This also requires training set data which led to other work described in the following sections.

4.2.1 Choice of approach in applying machine learning

The task requires traffic to be classified into two discrete categories, bot or legitimate user. Therefore logistic regression was selected as the most suitable algorithm to apply, since it can model the probability of a discrete outcome such as yes/no or true/false, given two input variables. Motivation for this approach was taken both from Coursera [30] and Rovetta et. al. [25].

The intended outcome being that with a trained model in place, a prediction would be output in the interval [0, 1]. Here, scores close 0 would indicate high probability of a bot and measures could be taken to make an appropriate cut off point, below which traffic categorized as such could be blocked.

4.2.2 Implementation and experimentation

With the goal of creating an implementable solution which should be optimized for calculation time as well as accuracy, it was most effective and appropriate to use library functions. PyTorch [23] was chosen as an appropriate library to use, with a number of highly optimized functions. Features including a dataset loader, data feature scaling and an easily adjustable model format with options to adjust the learning rate and optimization algorithms making it easy to use for experiments. Instructions on how to set up a ML-program with PyTorch was used [17], and with

a program in place experiments could be performed.

Due to data-set difficulties and delays, experimentation on a model was limited to use of generic open-source data acquired from Sklearn [28]. Aiming to fulfil initial goals with the thesis, a few tests with the data were carried out to help optimize and make a ML plug-in available for future use in Kerberos.

Experiments were done to find which learning rate would work best and how many rounds of training were required to produce accurate results efficiently. Another experiment consisted of comparing 2 optimizer algorithms found in the PyTorch library. A basic algorithm, Stochastic Gradient Descent (SGD), was compared with an advanced algorithm, ADAM, to see if better accuracy and performance could be obtained. The key difference in these algorithms is that SGD uses a fixed learning rate to generate incremental improvement, whereas, ADAM implements dynamic adjustment of the learning rate allowing faster initial adjustments before finely adjusting as the cost function gets smaller [23].

The value of these tests and results is limited due to the fact that relevant data was not gathered and used as a basis for experiments. Due to this, results of these experiments are presented in appendix C.

4.3 Distinguishing human and bot traffic

In order to create bot detection solutions, understanding the differences in bot versus legitimate traffic is essential. This sub-chapter describes how Instana, a web traffic monitoring tool was used. The aim of using this tool was to understand traffic patterns which could be used to create a bot detection framework. Furthermore, it was hoped, that using the tool a list of bot IP-addresses could be gathered in order to create a data-set for further ML experimentation. The same would also be done for legitimate traffic and together, these would form the training and test-set for a ML model.

4.3.1 Traffic analysis using Instana

Analysis of bot and legitimate traffic was done through manual analysis with help of the Instana traffic analysis dashboard. This tool enables powerful analysis of web traffic, providing helpful data visualization of a number of metrics. It also allows the construction of detailed filters to separate and isolate traffic in order to gain insights and analyze patterns of web traffic.

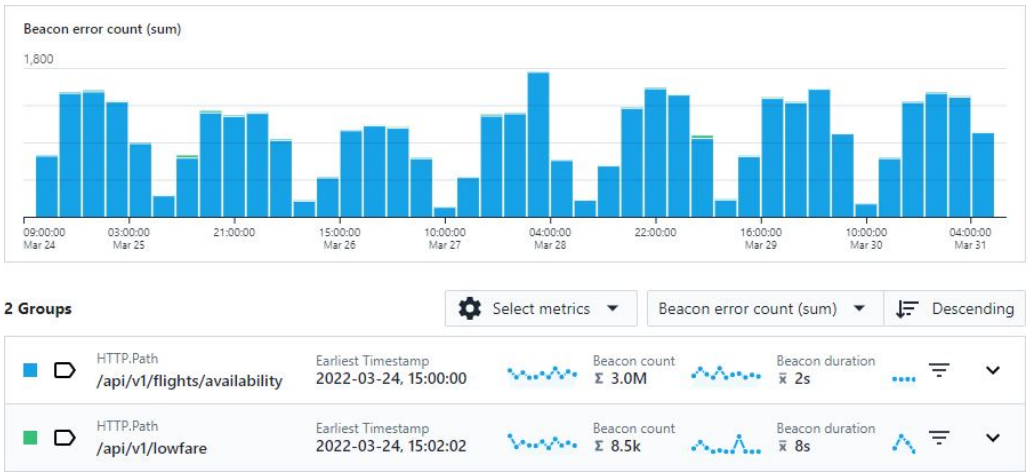

Figure 4.2: Visualization of traffic from Instana.

An example of the information provided by the Instana tool is seen in figure 4.2. The blue graph displays the varying volume of traffic over time. It was observed that bots generally operate by sending high impulses of traffic at various points in time. As such, one way to find bot sessions was to zoom in on a spike in traffic. A detailed list of the resources requested and accessed during a session is also provided as seen at the bottom of the figure. This information can be used to verify if the session was legitimate or not using a framework of known behavior patterns as listed below.

- For a legitimate user, the traffic should show a variation in the resource paths accessed. This can be seen in using Instana metrics including session amplitude or by looking at the list of resources accessed. Meanwhile bots generally request few types of resource and a high proportion of requests are to only one or two paths.
- Low error rate and trusted Internet Service Provider (ISP) are also characteristics connected to a legitimate user.

- Another important factor with a legitimate source is that the IP location should remain static and not change randomly.
- For bots, there are little variation in traffic rates in connection with high volumes of traffic.
- High error rate and blacklisted or flagged ISP are more common for bots.
- Another important factor is that a bots often interchange in IP region mid-sessions, which might be an indicator that the traffic uses a Virtual Private Network (VPN) in order to mask actual IP addresses.
- Bot traffic from a given IP address is often > 1000 HTTP-requests in a short space of time, while legitimate users tend to have sessions of 50-200 HTTP-requests.
- Bot sessions were generally short but frequent, leading to a high number of logins and lots of calls consisting only of a login and accessing of one or two resources.

This framework was discussed and conducted with insights from TML, as well as through experimental searching and observation using the tool. Another means of bot and legitimate traffic identification and verification, was the creation and use of Instana traffic filters. An example of how filters can be set up is shown in figure 4.3.

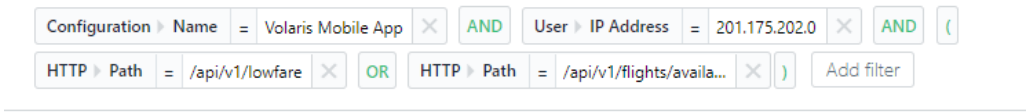

Figure 4.3: An Instana filter.

These filters can be constructed using a number of parameters, such as source IP address or range, server path (or paths) accessed and the number of error calls, amongst other things. In combination, the time span under observation can be adjusted from ranges of minutes to weeks in the settings, and subsequent

visualizations and sorting of the results gives insightful traffic information. Using these features allows for accurate analysis and the ability to evaluate the source of server traffic with a high degree of certainty. The results of Instana analysis are presented and discussed in the following chapters. It was hoped that the compiled list of IP addressed would be used to gather data-sets as previously mentioned. Unfortunately, this was not possible in the time frame of the project. Nonetheless, findings during this work provided insight into behavioral patterns and the basis of some key project findings and conclusions.

4.4 Optimization of Kerberos traffic classification

This sub-chapter describes an alternative solution to detect bots through separation (polarization) of traffic. The key aim was to provide an implementable solution, largely utilizing existing data-structures in Kerberos, which would provide a significant increase in the accuracy of bot detection. The approach to polarize traffic took inspiration from techniques applied in ML, including neural networking and feature engineering [27]. Appendix B is a detailed report made for TML, providing details of the algorithm, its theory, steps toward implementation, and how further development can be carried out.

The Kerberos program, developed in connection with this project provides an interface with live server traffic and subsequently processes the data, extracting relevant values from HTTP requests. It was observed during the project that Kerberos had created variables containing valuable metrics based on knowledge of traffic patterns and typical bot behavior versus that of legitimate users.

In order to polarize traffic, variable manipulation using a few simple mathematical operations was performed to highlight the differences in traffic patterns. This provides more clarity over the likely source of traffic, and a higher degree of certainty which is required if an IP address is to be classified as a bot. Various assumptions were used to highlight behavioral differences, including traffic volume and resource request patterns. These techniques were combined with intuitive knowledge about traffic patterns learned throughout the project.

The initial Kerberos algorithm takes a number of metrics, such as, the number of requests from a given IP address, and the ratio of visits to server paths where known behavioral differences existed. Partial scores were then calculated, for instance, if the ratio to a path known to be targeted by bot was above 55%. A few partial tests were carried out and a total score summed up with the output, 0 or 1, determining the source as bot or legitimate user.

The polarization of results solution provides an alternative scoring system with the aim of making a much clearer separation between traffic, whilst still being rooted in the same variables. Through variable manipulation and combining variables through multiplication, bot and legitimate traffic scores become distinctly polarized helping the detection process. An example of this is as follows.

- Bots have: high number of HTTP requests, a resource access ratio to the availability versus price ratio of 0.9-1 (av/pr), few resources (paths) requested, high use of top few (most visited) paths, short sessions but lots of them (many logins).
- Legitimate users have: fewer HTTP requests, a resource access ratio to the availability versus price ratio of 0.5, variation of paths requested, fewer sessions or logins, and a lower use of top paths.

This information can be used to shape an equation which handles the traffic from a given IP address as shown below. Here the following variables are used: *calls(lastHour)* is the total requests/calls from a given IP address during the past hour; *av* is the number of visits to an availability path; *pr* is the number of visits to a price path; *numPaths* is the number of unique paths visited during the time period; *topPathVisits* is the number of visits to the most visited path.

$$result = calls(lastHour)^2 \times (\frac{av}{pr})^3 \times (\frac{1}{numPaths})^2 \times (\frac{topPathVisits}{calls(lastHour)})^2$$

The results of each element in the equation are multiplied in order to separate (polarize) traffic behaviour. This is described more detailed in appendix B.

For typical bot values input, this produces a result of the magnitude:

$$1000^2 \times (0.9)^3 \times (\frac{1}{10})^2 \times (0.4)^2 = 1166$$

A typical legit user would result in a vastly different score as follows:

$$200^2 \times 0.5^2 \times (\frac{1}{50})^2 \times (\frac{1}{8})^2 = 0.0625$$

Once normalized, this score is scaled to the range [0, 1] where legitimate traffic would be scored close to 0, while bot traffic will be 1 or close to it. Through combining available resources with relatively simple operations, this solution provides a simple yet significant optimization to Kerberos which is detailed and gives a more accurate classification output.

Additional details of how the initial polarization equation was designed, a standardized approach to developing further equations as a part of the algorithm, a motivation of the mathematical mechanics used to achieved the separation of traffic, steps to normalize the output and other details helpful for implementation and further development are documented in appendix B.

4.4.1 Testing the polarization optimization

In order to evaluate if the polarization algorithm is an improvement on the initial Kerberos method of traffic analysis, testing was carried out. Since it was not possible, late in the project to have the polarization solution implemented into the Kerberos program, testing was done manually, gathering sessions at random. This was done using the instana analytical tool, by taking the most recent traffic IP addresses and gather traffic data from that sources from the previous 24 hours. Relevant values were then processed according to each algorithm to see how Kerberos would classify the traffic and if the polarization classification was an improvement. It is worth noting that polarization algorithm does not have a prescribed output score above which traffic is considered a bot. Instead it is intended to be experimented with, to establish where an appropriate 'cut' should

be made to classify the traffic into the two categories. For the purposes of this testing the output was first generated, that is, output values were generated, then an appropriate 'cut' value of 50 was chosen to classify the traffic, where scores above this were classified as bot and below as legitimate. The results of this testing are presented in the next chapter and subsequently discussed.

5 Results

The purpose of this degree project was to understand of how ML can be used in order to detect bot activity in the airline industry. This chapter summarizes the results and attempts to answer the research question of the thesis.

5.1 Instana results

Analyzing with Instana led to the following results in terms of manually gathering sources of suspicious and legitimate traffic to the server. These IP addresses were obtained and determined to be malicious by looking for parameters such as an even curve with the same number of calls, high or low duration, constant change of regions and higher failure rates, amongst other things.

Traffic source	~ 100% Availability	50/50 lowfare	> 40% to single path
Suspected bots			
78.138.00.00	✓	✗	✓
83.229.00.00	✓	✗	✓
209.99.00.00	✓	✗	✓
83.229.00.00	✓	✗	✓
148.74.00.00	✓	✗	✓
Legitimate users			
166.205.00.00	✗	✓	✗
73.116.00.00	✗	✓	✓
200.68.00.00	✗	✓	✗
172.58.00.00	✗	✓	✗
166.205.00.00	✗	✓	✗
177.236.00.00	✗	✓	✗
201.142.00.00	✗	✓	✗

Figure 5.1: IP addresses of detected bots and legitimate users.

The results provided in figure 5.1 can be used to gather a training set for ML training in order to make future predictions. The process, as described in the previous chapter can also be followed to gather more for future work. Note, part of the IP address is not published here to keep user information anonymous in keeping with data privacy requirements.

The IP addressed were gathered, in part by manually applying the initial Kerberos rules as described in chapter 4 to analyse traffic sessions. Reviewing resources accessed during a session in further detail also helped determine the source of calls and enabled the classification of sessions into the groupings ”suspected bots” and ”Legitimate users”. It should be noted that only in cases where, with a high degree of certainty, a session could be classified were these classifications made. If any doubts or uncertainty was held then these sessions were left out of results, as, gathering incorrectly classified session traffic would impede the success and accuracy of any future use of such data in ML training and testing.

5.2 Polarization versus Kerberos

Having carried out testing the following results were gathered to compare traffic analysis performed according to the Kerberos scoring system and the polarization scoring system.

Suspected bot (bool)	Kerberos result (bool)	Pol. result (bool)
×	×	×
×	✓	×
×	×	×
×	×	×
×	✓	×
×	✓	×
×	✓	×
×	×	×
×	×	×
×	✓	×
×	×	×
×	✓	×
×	✓	×
✓	✓	✓
✓	✓	✓
✓	✓	✓
✓	✓	✓
✓	✓	✓
×	×	×
×	×	×
✓	✓	✓

Figure 5.2: Traffic analysis of Kerberos versus Polarization.

As seen in figure 5.2 the Kerberos scoring system classifies traffic with reasonable success but a number of legitimate traffic sessions are incorrectly detected as being bots. Meanwhile the polarization scoring system correctly categorizes all 15 legitimate traffic sessions as legitimate and also correctly detects all 6 bot sessions. These results are summarized in the table below.

Traffic source	Sessions	Ker. correct predictions	Ker. accuracy	Pol. correct prediction	Pol. accuracy
Legitimate	15	8	53,00%	15	100,00%
Bot	6	6	100,00%	6	100,00%
Total	21	14	66,00%	21	100,00%

Figure 5.3: Classification accuracy of Kerberos versus Polarization.

As seen in figure 5.3 the polarization scoring system substantially outperformed that of Kerberos initial rule set in correctly classifying all 21 sessions analysed. Whilst the Kerberos system correctly detected the 6 bot sessions analysed and 66% accuracy in total, it showed significant shortcomings in analysing legitimate sessions. Its simple rules achieved only 53% accuracy in correctly classifying legitimate sessions. This is particular cause for concern given that one task of central importance is avoiding incorrectly blocking legitimate users.

5.3 Other Findings

This section will provide other key findings which were discovered during literature study, developed throughout work and in discussions with TML. The aim of findings is that they contribute to answering the key research question.

1. Research shows that there has been proven success in detecting web bots performing scraping tasks. A high detection rate was demonstrated in Rovetta et al. [25] through the use of clustering algorithms to perform initial analysis of traffic patterns prior to supervised ML categorization.
2. Brown and Doran [2] noted, an effective factor in detecting is behavior analysis based on resource uses. This was also observed throughout project work as being an area of central importance in implementing bot detection.

3. Path analysis can provide a powerful mechanism for detection [2]. Legitimate users follow paths into resources on a server during a browsing session as they navigate via links in web content. In contrast bots frequently hop, requesting resources systematically without following links. Algorithms can detect and provide usable metrics based on such behavior.

4. Regarding traffic volume and density, one unavoidable constant is that scraping mechanisms used by bots must generate a lot of traffic. In order to systematically gather server content this is necessary and must also increase further where detection mechanisms are implemented. Whilst possible to change IP address, doing so too frequently raises costs.

5. Feature engineering is where time and resources must be focused for successful ML implementation. ML software provides highly effective mathematical calculation tools for several tasks such as estimation and categorization. The tool is only as good as its input. Processing and intuitively selecting which data to use as input features for ML is paramount to what can be achieved. Notably it is also time consuming and generally requires specialized knowledge.

6. In connection with point 5, the degree of autonomy is difficult to estimate. Generally ML, especially supervised learning tested in this project, requires intuitive work to establish. Maintenance is also required both to check effectiveness and also the gathering of training set data. For instance in a categorization task, pre-categorized data must first be used to train an ML model before unsorted inputs can be processed. For TML, this would require manually gathering training sets of known bots and legitimate users.

7. It was observed that the particular problem at TML has a high degree of sensitivity. Incorrectly detecting a legitimate user as a bot and subsequently blocking it has significant consequences and is highly undesirable. This must be handled and in turn adds to complexity of the task.

8. In connection with point 7, assessing if detection predictions of an implementation were correct would be beneficial. It was noted that this is difficult given that user tests such as CAPTCHA can be forged by bots.

Development of strategies and implementations to handle this issue may be highly valuable in forming a successful implementation.

9. Implementation of honeypot traps in a servers could help provide metrics which could be used in detection algorithms [16]. Specifically if paths could be created which legitimate users cannot reach and scraping bots can.

6 Conclusions

This section contains the discussions and conclusions of this thesis, based on previous chapters, highlighting the positive sides and drawbacks during the project work. There is also a section about future work of the research area.

6.1 Discussion of results

Evaluation of the results will be centred on how the results produced answer by addressing the central research question along with its sub-questions.

6.1.1 Instana analysis

Instana results were initially gathered with the aim of using the IP addresses to gather in a test set of bot and one of legitimate users for ML experimentation. Working with the tool and creating filters in order to find bots or legitimate users led to useful insights, which are pointed below.

- It serves as proof that significant and measurable patterns of behavior are observable.
- Insightful knowledge of these patterns, such as which resources on the server bots primarily target, exists at TML.
- With very few filters in place, traffic can be helpfully separated giving a strong indication of the source of HTTP requests.

Whilst simple, these insights actually serve as a powerful basis in helping answer the research questions raised in this project. Behavior patterns not only exist, metrics can be used to separate them.

6.1.2 Kerberos optimization

Tests of the polarization algorithm in comparison with the initial set of rules implemented in Kerberos produced surprising results. It was surprising to see how the four static rules implemented in Kerberos led to a number of incorrect classifications of legitimate sessions. It was observed during testing that the reason for this was that if session behavior is slightly different from normal

assumptions on just one of the 4 rules, then this leads to it being classed as a bot. An example of incorrect classification was where one session had 12% of requests consisting of logins causing it to fail the check of being below 10% while the rest of the session behavior was consistent with legitimate use. This highlight the strength of the polarization algorithm, which, through combining and manipulating such inputs provides a more balanced and consistent output.

It should be noted that the initial four Kerberos rules were only an initial rough estimate set out to provide the basis for development and experimentation. Fine tuning these rules, would likely lead to slightly better results. It should also be noted that the testing carried out was limited, taking just 21 sessions randomly for analysis where if thousands of sessions were analysed results would give a more calibrated analysis of prediction accuracy and could be considered more robust. This was not possible within the course of the project since it was developed as an alternative solution late on in the project as an algorithm and was not possible to implement in the projects time-frame.

Nonetheless, the solution provides a simple yet powerful approach to handling the problem of determining the source of traffic. In particular, the algorithm and document of how it can be further developed provides a platform to powerfully separate traffic and analyse behavioral patterns. Given the surprisingly impressive results in initial testing, it may be worthy of future work and experimentation.

6.1.3 Key findings

From the early stages of the thesis project it was observed that the problem was highly complex and that any subsequent solutions applying ML would be also. As such it was decided that providing key research findings should form a part of deliverable results. The aim of these being that they would provide valuable information to TML and may also help in forming recommendations for implementation or future work.

Several interesting findings were discovered throughout literature study and work on the project. Various elements of these were discussed throughout collaboration with TML and deemed to be of interest. Having such findings concisely summarized was of help in developing future work recommendations and will also be generally helpful for readers. As such these results fulfill their aims and can be seen as a successful delivery.

6.1.4 Answers to the research question

Following research and experimentation, the following conclusions can be drawn in response to the research question and its sub questions described in section 1.2.

Bot traffic can be detected effectively. Known patterns of behavior and metrics by which to quantitatively handle are possessed by TML. Research presented in chapter 2 shows that using such metrics, detection has been successfully implemented using machine learning with highly accurate prediction rates [25]. Experimentation applying ML in this project was limited due to data gathering issues and results produced were subsequently not substantial in providing additional support in answering this sub question in connection with the particular case study at TML. However the alternative solution to polarize results in the Kerberos scoring system showed promising results and is worthy of consideration for additional experimental development as TML approaches how to detect and block bots.

Bots can be detected through many different approaches. Amongst other options, static rules and filters such as firewalls exist and can be effective, whilst they are generally limited in their effectiveness to stop advanced bots. This report has focused on researching how ML can be used. Whilst experimentation was limited to a generic dataset, findings show that highly accurate categorization predictions can be made using a logistic regression algorithm to analyse a training set of manually evaluated bot traffic in order to predict on future incoming server traffic. Research also showed that using unsupervised clustering to automatically find traffic patterns in data, then subsequently using logistic regression to categorize the clusters themselves, is a powerful approach [25].

Particular solutions for TML: A number of solutions and options have been presented for TML. These will also be summarized in the next section on future work. Details of how to implement a logistic regression algorithm as part of a bot detection solution have been presented. A particular solution to optimize the Kerberos program with a polarization algorithm was also put forward for consideration. A number of theoretical findings were also provided which might help TML in where to channel future efforts in handling the problem. It should be noted, that there is no one simple solution, rather, since the problem is complex and ever changing, the problem is addressed by offering tools and strategies which are deemed good options to begin implementing or focus future efforts on.

6.2 Future Work

This section describes limitations and other problems faced during the thesis, and how these could be improved in future work. This section also describes alternative solutions that could be useful for TML.

6.2.1 Training-set and feature engineering

The major issue and challenge when collecting bot traffic data, is to distinguish bots from crawlers. Automated filtering of the traffic do not guarantee that the actor is a bot or a human. Cleaning and improve the data collection in order to get more accurate results, would have been the main focus if given more time. In connection with this work on a training set, a focus on feature engineering would be advisable. As mentioned in findings, ML-algorithms are extremely powerful and efficient in processing data and creating a model for accurate predictions to the source of incoming traffic. These algorithms are however, only as powerful as the input given them provides. Processing data to give the most useful input features is thus the key to successful ML-implementations.

In the case of TML this would include, amongst other things, processing data to give metrics such as: the ratios given paths are requested, traffic volume, number of logins etc. In order to be effective this list should be exhaustively expanded and

experimented with. Research on unsupervised clustering techniques may reduce the manual work required for this. The authors also wish to note key findings which may be of value to develop and experiment with, are focusing on resource request patterns and possibly developing algorithms to analyse path hopping and navigation through a server where behavioural differences may exist.

6.2.2 Integration of ML model into Kerberos

Our ML model, designed to be a plugin to optimize the scoring system in Kerberos is documented in chapter 4, appendix A and appendix C. It is based on one test set on which experiments and development has been performed. Future application by TML would mean that this model needs integrating into their server in order to analyse real incoming server traffic. This will likely require the following:

- The type of data to use as input to the Kerberos ML-plugin, and a standardized training set size and timing of periodic training of the analysis. Also, documentation and comments on code to enable easy use of programs in the future for TML is needed.
- Size of the dataset and how many calls should be analysed, and how often retraining shall be performed, e.g. every 10 minutes/daily.
- Handling the prediction of the source output, e.g. the plugin may take a given number of inputs and produce an output x between 0 and 1, where x represents the likelihood of the given call being from a bot source.
- Maintenance, manually gather test sets and understanding and monitoring what the program is doing, is important in most ML implementations and should be kept in consideration throughout development. Another important note is getting feedback, that is, knowing if predictions made are correct or not remains an issue especially given the sensitivity of handling traffic without detriment to legitimate user experience.

The Kerberos optimization through polarization of results remains an option to explore. As detailed in results and appendix B, this is a simple solution utilizing many existing features and variables in the program. It is not ML, but a powerful in its simplicity and separation of scores according to known assumptions.

6.3 Project Reflection

This section describes the reflections of the project.

6.3.1 Sprint planning

By revising the results of this thesis, there were several things that could have been done differently in order to reach the project goal. One important improvement that could have been made, is to revise and do more clear milestones for each week, and take usage of sprint goals.

The work consisted of a timeplan with hard deadlines in order to reach the goals. However, the work was not entirely based on the timeplan, e.g. understanding ML and try to implement a useful solution. Working with Python and writing scripts for larger data sets, and learning embedded functions in the analytical tool Instana also took more time than expected.

6.3.2 Dataset delays, development dependencies and sliding scope

Having reached the conclusion of the project, hindsight provides a few take-away points to help improve and effectivize future work. From the outset, it was noted that the particular problem and the ML techniques were highly complex especially in relation to our prior knowledge of the subject. This made it difficult to define a substantial yet realistic project scope. As a result the scope was frequently adjusted throughout the project. At times during project work this led to difficulties in knowing what to do to progress toward concrete results.

This is not a major issue, but does highlight the value of experience and how well defined requirements and tasks can unlock effective work. Another issue connected to experience levels was the development of software in parallel to project work. This was not a problem in itself, but it was quickly observed that the programming levels of a professional versus a novice are vastly different.

In this case it led to a dependency on TML for the programming of Kerberos which interacts with servers providing TML services. This in turn meant that since time was scarce and recording of server traffic not as simple as thought, to it not being possible to get data sets to experiment with in detail. Ultimately, if a more detailed and implemented ML solution were the desired outcome, it may have been better have had access to simple processed data sets from an early stage.

This led to limitations in results, since experimentation was performed using a generic testset ans thus solutions produced were more theoretical as opposed to detailed implementation. In closing discussions however, TML stated, they consider the findings and insights developed throughout the project to be valuable and helpful to their work. The authors also feel that while the task was somewhat ambitious, a lot was learned and taken from the project including both technical skills as well as the handling of complex projects in general.

6.4 Final Words

A complex problem was presented by TML of the issues they are facing with web bots. Having carried out research, this thesis has analyzed these problems, and provided a number of alternative suggestions for future work. Bots which are designed specifically to get around bot detection and prevention software, are in a sense playing a game trying to circumnavigate developers wishing to block them. Whilst no simple solution to the problem exist yet, this report has found and presented useful research findings within the area. For the authors this thesis work have been an eye opening and educational experience. The question remains for TML if the obtained findings might be the correct way forward.

Appendices

Appendix A: Training set

This program was used in order to experiment, important functions are explained in order to facilitate future referencing and developments if such a solution were applied.

```
import torch
import torch.nn as nn
import torch.optim
import numpy as np
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import torchvision
from torch.utils.data import Dataset, DataLoader
import math
import time

tload = time.time()
class WineDataset(Dataset):

def __init__(self):

    # Initialize data, download, etc.
    # Read with numpy or pandas

    xy = np.loadtxt('./data/wine/wine.adj.csv', delimiter=',', dtype=np.float32, sk
    self.n_samples = xy.shape[0]

    # Here the first column is the class label, the rest are the features

    self.x_data = torch.from_numpy(xy[:, 1:]) # size [n_samples, n_features]
    self.y_data = torch.from_numpy(xy[:, [0]]) # size [n_samples, 1]
```

```
# Support indexing such that dataset[i] can be used to get i-th sample

def __getitem__(self, index):
    return self.x_data[index], self.y_data[index]

# We can call len(dataset) to return the size

def __len__(self):
    return self.n_samples

# Create dataset to work with
    wdataset = WineDataset()

# first_data = dataset[0]
# features, labels = first_data
# print("F & L = ", features, labels)

# train_loader = DataLoader(dataset=dataset,
# batch_size=4,
# shuffle=True)

# 0) Prepare data
# bc = datasets.load_breast_cancer()
X,y = wdataset.x_data,  wdataset.y_data

n_samples, n_features = X.shape
#print("samples, features =" , n_samples, n_features)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random
# Partitions the data ready for training and testing
# Test takes .2 of data randomly to test using train_test_split()
# May want to adjust to determine y
```

```
# Scale
# Adjusts data to have 0 mean and unit variance
# That is: features are evenly weighted regardless of their input values
# Ex: if 0.1 or if 5000 avg -> both scaled to 1.0
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Adjusts datatype of items
X_train = torch.from_numpy(X_train.astype(np.float32))
X_test = torch.from_numpy(X_test.astype(np.float32))
# y_train = torch.from_numpy(y_train.astype(np.ndarray))
# y_test = torch.from_numpy(y_test.astype(np.ndarray))

# Put result into 1 row 1 column according to size of y_train and y_test
y_train = y_train.view(y_train.shape[0], 1)
y_test = y_test.view(y_test.shape[0], 1)
# May adjust to get various printouts

# 1) Model
# Linear model f = wx + b , sigmoid at the end

class Model(nn.Module):
def __init__(self, n_input_features):
    super(Model, self).__init__()
    self.linear = nn.Linear(n_input_features, 1)
    # Number of input features, output features = 1

# Sets up so y = result of sigmoid(self.linear(x))
def forward(self, x):
    y_pred = torch.sigmoid(self.linear(x))
    return y_pred
```

```
model = Model(n_features)

# 2) Loss and optimizer
# Sets the number of rounds of analysis and the learning rate
# Could adjust for effectivity
num_epochs = 20
# print('Nr of epochs = ', num_epochs)
learning_rate = 0.2

# Binary Cross Entropy
criterion = nn.BCELoss()

# print('data loading time (ms) = ', (time.time()-tload)*1000)
t1 = time.time()

# Either of the optimisers can be chosen below(comment one out)
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
# optimizer = torch.optim.SparseAdam(model.parameters(), lr=learning_rate)
# optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
# scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'min')

# 3) Training loop
# PyTorch performs calculations such as gradient automatically

# Initiates prevloss variable to be updated
# Used to ensure that loss can only go down
prevloss = 100

for epoch in range(num_epochs):
# Forward pass and loss

    y_pred = model(X_train)
```

```
    loss = criterion(y_pred, y_train)
    if abs(loss)> abs(prevloss):
        print("stopped at epoch: ", epoch)
        break
prevloss = loss

# Backward pass and update
loss.backward()

# scheduler.step(loss)
optimizer.step()

# Zero grad before new step to recalculate
optimizer.zero_grad()

# Following line used to chart cost/loss function per epoch
if (epoch+1) % 1 == 0:
    print(f'epoch: {epoch+1}, loss = {loss.item():.4f}')

# for epoch in range(num_epochs):
# Forward pass and loss
    # for input, target in loader:
        # y_pred = model(X_train)
        # loss = criterion(y_pred, y_train)

# Backward pass and update
# loss.backward()
# scheduler.step(loss)
# optimizer.step()

# Zero grad before new step to recalculate
# optimizer.zero_grad()
```

```
# if (epoch+1) % 1 == 0:
    # print(f'epoch: {epoch+1}, loss = {loss.item():.4f}')
    # scheduler.step()

# Evaluation
t2 = time.time()
totalTime = ((t2 - t1) * 1000)
print('Algorithm evaluation time(ms) = ',  totalTime)

with torch.no_grad():
    y_predicted = model(X_test)

# yp = sc.inverse_transform(y_predicted)
# print("ypred = ", y_predicted)
# print("ypredsum= ", sum(y_predicted))
# print("yactsum = ", sum(y_test))
# print("factor = ", (sum(y_test)/sum(y_predicted)))

factor = (sum(y_test)/sum(y_predicted))
y_refactored = y_predicted * factor
y_predicted_cls = (y_refactored).round()
# for i in y_predicted_cls:
    # print("y pred = ",y_predicted_cls[i,0][0])
# print("ypred = ", y_predicted_cls)
# print("should be: ", y_test[:,0])

acc = y_predicted_cls.eq(y_test).sum() / float(y_test.shape[0])
print(f'accuracy: {acc.item():.4f}')
```

Appendix B: Optimization of Kerberos

This document is a report produced for TML providing details of the polarization idea.

Polarization of Kerberos bot scoring/prediction output

A particular optimization for the Kerberos bot detection software

Through knowledge of api traffic and patterns of bot vs legitimate user behavior a number of metrics can be produced. Through processing these metrics using a few mathematical operations and principles, equations can be produced to magnify differences in behavior. This leads to polarization of results and likely a muchI more accurate and usable prediction of the source of a call. Polarization of results can essentially be seen as a tool to help separate and provide more clarity, which in turn can be applied to detect and prevent bot attacks on a server.

Following an initial example as presented and discussed today (28/4/2022) detail will be provided of how such an equation can be developed, which assumptions can/must be made and used, a proof of the mathematics, and a normalization process to ensure the output value be in the interval [0, 1].

One or more equations will also be developed in order to implement and test in the Kerberos program. A method for testing, producing data results and a process of how to subsequently chart and analyze such results will also be made. Strengths and weaknesses of the idea will also be given as well as other comments and recommendations.
In connection with creation of a testing and analysis design, aim, limits and scope of this experiment will also be set to ensure it remains implementable, realistic and can produce results within a timeframe.

Following an example and motivation for this approach this document will provide:

- Aims, Scope and limits
- Theory
- Necessary assumptions/knowledge
- Which Variable/features are required/can be used
- Mathematical proofs and tools to use, as well as, normalization of results
- Details of how equations/algorithms can be developed
    - practical examples to implement after review
- Methods for experimentation and subsequent evaluation
- A proposed plan to implement
- critique/benefits

1: Initial example and reasoning:

through multiplication and manipulation scores can be polarized to magnify differences using known info about traffic.

- ***bots have****: high number of calls/ req, av/pr ratio ~.9-1, request few resources or paths, high use of top few paths/, short sessions but lots of them / many logins.*
- ***legit users have****: fewer calls/req, av/pr ratio ~.5, request many paths, lower use of top path/total calls, fewer sessions or logins*

This can be used to shape an equation as follows:
Result =
calls(last hour)^2 * (av/pr)^3 * 1/(numPaths^2) * (topPathVisits/calls(last hour))^2

for a typical bot we get a result:
*1000^2 * (.9)^3 * 1/ 10^2 * (.4)^2 =~ 1166*

while a typical legit user would be
*200^2 * .5^2 * 1/50^2 * (1 / 8)^2 =~ 0.0625*

OBS! Data used above is a general case based intuitively on knowledge of current traffic patterns. It is biased/exaggerated to make results closer (and not to make the algorithm appear better than reality).
Other operations can be done to further polarize, as well as, other features used such as top 3 paths, avg paths/session, num logins…
***captcha feedback** could also be used to update results and help ensure any legit users caught out be reevaluated i.e. after captcha result = sqrt(result) and let run for another 10 calls etc.*

Another example for Kerberos implementation for testing could be:
Result =
calls(last hour)^2 * (av/pr)^3 * 1/numPaths^2 * (topPathVisits/calls(last hour))^2

2: Aim, requirements, scope and limitation of the process

Aim

- The central aim of this process is an optimization of the results which the kerberos bot detection program outputs. Subsequently this should improve the accuracy of predicting the source of traffic. This should be established through experiments, data visualization and thorough comparison.
- If successful a number of recommendations can be formed for future implementation.
- Suggestions may also be made as to implementing a more automated approach to apply in the future if applicable.
- Results: Recommendations/ implementable ideas will be the primary result. It is also possible that basic testing and subsequent analysis to prove viability be delivered.
- The goal of analysis would be to show that through polarization, output from kerberos can more clearly be used to detect bot traffic.

Requirements

- The process of how such algorithms are created, should be standardized
- Results must be normalized to fit in the range [0,1]
- Should be implementable in that it is:
    - compatible with kerberos in its current state
    - Uses existing variables/ data available from kerberos data processing
    - New variables and their uses are explained
- Documented clearly

Scope and Limitation
Since the project is nearing a close and this is an additional solution/ recommendation the scope must be kept small and limited. It will serve as a basis for future testing and either no testing, or one simple test to check basic validity of the idea may be run. It will otherwise at the current time not be developed further as part of the thesis project though TML are welcome to use it as a basis for future work as desired.

3: Motivation

The main reasons for pursuing this options as an optimization to Kerberos bot detection are:

- Concern over future feedback
    - Supervised ML classification requires training a model using a number of previously categorized examples of bots and legitimate users. While this can be generated for a limited testset it cannot easily be repeated autonomously
    - Whilst there are options to use Captcha and Attestation, neither of these provide concrete certainty that a given source actually is bot or legitimate. This, since bots have methods of passing such and a real user can also fail such tests.
- Implementable
    - Since Kerberos is up and running and key steps of cleaning and processing data is in place optimization of its outputs may be a good way forward.
    - A body of knowledge exists about which metrics and traffic patterns can be used and exploited optimally in analyzing data in order to detect and subsequently prevent bots.
    - There is a possibility of performing scientific experiments to produce results/data output which can be analyzed and compared with the aim of making insightful findings and conclusions which can be applied and of value to TML
    - If developed, automated processes can be made to make this approach even more powerful and applicable as a long term solution to the bot problem.
- Other options
    - Cleaning and processing a data set of basic server json output into relevant features and metrics is complex and time consuming. In order to use current server data we currently have difficulty in identifying/categorizing enough traffic and even if enough was gathered it is questionable if this is a fruitful/viable use of time.
    - Once combined with concerns over feedback and the relevance of such an approach, even if data was processed this alternative could be deemed less viable than a polarization approach.

4: Theory

The idea behind this method is grounded in the following scientific approaches.

Machine Learning: The key aim in supervised machine learning is to analyze data in such a way that given a new input a subsequent output can be predicted with accuracy. The central mechanism to achieve this ‘training’ is by factoring and weighting the contributions of a number of features (fields of the input) on a model. Subsequently the features of the new input are simply multiplied/operated on according to the trained model. Whilst general approaches to simple data sets exist, with complex problems features must first be developed according to knowledge of the problem/task. In the case of bot detection using Kerberos this knowledge in generating features is linked to known traffic behavior patterns which indicate the source.

Whilst this polarization approach is not typical ML, it can be seen as utilizing several underlying principles including the selection and generation of relevant features and subsequently performing mathematical operations on these values as a means to highlight differences in data and separate results allowing for accurate predictions to be made. One area which may need development later on would be finding ways to automate this approach for analysis of results to provide a basis for ongoing accurate predictions in the future as changes occur.

Neural networking: This process of polarization takes inspiration and directly applies principles found in the ML concept of neural networking. The essence of this is manipulating and combining features to generate new features which provide insight and or better accuracy or findings in data under analysis. Schematically this can be visualized as follows

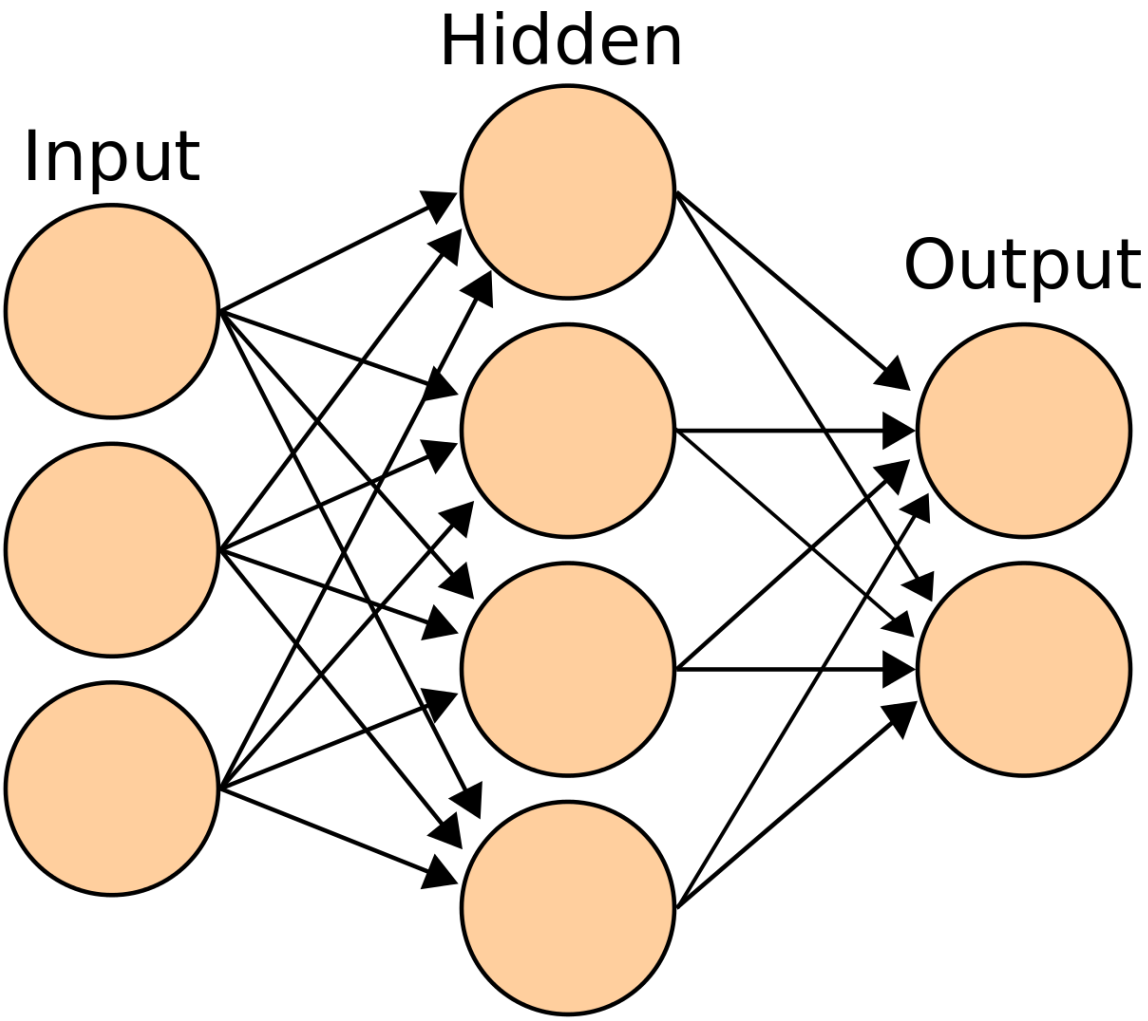

Fig. 1 general artificial neural network diagram(*https://en.wikipedia.org/wiki/Artificial_neural_network*)

The aim of figure 1 is to provide a visualization of the concept. The following example builds on this. Applying the basic conceptual model to make an example may look as follows (Obs. the following example would generate a similar but different structure):

input variables: a, b, c.
performing operations such as: d = a/b, e = a^2, f = a*b*c
which produce outputs of the form: (d + e) * f
Such outputs can be considered manipulated outputs and if combined with knowledge of the input data and weighted accordingly can be extremely powerful

Polarization in science and mathematics: Polarization is an approach used in various scientific fields including chemistry, biology, physics and mathematics. The essence is always the separation or an otherwise contracted or combined object. This is done in order to access or

analyze the contributing factors or to magnify the difference in a set of results to draw clear conclusions.

The mathematical principles and mechanics of this approach are discussed further and in detail in the proof section.

The application here is that Kerberos is likely to produce a relatively homogenous and simple output in its current form. Through manipulation using various mathematical operations such as multiplication and applying exponential powers to values, results can be magnified and separated in a cleaner and more accurate manner.

Proof and mathematical mechanisms

The mathematical principles applied in this algorithm are as follows:

Polarization algorithm:
In order to create seperation and magnification of differences in results the principle is to manipulate data in such a way that for outputs a, b where a < b, we get a<<<b (read a much smaller than b).

This can be achieved through multiplication and using powers though performing a series of operations and combinations of variables. The basic intuition is to take a basic data input where each field has an expected or assumed difference, that is, depending on the source, one variable should be larger than the other.

Ex: A1 < B1 , A2 > B2, A3<B3…. and so on. Based on known metrics in bot vs legitimate traffic.

From this point we can manipulate chosen variables using principles as described below to get a new set of fields of the form:

A1', A2', A3'…An' and B1', B2', B3'…Bn'
where, for every i, Ai' and Bi' it holds that Ai' < Bi' (read: every value in the A series is smaller than its corresponding value in the B series)

Further multiplication or exponential operations on each series of values can lead to the resulting A vs B series being clearly differentiated/SEPARATED.

This approach utilizes the following operations to manipulate data as desired:

Power operations(generally ^2 or ^3 will be used initially):
For two values 0 < a < b (read a is larger than 0 and. smaller than b), it is always the case that:
a^x < b^x , for every x > 1
further (b - a) < (b^x - a^x)
and also (b/a) < (b^x/ a^x)

This implies that through using a power operation on a, b with x the difference between the outputs grows. Thus helping separate results

A basic example would be:
a = 2, b = 3, x = 2
we get that the initial difference b - a = 1

while b^2 - a^2 = 4-9 = 5
and analysis with division shows b/a = 1.5
while b^2/a^2 = 9/4 = 2.25

Division(generally the form 1/x will be used):
For two values a, b, where 0 < a < b (read a is larger than 0 and smaller than b)
1/a > 1/b
This can be used to manipulate values to invert where one value is larger than another. Subsequently this can ensure that all values in a given series are larger than the corresponding in another series to implement the polarization algorithm as previously described.

Series Multiplication:
For two series of values A1, A2,..., An and B1, B2,..., Bn
Where Ai < Bi for every integer i
Multiplication of elements in each series will lead to the output that B1*B2*...*Bn will be larger and will grow faster than A1*A2*...*An. And as such differences in the initial individual input values are exaggerated causing a more polarized output.

If it were productive/ useful to TML, more formal induction proofs could be produced to support this. However, given the logic is relatively simple the above should suffice.

5: Equation/Algorithm development

Standardized steps to follow:
Developing equations can be done using the principles and tools as detailed above.

1. The initial step in this is to identify a number of data fields where an expectation or assumption can be used and exploited
2. Selection of which to use depending on confidence in assumption, purpose/ objective to evaluate
    a. Scraping or account hacking bot to detect.
    b. Inclusion/exclusion of given variable(s) in order to verify with/without dependency on them.
3. Manipulate and enlarge/weight variable values using tools as described
4. Combine variables through multiplication to further enlarge and polarize output. This also raises the likelihood of separated output and reduces dependency on a single variable.
5. Testing should be performed to evaluate output and make adjustments where necessary to things such as the normalization process.
6. Verification: by creating multiple such equations to polarize results, comparisons can be made to calibrate outputs, both reducing dependencies on single variables and providing a higher degree of certainty. It is also vital that results are checked against known traffic results to review the accuracy. Preventing legitimate users being identified as bots is a primary concern.

Assumptions and traffic patterns

In order to apply this algorithm, knowledge of behavioral differences between bots and legitimate users is required. This could be from intuitive knowledge or through testing and feature engineering such as clustering or other machine learning analysis.

Once reasonable and highly likely assumptions are taken they can be used according to the example in section 1 of this report, as well as, theory presented in section 4. Variables must be manipulated according to assumptions, ensuring that for all variables it can be assumed that bots will have a larger value than the same variable for a legitimate user.
Subsequently these variables can be operated on with powerss as described in section 4.
This operation and the subsequent multiplication of a series of such variables will ensure that polarization occurs and traffic can be distinguished.
It should be noted that a good degree of intuitive knowledge of the factors in this process is a requirement in order to effectively apply such measures.

If it were the case that one or two variables of a given input do not meet the assumptions, the likely outcome is that the output score will be output with a mid range score or classified as a legitimate user. This should be tested and monitored to ensure legitimate users are not falsely classified. Issues such as this will also be mitigated through a process of verification, where a few equations are run and calibrated in order to check that output classifications are consistent. This practice should reduce the dependency on one or a small number of variables and as such increase the robustness of this algorithm and approach.

Features/Variables

The following variables are likely to be useful in implementation. These are largely named after current variables used in Kerberos and otherwise appropriate descriptive names have been given.

Pre Existing:

CallVolume - which could be per session/ past hour/ 24 hours or other time period.
A minimum number of calls must be established for metrics to be valid
Get_LowPrice_Availability (GLPA) - Resource path
Get_LowPrice_Price (GLPA) - Resource path
TopPath - the most visited path by given IP
Top_Xnum_Paths - *it may be helpful to analyze either the resource uses of top 3 or 5 paths and or the percentage of total traffic from a given IP address to this given number of paths. This could form the basis of analyzable behavior patterns.*
NumPaths- the number of unique paths visited by IP
Dif_Logins - the count of login attempts using different usernames/passwords by same IP
Other possible
feedbackApproved - the number of times a Captcha or attestation test has been passed by IP

To produce/implement:

score - initial result of algorithm pre normalization score
maxScore - max initial score
minScore - minimum initial score
normalized_score - an updated score post normalization

**At this stage prior to testing it is difficult to say if it would be better to keep scores by ip address which are updated each request or if each request should be scored and logged as part of the processing. It is advised to try both approaches and observe what the effects are on the final output.*

Further potential variable in connection with analytics post scoring:
percentile - where the normalized score ranks among other scored traffic
cutPoint - the point above which traffic is deemed to be a bot

Normalization

As requested and specified in requirements, the following is a method for normalizing results.

The general normalization model is:
Xnorm = (X - Xmin) / (Xmax - Xmin)

Obs. this ensures that:
Xmin = (Xmin - Xmin) / (Xmax - Xmin) = 0/… = 0
Xmax = Xmax-Xmin / Xmax-Xmin = 1
All other results fall between these values.
Further. small results become smaller, while values closer to Xmax remain proportionally closer to their original value and also closer to 1.

Other methods could be explored but this is likely to suffice.

Concerns were identified that bot traffic scores have no upper limit which may lead to problems in normalizing results. There are likely complex solutions to this type of problem where a simple range does not exist. While these could be researched further they will not be done here. A simple implementable solution however is as follows:
Where it can be assumed such as in the example provided that legitimate user traffic should be <1 or have a low value, and bot traffic likely >1000 or have a significantly larger value.Then a sufficiently large max value can be set. In the example provided 1000 would be sufficiently large that any value close to this can be considered a bot with a high degree of certainty. From her scores can be handled as followed prior to normalization:
if (score > maxScore) -> score = maxScore
This will result in all such traffic being normalized as 1 and lead to results being more easily processed. There may be ways of dynamically setting / adjusting this maxScore to provide further optimization.

6: Evaluation, analysis and other recommendations for future implementation

To complement the theory and ideas presented in this document this section will close the report with some suggestions of how experimentation can be visualized and analyzed.

Mapping and analysis

Bell curves: since it is important that results are analyzed in relation to all scores not by their values themselves since following the normalization step they have been resized in a relational way. As such, the median value and percentile placement are the most valid ways of evaluation. This could be visualized using standard deviation bell curves, placing results accordingly to map the range and density. This may be helpful in choosing an appropriate cut off point, above or below which traffic is deemed as a bot.

polarisation evaluation
The intention of this algorithm/solution is that data should be output in such a way that a graph charting the values should be of a u shape. This implies high density of outputs close to 0 followed by a drop and very few outputs in a certain range and subsequently another range with high density which may be close to 1, or above a certain value. If such an output can be obtained through this process then it can be considered successful and useful. Further, if in combination with checks, it can be verified that outputs above a certain value correspond to the source being a bot then this method of evaluation may be an insightful and powerful method of bot detection.

Accuracy evaluation- Following analysis and choosing a cut off point. Results should be checked against real traffic. A similar test set could be used as described in the thesis report where traffic is manually gathered based upon known behavior patterns. When this algorithm is run, its output can be checked to see if predictions are accurate or not. Naturally the aim would be to produce the highest accuracy % possible. Special attention should also be given to check that no, or very few legitimate users are classified as bots.

Recommendations:
It is believed by the author that were this algorithm implemented the results would be as follows. It is likely to see a sliding scale where all new traffic starts close to 0 then bots move up toward 1 while legitimate traffic remain steady at low value. In accordance with this a number of requests must be appropriately decided upon from which to begin using scores. A figure around 50 would likely be appropriate, but this may need experimentation and adjusting.

As with machine learning, feature identification and analysis is the central and valuable element in the success of the solution. Furthermore, the more features which can be developed where concrete assumptions can be made, the more equations can be added to this approach, and the more accurate and reliable predictions will become. It is also possible to combine this approach with linear regression and use the equations above to generate features to input into such an algorithm. Alternatively, clustering or analysis to generate features could be applied in order to generate features which can be used in this polarization approach.

This approach is not typical machine learning and questions remain over how effective this approach would be. Maintenance and manual gathering of test sets may make it a time costly approach to development and questions of which period to retrain/shape the equations upon remain. It may also require too much specialized and particular knowledge. Nonetheless, the author puts it forward as an interesting option for experimental development which has potential to play a part in effective bot detection.

Appendix C: Machine Learning Results

Results

This program was used in order to provide recommendations to TML for future development with Kerberos. The test data was carried out to find the degree of accuracy which ML-algorithms can achieve, and the latency that processing lead to. In order to achieve optimal accuracy results and processing times, two algorithms, ADAM and Stochastic Gradient Descent (SGD), were selected from the PyTorch.optim library [23] for a comparison to form the basis for recommendations. The used platform to get the graph below was Python, mentioned more detailed in section 3.6.

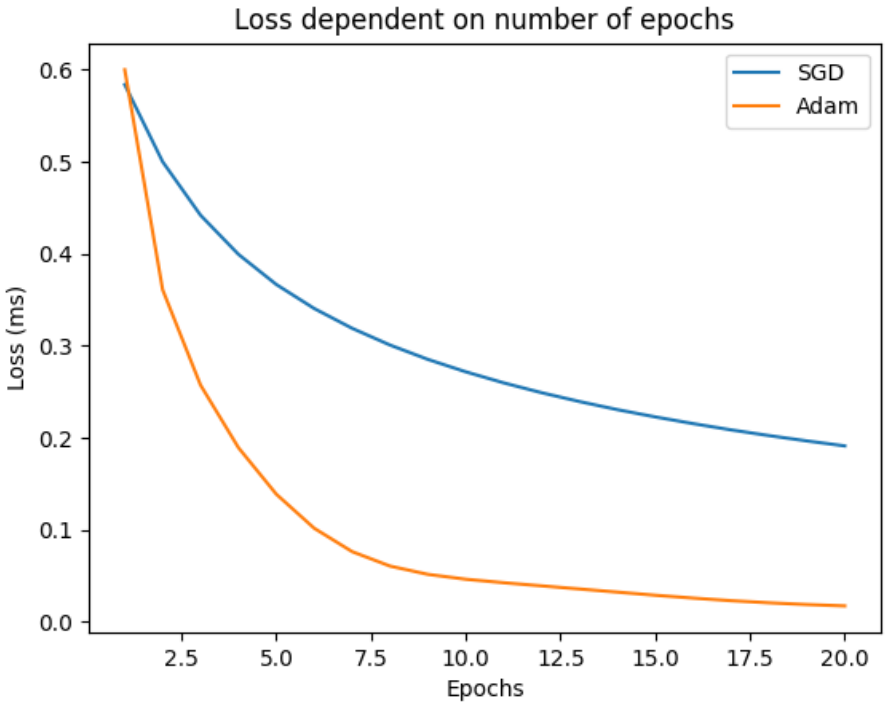

Figure .1: SGD versus Adam.

Analysis

Whilst experiments were performed using a generic dataset, the results produced were conclusive. Both algorithms compared had a similar training time and a rapid prediction time. However, while SGD had a good and steady reduction of the cost function, its model and subsequent accuracy in predictions, the ADAM algorithm outperformed it. Likely due to more advanced optimizations, such as a

dynamically adjusted learning rate, which enables it to begin with an initial large learning rate, meaning the model is significantly adjusted in a training round. As the cost gets closer to 0, this learning rate is reduced in a fine tuning stage, due to this, it can 'learn' in an accelerated manner before slowing down where appropriate. Subsequently, the performance of the ADAM algorithm is better and it is recommended as an implementable option that TML should go forward with a logistic regression ML solution.

In line with experiments, and general ML theory, another recommendation is that running 5-7 epochs of training is sufficient and that any reduction in cost and subsequent prediction accuracy is minimal hereafter and the time vs gain cost is not worthwhile, especially when processing large data sets. As mentioned the use of such a model is particularly appropriate due to the rapid output of predictions once a model has been trained.

However, consideration must be taken for the fact that ML training takes a significant amount of time on larger training sets, which would be the case when server traffic is being analysed. This can be seconds or minutes and thus it is not appropriate to train a model each incoming http-request. Rather, the gathering of test sets and subsequent training would be more efficiently and appropriately handled by doing so periodically. This is largely dependent on how frequently and significantly bot behavior changes. Ultimately this should be decided by TML, as anything from hourly to monthly might be a suitable time frame.

References

[1] Anaconda. *Anaconda packages*. URL: https://anaconda.org/anaconda/python. Accessed: 11.05.2022.

[2] Brown, Kyle and Doran, Derek. "Contrasting Web Robot and Human Behaviors with Network Models". In: *Journal of Communications* 13.8 (2018). URL: http://www.jocm.us/uploadfile/2018/0720/20180720041349446.pdf.

[3] Center, The International Risk Governance. *The Governance of Decision-Making Algorithms*. URL: https://infoscience.epfl.ch/record/261264/files/IRGC. Accessed: 14.05.2022.

[4] Cloud, Google. *Cbt tool reference*. URL: https://cloud.google.com/bigtable/docs/cbt-reference. Accessed: 20.05.2022.

[5] Cloud, Google. *Cloud Bigtable*. URL: https://cloud.google.com/bigtable. Accessed: 23.05.2022.

[6] Cloudflare. *What is a web crawler?* URL: https://www.cloudflare.com/en-gb/learning/bots/what-is-a-web-crawler/. Accessed: 17.02.2022.

[7] Cramer, J. S. *The origins of logistic regression*. 2002. URL: https://papers.tinbergen.nl/02119.pdf. Accessed: 30.05.2022.

[8] Datadome. *CAPTCHA vs. reCAPTCHA: Whats the difference?* URL: https://datadome.co/resources/captcha-vs-recaptcha-whats-the-difference/. Accessed: 25.02.2022.

[9] DATAQUEST. *Commonly Used Bot-mitigation Techniques*. URL: https://www.dqindia.com/commonly-used-bot-mitigation-techniques/. Accessed: 20.02.2022.

[10] Derek Doran, Swapna S. Gokhale. *Detecting Web Robots Using Resource Request Patterns*. International Conference on Machine Learning and Applications, 2012.

[11] Fireship. *Redis*. URL: https://www.youtube.com/watch?v=G1rOthIU-uo. Accessed: 20.05.2022.

[12] Garcia, Sebastian. *Botnets Behavioral Patterns in the Network. Analysis and Monitoring and Detection and Blocking*. ResearchGate, 2014. URL: https://www.researchgate.net/publication/282074318_Botnets_Behavioral_Patterns_in_the_Network_Analysis_Monitoring_Detection_and_Blocking. Accessed: 02.06.2022.

[13] Gentleman, R. and Carey, V.J. *Unsupervised Machine Learning. In: Bioconductor Case Studies. Use R!* Springer, 2008, pp. 137–157. ISBN: 978-0-387-77240-0. URL: https://doi.org/10.1007/978-0-387-77240-0_10.

[14] Imperva. *Bad Bot Report 2021*. URL: https://www.exclusive-networks.com/se/wp-content/uploads/sites/25/2020/12/Imperva-Bad-Bot-Report-2021.pdf. Accessed: 20.02.2022.

[15] Instana. *Instana - Enterprise Observability*. URL: https://www.instana.com. Accessed: 20.05.2022.

[16] Kaspersky. *What is a honeypot?* URL: https://usa.kaspersky.com/resource-center/threats/what-is-a-honeypot. Accessed: 19.04.2022.

[17] Loeber, Patrick. *PyTorch Beginner Tutorial*. URL: https://www.python-engineer.com/courses/pytorchbeginner/. Accessed: 23.05.2022.

[18] Maalouf, Maher. "Logistic regression in data analysis: an overview." In: *International Journal of Data Analysis Techniques and Strategies* 3.3 (2011), pp. 281–299. DOI: https://www.researchgate.net/profile/Maher-Maalouf-2/publication/283211221_IJDATS_Logistic_Regression_Rare_Events/data/562e0fb508ae518e34827577/IJDATS-Logistic-Regression-Rare-Events.pdf. Accessed: 03.06.2022.

[19] Mahdieh Zabihimayvan, Reza Sadeghi, Rude, H. Nathan, and Doran, Derek. *A Soft Computing Approach for Benign and Malicious Web Robot Detection*. Elsevier, 2017.

[20] Maulud, Dastan Hussen and Abdulazeez, Adnan Mohsin. *A Review on Linear Regression Comprehensive in Machine Learning*. 2020. URL: https://scholar.google.se/scholar_url?url=https://jastt.org/index.php/jasttpath/article/download/57/20&hl=sv&sa=X&

ei=tLuZYtSrM8LZmQGh5pWYBQ&scisig=AAGBfm3XdSh-dRBH_5UQ5Jj4WdX0m_JkzQ&oi=scholarr. Accessed: 03.06.2022.

[21] Nateski, Vladimir. *An overview of the supervised machine learning methods*. URL: https://www.researchgate.net/profile/Vladimir-Nasteski/publication/328146111_An_overview_of_the_supervised_machine_learning_methods/links/5c1025194585157ac1bba147/An-overview-of-the-supervised-machine-learning-methods.pdf. Accessed: 03.06.2022.

[22] Pozzana, Iacopo and Ferrara, Emilio. *Measuring Bot and Human Behavior Dynamics*. 2020. URL: https://www.frontiersin.org/articles/10.3389/fphy.2020.00125/full. Accessed: 02.06.2022.

[23] PyTorch. *pytorch.optim library*. URL: https://pytorch.org/docs/stable/optim.html. Accessed: 11.05.2022.

[24] Rouaud, Mathieu. *Probability, Statistics and Estimation*. 2013. URL: http://www.incertitudes.fr/book.pdf. Accessed: 30.05.2022.

[25] Rovetta, Stefano, Suchacka, Grazyna, and Masulli, Francesco. *Bot recognition in a Web store: An approach based on unsupervised learning*. 157 (2020) 102577. Elsevier, 2020.

[26] SAS. *Machine Learning - What it is and why it matters*. URL: https://www.sas.com/en_us/insights/analytics/machine-learning.html. Accessed: 20.02.2022.

[27] Science, Towards Data. *What is Feature Engineering*. URL: https://towardsdatascience.com/what-is-feature-engineering-importance-tools-and-techniques-for-machine-learning-2080b0269f10. Accessed: 30.05.2022.

[28] Scikit-learn. *Sklearn wine dataset*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_wine.html#sklearn.datasets.load_wine. Accessed: 11.05.2022.

[29] Techopedia. *HTTP header*. URL: https://www.techopedia.com/definition/27178/http-header. Accessed: 25.02.2022.

[30] University, Standford. *Machine Learning*. URL: https://coursera.org/share/f1160ba8e2f72c73b50cafff891ea67e. Accessed: 25.02.2022.

[31] Zyte. *Web Scraping*. URL: https://www.zyte.com/learn/what-is-web-scraping. Accessed: 16.02.2022.

TRITA-EECS-EX-2022:355