Machine learning model for bot and applications detection on Web Application Firewall data

Simone Palladino

August 2023

Contents

1 Abstract

The research question addressed by this thesis is whether the task of recognizing offline bots is (1) characterized by inherent differences between the behavior of legitimate users and that of automated software agents and (2) learnable by standard machine learning methods, such that unsupervised analysis is able to reveal meaningful and interesting information [4].

In this thesis, a new approach for addressing the problem of Web robot detection from Web-server access logs is introduced, in particular HTTP requests collected by a WAF. More specifically, a concept of session is first defined for grouping HTTP requests, then it is studied which features can characterize bots and which ones can characterize humans, and finally clustering alghoritms are applied to disgtinguish programmatic from non-programmatic sessions.

The theoretical result of this research is a success and opens the door for a concrete implementation of a solution to fight bots in online traffic.

2 Acknowledgments

I would first like to thank my thesis advisor Prof. *Luca Oneto* of the Computer Engineering programme at University of Genoa, for his immense availability, for his valuable lectures necessary for the accomplishment of this work, and for getting me in touch with the company where I developed my thesis, aizoOn Technology Consulting.

I would also like to thank the experts from aizoOn who were involved in this research project, my referent *Federica Bisio* and my colleagues *Ivan Russo* and *Daniele Ucci*, for their passionate participation and useful inputs.

I would also like to acknowledge my friend, colleague and mentor Dr. *Paolo Coletti*, Researcher in Computational Finance at the Faculty of Economics and Management of Free University of Bolzano, as the second reader of this thesis, and I am grateful to him for his valuable comments on this work.

I take this opportunity to express gratitude to my friend and colleague *Roberto Gnisci*, for sharing this entire college career with me from the first day to the last one, for being a great study partner and for always being a loyal friend.

Finally, I must express my very profound gratitude to *my parents* for providing me with unfailing support and continuous encouragement throughout my years of study .

Thank you.

3 Introduction

3.1 General context

In the current era of wide proliferation of the Internet and mobile technologies, many of our daily activities are moving to virtual platforms. Social life, communication, entertainment, shopping or searching for any kind of information are just a few examples. Contextually, web analytics and online marketing tools are increasingly being used to gain a competitive advantage in this newly outlined market. The rapid development of these technologies has caused the increase in web bot traffic, thanks to which advanced web-based applications can provide users with up-to-date, accurate and personalized services.

A Web bot, also called an Internet robot, Web agent, or intelligent agent, is a software tool that performs specific tasks on the Web, usually autonomously, following the hyperlink structure according to a specific algorithm. Many bots are benign and useful, such as search engine crawlers, shopping bots to gather information for product search engines or price comparators, link checkers to help website administrators detect broken and blacklisted links, or feed fetchers that transport website content to mobile applications. However, the activities of some bots raise concerns about the ethics and privacy of users, such as in the case of email address collectors, spambots, or bots that steal content. In addition, some bots are undoubtedly malicious: hacking bots used to steal sensitive data or inject malware, bots to generate click fraud in pay-per-click advertisements, or malware used for distributed denial of service (DDoS) attacks are just a few examples.

A significant portion of overall web traffic is generated by bots, many of which have clearly malicious objectives. Malicious bots tend to obscure their true identities by assuming user agent strings typical of legitimate web browsers and ignoring the robots.txt file that contains the website access rules for bots. This makes it difficult to identify bot traffic on web servers. In practice, relatively simple bot detection techniques are used, such as comparing an IP address or user agent string with a blacklist of known bot data or investigating a few keywords indicative of a bot in the user agent string. A request stream can also be tested for some atypical statistical characteristics, such as extremely short inter-arrival times, but these tests are often ineffective because bots tend to mimic human behavior[1].

As a result, there is a growing need to distinguish robots from humans when analyzing HTTP requests coming to web servers of interest, for a variety of reasons:

- Typically, analysis of a web server's HTTP interactions provides a wealth of information about the functionality, usability, design, and popularity of hosted sites and content. The basic premise is that HTTP interaction patterns reflect the choices end-users (customers) make when navigating within a Web site. In addition, reliable quantitative information that captures end-users' browsing choices is the basis for pay-per-click advertising,

one of the most popular and successful Internet business models. According to company reports [7], pay-per-click advertising generates 80% of Google's revenues and is adopted by other search engine giants. However, there is growing concern that this business model could be seriously damaged by click fraud, which involves, among other things, unwanted or malicious repetitive retrieval of advertising links by web robots that do not originate from known or stable IP addresses or Internet domains. Click fraud involves higher rates paid by advertisers for essentially useless HTTP requests and has raised concerns about the sustainability of the search engine business model, leading Google's CFO to declare that "click fraud is the 'greatest threat' to the Internet economy!".

- Automatic identification of malicious web robots based on their behavior, rather than their (possibly transient) IP addresses, may also be useful in an effort to address the problem of referral spam, which affects search engine ranking results.

- There is also concern that crawler-induced traffic accounts for a large portion of total HTTP traffic and that crawler activity may cause degraded performance of busy Web servers and network infrastructure, as well as increased error rates in Web caches. These concerns are supported by the few published articles that have investigated the behavior of web robots by analyzing the impact of known crawlers on different web servers [2].

- Finally, there is the need to distinguish between crawler and human traffic in cases where it is important to protect information of a temporary or sensitive nature, posted on intranet websites, from crawlers accidentally discovering such information and posting it via search engine databases.

To address the concerns mentioned above, it is necessary to be able to isolate the behavior of robots from that of the general population of web users (humans). Distinguishing web robots from humans will help marketing companies obtain more accurate statistics about the impact of online advertising and the interaction that real customers have with e-commerce sites. It will also help web administrators estimate the real side effects of crawler activity on web server performance. Finally, it can provide a basis for the development of intelligent access control systems that will protect websites from aggressive or unwanted crawlers[3].

3.1.1 Problems

The detection of web agents (bots) on the web presents several problems and challenges:

- **Camouflage of Bots**: Bots often try to camouflage themselves as human users by using fake or modified user agents. This makes it difficult to detect their true identity and distinguish between bots and real users.

- **Dynamic IPs**: Some bots use dynamic IP addresses or proxy server networks to hide their origin. This complicates detection, as IP addresses cannot simply be blacklisted.
- **False Positives and False Negatives**: Detection techniques can sometimes generate false positives (misclassifying human users as bots) or false negatives (not correctly detecting some bots). Finding a balance between accuracy and coverage is a challenge.
- **Evolution of Bots**: Bots can be updated or modified to circumvent existing detection techniques. This race between bot innovation and countermeasure development is an ongoing problem.
- **Benign Bots**: Not all bots are harmful. Some perform useful tasks such as search engine indexing. Distinguishing between useful and harmful bots is a challenge.
- **Complexity of Behavio**r: Bots can mimic human behavior in increasingly sophisticated ways, such as generating random requests or following complex navigation paths.
- **Privacy and Ethics**: Disclosure techniques could violate users' privacy because they involve monitoring their online activities. It is important to strike a balance between bot disclosure and respecting privacy.
- **Real-Time Detection**: Identifying bots in real time during active sessions can be complex, as it requires quick decisions based on a limited number of observations.
- **Traffic Variation**: Web traffic varies over time, both in terms of volume and behavior patterns. Detection techniques must be able to adapt to these variations.
- **Resources and Performance**: Implementing detection techniques can require significant resources and could affect web server performance.

In summary, web bot detection is a complex challenge that requires the use of advanced approaches and strategies to address the multiple dimensions of this problem.

3.2 Specific context

The problem domain are logs containing HTTP requests captured by a WAAP called Mithril, a product developed by the company aizoOn Technology Consulting.

aizoOn Technology Consulting is a global technology consulting company focused on innovation, which is structured in market areasincluding defense, energy, finance, transportation, industrial goods, and cybersecurity.

Mithril

Mithril is a web application security service that offers several solutions to protect websites from threats and attacks. Some of its main features include:

- **Mithril Overview Dashboard**: Provides an overview of website data in a single view, including blocked attacks and alerts. It also shows the total number of HTTP requests handled, blocked or served from the cache in a customizable time interval.
- **Blocked Requests View**: Allows you to view all blocked requests and alert events in a single view. You can filter and analyze each individual HTTP request, broken down by geolocation, device, browser type, or operating system type.
- **Data enrichment**: Enriches each request or alert with additional information about geolocation, device, session ID, device type, operating system type, and other details. It can correlate events and information such as users logging in from different geographical locations or with the same session ID.
- **Mithril Solutions**: Offers several cloud-based solutions, including web application firewall (WAF) deployment, DDoS attack protection, and API security.
- **Caching and Always Online**: Keeps the website always online and provides content to users even when the website is down. Uses caching to keep content available to users.
- **DDoS Mitigation and Protection**: Protects the website from DDoS attacks, keeping the website safe and available to users.
- **Virtual Patching**: Identifies and resolves vulnerabilities in the website and web application by patching without changing the source code.

The goal of this thesis is to develop an offline machine learning model capable of distinguishing programmatic traffic (bots) from non-programmatic traffic (humans) by only analyzing HTTP requests. The thesis aims to demonstrate the satisfiability of the goal, ignoring real time performance constraints.

3.2.1 Problems

Developing a machine learning model to distinguish programmatic from non-programmatic traffic using only a dataset of HTTP request logs presents several challenges and problems:

- **Data labeling**: Obtaining a properly labeled dataset can be difficult and require detailed analysis of HTTP request logs to distinguish between programmatic and nonprogrammatic traffic.

- **Pattern variability**: Programmatic traffic patterns can vary widely in terms of behavior and patterns, making it difficult to identify a single distinctive pattern.
- **Noise in the dataset**: HTTP request logs may contain noisy or irrelevant data that could adversely affect the model's ability to learn correctly.
- **Relevant features**: Identifying relevant features or variables in the dataset to discriminate between programmatic and non-programmatic traffic may require extensive analysis and domain expertise.
- **Changing concept**: The concept of programmatic traffic may evolve over time with new technologies and approaches, making an adaptable model necessary.
- **Business desires**: Criteria for distinguishing programmatic from non-programmatic traffic may vary according to business objectives, making a balance between accuracy and fault tolerance important.
- **Generalization**: The ability of the model to generalize correctly to new data could be challenged by the variety and dynamism of Web traffic.
- **Overfitting**: A model might overfit noise or overly specific training data, compromising its ability to generalize over new data.
- **Privacy**: Analysis of HTTP requests could involve sensitive or personal data, leading to concerns about privacy and regulatory compliance.

3.3 State of the art

In recent years, advances in bot technologies to perform deception in popular Web-based services have prompted research into specialized methods for bot detection, targeted at specific applications. This problem affects most Web sites and has direct economic implications for an online business. Many companies rely on commercial cybersecurity services or have studied their own countermeasures. However, academic literature can only account for methods that have been publicly disclosed. In contrast, many of these are trade secrets or use proprietary data, which does not contribute to the state of the art as it does not allow for replication, experimentation, third-party validation or further development.

In this paper, the focus will be on known methods described in academic sources. Some studies investigate bot behaviors at the network level [8]. On the other hand, many techniques are based on analyzing statistical differences in the behavioral patterns of bots and humans regarding application-dependent features. For example, [9] aims to detect fraud-prone apps in Google Play search ranking by analyzing a set of relational, behavioral, and linguistic features. It proposes click fraud detection in pay-per-click advertising by identifying duplicate clicks. Features derived from the analysis of user profiles and product

reviews are used to detect spammer groups and manipulation attacks in online recommendation systems.

A large number of bot detection approaches use supervised classification techniques to detect spambots, blog bots, shopping bots, and click fraud. Machine learning techniques have also proven effective in identifying fraud in online social networks [10]. These approaches, developed for specific Web services, rely on application-specific high-level features, which limits their application to only certain types of Web sites. Challenges include relevant feature selection, data collection and labeling, adaptation to new traffic patterns, and efficient implementation on different Web sites. Approaches based on the HTTP characteristics of traffic observed on Web servers without interfering with the Web site or server software have proven effective in discriminating between bots and legitimate users [11]. The difference in traffic patterns of bots and humans has been widely investigated, typically based on data recorded in Web server access logs.

However, all supervised learning approaches share a common disadvantage related to the difficulty of preparing a reliable training dataset, particularly in assigning accurate class labels to disguised robot sessions [4]. This disadvantage does not affect unsupervised learning, which consists of learning the intrinsic properties of data from unlabeled training samples.

HTTP feature-based approaches employ traffic pattern analysis or machine learning techniques. They often aim at offline bot detection, i.e., categorization of historical HTTP data over entire sessions completed on the server. Traffic pattern-based approaches use statistical properties of HTTP request characteristics, such as types of resources downloaded or inter-arrival times, to build probabilistic session models [3].

A limited fraction of bots can be identified by syntactic analysis of HTTP fields extracted from log entries, i.e., by examining robots.txt file access in sessions, inspecting specific keywords in user agent strings, or comparing IP addresses with a blacklist. This simple approach allows detection of only well-known and cooperative bots while remaining blind to new or evolving ones. Because of these limitations, more sophisticated approaches for offline bot detection have been proposed, including traffic pattern analysis and learning analytics.

Traffic pattern analysis looks for known differences in the interaction style between bots and legitimate users. Analytic learning, on the other hand, does not look for known patterns, but uses statistical or machine learning techniques to learn rules from browsing data and incorporate them into a probabilistic or formal machine learning model. Examples of probabilistic models include Bayesian approaches [12], as well as Markov models based on patterns of request arrival and types of resources requested.

Approaches exploiting machine learning techniques differ in the selection of relevant session features, the techniques used, and the methodology of session

extraction and experimental classifier evaluation. Supervised session classifiers have been implemented with decision trees [13], neural networks [14], logistic regression [15], support vector machines [14] and ensemble methods [15], among others.

Some work has been developed on session clustering to isolate bots. However, some of the unsupervised methods such as DBSCAN (Density-Based Spatial Clustering of Applications with Noise) have left some sessions unclustered and treated as noise [16]. Approaches such as SOM (Self-Organizing Map) and Modified ART2 (Modified Adaptive Resonance Theory 2) were used to gain a better understanding of the types and distribution of Web clients [17].

The challenge of detecting bots in real time has been addressed by only a few studies [1]. Approaches developed to detect camouflaged bots require adding traps or trap files to the site. Approaches that use HTTP-level characteristics of observed traffic on Web servers without interfering with the site or server software have been shown to be efficient discriminators between bots and legitimate users [6].

3.4 Proposed solution

The proposed solution aims to develop an unsupervised machine learning model, since the data domain on which it must operate are unlabeled HTTP request logs.
The approach involved:

- Designing the session concept and implementing an algorithm capable of grouping HTTP requests into these sessions.
- Identify meaningful features to distinguish human traffic from programmatic traffic.
- Processing the dataset to extract the designed features.
- Study the extracted features to identify any correlations.
- Choose and implement an unsupervised machine learning algorithm.
- Verify and optimize the result.

The result is a K-Means clustering model that, from a session, classifies the session according to the cluster it belongs to, then labels the cluster according to the dominant user agents in the cluster.

3.5 Innovative content

The innovative content of this work is a model capable of classifying bot traffic on a GENERAL domain. The state of the art offers solutions for specific domains,

for example, a model for a specific e-commerce [4], or a model for a specific social-network, or a model for a specific game. In the case studied, however, the domain is a dataset composed of HTTP requests collected by a firewall that defends a wide range of different websites, including e-commerce, banking, entertainment sites and so on. The model therefore has to recognize more general and less heuristically definable patterns, having to recognize different bots with different goals and behaviors from each other.

In addition, the model is of the unsupervised type, as the dataset contains no labels. This approach is advantageous in that much more data collected by the firewall can be exploited, as there is no need to manually insert labels to the collected data.

Last, but not least, the model does not overweight the user-agent as a feature, unlike other solutions present in the state of the art making it vulnerable to bots with obscured or masked user-agents, but it exploits the user-agent to label the defined clusters, instead giving weight to 20 other more meaningful features extracted from the sessions.

4 The development of the model

4.1 Problem

4.1.1 Problem definition

The environment considered are traffic logs gathered by a WAAF, so all traffic is based on the HTTP protocol, defined at the application layer of the ISO-OSI stack. Interaction via the HTTP protocol occurs between Web clients and Web servers. A single client-server transaction involves a request, issued by the client, and a response sent by the server. The request indicates one of the request methods, the most common of which are GET, POST, and HEAD, a header, containing meta-information including a user agent string to identify the client, a URI (Unified Resource Identifier) and a body. The response consists of a status line describing the result of the transaction by means of a status code and a verbal description, a header and a body containing the requested data.

Web clients are typically Internet browsers or mobile apps used by human users, but they can also be automated agents. In the case of a browser, interaction with the server occurs by downloading consecutive Web pages, typically linked together. For each new page, the browser generates a sequence of HTTP requests, first for the page description file and then for embedded objects, such as images. In the case of intelligent agents, however, a sequence of requests is not limited by the structure of the Web site, since the bots tend to traverse the site according to a given strategy, e.g., in depth or breadth, and can request only selected types of server resources.

HTTP is a stateless protocol that defines no permanent server-client connection. When it is necessary to maintain a session, such as during a customer's visit to a Web store, other mechanisms, such as cookies, are used. However, unlike HTTP transaction data, such additional information is not necessarily standardized among Web servers, nor easy to obtain by Web site administrators. Therefore, given only a sequence of HTTP requests observed on the server, a common practice is to define a session.

The goal is to design a method that is as universal as possible, one that would not require analysis of Web site semantics, changes in Web site structure, or a special data collection process on the server or client side. [1]

4.1.2 Formalization

Given an unlabeled dataset of HTTP request logs, the problem at hand is to develop an unsupervised machine learning model that can accurately distinguish between human traffic and programmatic (bot) traffic analyzing HTTP requests.

The following steps are involved in addressing this problem:

- **Session Definition and Grouping Algorithm**: Define and implement a session concept that groups a sequence of HTTP requests into sessions. Develop an algorithm capable of accurately identifying and grouping these

requests based on common attributes such as IP addresses, user agent strings, and time intervals between requests.

- **Feature Identification**: Identify relevant and meaningful features that can effectively differentiate between human and programmatic traffic within the context of these sessions. These features may include characteristics such as request frequency, types of resources accessed, request methods (GET, POST, HEAD), and other relevant parameters.
- **Feature Extraction**: Process the unlabeled dataset of HTTP request logs to extract the designed features from each session. This step involves transforming the raw log data into a format suitable for analysis and machine learning.
- **Feature Analysis**: Conduct an in-depth study of the extracted features to identify potential correlations and patterns that distinguish human traffic from bot traffic. This analysis aims to uncover inherent differences in behavior that can be exploited for accurate classification.
- **Unsupervised Machine Learning Algorithm**: Choose and implement an unsupervised machine learning algorithm to cluster sessions based on the identified features. The algorithm's objective is to group similar sessions together while maintaining a clear distinction between human and programmatic traffic.
- **Model Verification and Optimization**: Evaluate the performance of the clustering model by assessing its ability to correctly classify sessions into clusters. Fine-tune the model parameters and clustering configuration to optimize its accuracy in distinguishing between human and bot traffic.

4.1.3 Dataset

The dataset is composed by traffic logs collected by a WAAP (a sort of firewall), with the following header:
All the following examples are fictitious, they do not contain real data for privacy reasons

Log Entry Details

- **Timestamp** - The log's timestamp
  - *for example* Mar 24, 2023 @ 17:07:41.000
- **index** - Elastic search index
  - *for example* .ds-waap-logs-2023.03.23-000344
- **customer** - The name of the waap's customer

  - *for example* McDonald
- **Geoip.city_name** - The name of the client's city (ip)
  - *for example* Rome
- **Geoip.continent_name** - The name of the client's continent (ip)
  - *for example* Europe
- **Geoip.country_code2** - The country code of the client (ip)
  - *for example* IT
- **Geoip.country_name** - The country name of the client (ip)
  - *for example* Italy
- **Geoip.region_iso_code** - The iso_code of the client(ip)
  - *for example* IT-RM
- **Geoip.location** - The coordinates of the client (ip)
  - *for example* POINT(12.6843 56.1188)
- **Nodename** - The elasticsearch's node
  - *for example* ip-10-0-4-154.eu-central-1.compute.internal
- **Real_client_ip** - The client's ip
  - *for example* 134.30.168.24
- **service-id** - ID of a customer service
  - *for example* 54fd94af-c2b7-492a-bd6d-617f36bfd0b2

Transaction Details

- **transaction.producer.components** - Components of the transaction producer
  - *for example* OWASP_CRS/3.4.0-dev
- **transaction.producer.secrules_engine** - Status of the security rules engine
  - *for example* Enabled
- **transaction.request.body** - Request body content
  - *for example* (empty)

- **transaction.request.headers_json** - Request headers in JSON format,
  - *for example* {"user-agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/111.0.0.0 Safari/537.36", "X-Forwarded-Proto": "https", "sec-fetch-site": "same-site", "access-control-request-headers": "authorization,storecode", "accept": "/", "access-control-request-method": "GET", "origin": ".customername.it", "sec-fetch-mode": "cors", "X-Amzn-Trace-Id": "Root=1-641dcacd-65d99e7800477fab69c2742e", "Host": ".customername.it", "X-Forwarded-Port": "443", "referer": ".customername.it/", "X-Forwarded-For": "44.243.254.234", "sec-fetch-dest": "empty", "accept-encoding": "gzip, deflate, br", "accept-language": "it-IT,it;q=0.9,en-US;q=0.8,en;q=0.7"}
- **transaction.request.headers_map.content-lenght** - Content length of the request headers map
  - *for example* 593
- **transaction.request.headers_map.host** - Host in the request headers
  - *for example* host.cloud.customer.it
- **transaction.request.headers_map.origin** - Origin in the request headers
  - *for example* -
- **transaction.request.headers_map.referer** - Referer in the request headers map
  - *for example* -
- **transaction.request.headers_map.user-agent** - User agent in the request headers
  - *for example* Amazon-Route53-Health-Check-Service (ref 293dce71-3b67-498f-bd2c-4564e152a418; report amzn.to/1vsLAci)
- **transaction.request.headers_map.x-forwarded-for** - X-Forwarded-For in the request headers map
  - *for example* 44.253.252.234
- **transaction.request.headers_map.x-forwarded-port** - X-Forwarded-Port in the request headers map
  - *for example* 443
- **transaction.request.headers_map.x-forwarded-proto** - X-Forwarded-Proto in the request headers map

  - *for example* https

- **transaction.request.http_version** - HTTP version of the request
  - *for example* 1.1
- **transaction.request.method** - HTTP method of the request
  - *for example* GET
- **transaction.request.uri** - URI of the request
  - *for example* /v1/craftsmen?storeCode=001
- **transaction.request.uri_path** - Path of the URI in the request
  - *for example* /v1/craftsmen
- **transaction.response.body** - Response body content
  - *for example* (empty)
- **transaction.reponse.headers_json** - Response headers in JSON format
  - *for example* {"X-waap-Webapp-Group": "pub", "X-waap-Upstream-Latency": "5", "ETag": "W/2̈-vyGp6PvFi4sFtPoIWeDReyIC8̈", "Connection": "keep-alive", "X-Powered-By": "Express", "Content-Type": "application/json; charset=utf-8", "Content-Length": "2", "Date": "Fri, 24 Mar 2023 16:07:41 GMT", "X-waap-Proxy-Latency": "4", "Server": ""}

4.2 Methodology

4.2.1 Technologies and Subjects involved

This thesis is an application of machine learning to computer security. In order to develop the model and identify significant features, an in-depth study of cybersecurity was required, particularly the **HTTP protocol**, **WAF Mithril**, and the typical behavior of **various types of existing bots**.

The code to analyze and work with the data was written entirely in **Python**, through the following libraries: *Scikit-learn* for machine learning, *Pandas* and *NumPy* for handling data and *Seaborn* for plots.

Hypertext Transfer Protocol

The Hypertext Transfer Protocol (HTTP) is the foundation of communication between clients and servers on the Internet. It is an application-layer protocol that defines rules and conventions for the transfer of hypertext resources, such as Web pages, images, and files, between a client, usually a Web browser, and a Web server.

HTTP-based interactions follow a request-response model: the client sends an HTTP request to the server specifying the desired action (e.g., obtaining a

Web page) and the server responds by providing the requested resource or a status message. HTTP requests and responses are composed of a series of fields and headers that contain important information, such as content type, data length, and other metainformation.

A relevant aspect of the HTTP protocol is that it is stateless, which means that each request is processed separately, with no memory of previous requests. This has led to the widespread use of mechanisms such as cookies and sessions to maintain state and consistency between successive requests.

The HTTP protocol consists of a number of fields and headers, each with a specific description and functionality:

- **Method**: This field specifies the action to be performed on the server's resources. Common methods include GET (get a resource), POST (send data to the server), PUT (update a resource), and DELETE (remove a resource).
- **URI (Uniform Resource Identifier)**: The URI identifies the requested resource. It can be an absolute or relative path within the domain.
- **HTTP Version**: Indicates the version of the HTTP protocol used in the request or response, e.g., HTTP/1.1.
- **Headers**: Headers contain additional information about the request or response, such as content type, date and time, and other meta information.
- **Message Body**: This part carries the actual data of the request or response, such as form data to be sent to the server or content to be returned to the client.
- **Status Code**: This field is included in the server response and indicates the result of the request. For example, status code 200 indicates that the request completed successfully, while status code 404 indicates that the requested resource was not found.
- **Cookies**: This field contains data sent by the server to the client and retransmitted by the client in subsequent requests. Cookies are often used to maintain state between requests and for user tracking.
- **User-Agent**: This header indicates the client that is making the request, usually a Web browser. It can be used to tailor content to the characteristics of the client.
- **Host**: This header specifies the domain of the requested URL.
- **Content-Type**: This field indicates the type of content in the message body, such as text, image, or binary data.

WAF

A Web Application Firewall (WAF) is a critical component of information security that plays a crucial role in protecting Web applications from a variety of threats and attacks. The capabilities of a WAF are designed to identify, prevent, and mitigate risks and vulnerabilities that could compromise the security and integrity of Web applications. Some of the main functionalities of a WAF:

- **Request Filtering**: A WAF carefully analyzes incoming HTTP requests and applies filters to identify and block suspicious or malicious requests. This helps prevent attacks such as SQL injection, cross-site scripting (XSS) and other common vulnerabilities.
- **Data Validation**: A WAF can perform in-depth validation of user-submitted data, such as form input fields. This feature prevents malicious or invalid data entry that could exploit vulnerabilities in the application.
- **Protection of Sessions**: WAFs can monitor and protect user sessions to prevent identity usurpation, session fixation attack, and other session-related threats.
- **Content Filtering**: A WAF can perform content filtering based on certain criteria, allowing it to block or limit access to unauthorized or malicious content.
- **Threat Detection**: Using behavioral analysis techniques and attack signatures, a WAF can proactively detect suspicious activity or abnormal traffic patterns that could indicate an attack in progress.
- **Protection from Forgeries**: WAFs can detect and prevent cross-site request forgery (CSRF) attacks, which attempt to perform unauthorized actions on behalf of the user.
- **Mitigation of Distributed Attacks**: A WAF can protect web applications from Distributed Denial of Service (DDoS) attacks by identifying and blocking malicious traffic that seeks to overload the application.
- **Monitoring and Reporting**: WAFs provide robust monitoring and report generation capabilities, enabling administrators to keep track of detected threats, security events, and user activity.
- **Adaptability**: A WAF can be configured to fit the specific needs of the application, allowing administrators to define custom rules and security policies.
- **Updates and Patches**: WAF vendors provide regular updates and patches to address new threats and vulnerabilities, ensuring that the application remains protected over time.

Different types of Bots

In the online environment, there are several types of bots, each with distinct purposes, behaviors, and functionalities. These bots can be used for a variety of purposes, both legitimate and malicious. Below, we will explore some of the main types of bots along with their descriptions:

- **Web Crawlers:**
  Purpose: Web crawlers, also known as spiders or indexing bots, are used by search engines to explore the web and index pages. They can also be used to collect data or monitor changes on web pages.

  Behavior: Web crawlers automatically visit web pages, follow links, and collect information for creating search indexes.

  Functionality: Collecting data, indexing, monitoring changes.

- **Chatbots:**
  Purpose: Chatbots are artificial intelligence programs designed to interact with users through chats or messages. They can be used to answer questions, provide assistance, process orders and more.

  Behavior: Chatbots interpret users' messages and respond with predetermined or real-time generated responses.

  Functionality: Interacting with users, assisting customers, processing orders, providing information.

- **Social Media Bots:**
  Purpose: Social media bots can be used for a variety of purposes, such as increasing followers, automating publications, sharing content, or spreading spam messages.

  Behavior: Social media bots can automatically publish content, share posts, follow or interact with other users.

  Functionality: Automating social media activities, increasing engagement, disseminating content.

- **Malicious Bots:**
  Purpose: Malicious bots are designed to perform malicious activities, such as spreading malware, performing DDoS attacks, stealing sensitive data, or spreading spam.

  Behavior: Malicious bots can exploit vulnerabilities to infiltrate, distribute malware, perform large-scale attacks, or exploit stolen data.

  Functionality: DDoS attacks, spreading malware, data theft, spam.

- **Price Comparison Bots:**
  Purpose: Price comparison bots, or price comparison crawlers, are used to monitor the prices of products on different websites and compare them.

Behavior: These bots visit e-commerce websites, collect data on product prices and compare them to provide users with comparison information.

Functionality: Monitoring prices, comparing products, assisting shoppers.

- **Content Generators:**
  Purpose: Content generators automatically produce text, articles, reviews or other written content. They can be used to create content quickly or to fool ranking algorithms.

  Behavior: These bots generate text using natural language algorithms or word substitution algorithms.

  Functionality: Automated content creation, filling in blanks.

- **Web Scrapers:**
  Purpose: Web scrapers are automated tools or scripts that are responsible for extracting data from web pages in a systematic and structured manner. Their main purpose is to provide users with an efficient and accurate way to collect information from websites in order to support analysis, monitoring or other purposes.

  Behavior: The behavior of web scrapers is based on a set of well-defined actions. These tools navigate through web pages, downloading the content of those pages by extracting specific elements or desired data. This data can include text, images, links, tables and more. Once extracted, the data can be processed, analyzed or archived for later use.

  Functionality: Web scrapers can be developed using different programming languages and libraries. For example, languages such as Python can be used in conjunction with specialized libraries such as Beautiful Soup or Scrapy. These tools have a wide range of capabilities that allow them to extract data from various online sources.

  The applications of web scrapers are diverse and span various sectors. They can be used to collect product prices from e-commerce sites for comparison purposes, monitor social media activity, track updates and news from news sites, extract data from government websites for statistical analysis, and more. In essence, web scrapers offer a powerful means of capturing meaningful data from the Internet in an automated and structured way, expanding opportunities for online information analysis and use.

ElasticSearch

The logs collected by Mitrhil, thus the dataset for this thesis, are saved on an innovative database called ElasticSearch. Elasticsearch is a powerful search and analysis technology based on high-performance distributed data. Fundamentally, it is a search engine and analytics system that allows large volumes of data to be explored quickly and efficiently. Its distributed architecture allows it to scale horizontally across multiple nodes, enabling efficient data and query management.

At the heart of Elasticsearch is the indexing engine that stores, organizes, and analyzes data in a way that makes it easily searchable. The technology leverages the inverted index technique, which allows terms and words to be quickly associated with their relative positions in documents. This enables lightning-fast searches on even large texts.

Elasticsearch supports a wide range of data types, from strings to numbers, from geospatial data to complex JSON structures. It also offers powerful full-text search capabilities, complex queries, aggregations, and analysis of structured and unstructured data.

One of the distinguishing features of Elasticsearch is its RESTful API that simplifies interaction with the system. Developers can create, manage and query indexes, perform searches and obtain results through standard HTTP requests. This makes Elasticsearch extremely versatile and easy to integrate into a variety of applications.

To query the Elasticsearch database from Python, the official Elasticsearch-Py library provided by Elastic was used. This library greatly simplifies the interaction with an Elasticsearch cluster and offers a wide range of features for querying and analyzing data.

4.2.2 The Concept of Session

Web client interactions with the Web site can be represented as sessions. A Web session is defined as a sequence of HTTP requests from a client during a single visit. The HTTP protocol is stateless, and session information at the Web application level is not stored in access logs, so session identification at this level remains uncertain and heuristics must be used [3].

Typically, session identification is performed by first grouping all HTTP requests that originate from the same IP address and corresponding user agent (user-agent), and then using a timeout-based approach to split this grouping into several subgroups, such that the time interval between two consecutive subgroups is longer than a predefined threshold. A drawback of this method is that it is difficult to determine an appropriate threshold value because different user agents exhibit different browsing behaviors. Usually, a 30-minute period is adopted as a threshold in Web mining studies[5].

However, by performing experiments and finding support from research number [3], I noticed that using the 30-minute threshold as the sole criterion for splitting the click stream into sessions was not sufficient. I observed the extracted sessions using the 30-minute value and noticed that, for longer sessions (in terms of the number of requests), the click streams belonging to a semantically continuous browsing activity were being separated into distinct sessions.

To address this problem, I introduce a procedure that adjusts the threshold value dynamically, based on the number of session requests so far. Specifically, for sessions with less than $r_{\max}$ requests so far, I set the threshold value to t_1. When the number of requests reaches $r_{\max}$, I increase the threshold value to

$t_2 > t_1$. In other words, I allow a longer time interval between consecutive requests for larger sessions. By experimenting with various threshold values and studying the resulting sessions, I determined that setting $r_{\max}$ to 100, t_1 to 30 minutes and t_2 to 60 minutes gave the best results[3].

Based on the HTTP request fields available in the logs, additional session characteristics can be determined, such as total number of requests, session duration, average time per page, and many others.

4.2.3 Features Selection and Design

User Agent

Each time a network request is made, the User Agent is sent from the Browser to a Web server, so as to give more information about the system being used. It is thus a kind of "identification label" for the browser. The User Agent is a field in the HTTP protocol through which more or less in-depth information can be given about the device making the network request. This is done through the HTTP Header, and this information can be used, for example, to send certain elements only to those browsers that can actually process them.

To extract the user agent feature, I adopt the **bag-of words expression** , which is a general conversion process of text information[6]. The "Bag of Words" (BoW) representation is a common technique in natural language processing that finds application in extracting features from "user agents." Briefly, this technique involves analyzing and decomposing collected "user agents," which represent the identities of clients accessing resources on the Internet, in order to extract useful information.

It is important to perform a pre-processing step on these "user agents," which may include such operations as removing special characters, normalizing upper/lower case letters, and breaking text into individual words or "tokens."

Once pre-processed, the "tokens" extracted from the "user agents" are used to create a unique vocabulary representing all possible words or "tokens" in the collected data. This vocabulary will serve as the basis for creating feature vectors for each "user agent." Each vector will contain information on the frequency with which each word or "token" in the vocabulary appears in the corresponding "user agent".

In summary, the BoW technique allows "user agents" to be represented in numerical form, transforming them into feature vectors that can be used for analysis and recognition. This representation provides an effective way to extract important information from "user agents" and can be used in various contexts, such as client behavior analysis or access pattern recognition.

Initial State:
Mozilla/5.0 (Macintosh; Intel Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/12.1 Safari/605.1.15
Transformed:
mozilla / macintosh / intel / mac / os / x / applewebkit / khtml / like / gecko / version / safari

Total Number Of Requests

The total number of requests in a session is a critically important feature for machine learning models aimed at detecting bot-generated traffic. This parameter provides a direct measure of overall activity during the interaction of a user or automated agent with a Web application. In cases of bot traffic, it is often characterized by a large number of requests made in a short period of time, as bots are programmed to perform actions in rapid succession in order to exploit or harm the application's target. Consequently, the quantity of requests may prove to be a crucial indicator for identifying such fraudulent or suspicious activities.

A large number of requests in a single session might suggest the use of scraping strategies, in which bots collect data in a massive way from websites. In addition, the rapid accumulation of requests may be indicative of attacks such as distributed denial of service (DDoS), in which numerous bots attempt to overload a server with an intensive flow of traffic. Since real user behavior tends to have a more balanced distribution of requests over time, the total number of requests in a session can therefore serve as a key metric for identifying anomalies and inauthentic traffic patterns.

Total Volume

The volume of data sent to the client is an important feature for machine learning models dedicated to detecting bot-generated traffic. This parameter measures the amount of data transmitted from the Web server to the client (e.g., a browser) during a session. In cases of bot activity, there is often a high volume of data transmitted out of proportion to the normal behavior of real users.

Bots, designed to perform automated tasks, can generate excessive requests that cause a large amount of data to be sent to the client. This can include the massive collection of content from web pages, images, or resources, which can be a sign of malicious activity such as scraping or data theft. In addition, DDoS attacks may involve transferring an exceptionally large amount of data in order to overload the target client or server.

Detecting the abnormal increase in the volume of transmitted data can therefore provide a significant indication of inauthentic or suspicious activity. By integrating this feature into machine learning models, algorithms can be created that can identify bot traffic patterns based on the excessive amount of data sent to the client.

Total Time

The total session duration provides valuable insight into the total duration of a user's or agent's interaction with a Web site or Web application during a single session.

In the context of legitimate user sessions, total session duration tends to follow a specific range of values, reflecting typical user engagement patterns. However, in the case of bot traffic, anomalies can occur that lead to extremely short or unusually long session durations. Bots may generate rapid, automated interactions that result in very short sessions, or they may engage in prolonged interactions for malicious purposes, such as data scraping or other unauthorized activities.

The inclusion of total session duration as a feature in machine learning models makes it possible to capture and distinguish between sessions of normal users and bot-generated sessions. Deviations from the expected session duration interval can serve as a strong indicator of the presence of bots. Short, quick sessions may suggest automated interactions, while excessively long sessions may indicate aggressive scraping or other suspicious behavior.

Average Time Between Requests

The average time between requests provides a valuable indication of the frequency and pace at which requests are made during a Web session.

In the context of legitimate users' sessions, it is common to observe a certain temporal pattern between requests. Human users tend to browse more organically and follow consistent paths, which is reflected in a regular average time between requests. In contrast, bots often generate requests in an automated and continuous manner, resulting in significantly shorter average time between requests.

Using 'Average Time Between Requests' as a feature allows us to capture such variations in the temporal behavior of requests. Abnormally short intervals may suggest the presence of bot activity, such as trying to overload a server or performing repetitive actions too quickly.

Standard Deviation Time Between Requests

The standard deviation time between requests offers an additional perspective on the temporal pattern of requests within a Web session.

Human users tend to exhibit some variability in the time between requests, as their browsing behavior can be influenced by multiple factors, such as reading content, deciding which page to visit next, and interacting with interactive elements. This variability reflects the dynamic and unpredictable nature of human actions during online browsing.

On the other hand, bots often generate requests automatically and uniformly, resulting in times between requests with a lower standard deviation. Using the standard deviation time between requests as a feature allows the machine learning model to detect these differences in time variations between requests.

The presence of an abnormally low standard deviation could indicate bot

activities, such as automating certain actions or attempting to exploit vulnerabilities through fast and regular requests.

Rate of Night Requests

The percentage of requests made between 2:00 a.m. and 6:00 a.m. (local time) is a valuable feature for machine learning models dedicated to bot traffic detection. This metric provides an important temporal perspective on the activities of users and potential bots on the Web.

Human users tend to have distinct activity patterns during different times of the day. Typically, the nighttime hours, between 2:00 a.m. and 6:00 a.m., are characterized by a significant reduction in human online activity. During this period, most people are busy sleeping or inactive, which is reflected in the reduced amount of web requests made. As a result, the percentage of requests during these hours tends to be relatively low.

To calculate this feature, a conversion of the *Timestamp* field to local time was made, referring to the *Geoip.region iso code* field.

Repeated

The reoccurence rate of file requests provides insight into file access behavior by users and potential bots within a web session.

Human users tend to follow predictable browsing patterns during a web session. Typically, a person accesses a variety of files related to a particular activity or interest. This results in a high recurrence of requests for certain files within a session. For example, a user visiting an e-commerce site might access product images, descriptions, and detail pages multiple times during a single session.

On the other hand, bots may exhibit different behavior. Depending on their purpose, bots might request the same file repeatedly to exploit a vulnerability or conduct an attack. This anomalous behavior will be reflected in a high recurrence rate of file requests within a session.

Analysis of the reoccurence rate of file requests allows the machine learning model to identify significant inequalities in the frequency of file requests. The presence of an unusually high recurrence rate could suggest bot behavior, providing the model with an important indicator of suspicious activity.

Rate of Errors

In the context of HTTP requests, status codes indicate the outcome of the interaction between the client and the server. Codes with value ≥ 400 correspond to error or failure situations, such as '404 Not Found' or '500 Internal Server Error'. Human users generally generate only a small percentage of requests with such high status codes, since normal interactions often involve successful requests. In contrast, bots can cause a significant increase in the percentage of requests with status codes ≥ 400 due to their aggressive or malicious behavior.

The analysis of the percentage of requests with status codes ≥ 400 allows the machine learning model to detect situations where the session is characterized

by a disproportionate number of invalid or failed requests. This could indicate the actions of malicious bots attempting to exploit vulnerabilities or overload the server with malicious requests. The increase in such requests may be a significant sign of suspicious activity.

Inclusion of this feature in the model makes it possible to effectively monitor and capture anomalous patterns of requests with status code ≥ 400, providing a strong clue about the presence of malicious bot traffic.

Rate of GET methods

The GET method is one of the main HTTP methods used to request resources from a web server. Bots often differ from legitimate users in their use of HTTP methods. Bots can generate a significant proportion of GET requests, as they often try to extract information from web pages or perform scanning and data collection tasks. In contrast, human users can generate a variety of HTTP methods, including POST, PUT, DELETE, etc., reflecting more dynamic interaction with web applications.

Rate of POST methods

The POST method is widely used to send data to the web server, such as when filling out forms or sending sensitive information. However, bots often behave differently from legitimate users in using HTTP methods. POST requests are commonly adopted by bots to submit large amounts of data, perform spam activities, or attempt brute force attacks. Human users, in contrast, tend to use the POST method in a more balanced and circumstantial way.

Rate of HEAD methods

The HEAD method is mainly used to obtain information about the header of a resource without performing the actual data transfer. Although it is a legitimate and valuable tool used by web developers to obtain quick details about the resource, bots can exploit this method to perform suspicious or malicious activities. For example, bots can use the HEAD method to explore the content of a website without downloading the entire page content, thus avoiding traditional content-based detection.

Rate of OTHER methods

The percentage of requests with methods other than the usual ones, such as GET, POST, and HEAD, is an important feature to consider in a machine learning model aimed at detecting bot traffic. While standard methods are commonly used by legitimate users to interact with web resources, bots might use less common or exotic methods to perform malicious or invasive activities. The presence of a high number of requests using "unconventional" methods could suggest the activity of bots attempting to probe or exploit system vulnerabilities.

Width

The "Width" attribute is a key feature in the analysis of interactions within a browsing session. This attribute reflects the horizontal extent of the paths through which the user moves through the website. In other words, it measures how many distinct branches develop from the root node of the representative graph based on the URI names of the requested pages. When the amplitude is low, it means that requests are mainly focused on a single path, suggesting a more targeted or specific interaction with the site. In contrast, a higher amplitude reflects a more diverse navigation through different subsections or pages of the site.

For example, if a session contains requests for the following pages, { /A, /A/B, /A/B/C }, then its width will be 1.Basically, the width attribute measures the number of leaf nodes generated in the graph while the depth attribute measures the maximum depth of the tree(s) within the graph. Therefore, a session that contains requests for { /A, /A/B, /C, /D } will have a width of 3.

Depth

The attribute of "Depth" is equally significant in the evaluation of browsing sessions. This attribute reflects the vertical extent of the paths through which the user moves within the website. Depth measures how many hierarchical levels are reached during navigation, indicating how deeply the user explores different sections of the site. In the context of a graph based on the URIs of the requested pages, depth represents the maximum level of a tree generated by the graph. A greater depth may indicate a more detailed, engaging and complete navigation, typical of human users. On the other hand, a shallower depth might suggest more superficial or sequential behavior, often characteristic of automated tasks or bots.

For example, if a session contains requests for the following pages, { /A, /A/B, /A/B/C }, then its depth will be 3. Basically, the width attribute measures the number of leaf nodes generated in the graph while the depth attribute measures the maximum depth of the tree(s) within the graph. Therefore, a session that contains requests for { /A, /A/B, /C, /D } will have a depth of 2.

Rate of requests with Null Referrer

The referrer, or "referrer header," is an element of HTTP requests that provides information about the web page from which the request originated. In the case where the referrer is null, it indicates that the request did not originate from a previous web page. This situation is particularly relevant in the context of bot traffic identification, as bots often tend to execute direct requests to servers without following a typical page-to-page navigation path as human users do. Consequently, a significantly high percentage of requests with null referrers could suggest the presence of suspicious or automated activity, providing a valuable signal to the machine learning model in its bot identification process.

Max Sustained Click Rate

A "click" is configured as a request for an HTML file, and this feature identifies the maximum number of requests for HTML files made within a specific time window within a session. This approach is based on the intuition that there is an upper limit to the maximum number of "clicks" a human can make within a defined time interval t, which is influenced by human factors.

To capture this feature, initially the value of time window t is established. Then, a sliding window of length t within a given session is used in order to measure the maximum rate of "clicks" incurred in that session. For example, suppose we set t at 12 seconds and note that the maximum number of "clicks" within a time window of 12 seconds in that session is 36. We can conclude that the maximum sustained "click" rate is 3 "clicks" per second. This result indicates a behavior more like that of a robot than that of a human being. The sliding window approach starts from the first HTML-type request of a session and records the maximum number of "clicks" within each window, gradually moving the window one HTML request at a time until the last request of the session is reached. The maximum "clicks" per window provides the value of this feature. [3]

Robot.txt

The "robots.txt" file is a crucial resource used by search engines and other automated agents to understand the restrictions and permissions imposed by a website on their indexing and crawling behaviors. Therefore, monitoring access to this file can reveal valuable information regarding query behavior.

In the context of bot traffic detection, analysis of access to the "robots.txt" file reveals details about the client's willingness to comply with or circumvent website guidelines. Human agents tend to access the "robots.txt" file in a limited way, since the main purpose of this file is to communicate with search engines. On the other hand, bots may frequently access this file to obtain information about the structure and restrictions of the site before taking further actions. Therefore, analysis of access to the "robots.txt" file may reveal typical behavior patterns of bots, thus contributing to the accurate identification of automated activities.

Rate of Images requestes

Analysis of image requests within a Web site login session can reveal distinctive behaviors that are often associated with automated agents rather than human users.

Image requests are an essential part of a human user's browsing experience, as they contribute to the visual appearance and interaction with the site. However, bots may have a different pattern when it comes to image requests. For example, a bot might request a large number of images in a short period of time to collect data or perform automated tasks, while a human user would have a more varied and gradual behavior.

Measuring the "rate of image requests" allows the machine learning model

to capture this behavioral discrepancy. If a session has an unusually high rate of image requests relative to the total number of requests, it could be indicative of automated activities. Also, a consistent and uniform rate of image requests might suggest behavior more consistent with an automated agent.

The feature was extracted from the response header in the content-type entry

Rate of CSS elements requestes

Analysis of CSS style sheet requests within a Web site login session can reveal behavioral patterns that are often distinctive of automated versus human activities.

CSS style sheets are essential to the presentation and visual appearance of a web page, influencing formatting, layout, and graphical appearance. However, bots tend to behave differently than human users when it comes to CSS requests. For example, a bot might request a large number of style sheets in rapid succession in order to collect data or perform automated tasks, while a human user often follows a more varied and gradual pattern.

Measuring the "CSS request rate" allows the machine learning model to capture these behavioral differences. A high or excessively constant rate of style sheet requests might suggest the presence of an automated agent. On the other hand, a more natural distribution of CSS requests might reflect legitimate human behavior.

The feature was extracted from the response header in the content-type entry

Rate of JavaScript elements requestes

JavaScript scripts are critical to the interactivity and dynamism of Web pages, enabling advanced features such as animations, data validations, and user interactions. However, bots often behave differently than human users when it comes to JavaScript script requests. For example, bots may request a large number of scripts sequentially to collect data or perform automated actions or none at all, while human users tend to follow more varied and natural patterns.

The feature was extracted from the response header in the content-type entry

Rate of PDF elements requestes

PDF files are often used for sharing and viewing documents, reports, and online information resources. However, bots' interest in PDF files may differ considerably from human users. Bots may require numerous PDF files in sequence, for example, for content dragging or data extraction. This automated behavior can be distinguished from that of human users, who tend to request PDF files in a more random and variable manner.

The introduction of the "PDF request rate" feature into the machine learning model provides a powerful metric to detect such differences. A high or constant rate of PDF file requests may indicate the presence of bot activity, while a more diverse distribution reflects legitimate human behavior.

Twenty-two summary features descriptive of whole sessions were extracted from HTTP data, listed in the following table:

Feature Name	Feature Type	Feature Description
userAgent	String	User agent representing the client
noRequests	int	Total number of requests (session length)
volume	int	Total volume of data sent to the client [KB]
totalTime	int	Session duration expressed in seconds [s]
avgTime	int	Average time between requests [s]
stDevTime	int	Standard deviation of time between requests
Night	double	% of requests made between 2am to 6am (local time)
Repeated	double	Reoccurence rate of file requests
Error	double	% of requests with code status ≥ 400
GET	double	% of requests made with GET method
POST	double	% of requests made with POST method
HEAD	double	% of requests made with HEAD method
OTHER	double	% of requests made with other methods
Width	int	Width of the traversal (in the URL space)
Depth	int	Depth of the traversal (in the URL space)
nullReferrer	double	% of requests with a null referrer
MaxSustainedClickRate	double	Maximum number of requests in a given window
Robot.txt	bool	True if robot.txt is accessed, False otherwise
Image	double	% of requests requesting images elements
cssRequests	double	% of requests requesting css elements
jsRequests	double	% of requests requesting JavaScript elements
pdfRequests	double	% of requests requesting PDFs

Table 1: Features

4.2.4 Features Analysis

The plot of feature pairs is shown in the following section to study their significance and possible correlation. The plots refer to an experiment made taking a subdataset consisting of two and a half hours of network traffic, totaling about 8000 sessions, where each point in space represents a session.

Number of Requests and Volume

The distribution of points shows two very definite behaviors: the first is that as the volume of data transferred increases, the total number of requests remains limited to very small numbers (range 1 - 100), and reciprocally, as the number of requests increases, the volume remains limited in the vicinity of zero. The second, on the other hand, denotes a linear and directly proportional trend between the two features.

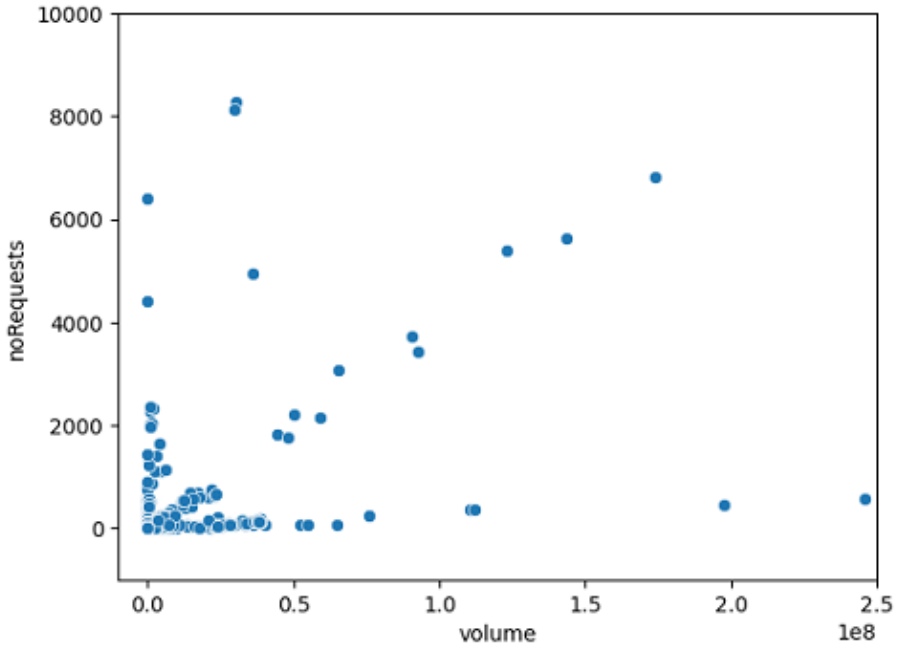

Figure 1: Number of Requests and Volume

Number of Requests and Total Time

As the number of requests increases, the total session time increases (as we might expect), but below 1000 requests, we instead have an almost even distribution on the axis of total time.

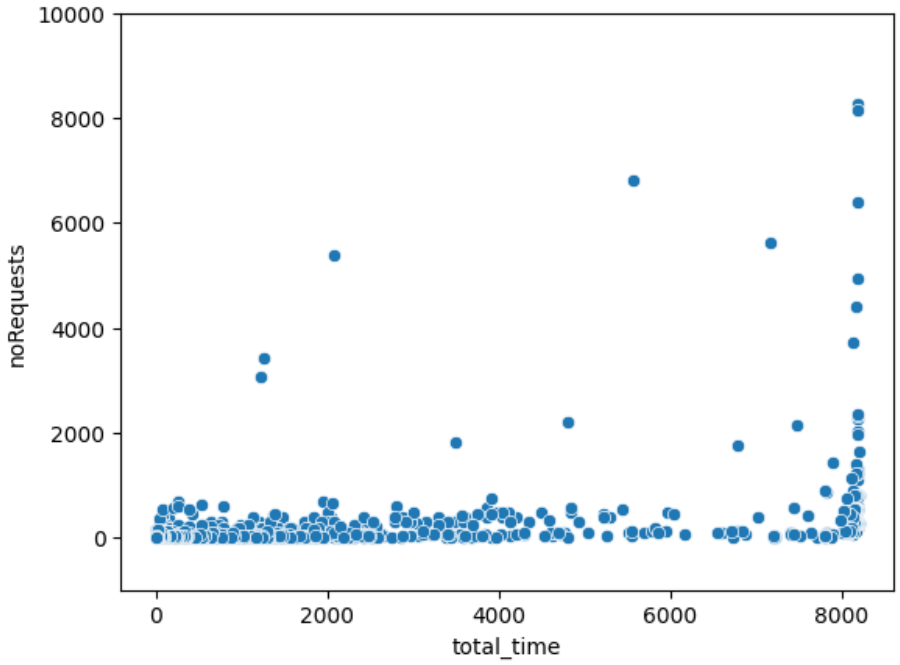

Figure 2: Number of Requests and Total Time

Number of Requests and Average Time

The insight this graph gives us is that as the number of requests increases, the average time between requests is close to zero, thus indicating extremely fast browsing, whereas as the average time increases we have a number of requests contained below 100.

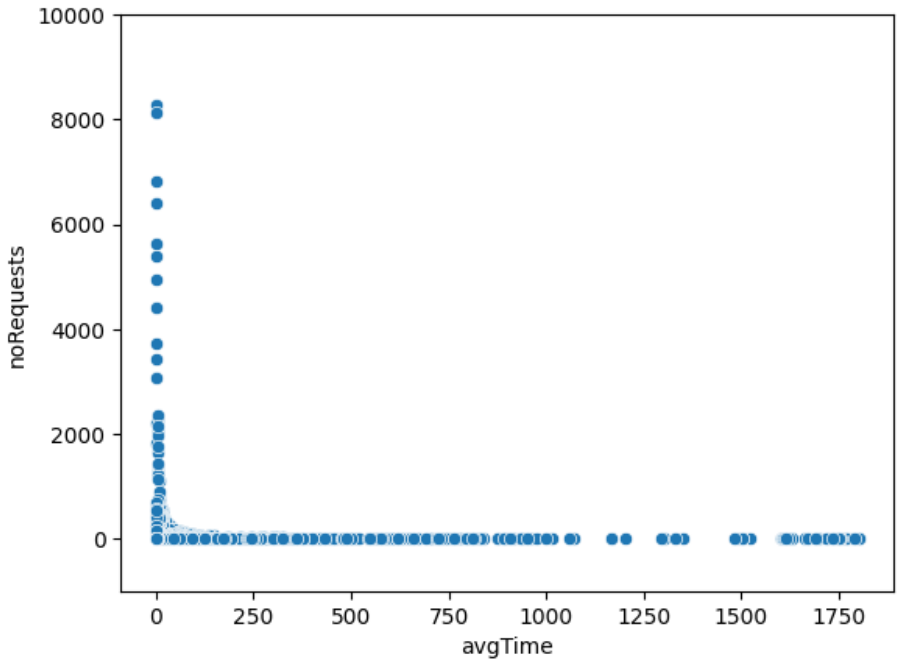

Figure 3: Number of Requests and Average Time

Number of Requests and Standard Deviation Time

The behavior of this graph is very similar to that of the ratio of number of requests to average time, but in a more relaxed manner. The tendency is to have a small number of requests as the standard deviation increases, and instead a zero standard deviation as the number of requests increases.

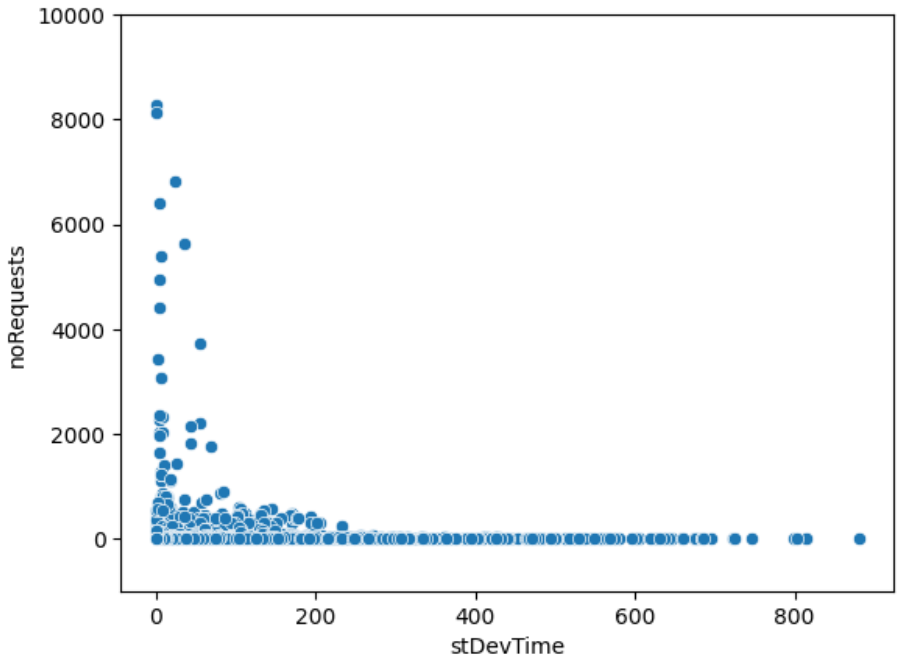

Figure 4: Number of Requests and Standard Deviation Time

Number of Requests and Null Referrer

The behavior of this graph is different from the previous ones, in that as the number of requests increases we can both have either a 0% rate of null referrers or a 100% rate of null referrers. Instead with a number of requests around 100, we can have a more even distribution on the null referrer axis

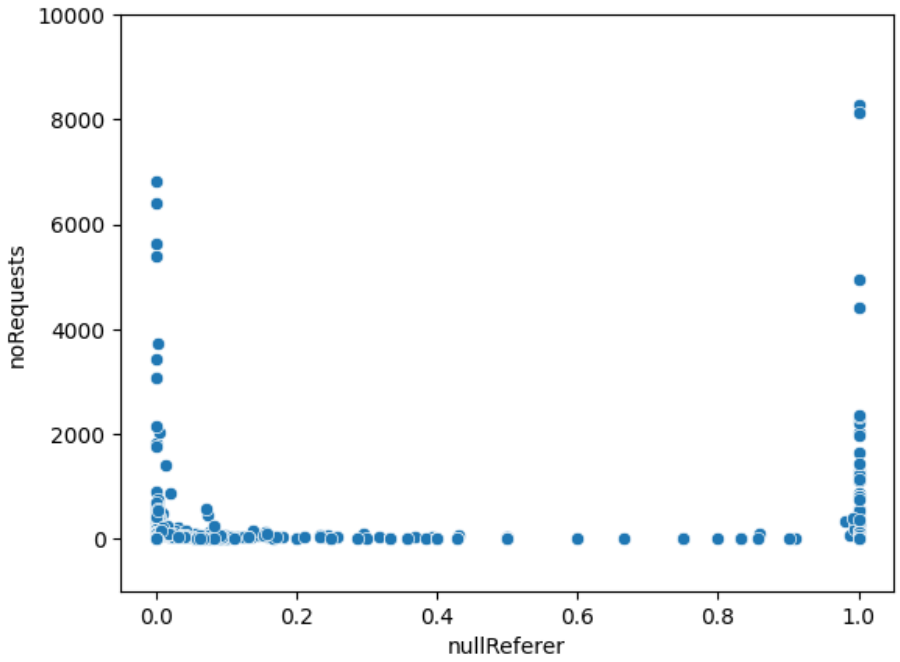

Figure 5: Number of Requests and Null Referrer

Number of Requests and Reoccurrence

The significance of this graph is that sessions with a number of requests greater than 500 tend to have a recurrence rate between 80% and 100%,while sessions with a lower total number of requests have may have any recurrence rate

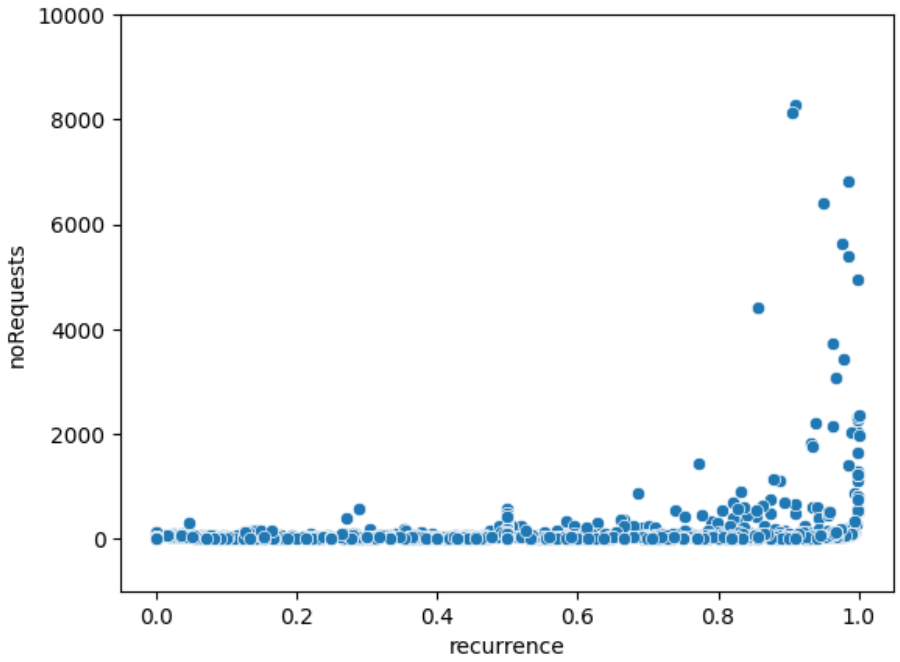

Figure 6: Number of Requests and Reoccurrence

Number of Requests and Images

The significance of this graph is that sessions with a high number of requests have an image request rate of either zero or around 30%, whereas sessions with a number of requests around 100, have an image request rate between 80% and 0%.

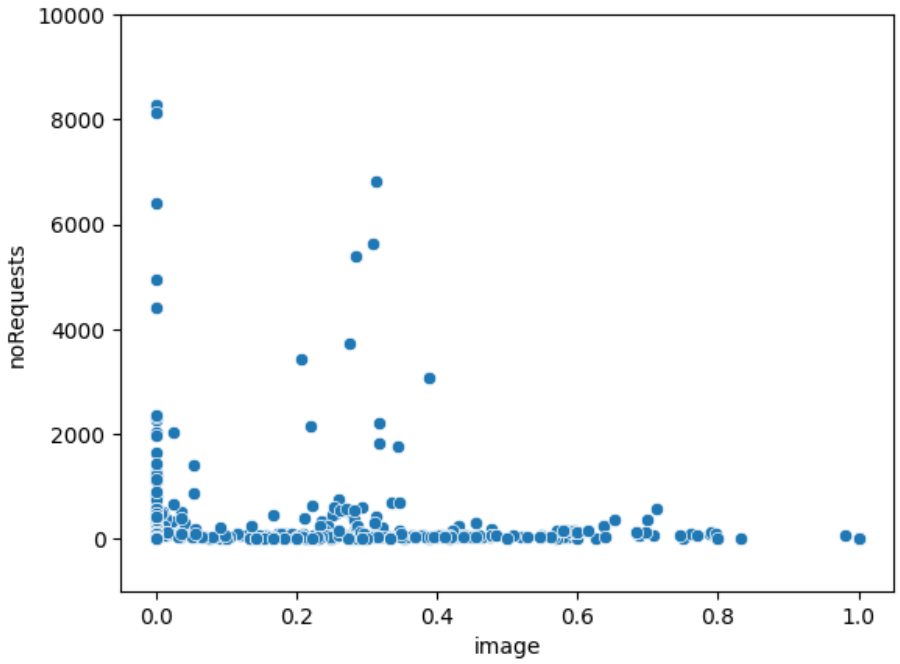

Figure 7: Number of Requests and Images

Number of Requests and GET

The insight that this graph gives us is that when a session has more than 1000 requests it tends to have a near 100% rate of GET method

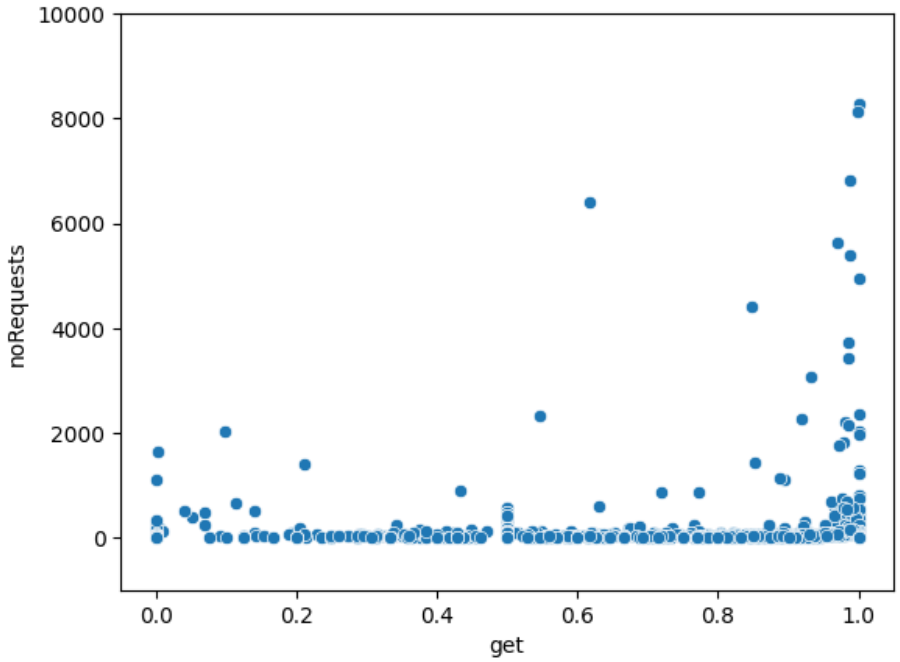

Figure 8: Number of Requests and GET

Number of Requests and POST

This graph denotes a complementary behavior of the POST method compared to the GET method, resulting in an almost mirrored graph compared to the previous one. This is a predictable behavior since GET and POST methods are two complementary methods and include almost all of the commonly used.

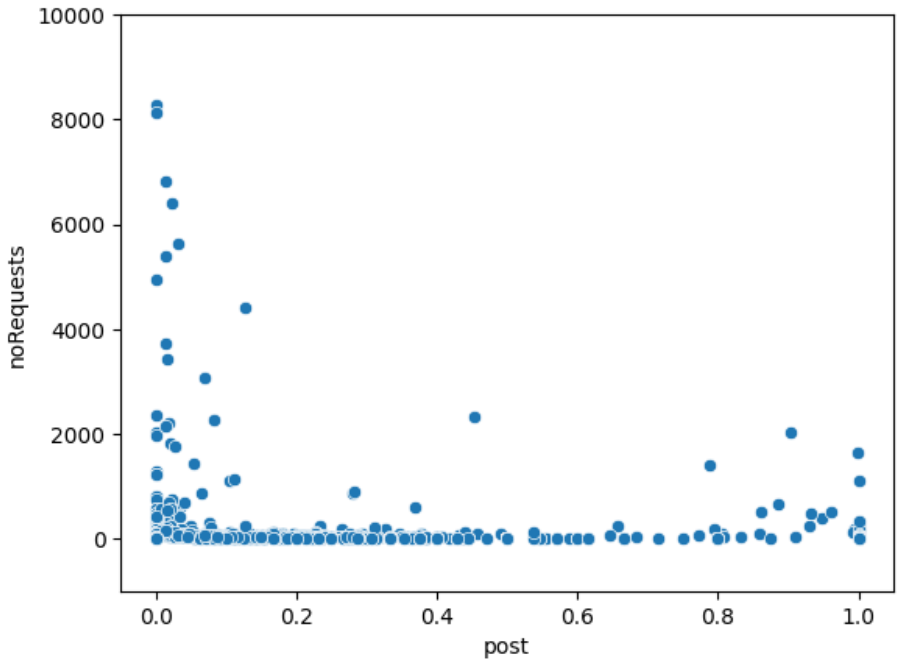

Figure 9: Number of Requests and POST

Number of Requests and Width

This graph shows us that as the number of requests increases, we have two peaks regarding the "width" of the paths explored, the first in the range 1-7, the second in the range 19-23. Another significant finding is that very high widths correspond to very low numbers of requests.

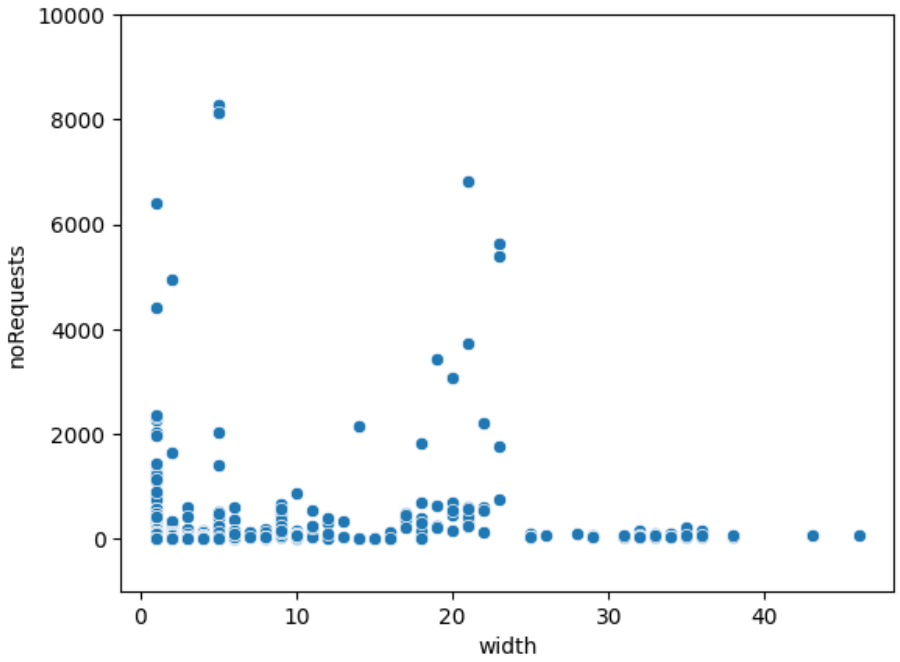

Figure 10: Number of Requests and Width

Volume vs Errors

The insight this graph gives us is that as volume increases we have a very small error rate, in the range 10% - 0% . In addition, all sessions with an error rate greater than 30% have a volume close to zero.

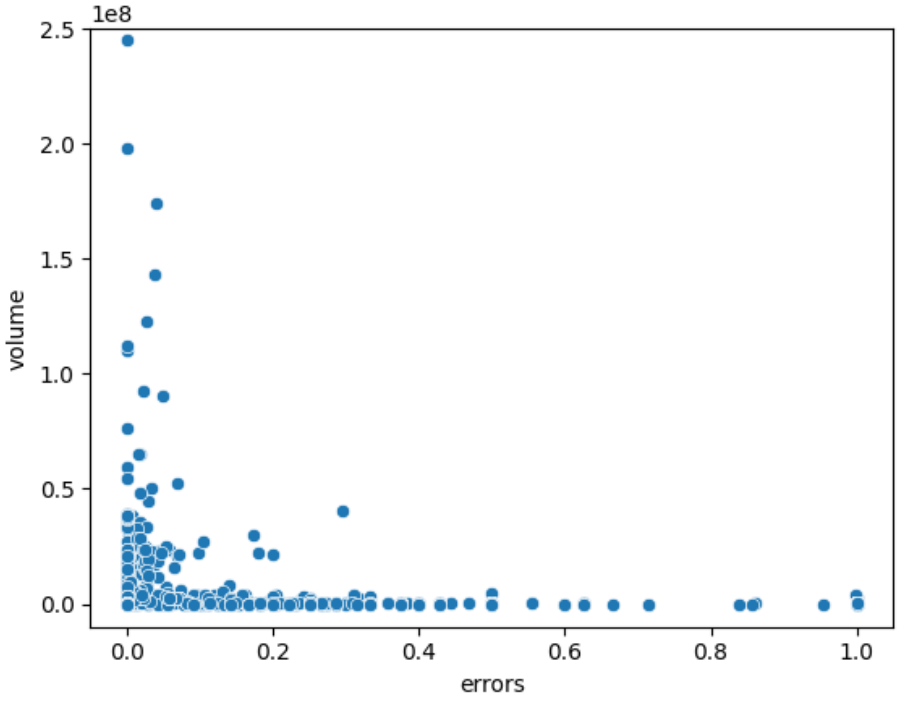

Figure 11: Volume vs Errors

Volume vs GET

This graph shows us that for volumes above 10GB, the rate of GET methods is close to 100% except for a few outliers.

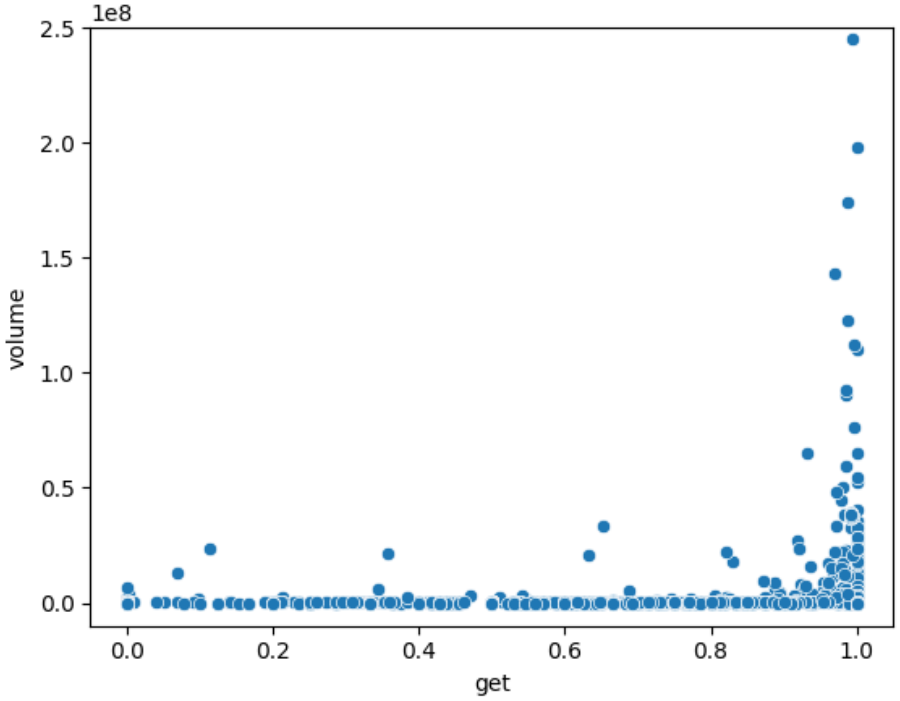

Figure 12: Volume vs GET

Volume vs POST

Again, this graph confirms for us the markedly complementary behavior of the POST method versus the GET method, as studied in the previous graphs.

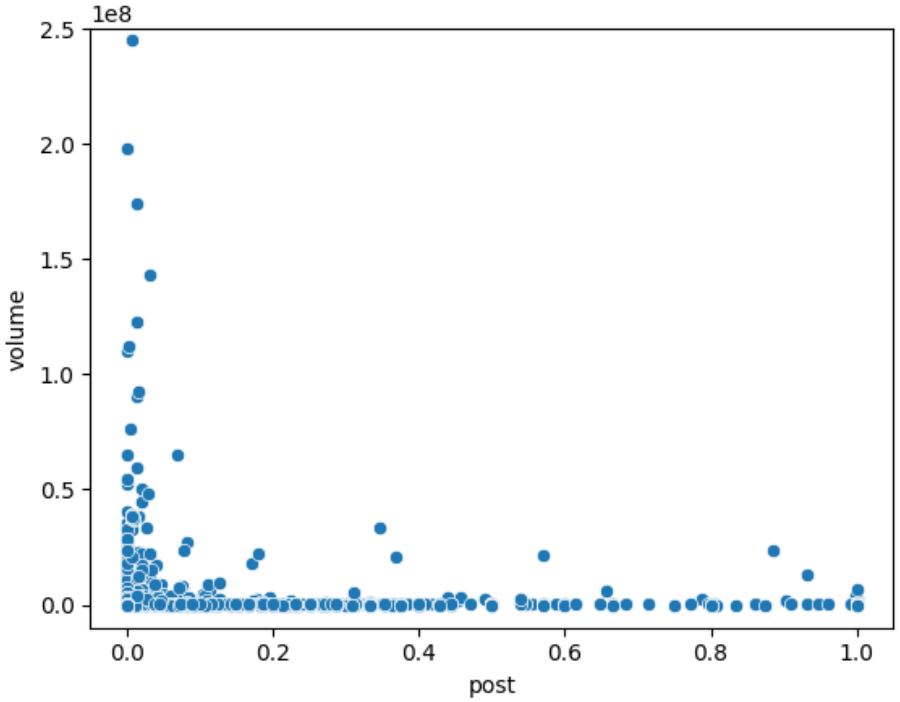

Figure 13: Volume vs POST

TotalTime vs Average Time

This graph has a more scattered behavior than the previous ones. There are two very interesting behaviors, the first is that as the total time increases we have a very narrow time average between one request and the next, which means a rapid succession of requests for very long sessions. The second remarkable behavior is that below 2000 seconds there is a linear and directly proportional relationship.

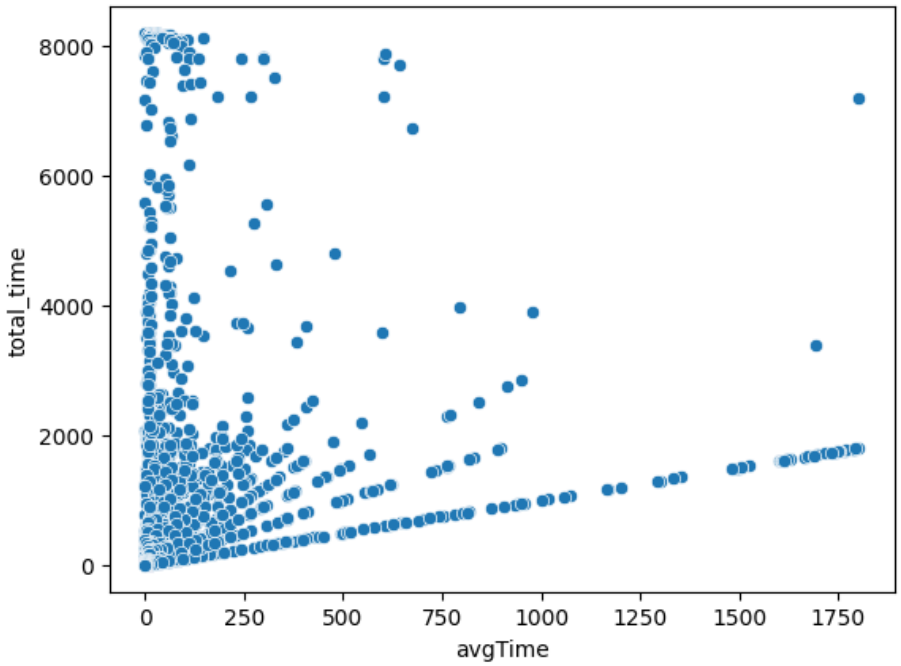

Figure 14: TotalTime vs Average Time

TotalTime vs Recurrence

This scatterplot indicates to us that for a total time less than 2000 seconds we have a recurrence rate almost equally distributed between 0% and 100%, but above 2000 seconds we have a more pronounced density indicating a recurrence rate greater than 50%, particularly in the range 80% - 100%

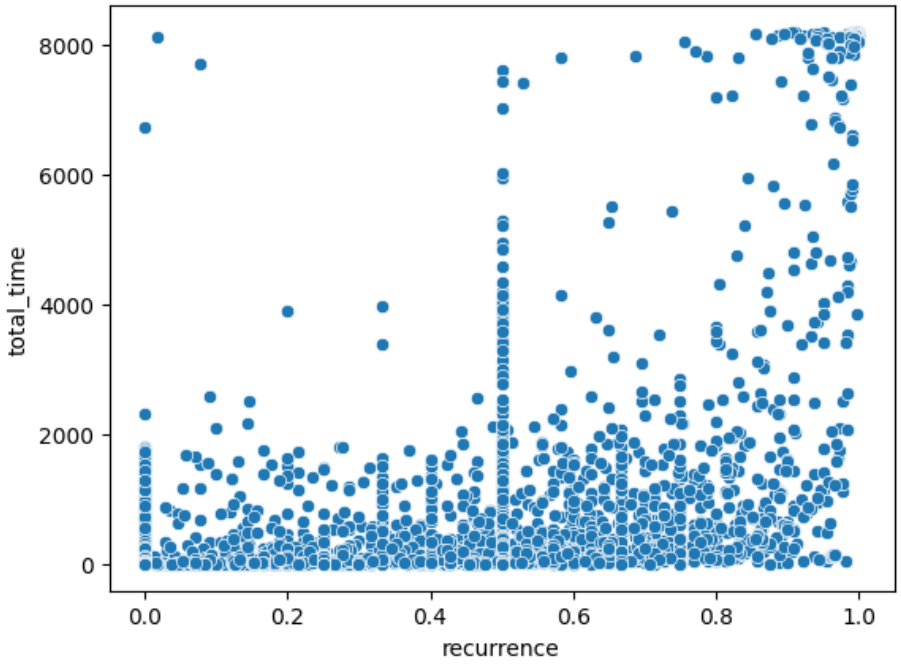

Figure 15: TotalTime vs Recurrence

Recurrence vs Errors

In the scatterplot regarding the relationship between recurrence rate and error rate, we have a result that I did not expect: at high recurrence rates we have a more pronounced distribution around low error rate values, which is counter-intuitive. But there is still a remarkable significance at error rates of 100%, where the distribution of points coincides with a high recurrence rate, which is the behavior I expected in the first instance.

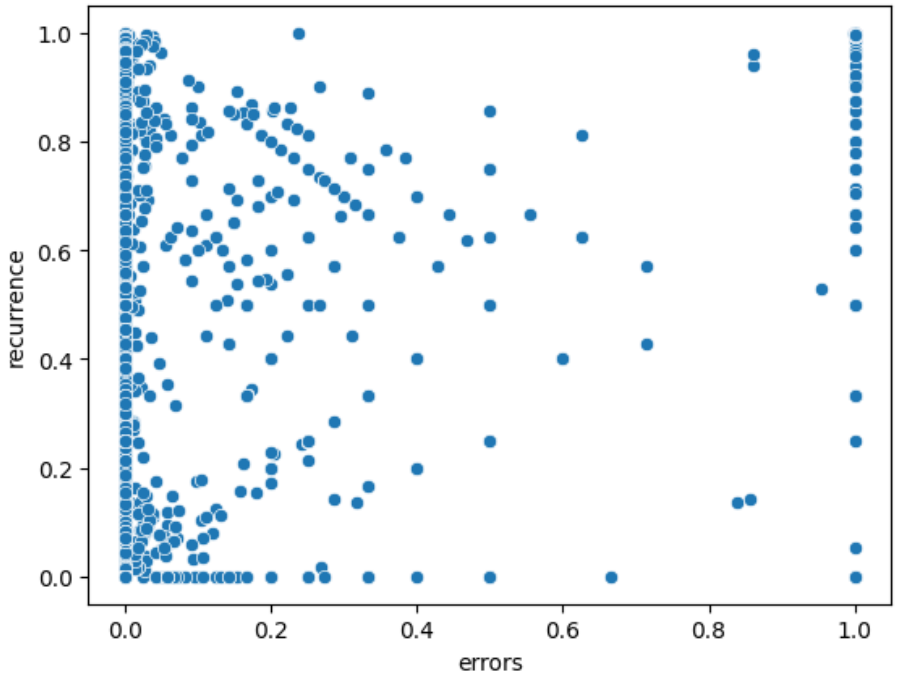

Figure 16: Recurrence vs Errors

Recurrence vs Image

The result of this scatterplot shows us that at high recurrence rates, the image request rate is limited in the 0%-40% range, and as the recurrence rate is lowered we can find sessions with a much higher image request rate, up to 80%, with some outliers at 100%.

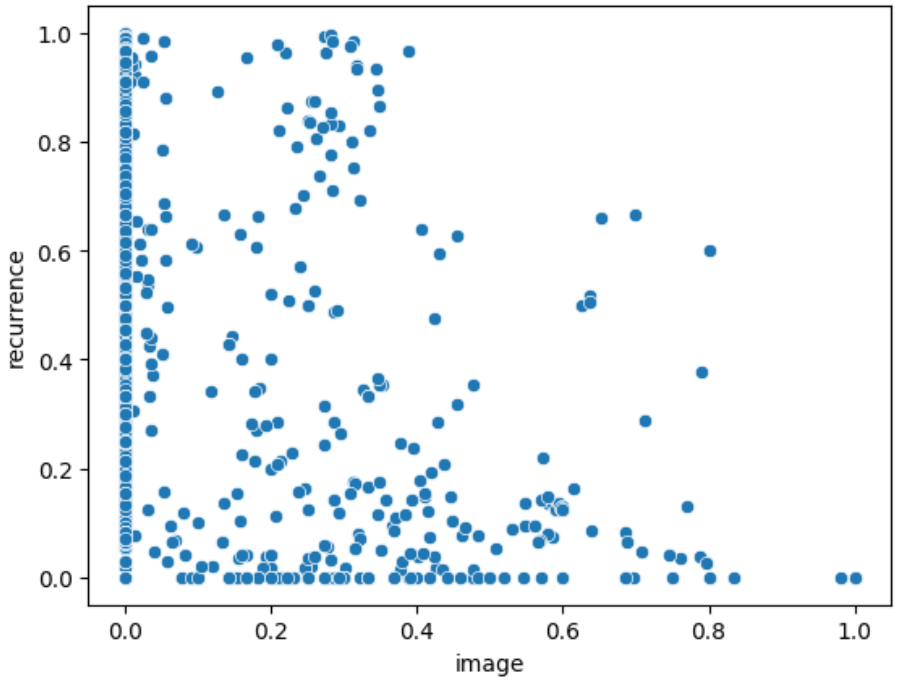

Figure 17: Recurrence vs Image

Errors vs Referrer Null

This scatterplot did not reflect my expectations, I would have expected a strong relationship between high error rates corresponding to high null referrers in the sessions, instead this marked relationship was not found

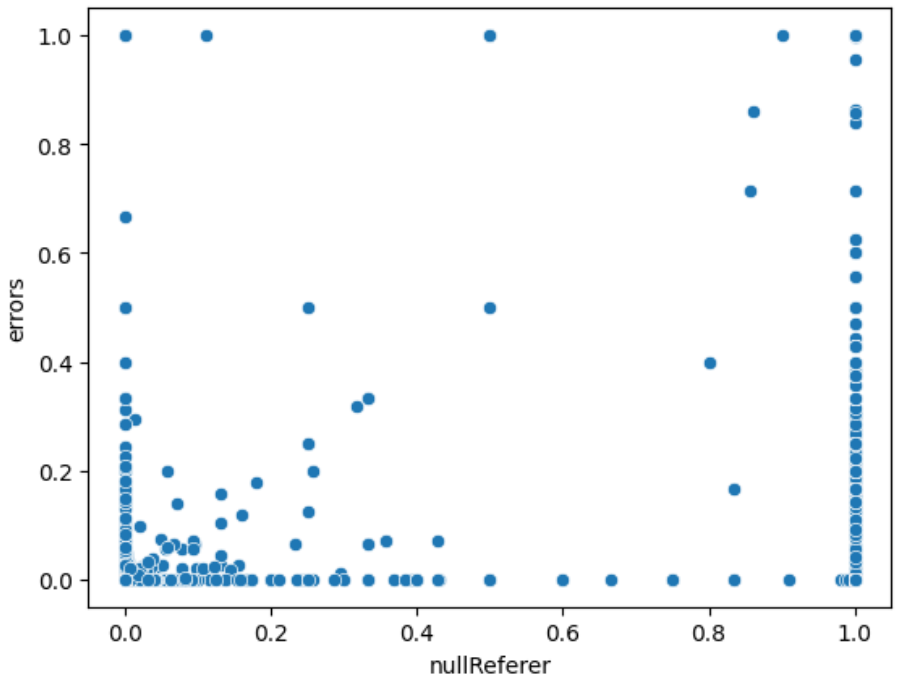

Figure 18: Errors vs Referrer Null

4.3 Machine Learning Model

4.3.1 Introduction to machine learning

Machine learning is a subfield of artificial intelligence (AI). The goal of machine learning generally is to understand the structure of data and fit that data into models that can be understood and utilized by people.

Although machine learning is a field within computer science, it differs from traditional computational approaches. In traditional computing, algorithms are sets of explicitly programmed instructions used by computers to calculate or problem solve. Machine learning algorithms instead allow for computers to train on data inputs and use statistical analysis in order to output values that fall within a specific range. Because of this, machine learning facilitates computers in building models from sample data in order to automate decision-making processes based on data inputs.

Any technology user today has benefitted from machine learning. Facial recognition technology allows social media platforms to help users tag and share photos of friends. Optical character recognition (OCR) technology converts images of text into movable type. Recommendation engines, powered by machine learning, suggest what movies or television shows to watch next based on user preferences. Self-driving cars that rely on machine learning to navigate may soon be available to consumers.

In machine learning, tasks are generally classified into broad categories. These categories are based on how learning is received or how feedback on the learning is given to the system developed.

Two of the most widely adopted machine learning methods are supervised learning which trains algorithms based on example input and output data that is labeled by humans, and unsupervised learning which provides the algorithm with no labeled data in order to allow it to find structure within its input data.

Supervised Learning In supervised learning, the computer is provided with example inputs that are labeled with their desired outputs. The purpose of this method is for the algorithm to be able to "learn" by comparing its actual output with the "taught" outputs to find errors, and modify the model accordingly. Supervised learning therefore uses patterns to predict label values on additional unlabeled data.

A common use case of supervised learning is to use historical data to predict statistically likely future events. For example it may use historical stock market information to anticipate upcoming fluctuations, or be employed to filter out spam emails.

Unsupervised Learning In unsupervised learning, data is unlabeled, so the learning algorithm is left to find commonalities among its input data. As unlabeled data are more abundant than labeled data, machine learning methods

that facilitate unsupervised learning are particularly valuable.

The goal of unsupervised learning may be as straightforward as discovering hidden patterns within a dataset, but it may also have a goal of feature learning, which allows the computational machine to automatically discover the representations that are needed to classify raw data.

Unsupervised learning is commonly used for transactional data. You may have a large dataset of customers and their purchases, but as a human you will likely not be able to make sense of what similar attributes can be drawn from customer profiles and their types of purchases.

Without being told a "correct" answer, unsupervised learning methods can look at complex data that is more expansive and seemingly unrelated in order to organize it in potentially meaningful ways. Unsupervised learning is often used for anomaly detection including for fraudulent credit card purchases, and recommender systems that recommend what products to buy next.

Clustering Clustering is basically a type of unsupervised learning method. It is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group and dissimilar to the data points in other groups. It is basically a collection of objects on the basis of similarity and dissimilarity between them.

Clustering is very much important as it determines the intrinsic grouping among the unlabelled data present. There are no criteria for good clustering. It depends on the user, and what criteria they may use which satisfy their need. For instance, we could be interested in finding representatives for homogeneous groups (data reduction), finding "natural clusters" and describing their unknown properties ("natural" data types), in finding useful and suitable groupings ("useful" data classes) or in finding unusual data objects (outlier detection). This algorithm must make some assumptions that constitute the similarity of points and each assumption make different and equally valid clusters.

Clustering Methods:

- Density-Based Methods: These methods consider the clusters as the dense region having some similarities and differences from the lower dense region of the space. These methods have good accuracy and the ability to merge two clusters. Example DBSCAN (Density-Based Spatial Clustering of Applications with Noise), OPTICS (Ordering Points to Identify Clustering Structure), etc.

- Hierarchical Based Methods: The clusters formed in this method form a tree-type structure based on the hierarchy. New clusters are formed using the previously formed one. It is divided into two category:

  - Agglomerative (bottom-up approach)
  - Divisive (top-down approach)

4.3.2 Brief review of algorithms

DBSCAN

DBSCAN stands for Density-Based Spatial Clustering of Applications with Noise. It groups 'densely grouped' data points into a single cluster. It can identify clusters in large spatial datasets by looking at the local density of the data points. The most exciting feature of DBSCAN clustering is that it is robust to outliers. It also does not require the number of clusters to be told beforehand, unlike K-Means, where we have to specify the number of centroids.

DBSCAN requires only two parameters: *epsilon* and *minPoints*. **Epsilon** is the radius of the circle to be created around each data point to check the density and **minPoints** is the minimum number of data points required inside that circle for that data point to be classified as a **Core** point.

In higher dimensions the circle becomes hypersphere, epsilon becomes the radius of that hypersphere, and minPoints is the minimum number of data points required inside that hypersphere.

DBSCAN creates a circle of epsilon radius around every data point and classifies them into **Core** point, **Border** point, and **Noise**. A data point is a Core point if the circle around it contains at least 'minPoints' number of points. If the number of points is less than minPoints, then it is classified as **Border** Point, and if there are no other data points around any data point within epsilon radius, then it treated as **Noise**.

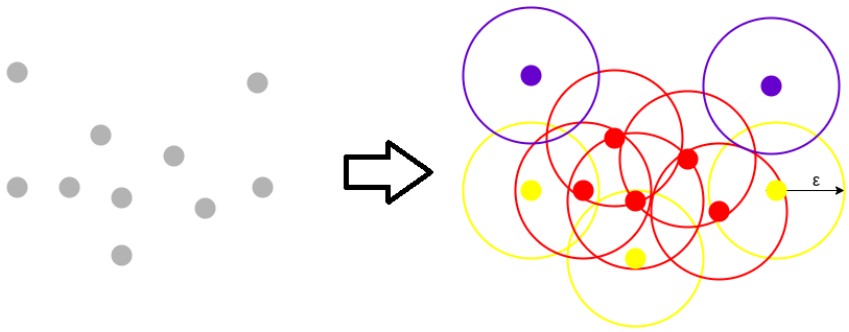

Figure 19: The above figure shows us a cluster created by DBCAN with minPoints = 3. Here, we draw a circle of equal radius epsilon around every data point. These two parameters help in creating spatial clusters.

All the data points with at least 3 points in the circle including itself are considered as Core points represented by red color. All the data points with less than 3 but greater than 1 point in the circle including itself are considered as Border points. They are represented by yellow color. Finally, data points with no point other than itself present inside the circle are considered as Noise represented by the purple color.

For locating data points in space, DBSCAN uses Euclidean distance, al-

though other methods can also be used (like great circle distance for geographical data). It also needs to scan through the entire dataset once, whereas in other algorithms we have to do it multiple times.

Parameter Selection in DBSCAN Clustering DBSCAN is very sensitive to the values of epsilon and minPoints. Therefore, it is very important to understand how to select the values of epsilon and minPoints. A slight variation in these values can significantly change the results produced by the DBSCAN algorithm.

The value of minPoints should be at least one greater than the number of dimensions of the dataset, i.e.,

$minPoints >= Dimensions + 1$

It does not make sense to take minPoints as 1 because it will result in each point being a separate cluster. Therefore, it must be at least 3. Generally, it is twice the dimensions. But domain knowledge also decides its value.

The value of epsilon can be decided from the K-distance graph. The point of maximum curvature (elbow) in this graph tells us about the value of epsilon. If the value of epsilon chosen is too small then a higher number of clusters will be created, and more data points will be taken as noise. Whereas, if chosen too big then various small clusters will merge into a big cluster, and we will lose details.

K-Means

K-means is a centroid-based clustering algorithm, where we calculate the **distance** between each data **point** and a **centroid** to assign it to a cluster. The goal is to identify the K number of groups in the dataset.

It is an **iterative process** of assigning each data point to the groups and slowly data points get clustered based on similar features. The objective is to minimize the sum of distances between the data points and the cluster centroid, to identify the correct group each data point should belong to.

Here, we divide a data space into **K clusters** and assign a **mean value** to each. The data points are placed in the clusters closest to the mean value of that cluster. There are several distance metrics available that can be used to calculate the distance.

The algorithm can be broken down into 4-5 steps:

1 **Choosing the number of clusters** The first step is to define the K number of clusters in which we will group the data

2 **Initializing centroids** Centroid is the center of a cluster but initially, the exact center of data points will be unknown so, we select random data points and define them as centroids for each cluster

3 **Assign data points to the nearest cluster** Now that centroids are initialized, the next step is to assign data points Xn to their closest cluster

centroid Ck. In this step, we will first calculate the distance between data point X and centroid C using Euclidean Distance metric. And then choose the cluster for data points where the distance between the data point and the centroid is minimum.

4 **Re-initialize centroids** We will re-initialize the centroids by calculating the average of all data points of that cluster.

5 **Repeat steps 3 and 4** We will keep repeating steps 3 and 4 until we have optimal centroids and the assignments of data points to correct clusters are not changing anymore

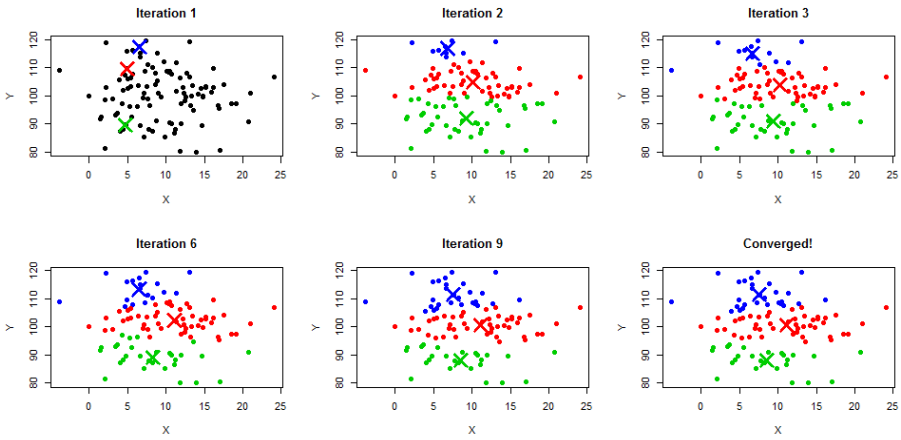

Figure 20: K-Means Iterations

4.3.3 Chosen algorithm

In this work, DBSCAN and K-Means clustering algorithms were used and compared. **K-Means performances were extremely better** than DBSCAN ones, so in this section the implementation of this algorithm will be analyzed.

The K-Means algorithm was implemented in **Python**, through the **scikit-learn library**. The code takes as input a dataset consisting of **rows of sessions** and **columns of features** discussed in previous chapters. All features are numeric, so they were normalized through the **StandardScaler** class of Scikit-learn. In contrast, the user agent, having been saved as a Bag of Word Expression, was normalized through the **CountVectorizer** class, again offered by scikit-learn.

In K-means clustering, **elbow method** and **silhouette analysis** techniques are used to find the number of clusters K in a dataset. The elbow method is used to find the "elbow" point, where adding additional data samples does not change cluster membership much. Silhouette score determines whether there are

large gaps between each sample and all other samples within the same cluster or across different clusters.

StandardScaler

StandardScaler is used to **resize the distribution of values** so that the mean of the observed values is 0 and the standard deviation is 1. StandardScaler is an important technique that is mainly performed as a preprocessing step before many machine learning models, in order to standardize the range of functionality of the input dataset.

StandardScaler comes into play when the characteristics of the input dataset differ greatly between their ranges, or simply when they are measured in different units of measure.

StandardScaler **removes the mean** and **scales the data** to the unit variance. However, outliers have an influence when calculating the empirical mean and standard deviation, which narrows the range of characteristic values.

These differences in the initial features can cause problems for many machine learning models. For example, for models based on the calculation of distance, if one of the features has a wide range of values, the distance will be governed by that particular characteristic.

The idea behind the StandardScaler is that variables that are measured at different scales do not contribute equally to the fit of the model and the learning function of the model and could end up creating a bias.

So, to deal with this potential problem, we need to standardize the data ($\mu = 0, \sigma = 1$) that is typically used before we integrate it into the machine learning model

CountVectorizer

CountVectorizer is a **text preprocessing technique** commonly used in natural language processing (NLP) tasks for converting a collection of text documents into a numerical representation. CountVectorizer operates by **tokenizing the text** data and counting the occurrences of each token. It then **creates a matrix** where the rows represent the documents, and the columns represent the tokens. The cell values indicate the frequency of each token in each document. This matrix is known as the "document-term matrix."
These are the **main advantages**:

- **Simplicity**: CountVectorizer is easy to use and understand. It has specific parameters and requires minimal configuration to get started with text preprocessing.
- **Speed and Efficiency**: CountVectorizer is computationally efficient and can handle large text datasets with many documents. It utilizes sparse matrix representations to save memory and processing time, especially when dealing with high-dimensional data.
- **Versatility**: CountVectorizer allows for flexible tokenization options, including handling n-grams (consecutive sequences of words) and custom

token patterns. It also provides opportunities for filtering stop words and controlling the vocabulary size.

- **Interpretable Results**: The resulting document-term matrix from CountVectorizer provides interpretable results. Each cell in the matrix represents the count or frequency of a token in a specific document, allowing for straightforward analysis and exploration

Elbow Method

The Elbow method is used to find the elbow in the elbow plot. The elbow is found when the dataset becomes flat or linear after applying the cluster analysis algorithm. The elbow plot shows the elbow at the point where the number of clusters starts increasing.

Recall that the basic idea behind partitioning methods, such as k-means clustering, is to define clusters such that the **total intra-cluster variation** [or total within-cluster sum of square (WSS)] **is minimized**. The total wss measures the compactness of the clustering, and we want it to be as small as possible. The elbow method runs k-means clustering (K-Means number of clusters) on the dataset for a range of values of K In the elbow method, we plot mean distance and look for the elbow point where the rate of decrease shifts. For each k, calculate the total within-cluster sum of squares (WSS). This elbow point can be used to determine K.

Perform K-means clustering with all these different values of K For each of the K values, we calculate **average distances to the centroid** across all data points. Plot these points and find the point where the average distance from the centroid falls suddenly ("Elbow"). At first, clusters will give a lot of information (about variance), but at some point, the **marginal gain will drop**, giving an angle in the graph. The number of clusters is chosen at this point, hence the "elbow criterion". This "elbow" can't always be unambiguously identified.

In this study, the elbow method was applied to estimate the number of K clusters between 1 and 50. The algorithm was implemented in Python by iterating the K-Means and plotting the obtained inertia values. **Inertia** is the sum of squared distances of samples to their closest cluster center. Finally, a **K-number of clusters of 12** was chosen, using the **KneeLocator library**.

We do not always have clear clustered data. This means that the elbow may not be clear and sharp, like in our study case

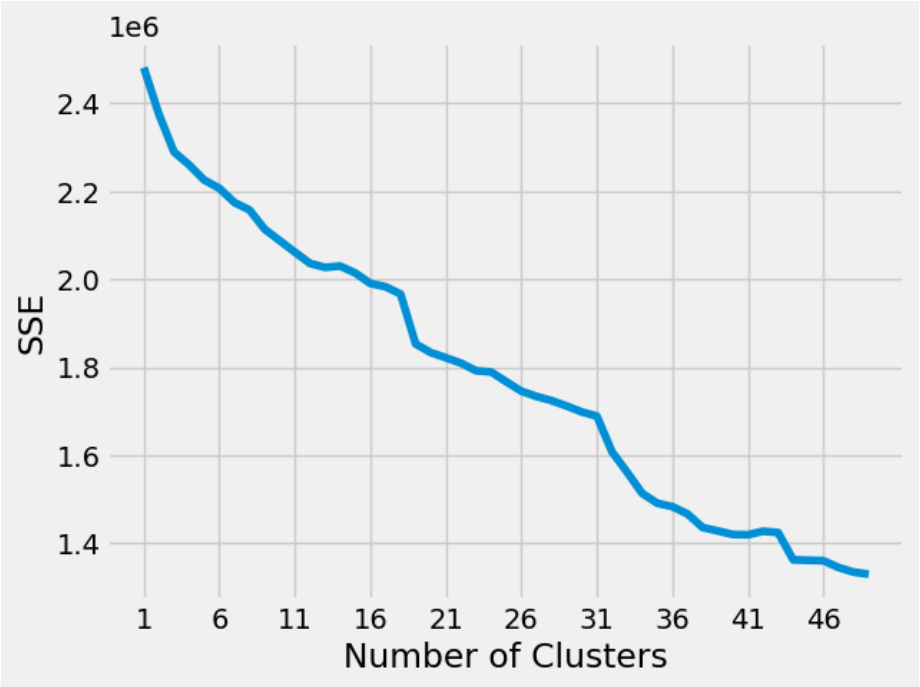

Figure 21: Elbow Method

Silhouette Score

Silhouette score is a measure of **how similar** a data point is within-cluster (**cohesion**) compared to other clusters (**separation**). The Silhouette score can be easily calculated in Python using the metrics module of the scikit-learn/sklearn library.

The equation for calculating the silhouette coefficient for a particular data point:

$$S(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

- S(i) is the silhouette coefficient of the data point i.
- a(i) is the average distance between i and all the other data points in the cluster to which i belongs.
- b(i) is the average distance from i to all clusters to which i does not belong.

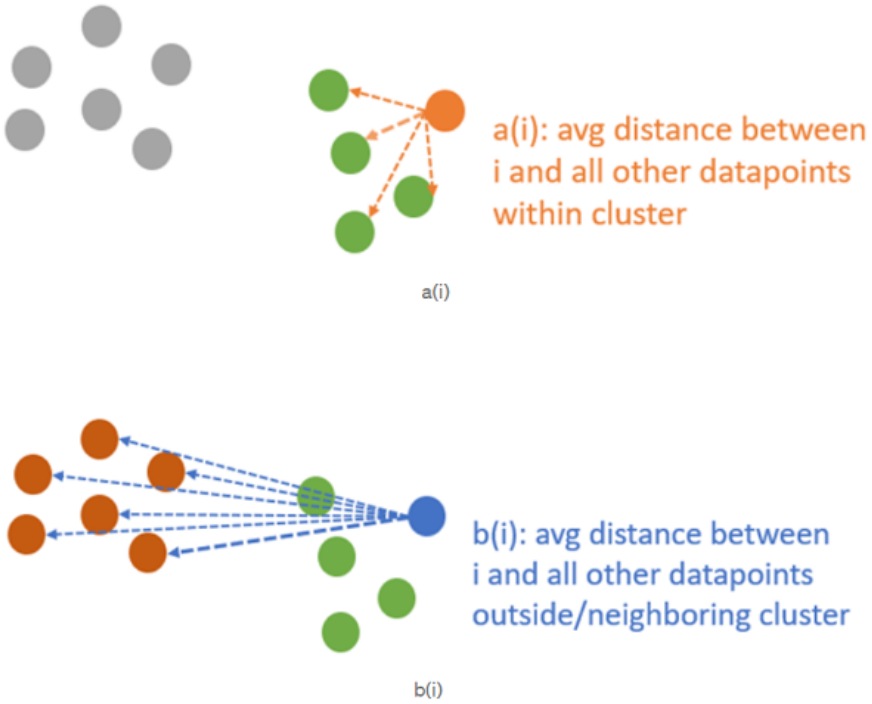

Figure 22: Silhouette Score Variables

We will then calculate the average Silhouette Score for every k with the formula:

$$\text{AverageSilhouette} = \text{mean}\{S(i)\}$$

Then plot the graph between Average Silhouette and K.

Points to Remember While Calculating Silhouette Coefficient:

- The value of the silhouette coefficient is between [-1, 1].
- A score of 1 denotes the best, meaning that the data point i is very compact within the cluster to which it belongs and far away from the other clusters.
- Values near 0 denote overlapping clusters.
- The worst value is -1.

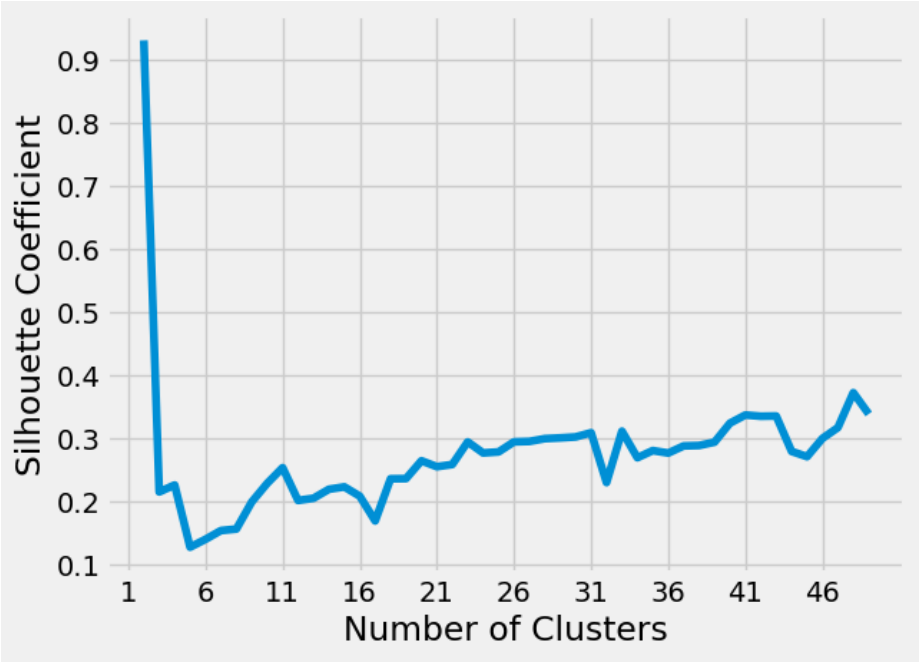

Figure 23: Silhouette Coefficient

In this study, the Silhouette Score was applied to estimate the number of K clusters between 2 and 50. The algorithm was implemented in Python by iterating the K-Means and plotting the obtained Silhouette Coefficients. Finally, a **K-number of clusters of 12** was chosen, confirming the result obtained with the Elbow Method.

5 Results

5.1 Cluster Analysis

The results obtained from clustering were analyzed by plotting **pairs** and **triples** of features against centroids with **Python**. To do this, the **matplotlib** library was used , iterating over all possible combinations of pairs and triples.

I would have initially expected to see some feature points around the centroids in a very pronounced way, but after analyzing the results, particularly the sparsity of the points around the centroids, I realized that the algorithm did not give more importance to some pairs or triples of features than others, which is a very good behavior.

Below are some plots of the features versus centroids:

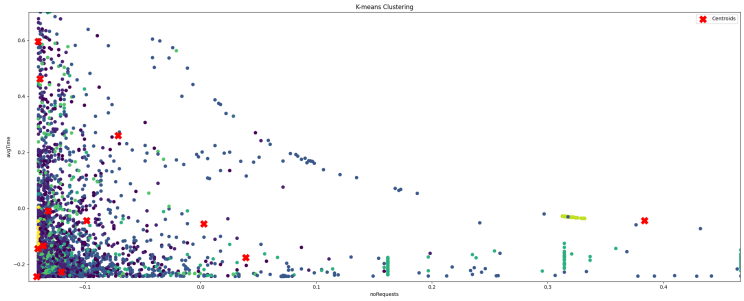

Figure 24: Centroids on Average Time vs Number of Requests

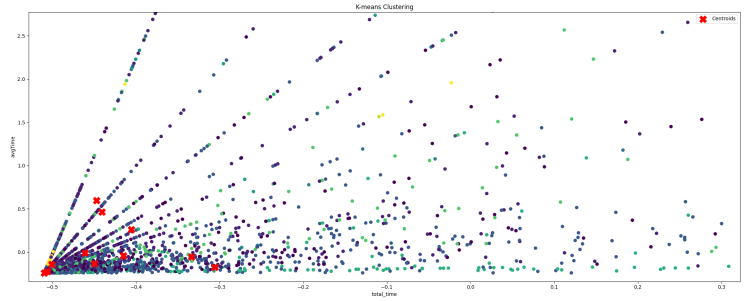

Figure 25: Centroids on Average Time vs Total Time

Figure 26: Centroids on Depth vs Recurrence

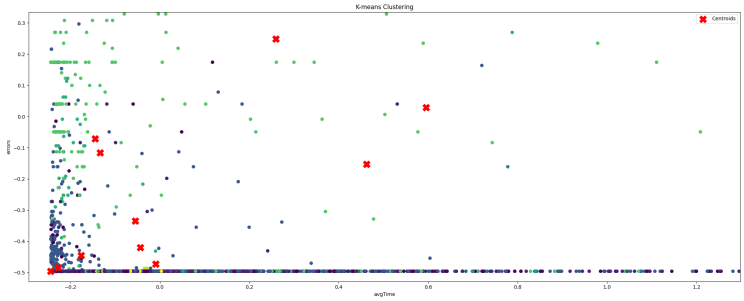

Figure 27: Centroids on Errors vs Average Time

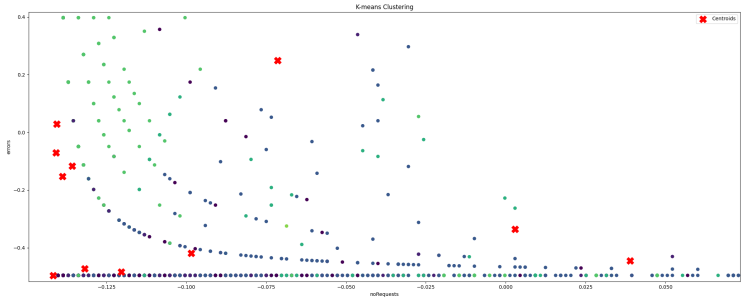

Figure 28: Centroids on Errors vs Number of Requests

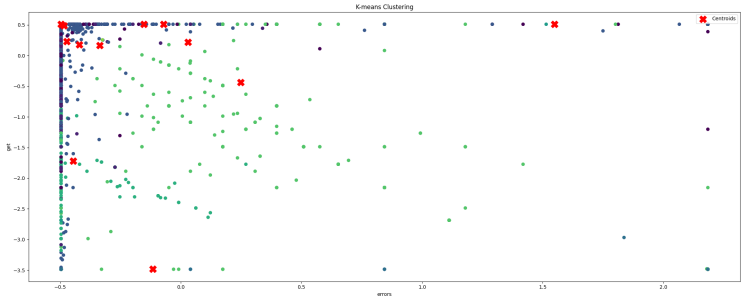

Figure 29: Centroids on Get vs Errors

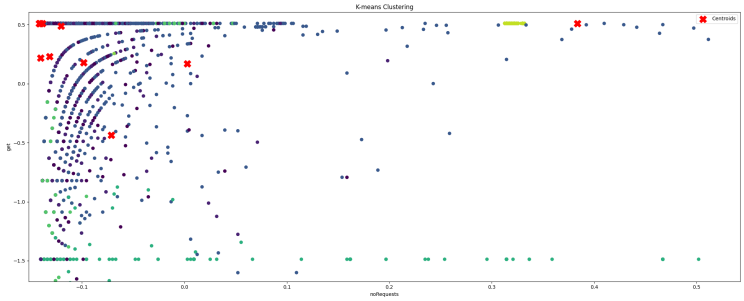

Figure 30: Centroids on Get vs Number of Requests

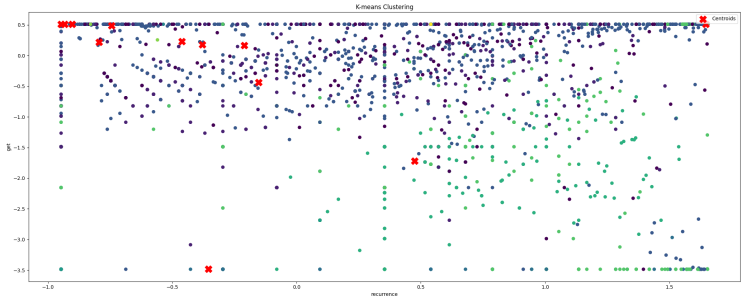

Figure 31: Centroids on Get vs Recurrence

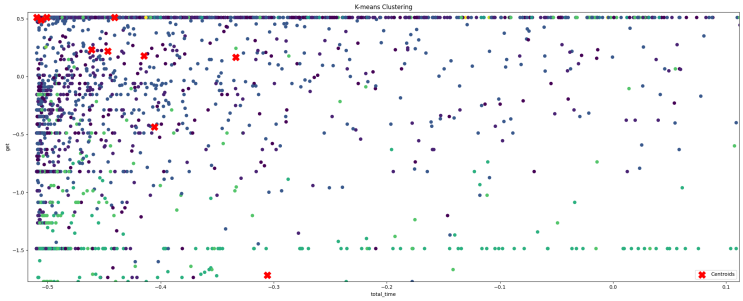

Figure 32: Centroids on Get vs Total Time

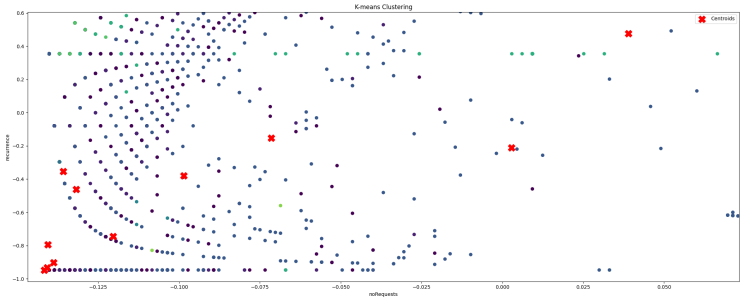

Figure 33: Centroids on Recurrence vs Number of Requests

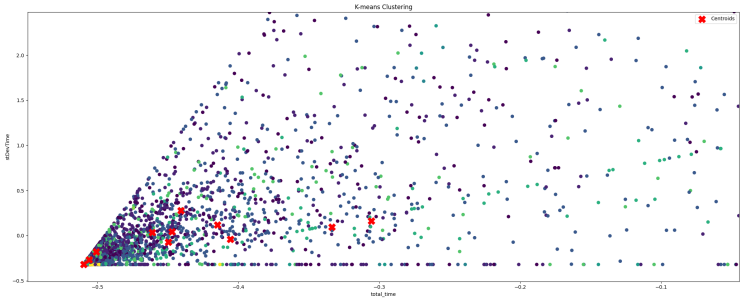

Figure 34: Centroids on Standard Deviation vs Total Times

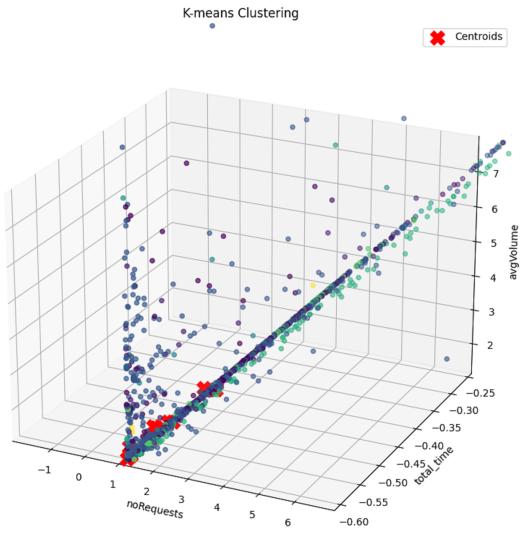

Figure 35: Centroids on Number of Requests vs Total time vs Average Volume

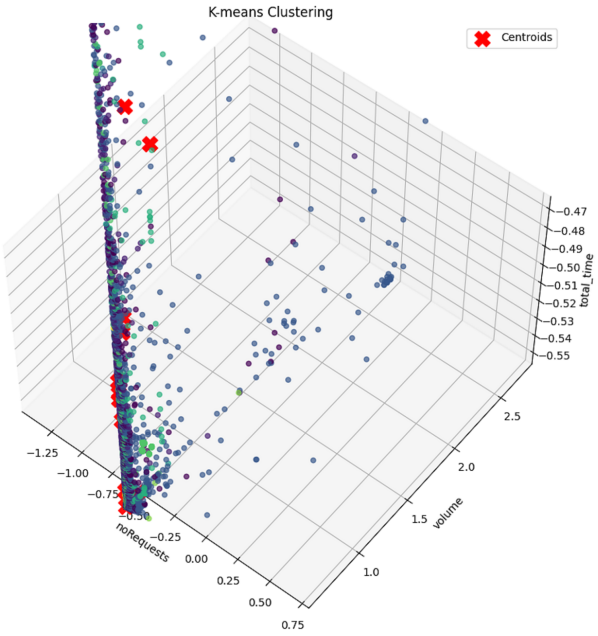

Figure 36: Centroids on Number of Requests vs Total time vs Volume

5.2 Result Analysis

In order to achieve the goal of this thesis, the result obtained from clustering was not enough. The next step was to create a **heuristic** to assign a label to each of the obtained clusters. This process was done through an algorithm written in **Python** with the **user_agents library**.
For each cluster, a **binary label** (Programmatic or Human) was assigned to each contained session by analyzing only the user-agent, then assigning to the cluster the label corresponding to the most frequent label for that cluster, and then calculating an expected accuracy by taking the most frequent label as a reference.

Table 2: Cluster Information

Cluster	Label	No of Bots	No of Humans	Expected Accuracy
0	Human	23	1092	97.94%
1	Human	10	1789	99.44%
2	Programmatic	5	0	100.00%
3	Human	28	1904	98.55%
4	Programmatic	148	0	100.00%
5	Programmatic	72	0	100.00%
6	Programmatic	134	0	100.00%
7	Human	3	496	99.40%
8	Programmatic	1016	32	96.95%
9	Programmatic	11	3	78.57%
10	Programmatic	1281	0	100.00%
11	Programmatic	309	0	100.00%

The goal of the thesis is met when in a cluster labelled as "Programmatic" we find some sessions labelled as "Human" (or when the expected accuracy is lower than 100%). This means that a session with a human user-agent had bot behavior and was assigned to a programmatic cluster. In other words, we uncovered a bot that was masking its user agent.

To make an example, in the previous table we found a total of 35 Bots, 32 belonging to *cluster 8* and 3 belonging to *cluster 9* respectively.

To verify that the result was reliable, for each of these 35 bots, the user-agent, service-id and timestamp related to the sessions were reported to **aizoOn's cybersecurity department** in order to **manually check** the behavior of those 35 users. **The response was positive**, confirming the successful detection of these bots that were faking their user-agent.

6 Conclusions and future developments

In this thesis a **novel method** for Web bot detection based on HTTP requests was proposed. The proposed **K-Means algorithm**, with the **selected features** obtained from our definition of the "**session**", demonstrated very good performance, obtaining **no false positives** within the programmatic clusters.

In contrast to other bot detection approaches, this method does not impose the domain to be a specific one, but it offers a **general solution** for every kind of domain, for example e-commerce traffic, banks traffic, personal web-sites and so on.

In addition, the method provides a **unsupervised machine learning solution**, which in this type of problem is a big advantage for two reasons. The first reason is that it is very difficult to obtain a dataset of labeled HTTP requests, as it would require human intervention. The second reason is that **the world of bots is constantly evolving**, so a dynamic approach such as clustering can remain valid over time.

The results provide a promising direction for future developments. With the aizoOn cybersecurity department, we are currently investigating the implementation of this machine model into the WAF Mithril, with the goal of offering a new service enhancing the product. The main challenges are to implement this model while taking into account the **computational and cost constraints of Mithril**, as it is a service that must operate on the order of milliseconds. The paths that will be explored are respectively an **offline approach**, thus without getting real-time results, and a **live approach**, looking for solutions to parallelize the creation of real-time sessions, thus moving the storage of HTTP requests from *ElasticSearch* to services such as *Redis* or *Kafka*.

7 References

- [1] Suchacka, G.; Cabri, A.; Rovetta, S.; Masulli, F. Efficient On-the-Fly Web Bot Detection. Knowl.-Based Syst. 2021, 223, 107074
- [2] V. Almeida, D. Menascé, R. Riedi, F. Peligrinelli, R.Fonseca, W. Meira Jr, Analyzing web robots and their impact on caching, in: Proceedings of the Sixth International Workshop on Web Caching and Content Distribution, June 2001, pp. 299–310.
- [3] A. Stassopoulou, M. Dikaiakos, Web, Robot detection: A probabilistic reasoning approach, Computer Networks 53 (2009) 265–278
- [4] S. Rovetta , G. Suchacka, F. Masulli , Bot recognition in a Web store: An approach based on unsupervised learning, Journal of Network and Computer Applications 157 (2020) 102577
- [5] J. Srivastava, R. Cooley, M. Deshpande, P.-N. Tan, Web usage mining: discovery and applications of usage patterns from web data, SIGKDD Explorations 1 (2) (2000) 12–23
- [6] T. Tanaka , S. Liaa, S. Nomura, Bot Detection Model using User Agent and User Behavior for Web Log Analysis, 24th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems
- [7] https://fourweekmba.com/google-revenue-breakdown/
- [8] Zhang, Yang, Detecting malicious activities with user-agent-based profiles. International Journal of Network Management 25.5 (2015): 306-319
- [9] M. Rahman, M. Rahman, B. Carbunar, D.H. Chau, Search rank fraud and malware detection in google play, IEEE Trans. Knowl. Data Eng. 29 (6) (2017) 1329–1342
- [10] V. Sharma, R. Kumar, W.-H. Cheng, M. Atiquzzaman, K. Srinivasan, A.Y. Zomaya, NHAD: Neuro-fuzzy based horizontal anomaly detection in online social networks, IEEE Trans. Knowl. Data Eng. 30 (11) (2018) 2171–2184
- [11] J. Lee, S. Cha, D. Lee, H. Lee, Classification of Web robots: An empirical study based on over one billion requests, Comput. Secur. 28 (8) (2009) 795–802
- [12] G. Suchacka, M. Sobków, 2015. Detection of Internet robots using a Bayesian approach. Proceedings of IEEE 2nd International Conference on Cybernetics. IEEE, pp. 365–370
- [13] C. Bomhardt, W. Gaul, L. Schmidt-Thieme, Web robot detection – preprocessing Web logfiles for robot detection, in: New Developments in Classification and Data Analysis, Springer, Berlin, Heidelberg, 2005, pp. 113–124

- [14] S. Rovetta, A. Cabri, F. Masulli, G. Suchacka, Bot or not? A case study on bot recognition from Web session logs, in: Quantifying and Processing Biomedical and Behavioral Signals, in: Smart Innovation, Systems and Technologies, vol. 103, Springer, 2019, pp. 197–206
- [15] D.S. Sisodia, S. Verma, O.P. Vyas, Agglomerative approach for identification and elimination of web robots from web server logs to extract knowledge about actual visitors, J. Data Anal. Inf. Process. 03 (2015) 1–10
- [16] M. Zabihi, M.V. Jahan, J. Hamidzadeh, A density based clustering approach to distinguish between web robot and human requests to a Web server, ISC Int. J. Inf. Secur. 6 (1) (2014) 77–89
- [17] D. Stevanovic, N. Vlajic, A. An, Detection of malicious and non-malicious website visitors using unsupervised neural network learning, Appl. Soft Comput. 13 (1) (2013) 698–708