

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

BotsBFUOD: Web Bot Detection using Biometric Features and Unsupervised Outlier Detection

Pedro C. Pereira



Mestrado em Engenharia Informática e Computação

FEUP Supervisor: Prof. Miguel Monteiro

Host Institution: Jscrambler S.A.

Jscrambler Supervisor: João Routar

October 17, 2022

BotsBFUOD: Web Bot Detection using Biometric Features and Unsupervised Outlier Detection

Pedro C. Pereira

Mestrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: Prof. Gil Gonçalves

External Examiner: Prof. Rolando Martins

Supervisor: Prof. Miguel Monteiro

October 17, 2022

Abstract

A bot is defined as a autonomous piece of software that wanders the application layer of the Internet to perform a certain task they were designed to. These can be classified as either "good" or "bad" depending on the intent with which they request resources from applications, if they follow the rules laid out for their activity, and what they do with the information they obtain. An example of a good robot would be GoogleBot, one whose purpose is to better match user's queries on their platform. Examples of bad bot activity include but are not limited to: vulnerability scanning, scraping (collecting proprietary data without the owners consent to use elsewhere), scalping, fraud, denial of service attacks etc. Though the existence of these agents on the Web is not something new, their numbers have been steadily increasing over the years, and have had an even larger growth than before in this current Pandemic. As with most issues of cybersecurity, this is one where there seems to be a never-ending battle between attackers and defenders. The role of the defender in this conflict, and the motivation of this work is to advance the science of detecting human and non-human or bot behaviour on the Web. This paper will present a proof of concept, for a system of Bot Detection based on Biometric Features and Unsupervised Outlier Detection. This system uses a combination of attributes found in literature and some novel ones to model user behaviour at a certain point in time in a session. It then analyses if at that point, the session is deemed anomalous, and assumed to be generated by a bot.

Keywords: Web Robot Detection, Biometric Patterns, Unsupervised Learning

Acknowledgements

As the main author of this work, I would like to use this section to acknowledge everyone who, directly or indirectly, assisted in the development of this thesis.

Firstly, as this work was only possible with their collaboration and cooperation, I would like to thank Jscrambler. This organization and its collaborators were vital to this work. From providing the necessary infrastructure, to volunteering your time and data, all your contributions supported this project. A special thanks to the Web Engineering team for their help in the implementation of necessary web resources. Namely, José Silva and Edgar Araújo. José for taking time-off from his vacation to squash a bug. Edgar, for all the meetings, the help with Web Dev, and the after hours casual talks.

Secondly, I would like to express my gratitude towards my friends and family. For every time you had to listen to me talk about this thesis. For every afternoon of work spent alongside you. For every time one of you gave me shelter, when I needed a change of scenery. And, for always believing in me, and helping me deal with any anxiety that came in this process. Thank you!

Finally, to my co-authors. Professor Miguel Monteiro of University of Porto and João Routar, Lead Research Data Scientist at Jscrambler. Thank you for your guidance and direction in this venture. For all your feedback, for all the help with bureaucracy and for all the weekly meetings. All your work was greatly appreciated!

Pedro C. Pereira

“Man is condemned to freedom”

Jean-Paul Sartre

Contents

1	Introduction	1
1.1	Context and Motivation	1
1.2	Objectives	2
1.3	Document Structure	2
2	Web Bots	3
2.1	Overview	3
2.1.1	Basic Concepts	3
2.1.2	Development Technologies & Tools	4
2.2	Detection State of the Art	6
2.2.1	Navigational Patterns	6
2.2.2	Biometric Patterns	11
2.2.3	Browser Fingerprinting	12
2.2.4	Other Approaches	13
2.2.5	Summary	14
3	Outlier Detection	19
3.1	Overview	19
3.1.1	Local Outlier Factor	20
3.1.2	Isolation Forest	21
3.1.3	Autoencoder	22
3.2	Data Preprocessing	24
3.3	Performance Evaluation	25
4	Detector Implementation	29
4.1	Detection Methodology	29
4.2	Data Extraction	29
4.2.1	Requirements	30
4.2.2	Biometric Data Extracted	30
4.2.3	Human Data	33
4.2.4	Bot Data	34
4.3	Data Processing	36
4.3.1	Session Aggregator	36
4.3.2	Data Preprocessing	37
4.4	Resulting training data	38
4.5	Classifiers Trained	39
4.6	System Overview	39

5	Experiments, Results and Discussion	41
5.1	Tuning Model Parameters	41
5.1.1	Parameters: Isolation Forest	42
5.1.2	Parameters: AutoEncoder	42
5.1.3	Parameters: LOF	43
5.2	Experiment 1: Packet-based Web Bot Detection	43
5.3	Experiment 2: Session-Based Web Bot Detection	51
5.4	Conclusions from Experiments	52
6	Conclusions and Future Work	53
	References	55
A	Appendix	59
A.1	Biometric Data Collector: bc.js	59
A.2	Detector Modules	70
A.3	Bots Developed	90

List of Figures

2.1	Works that utilize Nav. Features vs. Works that do not	7
2.2	Example Web Server Access Log Entry [Stevanovic et al., 2012]	8
3.1	Isolating an inlier (a/xi) vs. Isolating an outlier (b/xo) [Liu et al., 2009]	22
3.2	Examples of single (a) and multi-layer (b) ANNs [Aggarwal, 2016]	23
3.3	Example of autoencoder architecture [Aggarwal, 2016]	24
3.4	Confusion Matrix example	26
3.5	Precision-Recall Curve Example Comparing Multiple Algorithms [Aggarwal, 2016]	27
3.6	Example of Perfect Classifier and Random Classifier ROC [Alla and Adari, 2019]	28
4.1	Example Human Behaviour Diagram	33
4.2	Crawler Behaviour Diagram	35
4.3	Crawler Behaviour Diagram	35
4.4	Proposed Solution System Overview	40
5.1	Isolation Forest Confusion Matrix	44
5.2	Isolation Forest ROC Curve	45
5.3	Isolation Forest Precision-Recall Curve	45
5.4	AutoEncoder Confusion Matrix	46
5.5	AutoEncoder ROC Curve	47
5.6	AutoEncoder Precision-Recall Curve	48
5.7	LOF Confusion Matrix	49
5.8	LOF ROC Curve	49
5.9	LOF Precision-Recall Curve	50
5.10	BAARZS	51

List of Tables

2.1	Total Downloads for Selenium Python and Puppeteer.js from 12-19th Sep. 2022 .	4
2.2	Papers that use Nav. Feats. and papers that do not	7
2.3	Common Navigational Features	9
2.4	Summary of Learning Approaches	14
4.1	Biometric Features Selected	31
4.2	Data Set Properties	39
5.1	IForest Performance Metrics	44
5.2	AutoEncoder Performance Metrics	46
5.3	LOF Performance Metrics	48
5.4	Data Set Properties	52

Abbreviations

AI	Artificial Intelligence
(A)NN	(Artificial) Neural Network
CAPTCHA	Completely Automated Public Turing test to tell Computers and Humans Apart
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DDoS	Distributed Denial of Service
FN	False Negatives
FP	False Positives
HTTP	Hypertext Transfer Protocol
IForest	Isolation Forest
IP	Internet Protocol
LOF	Local Outlier Factor
OD	Outlier Detection
OS	Operating System
PoC	Proof of Concept
ROC	Receiver Operating Characteristic
SL	Supervised Learning
SOM	Self-Organizing Map
SVDD	Support Vector Data Description
TN	True Negatives
TP	True Positives
UA	User Agent
UL	Unsupervised Learning

Chapter 1

Introduction

1.1 Context and Motivation

Since the early days of the World Wide Web, individuals and organizations have developed software agents to automate different tasks. These agents that inhabit the Web are Web Robots or Web Bots. In the early 90s, a series of incidents involving automated agents led to the development of the Robot Exclusion Protocol/Standard. A file "/robots.txt" is the way for server administrators to dictate what is accessible to what bots on their website. However, due to the Request-Response nature of the Web, the standard is not enforceable [[Koster, 2007](#)].

A common classification for bots is distinguishing the "good" from the "bad". This system considers the intentions behind the requests, if the agents follow the Exclusion Protocol and what they do with the information obtained/accessed. Some applications of good bots are search engine indexers who work to better their engine's results, site monitoring bots who monitor performance and availability, etc. Even though these generally aren't harmful, detecting them may still be advised as it may be beneficial to control their access during periods of higher bandwidth usage [[DataDome, 2019](#)]. Regarding bad bots and their use cases, [[Watson and Zaw, 2018](#)] lists twenty-one events that include: vulnerability scanning, scalping, denial of service, and scraping (collecting data without the author's consent to use elsewhere) as reasons for wanting to detect these unethical agents.

As the Web grew and evolved, bot traffic also grew and the trend seems to be for it to continue to grow. A yearly industry report [[Imperva, 2021](#)] estimated 40.8% of all web traffic in 2020 was by bots, with 25.6% of the total being bad bot traffic. This report also notes how the pandemic situation boosted the growth of bot presence online, with a particular highlight on misinformation spreading bots and scalper bots.

Finally, the idea that bots can interfere with Web Usage Analysis has also been used as motivation for bot detection [[Lagopoulos et al., 2018](#)].

Regarding the motivation for the development of a solution based on Biometrics and Unsupervised Outlier Detection; as Chapter 2 will display, machine learning systems for bot detection have focused mainly on navigational characteristics of how a website is accessed (e.g. no. of

requests, error rates, etc.). [Tan and Kumar, 2002] When considering Biometric Features, these have been explored (often combined with navigational features) in supervised learning contexts, where assumptions are made about the data. [Chu et al., 2013] In contrast to this, this project will present a, proof of concept (PoC), purely biometric system that utilizes Unsupervised Outlier Detection, where the only assumption is that in the data the system will be exposed to, Bots are less prevalent. On that basis, outliers in this system will be considered bots.

1.2 Objectives

With the end goal of developing a Bot Detection system, the objectives of this work are threefold. The first step is to analyse the State of the Art in Bot Detection. By studying Bot Detection, what has been done and what can still be done will become more apparent. Current techniques and approaches will be examined and possible gaps in literature identified.

Following that, bot development must also be considered to achieve a higher understanding of the problem. By knowing how bots work, new, improved features can be designed for their detection. Furthermore, bots will need to be developed to test the solution once implemented.

To conclude, based on the knowledge exposed in the SoA sections of this work, a Bot Detection System will be implemented, and experimented upon. The resulting system is expected to, at least to some degree and with a reasonable error rate, be able to evaluate at some given points in a user's session, if this user is a Human or Web Robot.

The main contributions of this work are:

- the set of biometric features developed for detection;
- the study of how Biometric Features and Unsupervised Outlier Detection techniques can be used for Bot Detection.

1.3 Document Structure

The rest of this document is structured as follows. Chapters 2 and 3 cover the state of the art in the relevant fields for this project. The first, reviews literature on Bots. Be it detection and prevention, or bot development. The second, on Unsupervised Outlier Detection, covering basic concepts, evaluation methods and the algorithms picked for this project. Chapter 4 presents an overview of the developed system - from the features selected and how they were collected, through session rebuilding and data preprocessing. Following that, Chapter 5 describes the culmination of the work exposed in previous chapters in two experiments. Finally, chapter 6 concludes and suggests future work.

Chapter 2

Web Bots

To better understand the problem of Web Bot Detection, a literature review was conducted on the topic. In the interest of better comprehension of the subject matter and even to develop tests for the solution, literature on Bot Development, few as it may be, may not be neglected. As such, this chapter is composed of sections including both topics. The first section will cover basic concepts on Bots, use cases and development technologies and tools. The following section will narrow the focus to the State of the Art in Bots Detection.

2.1 Overview

2.1.1 Basic Concepts

The first concept to define is Bot. What is a bot? A (Ro)bot, is a piece of software that automates a given task. Web bots are bots that operate on the Web, performing tasks across any number of domains [[DataDome, 2019](#)].

In [[Doran and Gokhale, 2011](#)], the authors classify bots as either Good or Bad. Good bots explicitly follow the Robot Exclusion Protocol defined by the domain and don't interfere with other users' experience with the platform. Bots can be bad, either because they were built with ill-intent as automated attacks, or because they inadvertently affect other users' sessions. In the cited work, the authors give the example of a link verifying bot that sends GET requests instead of HEAD requests, thus consuming more bandwidth than necessary and possibly impacting others. Examples of purposefully bad bots will be discussed later in this section.

When discussing automated attacks, [[Chu et al., 2013](#)] categorizes bots by their behaviour or mode of operation. The paper focuses on three types of bots: Form Injection, Human Mimics and Replay bots. Form Injection bots directly send the HTTP request with the parameters filled to the target page. These bots don't interact through a browser. A step above in complexity, Human Mimic bots, use browsers and basic mouse movements and keyboard presses to try to emulate human behaviour. However, because human behaviour is often somewhat chaotic, these bots end up following betraying patterns. Finally, the most complex bots to detect are replay bots. These bots first record the actions of a human traversing the page and then replay the actions.

Bots have as many use cases as there are tasks that can be automated on the Web. Examples of generally considered "Good" use cases can be: search engine indexers (that crawl the web to add more relevant search results to their engines), site monitoring scripts, link checkers (that check if a given link is available to access) [DataDome, 2019]. In [Watson and Zaw, 2018], OWASP categorizes the possible automated attacks. These include: scraping, vulnerability scanning, denial of service and scalping. During the Pandemic, [Imperva, 2021] reported a rise in bot activity across the web. Scalping bots in particular were found to be the faster growing type of bot, due to the supply chain constraints that the Pandemic brought about.

Most research into Web Bot Development focuses either on how to more efficiently scrape information or how to train conversational bots (those that communicate with humans via a chat). Under the assumption that bot developers plan to follow the Robot Exclusion Protocol, from a scholarly point of view, there is little reason to try and mask a bot as a human. That's not to say there has been no research on this topic, however it has, mostly, been done in more informal settings.

2.1.2 Development Technologies & Tools

This section will cover important technologies, evasion tactics and tools used in bot development.

Automation tools allow developers to build more complex bots, that interact with a page, rather than just getting its content and processing it offline [Imperva, 2020]. The Web Browser Testing Framework Selenium is mentioned in literature as a starting point for these bots [Khder, 2021] [Zhao, 2017] [Amin Azad et al., 2020].

Another testing framework popular in the aforementioned informal settings is Puppeteer.js. There is a substantial amount of debate on which one is better as can be seen in [Wickramasinghe, 2021] and [Singh, 2021]. Total downloads in the week of 12th September 2022, for each tool, displayed in Table 2.1.

Tool	Downloads from 12-19th Sep. 2022	Source
Selenium Python	3,468,021	[PyPIStats, 2022]
Puppeteer.js	3,899,231	[npmjs, 2022]

Table 2.1: Total Downloads for Selenium Python and Puppeteer.js from 12-19th Sep. 2022

It is worth noting that, while the Selenium total downloads only consider the most commonly used language Python. The library itself is available in other languages like Java, Ruby, and others [Selenium, 2022].

As most commercial bot detection solutions are based primarily on device fingerprinting, some work has been done in simulating human device fingerprints. Often coupled with Selenium, there is, for example, the Selenium Stealth Library, that promises results even against some of the most well known bot detection solutions [Patra, 2020].

The field of mimicking human mouse movement has seen some research. Models that generate human-like trajectories using machine learning (specifically GANs) and datasets of real human

mouse movements have been proposed as can be seen in [Acien et al., 2020]. However, for the purpose of this project, developing such a solution was deemed unnecessary.

2.1.2.1 WindMouse Algorithm

This subsection will cover the WindMouse Algorithm. This algorithm was designed to create simple, yet human-looking, mouse trajectories to evade Bot Detection Systems. Furthermore, as performance was one of the author's main concerns, it consumes minimal processing time. As such, and given that it claims to have been used in production environments, [Land, 2021], it was considered relevant to include in its own section.

An algorithm, as its author puts it: "inspired by high-school physics", WindMouse uses the concepts of "gravity" and "wind" to create mouse paths not unlike those a human would create, at least to the naked eye [Land, 2021]. This algorithm calculates the trajectory between 2 points (x, y) in a two dimensional plane using 4 other inputs. The mouse cursor is treated as an object being impacted by 2 external forces that move it, Wind and Gravity (inputs 1 and 2). Gravity pulls the object towards its final destination, while Wind adds force in a random direction, smoothly changing in power and direction with time. At any given point the total force on the object is the sum of the wind at that point in time, with the gravity at that position. (The position of the object will also depend on the time passed).

$$\vec{F}(t) = \vec{W}(t) + \vec{G}(\vec{x}(t)) \quad (2.1)$$

F	Total Force on Object
t	Current Point in Time
W	Wind Force
G	Gravitational Force
x	Current Position

The algorithm's developer recalls Newton's 2nd Law, as "the force on an object is the second derivative of position". ($\vec{F} = m\vec{a} = m\ddot{\vec{x}}$) As such, they formulate that to find the position one needs to integrate the force to obtain the velocity over that period of time and after that integrate the velocity to obtain the position.

$$x(t) = \int_0^t \int_0^t \vec{F}(T) dT^2 \quad (2.2)$$

So as to not make impossible movement, a parameter for maximum velocity can be set (input 3). Finally, a distance can be set for which the speed will be reduced so that when the cursor is close to its target it is not thrown off route by the wind [Land, 2021].

2.2 Detection State of the Art

This section will review relevant current literature in bot detection technologies. The review of papers mainly includes work that was conducted after 2004. The year when Selenium, one of the most popular browser automation tools, was introduced. Tools like Selenium allow bot developers to create bots that interact with websites in much the same way humans do, generating mouse and keyboard events [Selenium, 2022]. In [Doran and Gokhale, 2011], the authors describe how Web Bot Detection has evolved into two main categories of systems: Analytical Learning and Turing Test Systems. As the name implies, in Analytical Learning systems, Artificial Intelligence (AI) models learn to classify behavior on the Web as Human or Bot made, by analyzing different features of it. Turing Test Systems challenge users to prove their humanity or trick bots into signaling their presence. As this project will focus on the development of a Learning System, this area will be analysed in more depth and will be the first subsection. Following that, and as it is relevant to current industry practices, the topic of browser fingerprinting will be explored. As it relates to the proposed solution, the learning of Biometric Patterns will have its own section. Finally, this section will briefly present areas less relevant to this undertaking but nonetheless important.

2.2.1 Navigational Patterns

The most prolific research area in Web Bot Detection is the learning of navigational features and patterns to identify Human vs. Bot behavior. These features and patterns give information on how the site was navigated in the course of a user's session. Examples of these can be: session duration, percentage of failed requests, types of requests and their ratios, etc. A more complete list of these characteristics can be seen in Table 2.3. In [Chen et al., 2020], the authors explore the three most active research areas in Bot Detection. Using the nomenclature from [Doran and Gokhale, 2011], what the authors of [Chen et al., 2020] consider the three major fields of BD fall in two categories: Turing Test Systems and Analytical Learning systems. The second category is interpreted as two separate in regards to whether they act online or offline in [Chen et al., 2020], yet in this section will be considered just the one. The initial claim is further supported by in the papers used for this state of the art review. This review aimed at exploring as many relevant approaches to Bot Detection as possible. To that end, the inclusion criteria requires either for an approach to introduce novel concepts or models, or for the paper to be cited by over ten others. In cases where multiple works analysed the very same approach, the most cited work, usually the original, is used. Of the 17 unique solutions to Bot Detection included, 12 utilized some sort of navigational attributes, the remaining 5 use either biometrics and/or Turing Tests. Figure 2.1 represents this ratio graphically. Table 2.2, names the papers in each category.

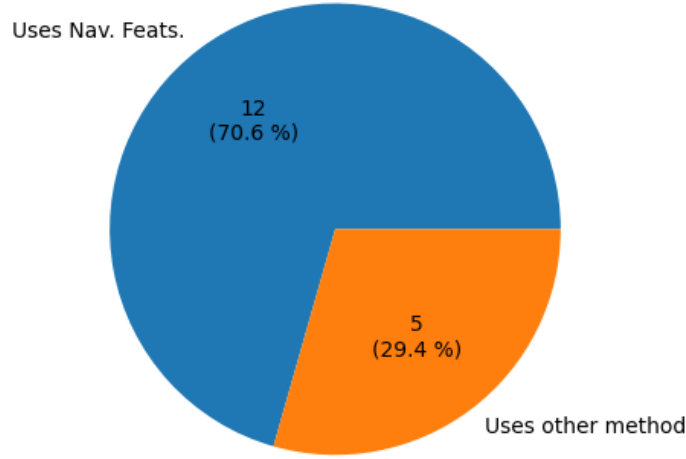


Figure 2.1: Works that utilize Nav. Features vs. Works that do not

Table 2.2: Papers that use Nav. Feats. and papers that do not

Use Nav. Feats.	Do not use Nav. Feats.
[Tan and Kumar, 2002]	[Hayati et al., 2010]
[Stassopoulou and Dikaiakos, 2009]	[Park et al., 2006]
[Suchacka and Sobkow, 2015]	[Chu et al., 2013]
[Stevanovic et al., 2012]	[Lagopoulos et al., 2018]
[Cabri et al., 2018]	[Vikram et al., 2013]
[Stevanovic et al., 2013]	[McKenna, 2016]
[Zabihimayvan et al., 2017]	
[Zolotukhin et al., 2014]	
[Jacob et al., 2012]	
[Rahman and Tomar, 2020a]	
[Iliou et al., 2021]	

These attributes are often extracted either from Server Access Logs, or from the requests as they come in [Chen et al., 2020]. Server Access Logs are files containing details on every web request to a domain. Each line in a log file contains the following data: request source IP address, a timestamp of the request, the HTTP method used, the requested file, the response code and finally, the UA of the request source. An example entry for a web log was taken from [Stevanovic et al., 2012] and can be seen in Figure 2.2.

```
122.248.163.1 - - [09/Jan/2011:04:37:38 -0500] "GET /course_archive/2008-09/W/3421/test/testTwoPrep.html HTTP/1.1" 200 5645
Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.-com/bot.html)
```

Figure 2.2: Example Web Server Access Log Entry [Stevanovic et al., 2012]

Initially, papers on this field mainly analyzed data from Access Logs and so had to extract session information after the fact. The process of session extraction is usually done with a timeout value and some comparable characteristics. The most basic approach just groups requests in a thirty-minute interval with the same IP address and UA (user agent) string. Other approaches consider dynamic timeout values and/or more complex contiguity checks, such as [Stassopoulou and Dikaiakos, 2009] and [Tan and Kumar, 2002].

2.2.1.1 Supervised Learning Approaches

Starting with the Supervised Learning (SL) Techniques, one of the most widely cited papers on the topic was [Tan and Kumar, 2002]. This work presented a classification approach based on some variations of traits in Table 2.3 and the C4.5 decision tree algorithm. Besides that, it also expanded the area of session identification, proposing a new scheme that could identify sessions across different IP addresses and UA names. To test the solution, Server Access Logs for the month of January, 2001, were collected. These counted 1.6 million entries belonging to an estimated, 180 thousand sessions. These were manually labeled to confirm detection results. At the time, the accuracy of 90% was achieved, which was fairly high by the standards of the time. Contributing to this was a new method for identifying mislabeled samples. By building multiple classifiers, the ensemble of their evaluations of a piece of data could be used to tell if the data was mislabeled in the first place. If a majority of the classifiers failed (either by false positive or negative) when classifying the sample, said sample could be considered mislabeled.

Another influential approach was introduced in [Stassopoulou and Dikaiakos, 2009], and that was the use of a Bayesian Network. This approach was also used with properties like the ones in Table 2.3. As the data set used at the time was deemed imbalanced, the model had a risk of growing a bias for the majority class. To fix this, the researchers employed several re-sampling techniques with varying degrees of success. The most successful strategy was to over sample the minority class to 50% of the data set volume in the training data. This achieved an F1-score of 0.903 on the testing data. The testing data was not submitted to sampling techniques, to preserve the natural imbalance. The testing data was manually labeled, as was the training data, and considered 685 human sessions and 99 crawler ones. The performance metric, however, was based on the initial manual labeling of sessions as human or bot made, and so may suffer from human error. Further, this project disregarded session with less than 5 requests which compromised its ability to deal with distributed attacks.

Table 2.3: Common Navigational Features

Id	Name	Description
0	robots.txt	A check to see if "robots.txt" was accessed
1	Pages Requested	Number of pages requested
2	Click Rate	Number of Requests in Session
3	Unassigned Referrer	% or number of requests with and unassigned referrer field
4	Total Time	Approximate, total session time
5	Request Type ratios	HTML to: image ratio, PDF files, etc.
6	Request Method ratios	% GETs, % HEADs etc.
7	Response Errors	Ratios or number of responses with a code ≥ 400
8	Page Depth	Page Depth in URL Space
9	Time per Page	Can be mean or associated to a certain page
10	Consecutive Sequential HTTP Requests	Sequential requests for pages with a shared path and in the same session
11	Data Volume	Size in Bytes/KBytes of data moved

Building upon the Bayesian Approach in [Suchacka and Sobkow, 2015], the authors augmented the usual feature set with domain specific attributes. As their data set was collected from an online bookstore, the researchers were able to extract more information on their users using contextually valid measurements, for example: if a purchase was made, number of pages traversed related to shipping information, number of attempts to register, etc. While these make for good predictors in the context of an online store, it is apparent that they can't be used in a general solution for all types of websites.

In [Stevanovic et al., 2012], the authors compare several SL techniques using a set with variations of attributes [0, 1, 3, 5, 6, 7] from the Table 2.3. Furthermore, they introduced the basis for the entries numbered 8 and 10. Their work relied on the WEKA software package implementation of the following algorithms: C4.5, RIPPER, Naive Bayes, Bayesian Network, K-nearest Neighbours, LibSVM and a Multilayer Perceptron Neural Network. In two experiments, first they used only "borrowed" features in their models, and then compared them to models trained with their added features. The results showed that their ideas improved classification outcomes and that the best overall model was the Neural Network. The authors did express that they found room for improvement in the C4.5 and RIPPER algorithms.

[Sisodia et al., 2015] compared Random Forests, C4.5 and Naive Bayes classifiers, using traditional navigation characteristics, to their ensemble following techniques of Bagging, Boosting and Voting. Though slightly better, the difference in performance between the ensembles and the Random Forest and C4.5 algorithms was not all that noticeable. Compared to a Naive Bayes approach however, the dissimilarity is more pronounced. In the original paper, the authors plot the precision, recall and f1-score for all models. In the mentioned plot, these variables hold the same

values for all models except for the Naive Bayes Classifier. In this particular case, all metrics drop.

Diverging from the common characteristics, [Hayati et al., 2010] instead focused on the set of web pages visited in a session to determine whether or not the traversal had been performed by a human. Using a data set that had twice the amount of bot records than human records, and the Support Vector Machine algorithm, the model reportedly reached a 96% accuracy. Using a metric of binary classification, not used by any other paper in this review, achieve an MCC of 81%. This metric varies from 1 to -1, a perfect classifier would be a 1, a random a 0 and an inverted one a -1.

The work up to this point was based on offline log analysis. [Cabri et al., 2018] introduced a model that consumes requests as they come in to determine if a session is controlled by a human or bot, before the session concludes. This method first and by looking at some major navigation characteristics determines if a single request is bot or man-made using a neural network. This request is added to the current set of requests being analyzed for a user. Then the current set of requests as a whole is passed through Wald's Sequential Probability Ratio Test to check if the set itself and the associated probabilities of each request can give a conclusion on the identity of the user. The most interesting result from this approach was the verification that human users were rarely confused with bot users and so their experience wasn't harmed by this system.

2.2.1.2 Unsupervised Learning Approaches

As SL techniques depend on the labeling of data by humans, the models will learn based on the human expertise behind the labeling which can lead to misclassification. To deal with this issue, a number of Unsupervised Learning (UL) and Optimization techniques, were also employed in Web Bot Detection.

In [Stevanovic et al., 2011] and [Stevanovic et al., 2013], the authors test two UL neural networks for web log analysis. Namely, the Self-Organizing Map (SOM) and a modified Adaptive Resonance Theory 2 (ART2) network. The second work follows closely the first and expands upon it refining the attributes selected. In both papers, the authors apply these algorithms to two subsets of the common features in Table 2.3 and analyze their results. To verify the results (not to train the models), the data set fed to the neural networks was preprocessed and each session was labeled as: normal browser, well-behaved crawler, malicious crawler or unknown visitor. Taking this into consideration, and looking at the results in the SOM mappings, particularly in the second paper, the 4 labels form 4 mostly distinguishable clusters. There are instances however, where sessions labeled as human appear in the same region on the map as the cluster of malicious crawlers. The researchers suggest these may be more stealthy bots that evaded the first labeling. Further, when originally labeled malicious crawlers are mapped to mainly human clusters the researches take issue. These were well crafted bots that had their creators just masked their known UA could've been mistaken for human browsers, which would defeat the detection scheme. As for unknown users, while in the first paper these were harder to assign to one specific cluster, in the second one they are more closely related to the human cluster. The results from the mART2 network help strengthen the conclusions from the SOM experiments, but by themselves are not as informative.

There are multiple overlaps in clusters from which little can be learnt if not backed by more concrete discrimination.

Taking into consideration most common navigation features and introducing a few novel ones, the SMART system presented in [Zabihimayvan et al., 2017] calculates the features that best characterize a domains traffic by applying Fuzzy Rough Set Theory. With the features selected the Markov Clustering algorithm is applied. The Clustering results are compared to a system modeled after [Stevanovic et al., 2013] and another DBSCAN model. The difference in performance between the SOM and mART system and SMART is not too noticeable, however SMART has a slight edge. As these Bot detection models were based on clustering, the metrics used to compare them stem from that field. In the paper they refer to Accuracy as Rand Index, and it is this that is used to compare algorithms. In regards to DBSCAN both models outperform this one.

A paper on Web Application Intrusion Detection [Zolotukhin et al., 2014] uses and compares several UL outlier detection techniques to process server log data. As the data set contained no harmful users, the models were able to learn the expected normal behaviour, and any divergence can trigger an alarm in the system. The models are trained to learn normal request parameters, query strings (alphanumeric characters masked to be indistinguishable), and user-agents. After training, the models classified attack data created by the authors to simulate anomalies in the three categories. The models trained were: SVDD, K-means, DBSCAN, SOM and LOF. In web resource anomalies all models had accuracy over 98%. In attribute values, the accuracy ranged from 95% (LOF) to 100% K-means. Finally, in user-agents DBSCAN and LOF stand out with an accuracy of 97.5%, followed by SVDD and SOM with accuracies of 90 and 87.5% and finally K-means performed the worst with 77.5% accuracy. Though not with the specific purpose of bot detection, this paper and its results show how outlier detection techniques can be used to mine data from server access logs to detect inadequate behaviour from users. It is also worth noting, the system presented is meant to run online and could detect a series of crafted attacks in minutes.

A generalized web bot detection system, with features specific for detection of distributed attacks was proposed in [Jacob et al., 2012]. This system was made of three primary components. The simplest, was a set of heuristics matched against access logs to look for hints of bot behaviour. The next step was an ensemble of supervised learning models trained to identify unusual traffic patterns. Finally, a clustering model was used to check for synchronicity and possible distributed attacks.

2.2.2 Biometric Patterns

The first approach to bot detection that had some relation to biometrics was proposed as early as 2006. Though now easily evaded, the system at the time proved to be effective against the bot technologies of its time. [Park et al., 2006] This system used the presence of mouse clicks and key presses to assert the identity of a user as human. As this was dependent on JavaScript and users could be disinclined to run such technologies, a blank CSS file was added to the pages under the assumption that bots would not request it as they had no need for it. This system has long been

surpassed by current browser automation technologies but it is worth mentioning as it was the start of the study of biometric for web user classification.

A study focused on blocking spam bots in blogs developed a model based on more complex biometric features in combination with learning algorithms. Before discussing the actual solution, another interesting contribution of this paper is the classification for bots based on apparent behaviour. The same classification that is used in this paper. Using data sets containing hours of human, mimic and replay bot interaction, a C4.5 decision tree was trained to classify users. The results show that this tactic was fairly accurate even at detecting replay bots. To prove this, the authors collected 239 hours of user data. 207 belonging to over 1000 human users and the remaining to a human mimic and a replay bot. On this data their model achieved an accuracy of around 99%. [Chu et al., 2013]

In [Rahman and Tomar, 2020a], hierarchical clustering was used to derive representative points from an initial data set. Though it is unspecified how the initial labels are assigned, supervised learning algorithms are then trained and tested on a data set containing bots programmed by the researchers. It is worth noting however, the biostatistics approach, combining biometric data from mouse and keyboard movement, with statistics such as session length and inter-request time. Results appear promising, reporting accuracies and f1-scores of over 90%, however, the lack of an explanation for the labeling process of the training data raises concern.

[Iliou et al., 2021] used convolutional neural networks to determine if the trace of a mouse's movement on the screen belonged to a human or bot. In addition, common navigational features and biostatistical features were used by SL classifiers. The data sets for training and testing the model were composed of bots programmed by the researchers and the authors and 28 others providing the human data. Despite the small sample, the levels of precision, recall and f1-scores are in the 93-99% range.

A framework for detecting bot behaviour by a multi-stage analysis is proposed in [Rahman and Tomar, 2020b]. Analyzing timing parameters related to requests, mouse movement and keyboard strikes, and error patterns. The last category being the most novel, works under the assumption humans make mistakes on a keyboard and so will press keys like delete and backspace. Using two data sets one of humans and another one of malicious bots, regression algorithms were run and differences in bot and human behaviour analysed. The results do not evaluate a particular bot detection practice, rather differences in the attributes measured.

2.2.3 Browser Fingerprinting

To understand how Browser Fingerprinting works, first it is necessary to define what is a User-Agent, as it is a principal concept of this methodology. The UA is a string field in HTTP requests usually containing information on the application, system and version of the user that sent the request. Browsers, as interfaces to the Web, have their own User-Agents [MDN, 2022]. When a user claims to be using a certain public browser, some supposed characteristics of this browser are known. Others can be learnt after analyzing enough requests with the same UA. Browser automation tools, widely used in Bot Development, as the name implies, use a browser to automate

a task. Often times, however, either by default or for some reason in the interest of developing the bot, these users will have attributes not consistent with their public UA. Bellow an example UA:

Mozilla/5.0 (Windows NT x.y; rv:10.0) Gecko/20100101 Firefox/10.0

As can be read in the UA, the system it's advertising is Windows. This could be a legitimate user. On the other hand, it could also be a bot, running for example on Linux, that chose to hide its real UA as Windows users are more common.

The practice of browser fingerprinting, often found in commercial bot detection solutions, looks for this mismatch in what is expected from a UA versus what actually is. The idea is to take in information that can betray the browser's identity. Certain functionalities of different browser automation tools are checked, information on hardware and OS is collected and much more. Algorithms trained to look for inconsistencies in fingerprints can be used to classify new fingerprints and by saving the results, a repeated fingerprint is automatically classified. This efficiency is in part what makes it attractive for commercial use. However, as two recent papers conclude, it is fairly easy to hide giveaways and inconsistencies [[Amin Azad et al., 2020](#)] [[Vastel et al., 2020](#)].

2.2.4 Other Approaches

For the domain of content-rich websites the semantic approach presented in [[Lagopoulos et al., 2018](#)] shows promise. The system uses topic mapping to assign different topics to the content pages of each website. The topics and the sequence they are traversed in along with common navigational features are used as feature vectors for the classifiers. Several configurations of ensemble classifiers are used and results seem appropriate. With f1-scores varying from 0.85 to 0.95. However, this technique relies on the domain being content-rich and as such could suffer in a generalized setting.

The following strategies could be considered Turing Test Models as they either lure bots into revealing themselves through creative HTML rendering or actively challenge users to prove their "humanity".

NOMAD, presented in [[Vikram et al., 2013](#)], introduced the concept of moving target defence against web bots. Moving target defense is the idea that defense systems should be complemented by a component that changes the appearance of the target to protect. The idea was to block bots that look for hints on targets through HTML tag ids and names. The authors created a symmetric key encryption system - a middle-ware component that encrypted property names and ids in transit to the client and then decrypted them before forwarding them to the server to handle the response. This added to fake invisible elements, diluting the pool from which the bot had to pick, and made for a harder traversal of the website for automated agents, while not bothering human users.

[[McKenna, 2016](#)] tried to use honeypots for bot detection. By creating appealing components, the authors expected to catch most crawlers that follow links. However the challenge of making these appealing to sophisticated bots and the narrow applicability of them make for reasons not to pursue this area.

No review on web bot detection techniques would be complete without mentioning the infamous CAPTCHA. Several variations of this challenge have appeared over the years. The two main issues with this technique are usability, to the extent where it can drive users away and even be unfair to the visually and hearing impaired, and the fact that most schemes in use have been broken [Banday and Shah, 2011].

Having reviewed possible solutions to the Bot Detection problem, and considering the nature of a possible data-set, the Unsupervised Outlier Detection field was selected as the approach for this project. As the data was to be collected from the company's public website there was no guarantee it would be free of Bot Sessions. Furthermore, as the features selected for characterization include some novel features and no expert was available, manual labelling was impossible. This topic will have its own review in Chapter 3

2.2.5 Summary

This section will present a table comparing the different learning approaches analysed up to this point. This table will describe the data used to train each model, the tests performed and the results obtained.

Table 2.4: Summary of Learning Approaches

Paper	Description	Training Data	Test Data	Results
[Tan and Kumar, 2002]	SL, C4.5, Nav Feats & Server Access Logs	Not available	1.6 M entries, estim. 180k sessions	90% ACC, 90% PRE, 82% REC
[Stassopoulou and Dikaikos, 2009]	SL, Bayesian NW, Access Logs	12k sessions, 10% crawlers, oversampled to 50%	685 Human sessions, 99 crawlers	85-90% ACC, 80-93% PRE, 80-95% REC, 85-90% F1
[Suchacka and Sobkow, 2015]	SL, Bayesian NW, Access Logs, Nav + Domain Specific Feats.	12.5k Sessions 25-33% crawlers	2, 13k Session Sets same proportion of bots	90% ACC, 6% Error Rate (Missed Bots)
[Stevanovic et al., 2012]	SL, 7 models: C4.5, RIPPER, Naive Bayes, Bayesian NWs, kNNs, LibSVM and NNs.	96k (human + good bot) sessions vs. 2.5 bad bot sessions + unsepc. oversampling of minority class (bad bots)	37k humans and good bots vs 925 bad bots	Best model: NN, 99% ACC, 80% PRE, 80% REC, 99% F1

Continuation of Table 2.4				
Paper	Description	Training Data	Test Data	Results
[Sisodia et al., 2015]	SL, Simple models of C4.5, Random Forests and Naive Bayes vs. Bagging and Boosting (C4.5) and Voting Ensemble (all 3)	5 Datasets: 10-30k humans and 10% of that in bots, 1-3k bots	Same logic as training data	Naive Bayes: $\approx 20\%$ for all metrics. Other models: 80% PRE, 60% F1, 40% REC
[Hayati et al., 2010]	SL: SVM, pages as actions. 34 actions. Total Data: 11k bot records, 5.5k human records, 4.2k sessions	2/3 of data set	1/3	95.5% ACC, 81% MCC
[Cabri et al., 2018]	SL: Deep NNs, Nav. feats., HTTP Request, WSPRT sequence of requests.	N/a	13k sessions, 6k bots, 7k humans	96% ACC, 98% PRE, 94% REC, 96% F1
[Stevanovic et al., 2013]	UL, SOM and mART2. Clustered sessions and used the testing data to id the sessions and label in each cluster.	53k humans, 7.6k good bots, 287 bad bots and 4042 unknown.	N/a	Clustering results not applicable
[Zabihimayvan et al., 2017]	UL, clustering, Nav. feats. HTTP requests, SMART, DBSCAN, and SOM and mART2 from [Stevanovic et al., 2013]	N/a	D1: 2.5k sessions 139 good bots 241 bad bots 2k humans D2: 2.6k sessions 916 good bots 680 bad bots 1k humans	DBSCAN D2: 52% ACC, otherwise: 86-90% ACC (as RI).

Continuation of Table 2.4				
Paper	Description	Training Data	Test Data	Results
[Zolotukhin et al., 2014]	UL Anomaly Detection, intrusion detection (bots as intruders)	N/a	5 attacking sessions	LOF, 95% ACC Other Models: 98-99% ACC All bots and attacks were eventually detected.
[Jacob et al., 2012]	Naive Bayes, SVM + Association Rules, Nav. feats, Time-series clustering, distributed attacks	73 million requests, 813 IPs	62 million requests, 763 IPs	Bot Detection: ACC on Bots: 95-99% ACC on Humans: 78-83% Detection of Distrib. Attacks: 94% ACC, 99% PRE, 85% REC
[Park et al., 2006]	SL ensemble learning of nav features + heuristic detection of human proof, detecting mouse and keyboard action, honeypots. Initial Data: 43k human, 125k bot sessions	50% data	50% data	91-95% ACC
[Chu et al., 2013]	SL C4.5 decision trees, biometric data as actions	Human data: 1078 signed in users at a blog, multiple 2 hr periods, 1 day Mimic Bots: 30 hrs Replay Bots: 2 hrs	4.5 million events, 200k actions, 207 hrs of humans and 32 of Bots	99% ACC

Continuation of Table 2.4				
Paper	Description	Training Data	Test Data	Results
[Rahman and Tomar, 2020a]	Clustering, Data Reduction, Naive Bayes, biometric features. D1: 16k human , 1.8k bot requests; D2: 9k humans, 11k bots	1/10 of data	9/10 of data	91% ACC, 86% PRE, 83% REC, 84% F1, 78% MCC
[Iliou et al., 2021]	Ensemble of CNNs, mouse trace, SL models and ensembles, Nav. feats.	35 human 35 moderate bot 35 advanced bot sessions	15 human 15 moderate bot 15 advanced bot sessions	97% ACC

Chapter 3

Outlier Detection

As the proposed solution contemplates an Outlier Detection (OD) model for detecting bots, this chapter will be dedicated to this field of study. First, a general review covering basic concepts, some recent surveys and relevant algorithms will be presented. Then, the field of Data Processing will be briefly discussed. This field though not unique to OD, is necessary for many applications of Machine Learning, OD included. Finally, the used performance metrics will be covered. As this project and the models included treat OD as a Binary Classification problem, there is some overlap in the metrics used.

3.1 Overview

An outlier is a point in a set that differs significantly, by some measure, from the remaining. The term is often used interchangeably with anomaly. A more precise definition, however, considers that outliers are expected to be present to some extent in real and training data, whereas anomalies are points that do not occur in the training data and appear completely new in real data. OD learning models can be either SL, UL or semi-supervised. However, the need for labelling in a field where one would want to find outliers not seen before often leads practitioners to focus on UL based models. For this project, the same rationale was adopted, and as such this section will focus on Unsupervised Outlier Detection (OD) [[Boukerche et al., 2020](#)].

Outliers are often categorized into point, local or collective outliers. Data points that are anomalous compared to the entirety of the data set are point-based outliers. Local outliers are points that deviate from their nearest neighbours. Sometimes, instead of local outliers, the related concept of contextual outliers is introduced. Context-based outliers are points that in a different environment (a variable set/unset, given time-frame, etc.) could be considered inliers. Finally, collective outliers are sets of points, where individual points may not be considered anomalous but their set is [[Alla and Adari, 2019](#)].

The original motive for OD was data cleansing. Deleting rare values so statistical models could better fit data. Currently, however, the focus of this field is on the rare values as they can provide interesting and unexpected information. With this target in mind, OD models have been applied to

fraud and intrusion detection systems, defect detection and many other fields. This project hopes to aim OD at biometric descriptions of a user's behaviour on a website. By analyzing how the user interacts with the computer's peripherals (pointing devices and keyboard), assuming the majority of the users of a website are human, bots may be found in the outlier pool. In [Boukerche et al., 2020], the authors categorize UL methods of OD into Proximity or Projection-based approaches.

- Proximity based approaches are those that use some type of measure of distance between a point and its closest peers. If a point has under a set number of neighbours in a given distance, it may be classified as an outlier. Examples of proximity based approaches are LOF and Nearest Neighbours.
- Projection-based approaches are ones that convert data to an almost equivalent, if less complex and complete, space. Deep Auto-encoders and Isolation Forests are considered projection-based.

The survey in [Domingues et al., 2017] submitted popular UL OD models (including LOF, SOM and Isolation Forest) to test them in terms of performance, time and space consumption and scalability. In total, 14 algorithms were tested, on 12 publicly available data sets and 3 proprietary ones. These data sets differed in size from a few hundred, to twenty thousand entries, and from 6 to 107 features. Model performance is compared using the area under the ROC and precision-recall curves. The overall best performer in most categories was the Isolation Forest Algorithm. The worst performers identified were: LOF, SOM and ABOD (angle-based OD).

3.1.1 Local Outlier Factor

This historical algorithm has seen multiple improvements and inspired many variations across the years, such as COF and LoOP [Boukerche et al., 2020]. Though more recent solutions have perform better in regards to both temporal and spatial complexity and even accuracy, it is still worth reviewing a paper that introduced the concepts of an Outlier Score and local outliers.

In the original paper, [Breunig et al., 2000], the authors list the approaches for OD at the time and argue that all have their issues that LOF wants to improve upon. The authors start by dissecting the Distribution-based solutions. These attempted to fit a standard distribution to the data. This strategy had two major drawbacks as most distributions were univariate (considering only one variable) and required a priori knowledge of the data distribution, which for real-world applications was often unfeasible. Following that, the early depth-based approaches are criticized, as they struggled heavily in large, multidimensional data sets (starting with 4 dimensions). Clustering approaches, the authors argued, focused on clustering so were not optimized for OD, and lacked a way to calculate an Outlier Score. Finally, the contemporary distance-based approaches were shown to fail on examples of data sets with clusters of different density.

The algorithm only takes one parameter k , the minimum number of points necessary to be present in a neighborhood. Following that, points are classified as outliers based on how close it is to its k -neighbors, and how dense its neighbours are in their own neighborhoods.

The k -distance of an object o is defined as the distance between o and another object p such that:

For at least k objects q , other than p and o it's true that:

$$d(o, q) \leq d(o, p) \quad (3.1)$$

For at most $k - 1$ objects q , other than p and o it's true that:

$$d(o, q) < d(o, p) \quad (3.2)$$

The k -distance neighborhood of an object contains all the objects that are only as far from the object as the k -distance.

The reachability distance of an object o in regards to another p is defined in Equation 3.3:

$$reachability = \max(k_distance(o), d(p, o)) \quad (3.3)$$

With that, the *localreachabilitydensity* (lrd) of a point can be calculated, which can essentially measure a points neighborhood density. Finally, the local outlier score of a point p is calculated with regards to its o -nearest neighbours. The higher the *LOFscore* the more likely a point is an outlier. This score is proportional to the sum of the lrd of the point p over the lrd of all its O nearest neighbours. Considering NN_p the set of the o -nearest neighbours to p .

$$LOF(p) \propto \sum_{o \in NN_p} \frac{lrd(p)}{lrd(o)} \quad (3.4)$$

As can be seen from Equation 3.4, as the neighbors neighborhood density increases, so too must the point's neighborhood's or the *LOFscore* will increase. It's from this concept of *LOFscore* that originates the idea for Outlier Scores. Furthermore, by comparing each point to its neighbors, the concept of Local Outlier was born.

Tests at the time showed that the relationship between the number picked k and performance was not entirely straightforward. Still the authors proposed heuristics to help selecting an adequate k value [Breunig et al., 2000].

3.1.2 Isolation Forest

Introduced in 2008, yet still comparable if not better than more recent models [Domingues et al., 2017], the Isolation Forest Algorithm was the first to use the concept of isolation for OD. To isolate an object in a data set, is to partition the data so that one partition only contains the point we want to isolate. To partition the data the algorithm randomly picks an attribute to base the partition on, then randomly selects an existing value between that attribute's maximum and minimum. Finally,

the data is divided between points with that value being bigger than the selected and points with that attribute being smaller. The idea behind using isolation is that, as inliers are closer to each other, if random partitions are used, more cuts will be needed to isolate an inlier versus an outlier. This idea is graphically demonstrated in the original paper [Liu et al., 2009] and in Figure 3.1.

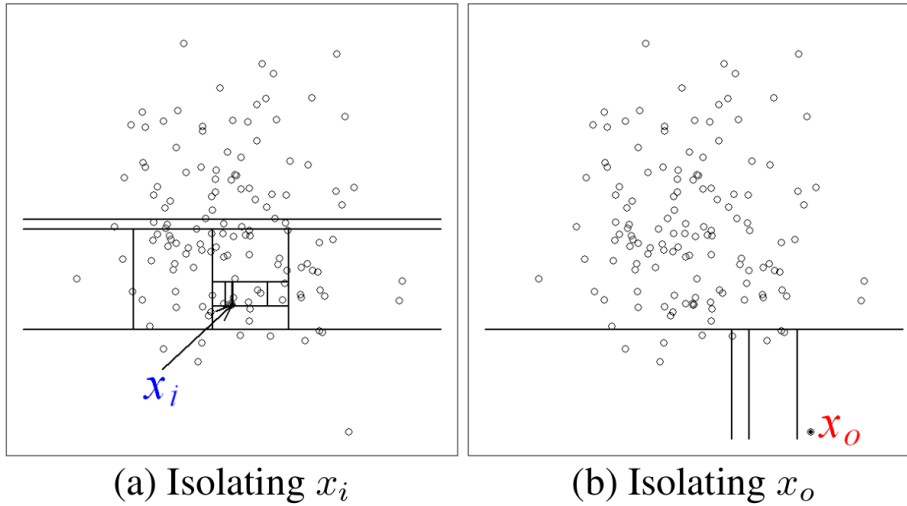


Figure 3.1: Isolating an inlier (a/ x_i) vs. Isolating an outlier (b/ x_o) [Liu et al., 2009]

The partitions of a data set over one feature can be represented by a Binary Tree Structure. The number of partitions to isolate a point, will be roughly equivalent to the path length to reach that point's node. Building a tree to partition the data over each feature (and possibly combinations of features), one gets an Isolation Forest. Calculating the average path across all trees the expected path length to a point is achieved. If the final path is smaller than the expected, the point can be considered an outlier. Its Outlier Score will be the path size [Liu et al., 2009].

This model was demonstrated to work well in high-dimensional spaces and in problems with a substantial amount of irrelevant attributes. Furthermore, and contrary to most methods of OD where the more data the better, iForest works best with relatively small sample sizes, as large sample sizes reduce the model's ability to isolate outliers. There are two parameters to tune for this model. First is the sub-sampling size, that controls how many instances will be used to build the trees, and therefore, the maximum tree depth. Secondly, one must consider the number of trees to be built. If the number of trees is greater than the number of features, combinations of features can be used [Liu et al., 2009].

3.1.3 Autoencoder

An area of Machine Learning that has seen ever increased interest from the community is Deep Learning. As such, an algorithm from this class was chosen to be evaluated in this project. Deep Learning Neural Networks (NNs) are those that have more than one hidden processing layer. Autoencoders are a type of Deep Learning Neural Networks focused on Dimensionality Reduction.

Before diving into what is an Autoencoder, it is first mandatory to understand the underlying technologies.

Previous to Autoencoders, Dimensionality Reduction had already been used for OD. (e.g. Principal Component Analysis). The general idea behind OD through Dimensionality reduction is if data attributes are correlated, which they usually are for real-world applications, we can predict some values based on others. With that, we can create a compact representation of the parameters that can then be used to reconstruct the original object. Once reconstructed, the resulting object can be compared to the original and the reconstruction error calculated. As models train to reconstruct inliers and how their values relate, outliers will be reconstructed in a "lossier" manner, the higher the reconstruction error, the more likely something is an Outlier[Aggarwal, 2016].

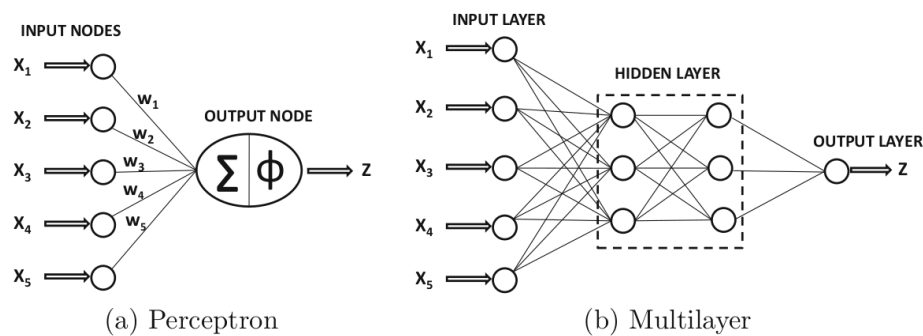


Figure 3.2: Examples of single (a) and multi-layer (b) ANNs [Aggarwal, 2016]

As can be seen in Figure 3.2, NNs are graph-based machine learning data structures. The name derives from the inspiration it takes from the model of human neurons and how information courses through the brain. To our neurons, information is picked up on the dendrites - the entry point to the cell. Following that, and under certain conditions, a neuron can "fire" a synapse and output a signal to the dendrites of the next neuron. An Artificial NN is a layered graph, consisting of at least one input and one output layer and zero (perceptron) or more hidden (multilayer), intermediate layers. Nodes in the graph are called neurons. Neurons on the input layer receive raw data, usually a feature of the training data [Aggarwal, 2016]. As data enters the network, when passing from one layer to the other, nodes are connected through weighted edges that alter the data. When the weighted data reaches a neuron, an activation function further manipulates the value before passing the result along through another weighted edge to the following layer. All nodes in the hidden layer use the same activation function. The final layer is the output layer, uses its own activation function that depends on the problem to solve. As input data is processed to output, the output is tested for accuracy, and through a process called backpropagation. The output will be passed down each layer in reverse to recalculate weights at each node. A cost function is used to estimate how the training is progressing by calculating how off a result was [Alla and Adari, 2019]. NNs can be interpreted as function approximators that learn the weights to turn an input to an output through the various layers. If a non-linear hidden activation function is used the model can train to reproduce a non-linear function. [Aggarwal, 2016]

Autoencoder networks can be seen as two symmetrical, connected NNs. The idea is that, one half learns a dimension reduction function while the other learns how to expand the reduced dimension. As data flows inward, first the number of nodes per layer decreases, forcing data compression. At the middle section of the network, the deepest compression is attained and the number of nodes per layer starts to increase so that the data is reconstructed. Comparing the decoded version to the input the reconstruction error can be estimated. As was the case for general dimensional reduction for OD, the reconstruction error here can also be used as a measure of Outlier Score [Alla and Adari, 2019]. See Figure 3.3 for a graphical representations of an autoencoder.

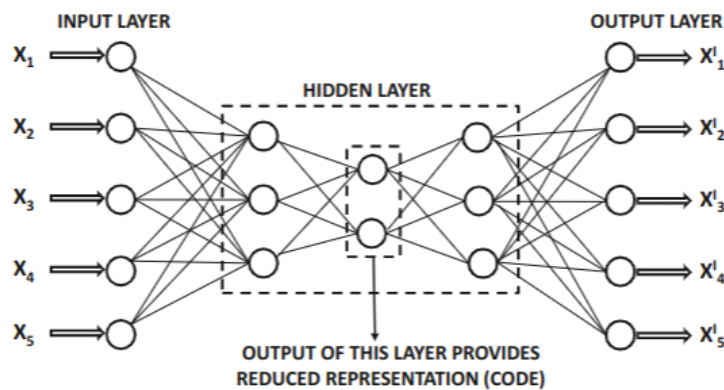


Figure 3.3: Example of autoencoder architecture [Aggarwal, 2016]

3.2 Data Preprocessing

Data preprocessing is a field that aims to improve results in AI models by refining the data, as these benefit from better quality data. This quality increase is achieved by encoding, transforming, and manipulating information in 4 different areas [Baheti, 2021].

Data Cleaning focuses on fixing missing values and squashing noise/removing outliers. In real world data, attribute values are often lost due to unforeseeable circumstances. When simply discarding those values is not an option, techniques have been developed to approximate the lost value. For some applications, though not OD, noise and outliers are simply not beneficial for the models. For that reason, noise and outlier removal are sometimes employed.

Data Integration is necessary when there can be multiple sources for data in different formats.[Baheti, 2021]

Data Transformation prepares data to be consumed by the model to be trained. A critical step of this process is Feature Selection. In this process, compound attributes can be created from existing ones, and irrelevant attributes are dropped. Most models deal only with numerical values, as such, when categorical variables are used, these need to be encoded. Another usage for Encoding variables is when it is required to discretize continuous variables[GeeksForGeeks, 2021a]. Additionally, AI models are usually expected to predict a value based on inputs. However,

if these inputs are not in the same order of magnitude, larger values may be considered more important[[Scikit-Learn, 2022b](#)]. To solve this issue, features can be scaled to better represent their variation over their absolute value.

When the volume of data is too large, **Data Reduction** techniques may be needed [[Baheti, 2021](#)].

3.3 Performance Evaluation

Most Learning Systems for Bot Detection, as those discussed in Chapter 2, identify users as belonging to one of two classes: Bot or Human. This falls into the category of a Binary Classification, as such the metrics used in accessing the performance in problems of that broader category are used. Furthermore, Outlier Detection models also tend to classify inputs as either in-lying or outlying. Though some give an outlier score between zero and one, a threshold for that can be set so that a continuous score is discretized. Doing so, allows for the evaluation of OD systems using Binary Classification metrics. In this work, OD was used for the purpose of Web Bot Detection under the assumption that outliers were to be considered Bots, and inliers, Humans. As such, the results in Chapter ?? will be interpreted using Binary Classification metrics, which will be explained in this section.

The standard in Binary Classification is to label one class as a Positive and the other a Negative. When testing a classifier's performance, 4 variables will be used to derive all metrics. First, the numbers of predicted positives and negatives are taken in consideration: "how many tested points fall into each category?". Then these are compared to the correctly and incorrectly predicted classes, to calculate the initial variables: True Positives (TP), True Negatives (TN), False Positives (FP), False Negatives (FN).

Often times, the previous values will be displayed in a graphic format dubbed Confusion Matrix. An example of this structure can be seen in Figure 3.5. On the horizontal axis one can see the result of the classification process in the number of predicted labels. On the vertical axis the actual label of the point is referenced. As such, the first quadrant on the top left corner corresponds to TNs. The second quadrant on the top right corner to FNs. Below that the TPs, and finally, on the bottom left quadrant the FPs. Furthermore, the darker the color of the quadrant the more points in it [[Scikit-Learn, 2022a](#)].

The first metric to be derived from the Confusion Matrix is Accuracy. Measures how many predictions were correct. Whether True Positives or True Negatives over the total number of points [[Alla and Adari, 2019](#)].

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (3.5)$$

In [[Stassopoulou and Dikaiakos, 2009](#)], the authors give this illustrative example on the need for other metrics other than accuracy: "assume a dataset with 100 cases out of which 90 cases

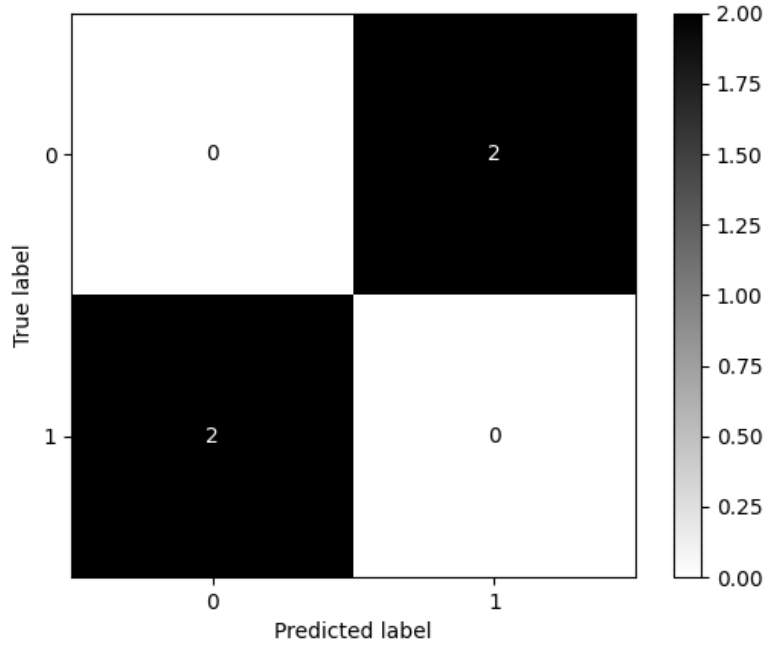


Figure 3.4: Confusion Matrix example

belong to the majority class and 10 cases belong to the minority class. Then a classifier that classifies every case as a majority class will have 90% accuracy, even though it failed to detect every single target of the minority class." Furthermore, for the purpose of OD wherein identifying outliers, usually the positive class, is of particular interest these other metrics are also used. Precision evaluates how many predicted true positives are actual positives. Recall measures how many of the total positives were identified. Finally, the F1-score combines Precision and Recall into a metric that highlights identification of the positive class and penalizes miss-classifications of any type [Alla and Adari, 2019].

$$Precision = \frac{TP}{TP + FP} \quad (3.6)$$

$$Recall = \frac{TP}{TP + FN} \quad (3.7)$$

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2 * TP + FP + FN} \quad (3.8)$$

While Precision and Recall can be metrics by which to evaluate a model, they can be defined in relation to another variable. OD models most often output outlier scores, of those scores only a threshold of them will be classified as actual outliers. This can be tweaked, setting it too low will cause outliers to go undetected. On the other hand, a high contamination value, as this threshold is usually defined by estimated volume of outliers (contamination) in the training data, may lead to

high rates of false positives. This trade-off can be studied by plotting a precision-recall curve by varying the threshold value. As precision and recall are connected through this threshold variable, and since the variation of this value is not necessarily monotonic it is worth exploring these curves. From there we can extract how each algorithm behaves at a given threshold and get more insight on their performance [Aggarwal, 2016].

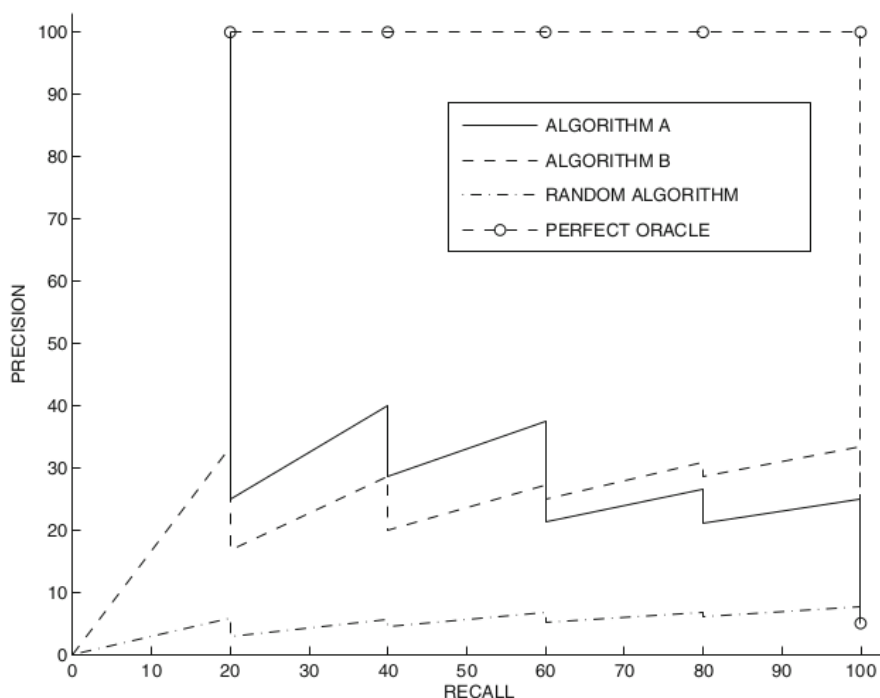


Figure 3.5: Precision-Recall Curve Example Comparing Multiple Algorithms [Aggarwal, 2016])

Now reconsidering recall as a performance variable. Recall can also be referred to as True Positive Rate (TPR). If compared to the False Positive Rate the results can also be interpreted as a measure of performance.

$$FPR = \frac{FP}{FP + TN} \quad (3.9)$$

As its name implies, FPR is the rate at which points that should be classified as negative, are classified as positive. In the context of outlier detection, what percentage of inliers are being considered outliers. Plotting how one rate impacts the other one gets a Receiver Operating Characteristic Curve (ROC). As in OD, identifying the positives is what is most desired, this graph displays how by making the model more sensitive, and thus incurring in a higher FPR, can also raise the TPR. However, for the purpose of OD it is also not ideal to have a high FPR as it would pollute the outlier pool with inlying points [Alla and Adari, 2019]. The area under the ROC (AUROC) can be interpreted as the probability of a randomly picked outlier or of a randomly picked inlier to be correctly classified [Aggarwal, 2016]. In that sense, a perfect classifier would have a fixed TPR of 1 and regardless of FPR this would remain, and so the AUROC would be equal to

1. In the case of a Binary Classification, where the system randomly picks the category, the TPR would rise with the FPR, resulting in an AUROC of 0.5. In the case of OD, if the AUROC is under 0.5, it is assumed that the system is learning, just the labels are switched. From 0.5 to 1, the higher the AUROC the better, as it reflects how correct the models predictions are. Authors advise, however, to be careful of an abnormally high AUROC (over 0.99) as it can be a sign of overfitting. When models overfit to training data they may have hindered performance against new, unseen data [[Alla and Adari, 2019](#)].

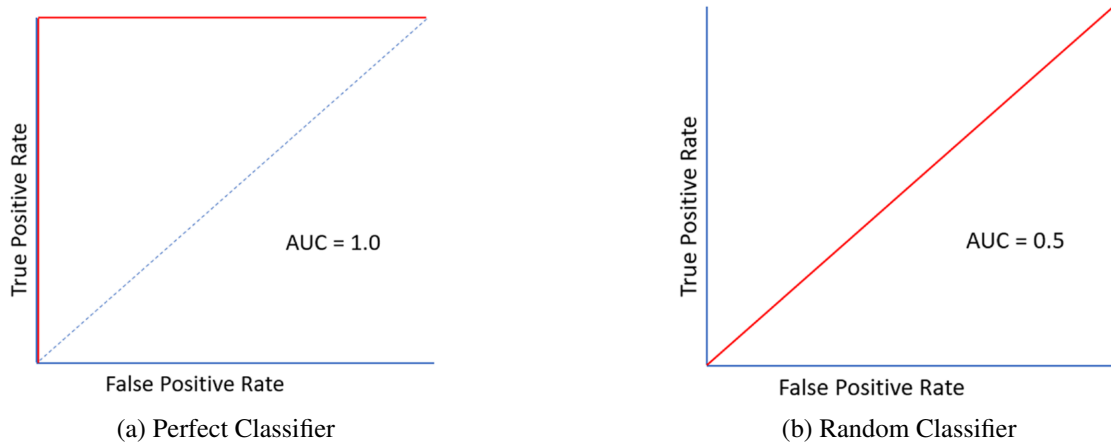


Figure 3.6: Example of Perfect Classifier and Random Classifier ROC [[Alla and Adari, 2019](#)]

Chapter 4

Detector Implementation

This chapter will cover all aspects of the Solution Implementation. From data gathering and processing, to the classifiers trained. It will conclude with a walk-through of the end system.

4.1 Detection Methodology

The proposed solution will use unsupervised Outlier Detection on a set of biometric features to identify outliers as bots. The usage of an unsupervised approach stems from the characteristics of available data. Data was collected through a public access website, for that reason contamination (by bots) is expected. The usage of biometrics forces bots to invest considerable time into mimicking human interactions. If bots are effectively forced into becoming as slow as humans then a key advantage in their usage is removed. Furthermore, if at a point in time all users behave following the same biometric patterns, then detection won't have to focus on the bot aspect but rather on the actions automated by the bots. User access control can focus on specific interactions to ban, instead of focusing on the nature of the user. User actions can, for example, be checked to see if they follow a pattern of vulnerability scanning.

The detection process starts with a script embedded in a website. This script captures biostatistical data, mainly in regards to movements performed on the human interface devices (mouse, keyboard, touch, pointer and screen). Now as this script will reload at every page, the session must be rebuilt from the event logs. That is handled by the Session Aggregator Module. After that, the data preprocessing module fixes missing values and, if need be, scales the data. Finally, the OD classifier is trained to determine if a session and its recent interactions are inliers (humans) or outliers (robots).

4.2 Data Extraction

This section will demonstrate the multiple steps in the process of gathering data to train the Detection Models. The first section will outline some requirements of the data collecting tool. Following

that, the data itself will be described. Finally, the systems used to gather verified human and bot data will be described in their respective sections.

4.2.1 Requirements

As previously mentioned, the data used in this project was collected from the partner company's public website, Jscrambler. Since this constitutes a production environment some requirements were mandated to minimize the impact of data collection on user experience.

As was noted in [Amin Azad et al., 2020], ease of implementation is often considered key by developers of commercial bot detection solutions. This requirement was further enforced so as to minimize the workload of the Web Engineering team at Jscrambler that assisted with the implementation of the collection script.

A critical requirement to assure user experience would not be hindered by running the collection tool, is bandwidth it consumes. To that end, the Web Engineering team ran performance tests to ensure the tool does not impact the end user of the website. Further, data is only sent on specific events, to avoid server overload:

- When page loads or unloads.
- On "touchend" event (only available for the touch peripheral). When a the user lifts their finger from the touch screen.
- When keyboard, mouse or scroll are idle for 500ms after movement.

More than ever, anonymity is taken into consideration when dealing with people's data. Following that trend, this system only uses data for the purpose of detection. Identifying data is used to reconstruct sessions but is discarded after this is done, to ensure confidentiality. Furthermore, communications with the collecting server are encrypted through HTTPS.

4.2.2 Biometric Data Extracted

A full listing of the features collected client-side can be seen in Table 4.1. These features are not necessarily the final version of the data to be analyzed. As the script reloads on navigation, when the data reaches the server the session it belongs to needs to be reconstructed, more on this in the Section 4.3.

The captured attributes model the user's behaviour in terms of their interactions with the page. These attributes can be thought of as movement, pressure, time and typing-based along with a miscellaneous of others. For movement-based features, two-dimensional (x and y) data is captured for the mouse, touch screen and page scroll. Those are displacement, speed, acceleration which are common movement description values. A not so common movement variable also used in this work is the acceleration derivative, that can be used as a measure of the smoothness of movement, "jerk" [Roren et al., 2022]. Pressure statistics are measured for devices that allow it (mouse, pointer and touch) using the Pressure.js library [Yamartino, 2016]. Time-based metrics include

time elapsed since script was loaded, average time holding a touch, a mouse button or a key. For typing statistics characters are split into alphanumeric, correction (Delete and Backspace) and others, then the standard deviation of the type of character, as well as the entropy for that set of data is calculated. Strike speed and acceleration is also calculated. Finally, variables like whether the script was loaded on a mobile, how many times the page gained focus, and number of cuts, copies and pastes fit the miscellaneous category. To determine if a browser is mobile, a UA check is performed using the REGEX from [Smith, 2014]. If that fails, the maximum screen width is queried, if it's smaller than 1024 pixels, the user is considered to be on mobile.

Table 4.1: Biometric Features Selected

ID	Name	Description	Compound
0	Time Elapsed	Since script was loaded	True
1	No. of Packets	Number of packets in a session	True
2	Mobile	Boolean to check if user is on mobile	False
3	No. of Reloads	Humans often reload a page waiting for resources to arrive. Bots won't.	True
4	No. of Back Forward	Humans will go back and forth through their cached pages as they may not get all the information they wanted at first glance.	True
5	Focus Shifts	Number of times a windows gains focus	True
6	Mouse Displacement X	Amount of mouse movement in horizontal direction	True
7	Mouse Displacement Y	Amount of mouse movement in vertical direction	True
8	Mouse Speed X	(Mouse Displacement X)/Time	False
9	Mouse Speed Y	(Mouse Displacement Y)/Time	False
10	Mouse Acceleration X	(Mouse Speed X)/Time	False
11	Mouse Acceleration Y	(Mouse Speed Y)/Time	False
12	Mouse Jerk X	(Mouse Acceleration X)/Time	False
13	Mouse Jerk Y	(Mouse Acceleration Y)/Time	False
14	Touch Displacement X	Amount of touch movement in horizontal direction	True
15	Touch Displacement Y	Amount of touch movement in vertical direction	True
16	Touch Speed X	(Touch Displacement X)/Time	False
17	Touch Speed Y	(Touch Displacement Y)/Time	False
18	Touch Acceleration X	(Touch Speed X)/Time	False

Continuation of Table 4.1			
ID	Name	Description	Compound
19	Touch Acceleration Y	(Touch Speed Y)/Time	False
20	Touch Jerk X	(Touch Acceleration X)/Time	False
21	Touch Jerk Y	(Touch Acceleration Y)/Time	False
22	Scroll Displacement X	Amount of touch movement in horizontal direction	True
23	Scroll Displacement Y	Amount of touch movement in vertical direction	True
24	Scroll Speed X	(Scroll Displacement X)/Time	False
25	Scroll Speed Y	(Scroll Displacement Y)/Time	False
26	Scroll Acceleration X	(Scroll Speed X)/Time	False
27	Scroll Acceleration Y	(Scroll Speed Y)/Time	False
28	Scroll Jerk X	(Scroll Acceleration X)/Time	False
29	Scroll Jerk Y	(Scroll Acceleration Y)/Time	False
30	Average Mouse Click Pressure	Measured with Pressure.js	False
31	Average Touch Pressure	See 29	False
32	Average Pointer Pressure	See 29	False
33	Mouse Button 0 Click Number	Number of times mouse button 0 was clicked	True
34	Average Mouse Button 0 Click Time	Average time spent between clicking and releasing mouse button 0	False
35	Mouse Button 1 Click Number	Number of times mouse button 1 was clicked	True
36	Average Mouse Button 1 Click Time	Average time spent between clicking and releasing mouse button 1	False
37	Mouse Button 2 Click Number	Number of times mouse button 2 was clicked	True
38	Average Mouse Button 2 Click Time	Average time spent between clicking and releasing mouse button 2	False
39	Mouse Button 3 Click Number	Number of times mouse button 3 was clicked	True
40	Average Mouse Button 3 Click Time	Average time spent between clicking and releasing mouse button 3	False
41	Mouse Button 4 Click Number	Number of times mouse button 4 was clicked	True
42	Average Mouse Button 4 Click Time	Average time spent between clicking and releasing mouse button 4	False
43	Number of Touches	In name	True

Continuation of Table 4.1			
ID	Name	Description	Compound
44	Average Touch Press Time	In name	False
45	Strikes	Number of key downs	True
46	Presses	Number of key ups	True
47	No. Alphanumeric Keys	Number of alphanumeric characters input	True
48	No. of Delete and Backspace Keys	In name	True
49	No. of Other Keys	Non-alphanumeric characters and special keys	True
50	Ratio of Alphanumeric Keys to all others	(alphanumeric count)/total	True
51	Ratio of Correction Keys to all others	(correction count)/total	True
52	Ratio of Other Keys to all others	(other count)/total	True
53	Standard Deviation of Strike Type	Considering the 3 types the ones in rows 42, 43, 50	False
54	Entropy of Strike Type	See 45	False
55	Average Key Hold	Average time holding a key	False
56	Strike Speed	Strikes/Time	False
57	Strike Acceleration	Strike Speed/Time	False
58	Average Flight Time	Time between releasing a key and pressing another	False
59	No. of Cuts	In name	True
60	No. of Copies	In name	True
61	No. of Pastes	In name	True

4.2.3 Human Data

To identify human data, the existing Google *id* cookie was leveraged. This *id* is unique for each user that is logged in to their Jscrambler account and has cookies enabled. As most employees of the company have such an account and in general accept cookies, a collaboration request was put forth. An endpoint only accessible to logged in Jscrambler collaborators was put online. By navigating to this endpoint, the *id* cookie was stored in an anonymous fashion to protect employees identity. 44 (forty-four) employees voluntarily registered their *ids* at the endpoint, resulting in 569 (five-hundred sixty-nine) sessions collected from July 2022, to the 10th of August of the same year. Figure 4.1 represents graphically, a random (human's) behaviour for a random session.



Figure 4.1: Example Human Behaviour Diagram

4.2.4 Bot Data

To acquire labeled bot data, without resorting to manual labeling or proprietary tools, Web Bots were developed with Selenium Python; and aimed at Jscrambler's website. Later, their sessions were retrieved through their unique UAs. Two common bot attack scenarios were implemented in two levels of complexity: data scrapers and password crackers. The most basic bots used Selenium's basic methods to move the mouse on screen and instantly inject text into inputs. The more advanced bots implement the WindMouse algorithm to generate mouse trajectories, resulting in movements harder to distinguish from human. Furthermore, typing and scroll speed is slowed down to something that appears human-like, to the naked eye.

- When typing, characters in the input string are cycled through, being input one at the time. In between inputting characters the bot waits between .01 and 1 second.
- When scrolling, the distance to the element to scroll to is taken into consideration.
 - If this distance is greater than 400 pixels, the next scroll distance will be randomly picked from 150 to 400 pixels
 - Otherwise, scroll distance range is from 2 to 20 pixels
- In between scrolls, the bot waits from .01 to .2 seconds.

As stealthy bots often do, the ones developed for this project accept cookies when first arriving at the website. To ensure the creation of bots that follow a common interaction flow an interaction extractor was developed to compile the recurrent interactions. These can be moving a mouse on a page, navigating to a page, typing and clicking mouse buttons, etc. Having extracted the common interaction flows, it was determined these behaviours to already be implemented in the more advanced scrapers.

4.2.4.1 Scraper

After reaching the landing page and accepting cookies, the crawlers scrape the areas of the website accessible through the navigation bar (excluding the 'Documentation' section as the data collecting script was not available on that subdomain). Whenever a new page is loaded, the window is scrolled to the footer to simulate a human reading. Figure 4.2 represents graphically, the scraper bots' behaviour.

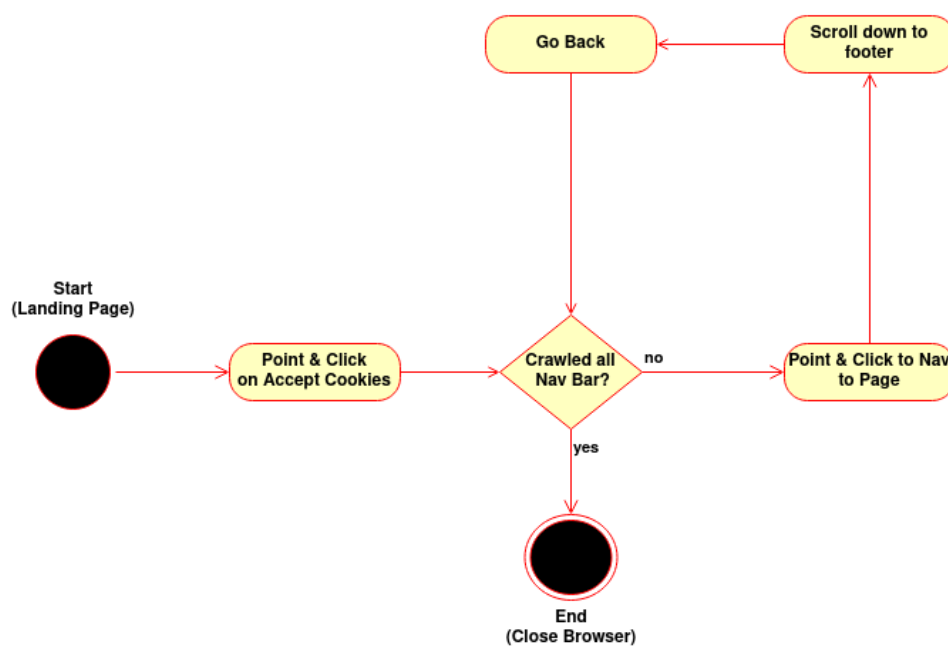


Figure 4.2: Crawler Behaviour Diagram

4.2.4.2 Cracker

Entering through the landing page, these bots navigate to the login page. There, a fake email is input and semi-random strings are passed as password values. These strings consist of 10 plus however many passwords have been input, alphabetical characters, selected using Python's "random" library, specifically the "choice" function. When both fields are filled, the bot attempts to login. Once it fails, it deletes the contents of the password field input and tries again. As the idea is to detect bots in as few packets as possible, this bot only attempts 5 passwords. Figure 4.3 represents graphically, the scraper bots' behaviour.

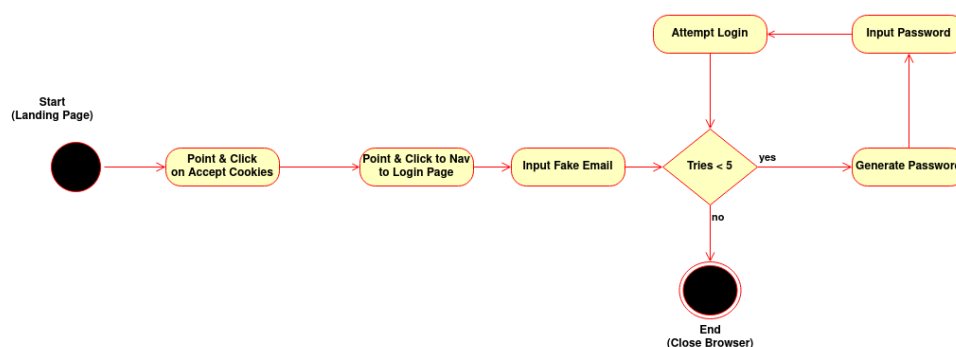


Figure 4.3: Crawler Behaviour Diagram

4.2.4.3 Interaction Flow Extraction

As previously stated, this tool was created to ensure bots covered normal interactions with the website. The idea was to reinforce the notion that detection is based on biometric anomalies rather than the pages visited. To extract common behaviours, sequential packets belonging to sessions in the training data were compared. When a page first got into focus, a navigation was assumed. When the mouse statistics varied in consecutive packets, a mouse movement was detected. The same logic was applied for scrolling, typing and clicking on the mouse. Once these behaviours were extracted, their occurrences were counted, in the hopes of finding something new that real users were doing and that the developed bots weren't. However, after analysing said behaviours, it was concluded that those that repeated a substantial amount, were also emulated by the bots. The most common being moving the mouse in between scrolls of a page, something that was done by the more advanced scraper.

4.3 Data Processing

Data preprocessing, in the context of this project, is dependent on the session a piece of data belongs to, as there are variables that compound throughout the session. As such, this section will first describe the Session Aggregator module before addressing data preprocessing.

4.3.1 Session Aggregator

As data packets come in they need to be associated with the session that produced them. The most common method for rebuilding sessions considers two packets to belong to the same session if they have the same IP, UA combination and are within a thirty minute interval of each other [Doran and Gokhale, 2011]. As such, this was the method used in this work with a modification to track sessions across proxies. For sessions that accept cookies a Google Analytics id is assigned to them. When available, if there exist two packets with the same id in a 30 minute timeout, they are considered to belong to the same session. Sessions are kept in a dictionary, indexed by a key that is either the IP, UA combo or just the Google id. Some variables are of immediate interest, these are calculated client-side, and when they reach the server they update their values on the session variable. These are: all variables of speed, acceleration and jerk; click/pointer/touch pressures; click/touch/key hold time and standard deviation and entropy of strike type. Other variables are compounded to describe the session up to that point. (e.g. total time, number of clicks, total mouse displacement, etc.). That last column in Table 4.1 refers to if that variable is immediate or compound. To calculate variations in compound variables, each session keeps a dictionary of the last packet of each page it has already visited. When a packet comes in, the variations in compound attributes can be calculated based on the values of the last packet for that page.

4.3.2 Data Preprocessing

This section will analyse if and how the four areas of Data Preprocessing were applied in the context of this project.

As the data set for this project is not free from missing values, this step of **Data Cleaning** had to be implemented. As for noise reduction and outlier removal, since the objective of the system is to identify outliers and contamination of the data is to be expected, it would change the philosophy of detection if outliers were removed beforehand.

In this project data, was collected in a single, centralized server. For that reason, the activity of **Data Integration** was deemed unnecessary.

Data Transformation processes, whether implicit or explicitly were applied throughout this project. **Feature selection** work was done in an informal fashion when originally determining the features to characterize biometric behaviour. As the attributes selected had no categorical values to encode, and the continuous ones did not require discretization, encoding was not implemented. Of the three models used in this project, only one did not require some **Normalization**, the Isolation Forest[Liu et al., 2009].

The volume of data in this project did not require **Data Reduction** techniques.

The following subsections will cover how the necessary steps of Data Preprocessing were implemented in this project.

4.3.2.1 Handling Missing Values

Analysing the data, features that were found to contain missing values were identified and when possible so were the causes for their disappearance.

- In pages where it is not possible to scroll horizontally, some browsers assumed Null values for the scroll's horizontal displacement. In this case, fixing the missing value was simply setting it to zero.
- A similar case occurs in browsers that do not support mouse buttons 3 and 4. Values for clicks and other statistics may be excluded, the fix is simply setting these values to zero.
- A value that was excluded seemingly at random was the page value. It is based on the 'window.location' property of the DOM which should be implemented on all browsers [MDN, 2021]. While this attribute is not directly used to classify a Session, it is based on it that accumulated statistics will be calculated during reconstruction. As such, 4 strategies were experimented with, with three achieving similar results and one trailing in performance.
 - The simplest strategy, which led to the worse results, was dropping rows of data with that feature missing.
 - The next two experiments consisted in assigning a value for that feature and considering all packets missing this to be from the same page. As was often the case, though the page value was missing, the packets could be seen as sequential and coming from

the same pages as their accumulated values were seen to follow a logical flow of navigation in one page (e.g. time elapsed kept increasing, number of clicks kept increasing smoothly, etc.).

- The value assigned was first a new fixed value " ", so as to not interfere with other packets that may be from pages reporting their actual page.
- After that, the strategy of using the mode value was tested. That meant that when a packet came in with no page it was assumed to be the landing page of the website. This had no impact when compared to the previous test.
- The final experiment while also not improving the detection performance, made the most semantic sense. If the first packet in a session had no page it is assumed it is the landing page. For all subsequent packets, if the page attribute is missing it is assumed to be the last one reported.

4.3.2.2 Data Scaling

Data scaling has two main branches: standardization and normalization. Standardization fits each column to a normal distribution of mean zero and variance one. Though it works best for data already normally distributed, if the actual distribution is not too far off from the normal, it can successfully scale data. All the while preserving negative values and being robust to outlier presence. Normalization, on the other hand, does not assume the data to follow any distribution. Instead, the strategy employed is to scale every feature between its minimum and maximum values [Baheti, 2021]. In [GeeksForGeeks, 2021b] the author claims, normalization may be hindered if the data contains outliers. Seeing as it takes the minimum and maximum of each feature as bounds, the claim seems reasonable. After experimenting with both Autoencoders and LOF models, Standardization was the chosen method of scaling as it provided better results.

4.4 Resulting training data

Training data was captured by embedding the collecting script (bc.js) into the Jscrambler's public website and storing it on their server. From there, all interactions with the website in June 2022 were recorded. As the website could be exposed to stealthy web bots it is assumed they contaminate the data set. As previously mentioned, it is for this reason that the strategy of unsupervised outlier detection was picked. After processing the training data, the resulting data set contained a total of 113 603 (one-hundred and thirteen thousand six-hundred and three) sessions; of varying lengths. Studied data set properties are presented in Table 4.2

Table 4.2: Data Set Properties

Total Sessions	113 603
No. of Mobile Sessions	26429 (23%)
No. of Computer Sessions	87174 (77%)
Avg. Time Spent (s)	1281 (21.3 mins)
Avg. No. of Packets	2.8

Interesting results from the exploration of the data set:

- Percentages of Mobile and PC users (23/77%)
- Avg. Time Spent (21 mins) vs. Avg. Number of Packets (approx. 3 packets)
 - It may be worth exploring further how the average session duration is 21 minutes, while the average number of packets is "only" 3. This may be due to the fact that a section of the website is not (namely the Documentation) is not covered by the collecting script.

4.5 Classifiers Trained

The classifiers trained for this project are the ones covered in Chapter 3, these are: LOF, Isolation Forest and an AutoEncoder. LOF was included due to its historical significance for the field of OD. Isolation Forest because of its ability to learn from sparse data, and performance when compared to other OD models [Domingues et al., 2017]. Finally, AutoEncoders are included to represent the increasingly researched area of Deep Learning. The tuning process for the model parameters, and experiments on the classifiers is covered in Chapter 5.

4.6 System Overview

This section will follow the data from its creation to its input into the OD module. Figure 4.4 describes the system's architecture.

As a user is browsing the protected website, on the aforementioned events, their biometric data will be sent to the server. There, if previous information on the session is available, that information will be updated. If not, a new session will be considered. From there, the data preprocessing routines will fix any issue in the data. Finally, the point containing both immediate and aggregate session information will be analyzed by the OD module and the session labeled as belonging to a Human or Bot.

The code for the AI models used in this project comes from the PyOD, (Python Outlier Detection), and Scikit-Learn packages, two Python based frameworks. The first aggregates OD models from several other libraries, including the second. For these reasons, and for consistency's sake, all implementations are fetched from PyOD. Scikit-Learn is an older, established package, includes

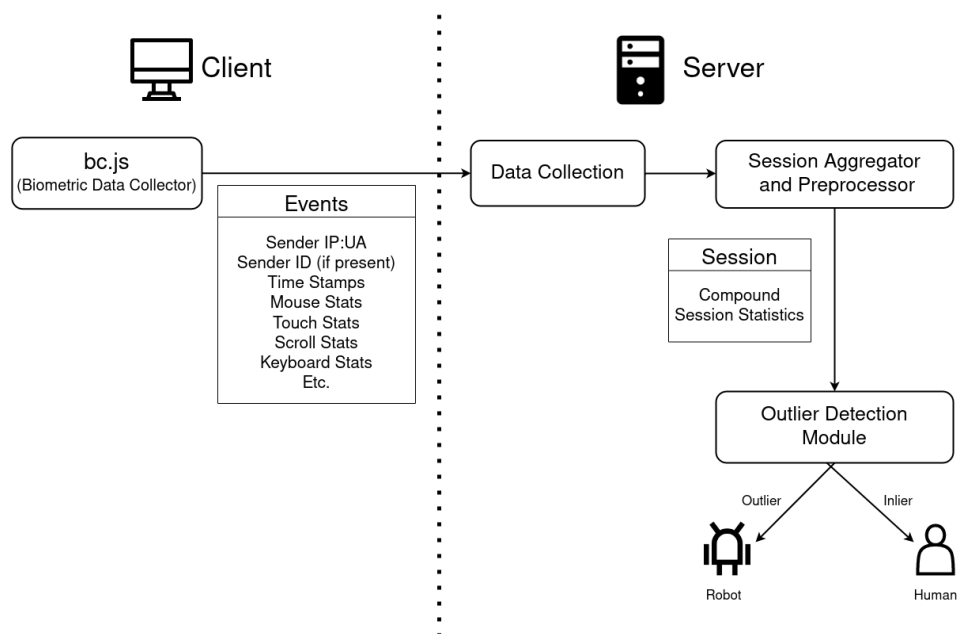


Figure 4.4: Proposed Solution System Overview

less models than PyOD. That being said, it includes other necessary components for OD and other machine learning applications, such as Scaling and Normalization functions, and Performance Evaluators [Zhao et al., 2019] [Pedregosa et al., 2011].

Chapter 5

Experiments, Results and Discussion

In this chapter, various experiments on the system will be discussed. The first set will focus on the tuning and training process of the different OD models. Following that, PoC experiments. Experiment 1 will analyze system performance according to standard metrics for evaluating OD models using random points proportionally picked from the test data. Experiment 2 will run through all testing instances and test the conceptualized system end-to-end, considering the context of a session. That is, as points come in, processing them, associating them to a session, and considering if the whole session is classified as a Bot. Furthermore, this experiment was also conducted to determine how many points were needed to classify the user. An unforeseen but valuable result was extracted from this experiment, that allowed to better the system's performance.

5.1 Tuning Model Parameters

A parameter that affects all models, and so was the first to be fixated, is the contamination percentage. The systems employed for OD require a parameter of contamination. This value estimates how many points in the original, unsupervised, data set are outliers. It is used to determine how outputs are interpreted as either outliers or inliers. (i.e. what range of outlier scores is considered a true outlier). During the initial training phase, small, earlier versions of crawlers developed, both simple and more advanced were run. Then the data they generated added to what was at the time, training data. Finally, the models were run on their default parameters and the results checked. The small crawlers acted as "canaries in a coal mine", the contamination parameter was gradually increased until the crawlers were identified as outliers. In the final training data set, the initial crawlers were removed so as to not further pollute the data. During this time, the detection seemed to achieve adequate performance, at least by detecting the "canaries", at 25% contamination. Seeing as some reports place Total Internet Traffic at around 40% being bot generated [Imperva, 2021], 25% was deemed not unreasonable as the data was collected from a public company website. After this parameter was temporarily set, the tuning specific to each model began. When that was concluded, testing while varying both contamination and other parameters was done, yet the best results were given at the original 25% value.

5.1.1 Parameters: Isolation Forest

As mentioned in Chapter 3 the Isolation Forest Algorithm has two parameters to tune. The simplest to understand is the number of trees to be built. Unless otherwise specified, every time a tree is built, it will select one feature from all the available and recursively insert up to a maximum number of samples in a binary tree. At each step picking random values in between the minimum and maximum for that partition over that attribute. As the authors mentioned in the original article, the number of trees, even for large complex data sets, is usually optimal with less than 100 (one-hundred) trees. [Liu et al., 2009] With that in mind, the first value estimated was that of at least as many trees as there are features, so that all features at least get some chance of being represented. So that was 62 trees. Through testing, from 62 to 70 estimators the performance did not drop. Using less than 62 trees however, did if ever so slightly worsen performance. As such, the number was fixed at 65 trees. Parallel to this tuning, the number of samples to create each tree was also tuned. Behaviour of this parameter, was more predictable than the number of trees. It defaulted to 512 points, which was fairly minuscule compared to the data set of 110 thousand entries. Further, at that point, not many crawler packets were being found in proportion to their prevalence. Doubling the number of entries to 1024, there was a performance upgrade across different numbers of estimators. Again doubling that to 2048 points achieved the best performance. After much testing, the number of entries from 1500 to 2500 do not impact much the performance, so 2048 samples was the number fixed.

5.1.2 Parameters: AutoEncoder

The default parameters for the AutoEncoder showed promise. The first tweaks were performed on the structure of the hidden layers. Having tested many layouts, the better performing provided similar results to the default. That being said, of all architectures, the one that was more in line with the characteristics of the data was one with 7 hidden layers, where:

- Layers 1 and 7 have 50 neurons
- Layers 2 and 6 have 38 neurons
- Layers 3 and 5 have 26 neurons
- Layer 4, the central one, has 14 neurons

As different architectures were tested, the number of epochs was also explored. However, it was evident that the model converged after around 60 epochs, and so 65 was the number selected.

Finally, across multiple architectures, the batch size was tweaked, this controls the number of points judged before re-weighting the nodes. This parameter proved to mainly speed up or down the learning process over the same number of epochs. A value was chosen to maximize speed while not interfering with detection results of 2048 points.

5.1.3 Parameters: LOF

The one parameter of LOF that can be tweaked, the number of neighbours considered, altered performance in a rather unpredictable way. As the authors themselves acknowledge in their original work [Breunig et al., 2000]. After several runs, the default value of 20 neighbours was kept as the best performer.

5.2 Experiment 1: Packet-based Web Bot Detection

As the contamination parameter was set to 25%, the testing data should follow the same principle to accurately measure performance. As such the training data consists of 522 randomly selected packets originating from humans and Jscrambler collaborators, and the remaining 174 existing bot packets. (Originating from the bots created). As there are more sessions than the number of human packets selected, this experiment was run several times to confirm its results. In reshuffling the human data set to cover all packets, the performance metrics did not deviate markedly from one run to another, so the results are presented from one random run.

IForest

Starting with the model that achieved the best performance, the Isolation Forest. As can be seen in Figure 5.1, the Confusion Matrix, the model predicted:

- True Negatives, or actual humans: 397 points
- True Positives, or actual bots: 120 points
- False Positives, or humans mistaken for bots: 125 points
- False Negatives, or bots mistaken for humans: 54 points

In Table 5.1 the numerical performance metrics, accuracy, precision, recall and f1-score are displayed for the Isolation Forest Model. On one side, metrics are calculated for the purpose of OD. Wherein, metrics are calculated in relation to the positive class. For the purpose of this project, the identification of bots. However, a system for bot detection may not trigger many false alarms. As such it is interesting to analyze the same metrics considering the classification of both classes. For the purpose of OD, an accuracy of 0.74 and f1-score of 0.57 may not be optimal, but indicate that the model is at least somewhat capable of identifying bots vs. humans.

Figure 5.2 displays the ROC for the current model. Its Area Under Curve can be interpreted as the probability, of given a packet of unknown origin, the classifier may correctly classify it. It's interesting to note, if not obvious at first, that this value is closely related to the balanced accuracy that can be seen in Table 5.1. (AUROC = 0.73, Balanced Accuracy = 0.73)

Analysing the PR curve in Figure 5.3, does not yield much information and precision remains constant as recall reaches its true value, after which it drops off. Nonetheless the AUC for the PR curve can also be used as a metric for model performance.

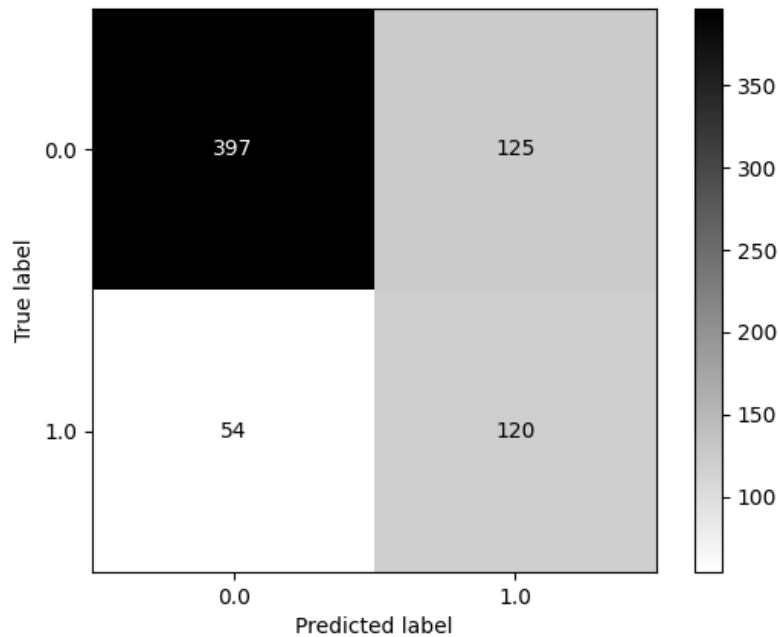


Figure 5.1: Isolation Forest Confusion Matrix

Table 5.1: IForest Performance Metrics

Based on Confusion Matrix		Adjusted for class imbalance	
Accuracy	0.74	Balanced Accuracy	0.73
Precision	0.49	Balanced Precision	0.68
Recall	0.69	Balanced Recall	0.72
F1-score	0.57	Balanced F1-score	0.69

As will be demonstrated in the following sections, performances of the other models tested were inferior to that of the Isolation Forest across all metrics.

AutoEncoder

The second best model was the AutoEncoder. As can be seen in Figure 5.4, the Confusion Matrix, the model predicted:

- True Negatives, or actual humans: 395 points
- True Positives, or actual bots: 104 points
- False Positives, or humans mistaken for bots: 127 points
- False Negatives, or bots mistaken for humans: 70 points

In Table 5.1 the numerical performance are displayed for the AutoEncoder Model. For the purpose of OD, an accuracy of 0.72 and f1-score of 0.51 may not be optimal. In fact, as previously

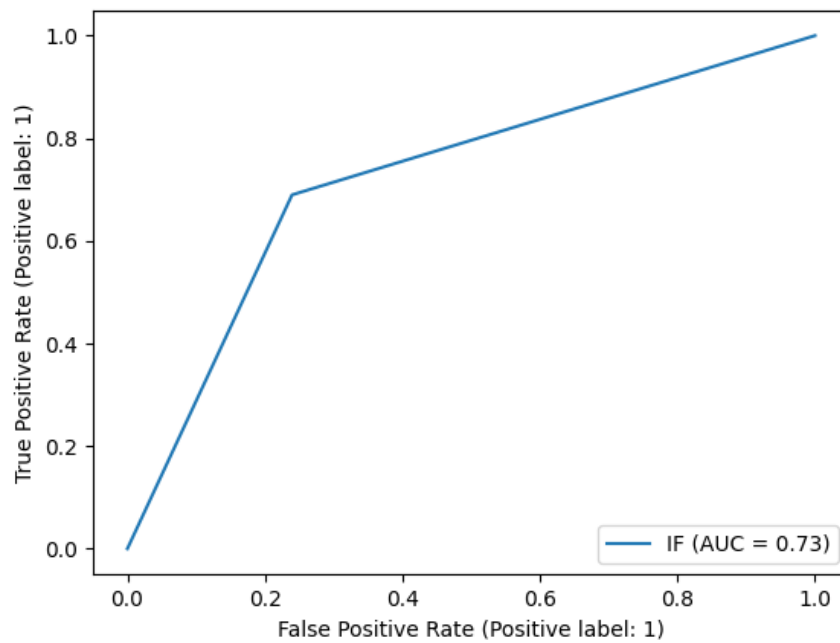


Figure 5.2: Isolation Forest ROC Curve

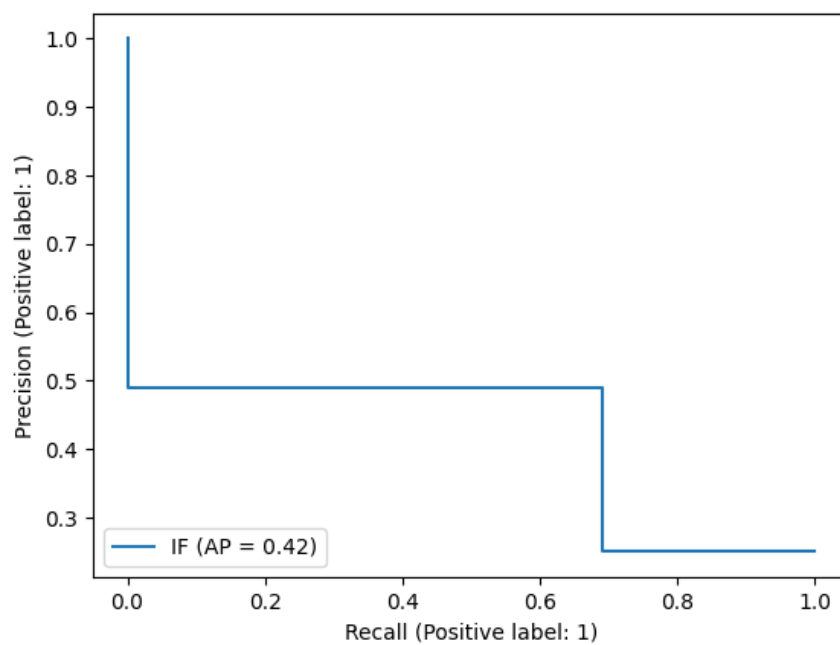


Figure 5.3: Isolation Forest Precision-Recall Curve

stated, they're clearly worse than the previous model, yet it remains that the model is at least somewhat capable at the task of identifying bots.

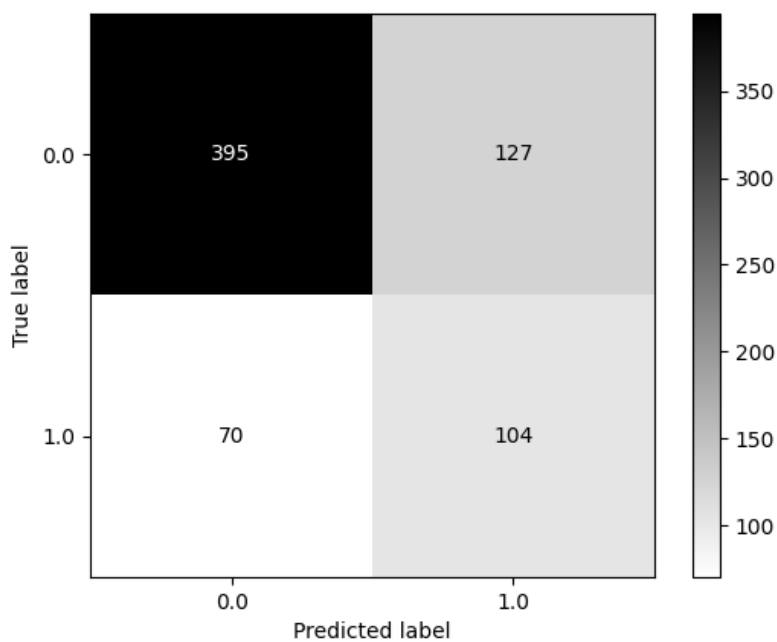


Figure 5.4: AutoEncoder Confusion Matrix

Table 5.2: AutoEncoder Performance Metrics

Based on Confusion Matrix		Adjusted for class imbalance	
Accuracy	0.72	Balanced Accuracy	0.68
Precision	0.45	Balanced Precision	0.65
Recall	0.60	Balanced Recall	0.68
F1-score	0.51	Balanced F1-score	0.66

The ROC curve for the AutoEncoder model is presented in Figure 5.5. Once again, the AU-ROC is equal to the model's balanced accuracy. ($0.68 = 0.68$)

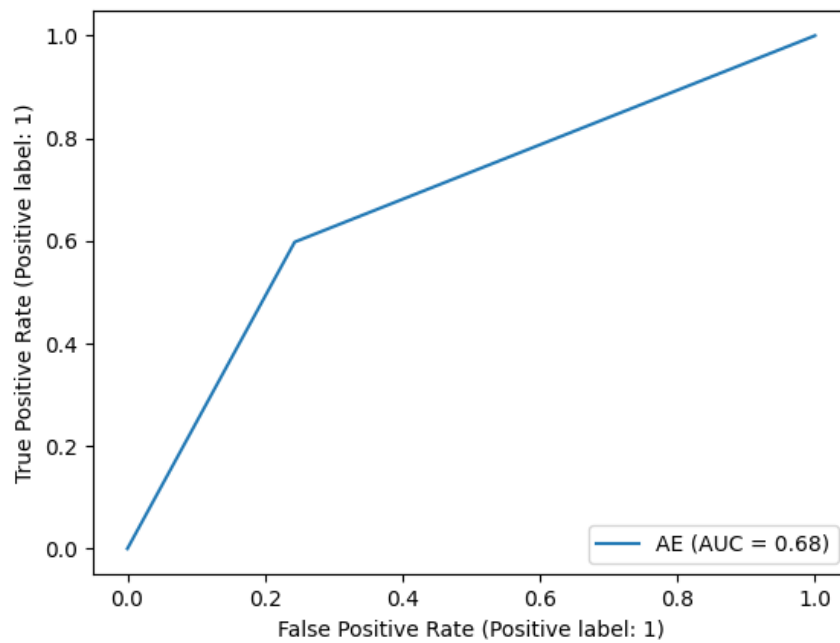


Figure 5.5: AutoEncoder ROC Curve

As was the case for the previous model, not much can be taken from the PR curve, Figure 5.6. Only that in comparison this models PR AUC is lower than the Isolation Forest Model. (0.37 vs. 0.42)

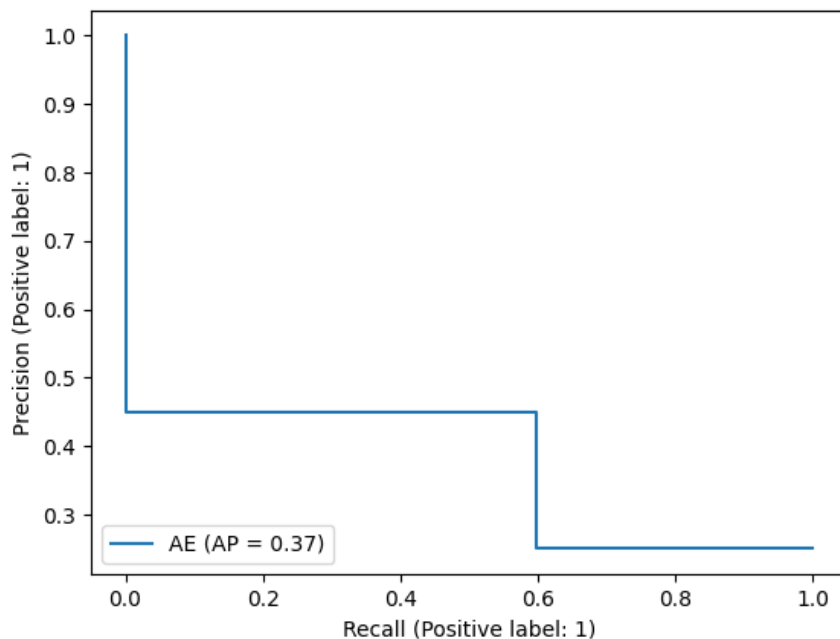


Figure 5.6: AutoEncoder Precision-Recall Curve

Though performance of the AutoEncoder is markedly worse than the Isolation Forest model in terms of metrics. Comparing both confusion matrices, one can see how the difference in the actual numbers is not that significant. Furthermore, these differences may be exacerbated by the smaller than desirable sample size.

LOF

The last model trained, was also the worse performing. The previous models, though there was a clear "winner", are still close enough to compare. The LOF, however, demonstrates a dip in performance, that coupled with a large amount of time spent training the model, make it the worst by far. For comparison purposes, the results are nonetheless included. Figure 5.7, presents the following predictions:

Table 5.3: LOF Performance Metrics

Based on Confusion Matrix		Adjusted for class imbalance	
Accuracy	0.65	Balanced Accuracy	0.63
Precision	0.38	Balanced Precision	0.60
Recall	0.58	Balanced Recall	0.63
F1-score	0.45	Balanced F1-score	0.60

An accuracy of 0.65 and an 0.45 F1-score, highlight how this model is not learning at the same pace as its more recent peers.

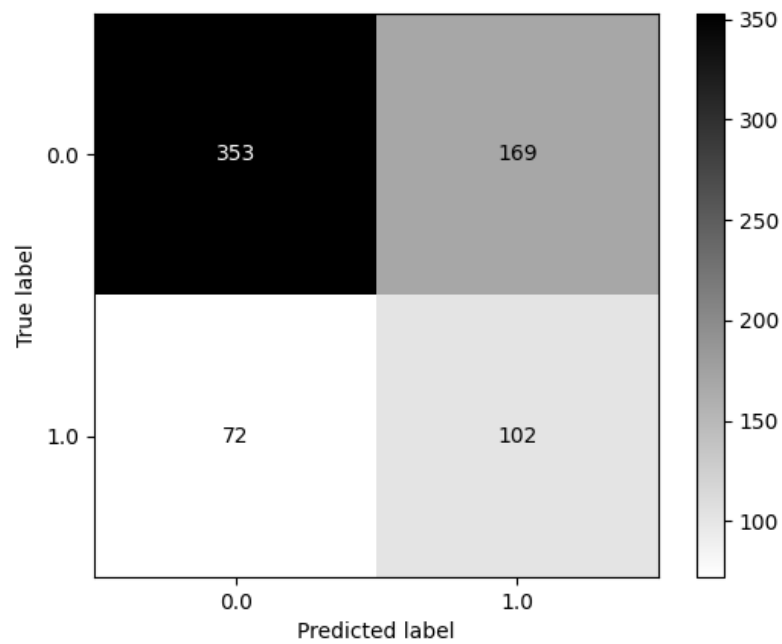


Figure 5.7: LOF Confusion Matrix

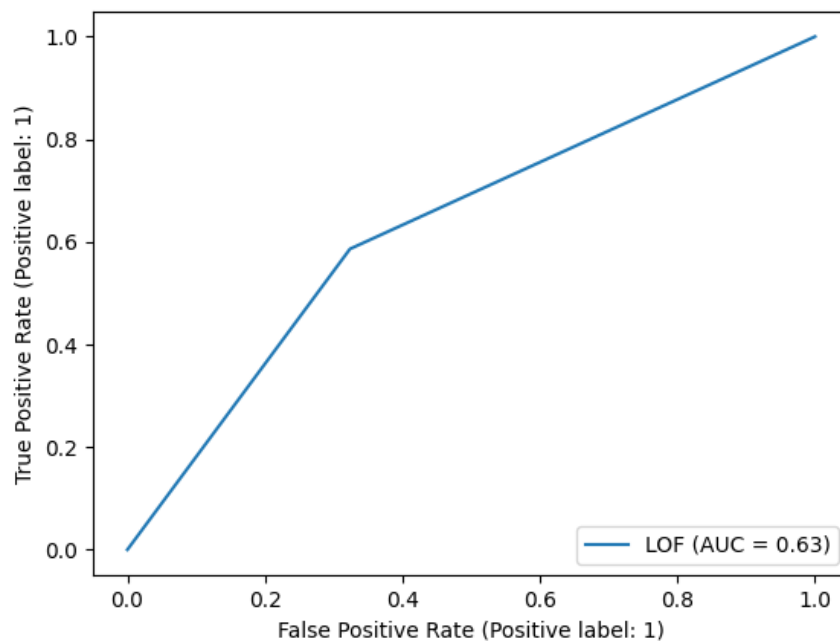


Figure 5.8: LOF ROC Curve

Figures 5.8 and 5.9 give more insight on how this model is worse than the previous across all major metrics.

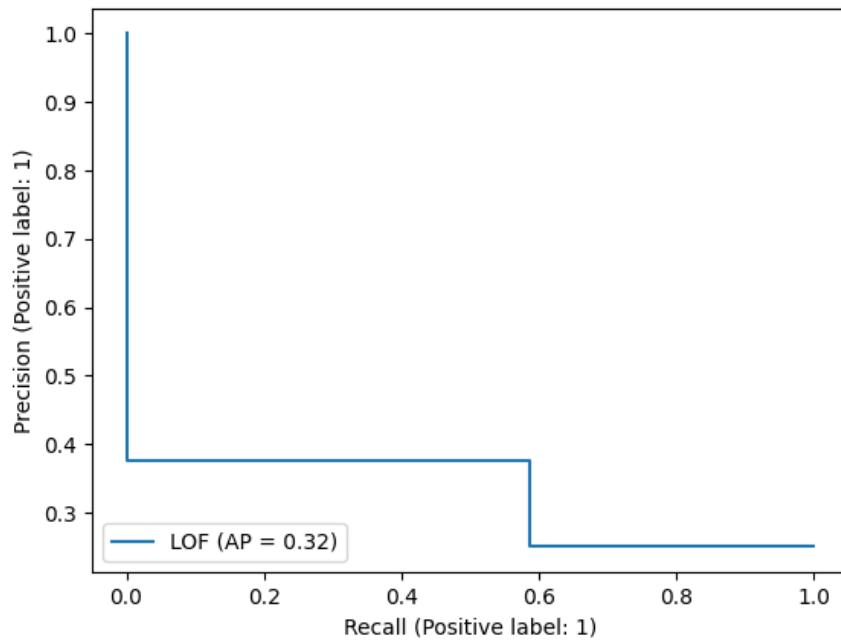


Figure 5.9: LOF Precision-Recall Curve

Model Comparison

Though many metrics can be calculated as can be seen in the above sections, as their interpretations often overlap, one can compare models using only a subset of these evaluators. In Figure 5.10 the metrics of Precision, Recall and F1-score are graphed. It is clear just by looking at the graph, the order of performance when comparing the three models.

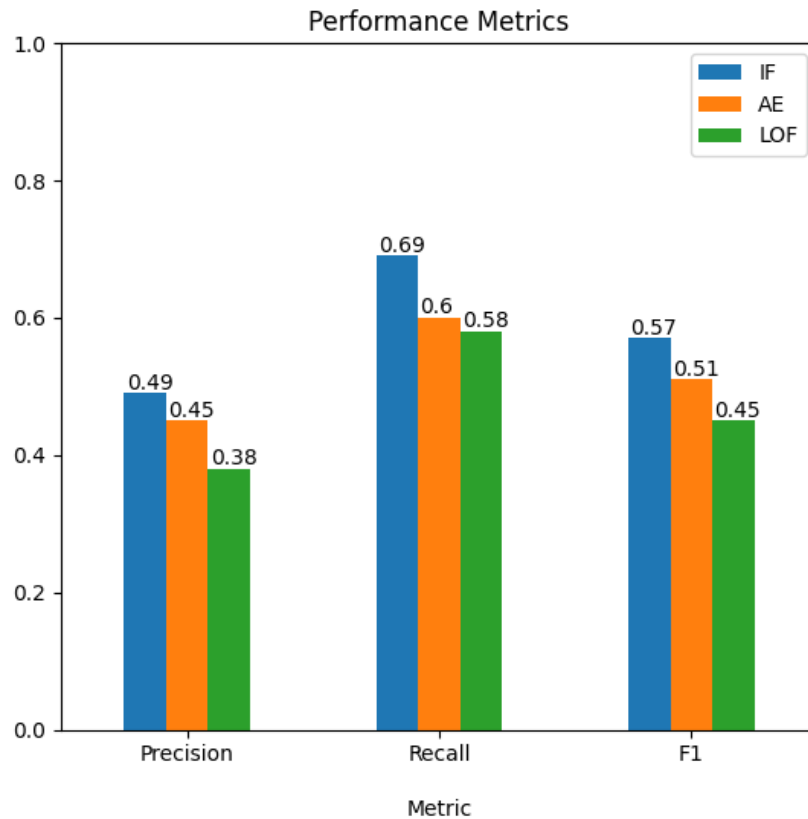


Figure 5.10: BAARZS

5.3 Experiment 2: Session-Based Web Bot Detection

Though the results in the first experiment were not the most exciting, the ones from this one may be more so. This experiment placed the system into a simulation of its real-world application. The testing data was read from its original, unprocessed, JSON state. As sessions were rebuilt, they were classified on their current state. If at one point a session was considered anomalous, the packet count, for that session, at which this occurred and the session key were recorded. During a first round of experimentation, many humans were being flagged as robots on their first packet. To counteract this, the first packet of every session was not considered for classification. With that, the number of misclassified human sessions dropped. All the while, bot sessions kept getting detected a few packets in.

This experiment demonstrated that:

- All bots can be identified in the first 4 packets by both models.
- Out of 569 human sessions:
 - The IForest only regarded 89 as anomalous, or 16% of the sessions

- The AutoEncoder model was slightly less accurate, regarding 95 sessions as outlying, or 17%.

This experiment raises an interesting question. How can the model have a worse performance when analyzing individual packets vs. when analyzing sessions as a whole. A possible explanation is that once a session becomes outlying, it will remain so. Furthermore, considering that sessions, and particularly human sessions vary greatly in number of packets, it would not be unreasonable to conclude that these anomalous sessions are longer than their normal counterparts. As in that case, the human data will have a higher proportion of these abnormal human sessions and so throw off the balance in the testing data for the first experiment.

It is worth describing certain properties of the data set used in these experiments. Table 5.4, analyses the same properties as Table 4.2 did for the training data.

Table 5.4: Data Set Properties

Total Sessions	573 (569 human)
No. of Mobile Sessions	0 (0%)
No. of Computer Sessions	573 (100%)
Avg. Time Spent (s)	380 (6.3 mins)
Avg. No. of Packets	3

As the data was collected passively from consenting Jscrambler collaborators, no restrictions were enforced on the data. As such, the testing sample did not include mobile users, which is something to consider and test for in the future.

5.4 Conclusions from Experiments

From the first experiment, two things are worth noting. First, from analysis of the ROC curves and accuracy related metrics, though there is room for improvement, the models are able, to some extent, to infer the identity of a user based on a random snapshot of their session. Where the need for improvement is most felt is in the variable of Precision, where the False Positives are considered. However, this variable may also be impacted by the difference in the number of sessions in the testing data when compared to training data.

This is more understandable when taking into consideration results from the second experiment. In this, the context of session was added, and in only 16% of the sessions did Isolation Forest misclassify the user. This means, though a lot of points were deemed anomalous when picked at random, they belonged to the same 16% of sessions that may have been longer and so had more points to use in the first experiment.

Though not completely clear from the results of the first experiment, when considering both, the results seem to indicate there is something of value in the approach of bot detection using biometric features and unsupervised outlier detection.

Chapter 6

Conclusions and Future Work

As bot detection evolves, so do bots. As such, it is necessary to keep improving their detection and containment. This paper presents a project that tried to do just that. Though the resulting models and the underlying feature set still have a lot of room for improvement, this project can be seen as a proof of concept for a biometric based, unsupervised, bot detection system.

Experiments in this project demonstrated a few interesting results. Though results in the first experiment show a higher than desirable number of false positives, this is influenced by the size of the data set. From the second experiment, where all sessions in their entirety are considered, the number of false positives drops, so we can conclude, detection considering all packets in a session is more accurate than if based on just 1 random packet. Furthermore, in both experiments the model with the best classifying performance was the Isolation Forest. Not only that, it is also the fastest to train and the one that requires the least amount of data. For these reasons, and for this feature set, the Isolation Forest was deemed the best model.

Some future work could include calculating how relevant each feature is for detection. With that not only could more variables be derived, obsolete variables could be dropped. Unlike the Isolation Forest [Liu et al., 2009], the other models used in this project were negatively impacted by features that did not offer much relevant information.

Furthermore, from the creation of this data set new features can be explored. A prime example of this would be the entropy of some continuous variables like mouse displacement in a movement. To calculate the entropy of a continuous variable one must first calculate its probability distribution. From the data collected in this project, the required distributions could be calculated.

The existence of a human verified data set, and of a mechanism to expand it being in place, opens a sub-field of OD to explore this feature set with. Namely, Anomaly Detection, in which models are trained by looking at only normal points, to then predict new unseen points.

As this project was developed with cooperation from a company, it is also important to look at it from a product point of view. If this solution was to be packaged as an anti-bot one, it would be interesting to include a module to detect synchronicity of requests and the possibility of a distributed attack as is done in [Jacob et al., 2012]. Furthermore, it might still be worth expanding the feature set by including more navigational attributes. As Chapter 2 reviewed, a lot of quality

work has been done in this field and could only improve the overall solution. The attributes required could also be extracted from requests and make the system less reliant on JavaScript.

While in the field of Bot Detection, the main goal of this system isn't necessarily to stop bots, but to try to guide their evolution towards being more human in behaviour, particularly speed. If bots are as slow as humans, then their existence isn't an issue, malicious usage by any actor is, but to detect malicious usage of a website, different, more specialized, tactics can be employed.

References

- [Acien et al., 2020] Acien, A., Morales, A., Fierrez, J., and Vera-Rodriguez, R. (2020). Becaptcha-mouse: Synthetic mouse trajectories and improved bot detection. *Pattern Recognition*, 127.
- [Aggarwal, 2016] Aggarwal, C. C. (2016). *Outlier Analysis*, chapter 3. Springer Publishing Company, Incorporated, 2nd edition.
- [Alla and Adari, 2019] Alla, S. and Adari, S. K. (2019). *Beginning Anomaly Detection Using Python-Based Deep Learning*. Apress Berkeley, CA.
- [Amin Azad et al., 2020] Amin Azad, B., Starov, O., Laperdrix, P., and Nikiforakis, N. (2020). Web runner 2049: Evaluating third-party anti-bot services. In *Detection of Intrusions and Malware, and Vulnerability Assessment: 17th International Conference, DIMVA 2020, Lisbon, Portugal, June 24–26, 2020, Proceedings*, page 135–159, Berlin, Heidelberg. Springer-Verlag.
- [Baheti, 2021] Baheti, P. (2021). A simple guide to data preprocessing in machine learning. Available at <https://www.v7labs.com/blog/data-preprocessing-guide>.
- [Banday and Shah, 2011] Banday, M. T. and Shah, N. (2011). A study of captchas for securing web services.
- [Boukerche et al., 2020] Boukerche, A., Zheng, L., and Alfandi, O. (2020). Outlier detection: Methods, models, and classification. *ACM Computing Surveys*, 53:1–37.
- [Breunig et al., 2000] Breunig, M. M., Kriegel, H.-P., Ng, R. T., and Sander, J. (2000). Lof: Identifying density-based local outliers. *SIGMOD Rec.*, 29(2):93–104.
- [Cabri et al., 2018] Cabri, A., Suchacka, G., Rovetta, S., and Masulli, F. (2018). Online web bot detection using a sequential classification approach. In *2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPC-C/SmartCity/DSS)*, pages 1536–1540.
- [Chen et al., 2020] Chen, H., He, H., and Starr, A. (2020). An overview of web robots detection techniques. In *2020 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*, pages 1–6.
- [Chu et al., 2013] Chu, Z., Gianvecchio, S., Koehl, A., Wang, H., and Jajodia, S. (2013). Blog or block: Detecting blog bots through behavioral biometrics. *Computer Networks*, 57(3):634–646.
- [DataDome, 2019] DataDome (2019). Good bots vs. bad bots and when you should block them. Available at <https://datadome.co/bot-management-protection/good-bots-vs-bad-bots-and-when-you-should-block-them/>.

- [Domingues et al., 2017] Domingues, R., Filippone, M., and Zouaoui, J. (2017). A comparative evaluation of outlier detection algorithms: Experiments and analyses. *Pattern Recognition*, 74.
- [Doran and Gokhale, 2011] Doran and Gokhale (2011). Web robot detection techniques: overview and limitations. *Data Mining and Knowledge Discovery*.
- [GeeksForGeeks, 2021a] GeeksForGeeks (2021a). Data preprocessing in data mining. Available at <https://www.geeksforgeeks.org/data-preprocessing-in-data-mining/>.
- [GeeksForGeeks, 2021b] GeeksForGeeks (2021b). Normalization vs standardization. Available at <https://www.geeksforgeeks.org/normalization-vs-standardization/>.
- [Hayati et al., 2010] Hayati, P., Potdar, V., Chai, K., and Talevski, A. (2010). Web spambot detection based on web navigation behaviour. pages 797–803.
- [Iliou et al., 2021] Iliou, C., Kostoulas, T., Tsikrika, T., Katos, V., Vrochidis, S., and Kompatsiaris, I. (2021). Detection of advanced web bots by combining web logs with mouse behavioural biometrics. *Digital Threats: Research and Practice*, 2(3).
- [Imperva, 2020] Imperva (2020). Thirteen questions you should ask your bot mitigation vendor - whitepaper. Technical report, Imperva.
- [Imperva, 2021] Imperva (2021). Bad bot report. Technical report, Imperva.
- [Jacob et al., 2012] Jacob, G., Kirda, E., Kruegel, C., and Vigna, G. (2012). Pubcrawl: protecting users and businesses from crawlers. pages 25–25.
- [Khder, 2021] Khder, M. (2021). Web scraping or web crawling: State of art, techniques, approaches and application. *International Journal of Advances in Soft Computing and its Applications*, 13:145–168.
- [Koster, 2007] Koster, M. (2007). The web robots pages. Available at <http://www.robotstxt.org/>.
- [Lagopoulos et al., 2018] Lagopoulos, A., Tsoumakas, G., and Papadopoulos, G. (2018). Web robot detection: A semantic approach. In *2018 IEEE 30th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 968–974.
- [Land, 2021] Land, B. J. (2021). Windmouse, an algorithm for generating human-like mouse motion. Available at <https://ben.land/post/2021/04/25/windmouse-human-mouse-movement/>.
- [Liu et al., 2009] Liu, F. T., Ting, K., and Zhou, Z.-H. (2009). Isolation forest. pages 413 – 422.
- [McKenna, 2016] McKenna, S. F. (2016). Detection and classification of web robots with honey-pots.
- [MDN, 2021] MDN, W. D. (2021). Window.location. Available at <https://developer.mozilla.org/en-US/docs/Web/API/Window/location>.
- [MDN, 2022] MDN, W. D. (2022). User-agent. Available at <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/User-Agent>.

- [npmjs, 2022] npmjs (2022). puppeteer. Available at <https://www.npmjs.com/package/puppeteer>.
- [Park et al., 2006] Park, K., Pai, V., Lee, K.-W., and Calo, S. (2006). Securing web service by automatic robot detection. pages 255–260.
- [Patra, 2020] Patra, D. (2020). selenium-stealth. Available at <https://github.com/diprajpatra/selenium-stealth>.
- [Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.
- [PyPIStats, 2022] PyPIStats (2022). selenium. Available at <https://pypistats.org/packages/selenium>.
- [Rahman and Tomar, 2020a] Rahman, R. U. and Tomar, D. S. (2020a). New biostatistics features for detecting web bot activity on web applications. *Computers Security*, 97:102001.
- [Rahman and Tomar, 2020b] Rahman, R. U. and Tomar, D. S. (2020b). A new web forensic framework for bot crime investigation. *Forensic Science International: Digital Investigation*, 33:300943.
- [Roren et al., 2022] Roren, A., Mazarguil, A., Vaquero-Ramos, D., Deloose, J.-B., Vidal, P.-P., Nguyen, C., Rannou, F., Wang, D., Oudre, L., and lefevre colau, m.-m. (2022). Assessing smoothness of arm movements with jerk: A comparison of laterality, contraction mode and plane of elevation. a pilot study. *Frontiers in Bioengineering and Biotechnology*, 9.
- [Scikit-Learn, 2022a] Scikit-Learn (2022a). Metrics and scoring: quantifying the quality of predictions. Available at https://scikit-learn.org/stable/modules/model_evaluation.html.
- [Scikit-Learn, 2022b] Scikit-Learn (2022b). Preprocessing data. Available at <https://scikit-learn.org/stable/modules/preprocessing.html#preprocessing>.
- [Selenium, 2022] Selenium (2022). Selenium. Available at <https://www.selenium.dev/>.
- [Singh, 2021] Singh, P. (2021). Is puppeteer better than selenium ?? Available at <https://chatbotslife.com/is-puppeteer-better-than-selenium-d348576787a8>.
- [Sisodia et al., 2015] Sisodia, D., Verma, S., and Vyas, O. (2015). Agglomerative approach for identification and elimination of web robots from web server logs to extract knowledge about actual visitors. *Journal of Data Analysis and Information Processing*, 03:1–10.
- [Smith, 2014] Smith, C. (2014). Detect mobile browsers. Available at <http://detectmobilebrowsers.com>.
- [Stassopoulou and Dikaiakos, 2009] Stassopoulou, A. and Dikaiakos, M. (2009). Web robot detection: A probabilistic reasoning approach. *Computer Networks*, 53:265–278.
- [Stevanovic et al., 2012] Stevanovic, D., An, A., and Vlajic, N. (2012). Feature evaluation for web crawler detection with data mining techniques. *Expert Systems with Applications*, 39:8707–8717.

- [Stevanovic et al., 2011] Stevanovic, D., Vlajic, N., and An, A. (2011). Unsupervised clustering of web sessions to detect malicious and non-malicious website users. *Procedia CS*, 5:123–131.
- [Stevanovic et al., 2013] Stevanovic, D., Vlajic, N., and An, A. (2013). Detection of malicious and non-malicious website visitors using unsupervised neural network learning. *Applied Soft Computing*, 13:698–708.
- [Suchacka and Sobkow, 2015] Suchacka, G. and Sobkow, M. (2015). Detection of internet robots using a bayesian approach. pages 365–370.
- [Tan and Kumar, 2002] Tan, P.-N. and Kumar, V. (2002). Discovery of web robot sessions based on their navigational patterns. *Data Min. Knowl. Discov.*, 6:9–35.
- [Vastel et al., 2020] Vastel, A., Rudametkin, W., Rouvoy, R., and Blanc, X. (2020). Fp-crawlers: Studying the resilience of browser fingerprinting to block crawlers.
- [Vikram et al., 2013] Vikram, S., Yang, C., and Gu, G. (2013). Nomad: Towards non-intrusive moving-target defense against web bots. pages 55–63.
- [Watson and Zaw, 2018] Watson, C. and Zaw, T. (2018). OWASP automated threat handbook: Web applications, version 1.2. Technical report, OWASP.
- [Wickramasinghe, 2021] Wickramasinghe, S. (2021). Selenium vs puppeteer: Which is better? Available at <https://www.blazemeter.com/blog/selenium-vs-puppeteer>.
- [Yamartino, 2016] Yamartino, S. (2016). Pressure.js. Available at <https://pressurejs.com/>.
- [Zabihimayvan et al., 2017] Zabihimayvan, M., Sadeghi, R., Rude, H. N., and Doran, D. (2017). A soft computing approach for benign and malicious web robot detection. *Expert Systems with Applications*, 87:129–140.
- [Zhao, 2017] Zhao, B. (2017). *Web Scraping*, pages 1–3.
- [Zhao et al., 2019] Zhao, Y., Nasrullah, Z., and Li, Z. (2019). Pyod: A python toolbox for scalable outlier detection. *Journal of Machine Learning Research*, 20(96):1–7.
- [Zolotukhin et al., 2014] Zolotukhin, M., Hamalainen, T., Kokkonen, T., and Siltanen, J. (2014). Analysis of http requests for anomaly detection of web attacks. pages 406–411.

Appendix A

Appendix

Code Listings for every component of this Project.

A.1 Biometric Data Collector: bc.js

The collection tool is split into three scripts. Two auxiliary classes and one main file. The base class is Move (move.js), representing the movement of either the Mouse, Touch or Scroll. That class is used to create the Stats class that compiles the features to send to the server. Finally, the script that joins everything and defines the triggers to send data to the server is bc.js.

move.js

```
1  //Describes a movement in terms of displacement, speed, aceleration and their  
   standard deviations  
2  class Move {  
3      constructor() {  
4          this.timestamp = 0;  
5          (this.x = 0), (this.y = 0);  
6          (this.totalX = 0), (this.totalY = 0);  
7          (this.speedX = 0), (this.speedY = 0);  
8          (this.acelX = 0), (this.acelY = 0);  
9          (this.jerkX = 0), (this.jerkY = 0);  
10     }  
11  
12     move(x, y, debug) {  
13         let now = Date.now();  
14         let dt = now - this.timestamp;  
15         let distanceX, distanceY;  
16         distanceX = Math.abs(this.x - x);  
17         distanceY = Math.abs(this.y - y);  
18         this.totalX += distanceX;  
19         this.totalY += distanceY;
```

```

20     this.x = x;
21     this.y = y;
22     this.timestamp = now;
23
24     let lastSX = this.speedX;
25     let lastSY = this.speedY;
26
27     this.speedX = distanceX / dt;
28     this.speedY = distanceY / dt;
29
30     let lastAX = this.acelX;
31     let lastAY = this.acelY;
32
33     this.acelX = (this.speedX - lastSX) / dt;
34     this.acelY = (this.speedY - lastSY) / dt;
35
36     this.jerkX = (this.acelX - lastAX) / dt;
37     this.jerkY = (this.acelY - lastAY) / dt;
38
39     if (debug) console.log(this);
40 }
41 }
42
43 module.exports = Move;

```

stats.js

```

1 let Move = require("../move");
2
3 const start = Date.now();
4
5 class Stats {
6     constructor() {
7         // Since script loaded
8         this.timeElapsed = 0;
9         // Is script running on mobile device?
10        this.mobile = false;
11        // Booleans to see if page was reloaded or loaded from cache through the
12        // arrows in the browser window
13        (this.reload = false), (this.bckFwd = false);
14        // Focus Shift counter: counter for how many times the window has lost and
15        // gained focus
16        this.fs = 0;
17        // Movement Statistics
18        this.mouseM = new Move();
19        this.touchM = new Move();

```

```

18     this.scrollM = new Move();
19
20     // Click Pressure Statistics not currently used
21     this.currClickPresh = 0;
22     this.avgClickPresh = 0;
23     this.noClickP = 0;
24     this.currTouchPresh = 0;
25     this.avgTouchPresh = 0;
26     this.noTouchesP = 0;
27     this.currPointerPresh = 0;
28     this.avgPointerPresh = 0;
29     this.noPointerP = 0;
30
31     // One for each button, 0 is Left, 1 is Middle/Wheel and 2 is Right
32     this.buttons = [
33         { clickTimestamp: 0, clicks: 0, avgClickHold: 0 },
34         { clickTimestamp: 0, clicks: 0, avgClickHold: 0 },
35         { clickTimestamp: 0, clicks: 0, avgClickHold: 0 },
36         { clickTimestamp: 0, clicks: 0, avgClickHold: 0 },
37         { clickTimestamp: 0, clicks: 0, avgClickHold: 0 },
38     ];
39
40     // Obj to keep track of touch data
41     this.touch = { touchTimestamp: 0, touches: 0, avgTouchHold: 0 };
42
43     this.delKeys = 0; // Number of times delete or backspace key was pressed
44     this.alphanumKeys = 0; // Number of [a-zA-Z0-1] characters hit
45     this.otherKeys = 0; // Other characters
46     this.strikeStdDev = 0;
47     this.strikeEntropy = 0;
48
49     (this.cuts = 0), (this.cpys = 0), (this.pastes = 0);
50
51     // Map with the key and the time holding it n vars to calculate avg key
52     // hold and typing speeds
53     this.keys = new Map();
54     (this.avgKeyHold = 0),
55     (this.keyPresses = 0),
56     (this.strikes = 0),
57     (this.lastStrike = 0),
58     (this.strikeSpeed = 0),
59     (this.strikeAcel = 0);
60
61     // Vars to help calculate key flight time
62     (this.avgFlightTime = 0),
63     (this.flights = 0),
64     (this.flightTimestamp = 0);
65 }

```

```

66   static mean(arr) {
67       return (
68           arr.reduce((acc, curr) => {
69               return acc + curr;
70           }, 0) / arr.length
71       );
72   }
73
74   static standardDeviation(arr) {
75       let mean = this.mean(arr);
76
77       let squareRay = arr.map((el) => {
78           return (el - mean) ** 2;
79       });
80
81       let sum = squareRay.reduce((acc, curr) => acc + curr, 0);
82
83       return Math.sqrt(sum / arr.length);
84   }
85
86   setTime() {
87       this.timeElapsed = Date.now() - start;
88   }
89 }
90
91 module.exports = Stats;

```

bc.js

```

1  "use strict";
2
3  // Pressure.js library needed to read mouse and touch pressure
4  const Pressure = require("pressure");
5  const Stats = require("./stats");
6
7  let statistics = new Stats();
8
9  function statCleaner(key, value) {
10     if (
11         key == "keys" ||
12         key == "flights" ||
13         key == "flightTimestamp" ||
14         key == "lastStrike" ||
15         key == "currClickPresh" ||
16         key == "currTouchPresh" ||
17         key == "currPointerPresh" ||

```



```

18     key == "noClickP" ||
19     key == "noTouchesP" ||
20     key == "noPointerP"
21 )
22     return undefined;
23 if (key == "mouseM" || key == "scrollM" || key == "touchM") {
24     let newVal = { ...value };
25     delete newVal.x;
26     delete newVal.y;
27     delete newVal.timestamp;
28     return newVal;
29 }
30
31 if (key == "touch") {
32     let newVal = { ...value };
33     delete newVal.touchTimestamp;
34     return newVal;
35 }
36
37 if (key == "buttons") {
38     let newVal = { ...value };
39     delete newVal[0].clickTimestamp;
40     delete newVal[1].clickTimestamp;
41     delete newVal[2].clickTimestamp;
42     delete newVal[3].clickTimestamp;
43     delete newVal[4].clickTimestamp;
44     return newVal;
45 } else return value;
46 }
47
48 function sendStats() {
49     statistics.page = document.location.href;
50     let data = new FormData();
51     data.append("obj", JSON.stringify(statistics, statCleaner));
52     navigator.sendBeacon(
53         "http://localhost/capazpsicologia/testLog.php",
54         JSON.stringify(statistics, statCleaner)
55     );
56 }
57
58 function checkForMobile(){
59
60     ((a) => {
61         if ((
62             /(android|bb\d+|meego).+mobile|avantgo|bada\/|blackberry|blazer|compal|
                elaine|fennec|hiptop|iemoible|ip(hone|od)|iris|kindle|lge |maemo|
                midp|mmp|mobile.+firefox|netfront|opera m(ob|in)i|palm( os)?|phone|
                p(ixi|re)\|plucker|pocket|psp|series(4|6)0|symbian|treo|up\.(
                browser|link)|vodafone|wap|windows ce|xda|xiino/i.test(

```

```

63         a
64     ) ||
65     /1207|6310|6590|3gso|4thp|50[1-6|i|770s|802s|a wa|abac|ac(er|oo|s\-)|ai
        (ko|rn)|al(av|ca|co)|amoi|an(ex|ny|yw)|aptu|ar(ch|go)|as(te|us)|
        attw|au(di|\-m|r |s )|avan|be(ck|ll|nq)|bi(lb|rd)|bl(ac|az)|br(e|v)
        w|bumb|bw\-(n|u)|c55\|capi|ccwa|cdm\|-|cell|chtm|cldc|cmd\|-|co(mp|
        nd)|craw|da(it|ll|ng)|dbte|dc\-s|devi|dica|dmob|do(c|p)o|ds(12|\-d)
        |el(49|ai)|em(l2|ul)|er(ic|k0)|esl8|ez([4-7]0|os|wa|ze)|fetc|fly
        (\-|_)|gl u|g560|gene|gf\5|g\mo|go(\.w|od)|gr(ad|un)|haie|hcit|hd
        \-(m|p|t)|hei\|hi(pt|ta)|hp( i|ip)|hs\|ht(c(\-| _|a|g|p|s|t)|tp
        )|hu(aw|tc)|i\-(20|go|ma)|i230|iac( |\-|\/)|ibro|idea|ig01|ikom|
        im1k|inno|ipaq|iris|ja(t|v)a|jbro|jemu|jigs|kddi|keji|kgt( |\\/)|
        klon|kpt |kwc\|kyo(c|k)|le(no|xi)|lg( g|\/(k|l|u)|50|54|\-[a-w])|
        libw|lynx|m1\-w|m3ga|m50\|ma(te|ui|xo)|mc(01|21|ca)|m\-cr|me(rc|ri
        )|mi(o8|oa|ts)|mmef|mo(01|02|bi|de|do|t(\-| |o|v)|zz)|mt(50|p1|v )|
        mwbp|mywa|n10[0-2]|n20[2-3]|n30(0|2)|n50(0|2|5)|n7(0(0|1)|10)|ne((c
        |m)\-|on|tf|wf|wg|wt)|nok(6|i)|nzph|o2im|op(ti|wv)|oran|owg1|p800|
        pan(a|d|t)|pdxg|pg(13|\-([1-8]|c))|phil|pire|pl(ay|uc)|pn\2|po(ck|
        rt|se)|prox|psio|pt\|g|qa\|a|qc(07|12|21|32|60|\-[2-7]|i\)|qtek|
        r380|r600|raks|rim9|ro(ve|zo)|s55\|sa(ge|ma|mm|ms|ny|va)|sc(01|h
        \-|oo|p\)|sdk\|se(c(\-|0|1)|47|mc|nd|ri)|sgh\|shar|sie(\-|m)|sk
        \-0|sl(45|id)|sm(al|ar|b3|it|t5)|so(ft|ny)|sp(01|h\|v\|v )|sy(01|
        mb)|t2(18|50)|t6(00|10|18)|ta(gt|lk)|tcl\|tdg\|tel(i|m)|tim\|t\|
        mo|to(pl|sh)|ts(70|m\|m3|m5)|tx\9|up(\.b|gl|si)|utst|v400|v750|
        veri|vi(rg|te)|vk(40|5[0-3]| \-v)|vm40|voda|vulc|vx
        (52|53|60|61|70|80|81|83|85|98)|w3c(\-| )|webc|whit|wi(g |nc|nw)|
        wmlb|wonu|x700|yas\|your|zeto|zte\-/i.test(
66         a.substr(0, 4)
67     )
68 )) {
69     statistics.mobile = true;
70 }
71 )) (
72     navigator.userAgent || navigator.vendor || window.opera,
73 );
74
75 if (window.matchMedia("only screen and (max-width: 1024px)").matches)
76     statistics.mobile = true;
77 }
78
79 // To measure average click/touch/pointer pressure
80 Pressure.set(
81     "*",
82     {
83         change: (force, _) => {
84             statistics.currClickPresh = force;
85             statistics.setTime();
86         },

```

```

87         end: function () {
88             statistics.avgClickPresh =
89                 (statistics.avgClickPresh * statistics.noClickP +
90                  statistics.currClickPresh) /
91                 (statistics.noClickP + 1);
92             statistics.noClickP++;
93             statistics.setTime();
94         },
95     },
96     { only: "mouse", preventSelect: false, polyfil:false }
97 );
98
99 Pressure.set (
100     "*",
101     {
102         change: (force, _) => {
103             statistics.currTouchPresh = force;
104             statistics.setTime();
105         },
106         end: function () {
107             statistics.avgTouchPresh =
108                 (statistics.avgTouchPresh * statistics.noTouchesP +
109                  statistics.currTouchPresh) /
110                 (statistics.noTouchesP + 1);
111             statistics.noTouchesP++;
112             statistics.setTime();
113         },
114     },
115     { only: "touch", preventSelect: false, polyfil:false }
116 );
117
118 Pressure.set (
119     "*",
120     {
121         change: (force, _) => {
122             statistics.currPointerPresh = force;
123             statistics.setTime();
124         },
125         end: function () {
126             statistics.avgPointerPresh =
127                 (statistics.avgPointerPresh * statistics.noPointerP +
128                  statistics.currPointerPresh) /
129                 (statistics.noPointerP + 1);
130             statistics.noPointerP++;
131             statistics.setTime();
132         },
133     },
134     { only: "pointer", preventSelect: false, polyfil:false }
135 );

```

```

136
137 // Counting Focus Shifts
138 window.addEventListener("focus", () => {
139     statistics.fs++;
140     statistics.setTime();
141 });
142
143 // Keeping track of cuts, copys and pastes
144 window.addEventListener("cut", () => statistics.cuts++);
145 window.addEventListener("copy", () => statistics.cyps++);
146 window.addEventListener("paste", () => statistics.pastes++);
147
148 // Keydown Events
149 window.addEventListener("keydown", (event) => {
150     statistics.strikes++;
151     const re = /^[a-zA-Z0-9]$/;
152
153     if (event.key === "Backspace" || event.key === "Delete")
154         statistics.delKeys++;
155     else if (re.test(event.key)) statistics.alphanumKeys++;
156     else {
157         statistics.otherKeys++;
158         console.log("got key: " + event.key);
159     }
160
161     statistics.strikeStdDev = Stats.standardDeviation([
162         statistics.alphanumKeys,
163         statistics.delKeys,
164         statistics.otherKeys,
165     ]);
166
167     let entA = isNaN(
168         (statistics.alphanumKeys / statistics.strikes) *
169         Math.log2(statistics.alphanumKeys / statistics.strikes)
170     )
171         ? 0
172         : (statistics.alphanumKeys / statistics.strikes) *
173           Math.log2(statistics.alphanumKeys / statistics.strikes);
174     let entD = isNaN(
175         (statistics.delKeys / statistics.strikes) *
176         Math.log2(statistics.delKeys / statistics.strikes)
177     )
178         ? 0
179         : (statistics.delKeys / statistics.strikes) *
180           Math.log2(statistics.delKeys / statistics.strikes);
181     let entO = isNaN(
182         (statistics.otherKeys / statistics.strikes) *
183         Math.log2(statistics.otherKeys / statistics.strikes)
184     )

```

```

185         ? 0
186         : (statistics.otherKeys / statistics.strikes) *
187           Math.log2(statistics.otherKeys / statistics.strikes);
188
189     statistics.strikeEntropy = 0 - entA - entD - entO;
190
191     if (statistics.strikeEntropy == null) statistics.strikeEntropy = 0;
192
193     if (!statistics.keys.has(event.key)) {
194         statistics.keys.set(event.key, Date.now());
195     }
196 });
197
198 let kbSendData = setTimeout(sendStats, 500);
199 // Keyup Events
200 window.addEventListener("keyup", (event) => {
201     statistics.setTime();
202     clearTimeout(kbSendData);
203     if (statistics.flightTimestamp === 0)
204         statistics.flightTimestamp = Date.now();
205     // To calculate avg time holding key
206     if (statistics.keys.has(event.key)) {
207         statistics.keyPresses++;
208         let dt = Date.now() - statistics.keys.get(event.key);
209         statistics.avgKeyHold =
210             ((statistics.keyPresses - 1) * statistics.avgKeyHold + dt) /
211             statistics.keyPresses;
212
213         if (statistics.avgKeyHold == null) statistics.avgKeyHold = 0;
214         statistics.keys.delete(event.key);
215
216         dt = Date.now() - statistics.lastStrike;
217         let lastSS = statistics.strikeSpeed;
218         statistics.strikeSpeed = statistics.keyPresses / dt;
219         statistics.strikeAcel = (lastSS - statistics.strikeSpeed) / dt;
220         statistics.lastStrike = Date.now();
221     }
222
223     // To calculate key flight time
224     let ft = Date.now() - statistics.flightTimestamp;
225     statistics.flights++;
226     statistics.avgFlightTime =
227         (statistics.flights * statistics.avgFlightTime + ft) /
228         (statistics.flights + 1);
229     statistics.flightTimestamp = Date.now();
230
231     kbSendData = setTimeout(sendStats, 500);
232 });
233

```

```

234 // Touch Screen movement event
235 window.addEventListener("touchmove", (event) => {
236     statistics.touchM.move(
237         event.touches[0].pageX,
238         event.touches[0].pageY,
239         false
240     );
241 });
242
243 // Touch Start event
244 window.addEventListener("touchstart", () => {
245     statistics.touch.touchTimestamp = Date.now();
246     statistics.touch.touches++;
247 });
248
249 // Touch End event
250 window.addEventListener("touchend", () => {
251     statistics.setTime();
252     let dt = Date.now() - statistics.touch.touchTimestamp;
253     statistics.touch.avgTouchHold =
254         (statistics.touch.touches * statistics.touch.avgTouchHold + dt) /
255         (statistics.touch.touches + 1);
256     statistics.touch.touchTimestamp = Date.now();
257     sendStats();
258 });
259
260 let mouseSendData = setTimeout(sendStats, 500);
261
262 // Mouse movement event
263 window.addEventListener("mousemove", (event) => {
264     statistics.setTime();
265     clearTimeout(mouseSendData);
266     statistics.mouseM.move(event.pageX, event.pageY, false);
267     mouseSendData = setTimeout(sendStats, 500);
268 });
269
270 // Mousedown Events
271 window.addEventListener("mousedown", (event) => {
272     let i = event.button;
273     statistics.buttons[i].clickTimestamp = Date.now();
274     statistics.buttons[i].clicks++;
275 });
276
277 // Mouseup Events
278 window.addEventListener("mouseup", (event) => {
279     let i = event.button;
280     let ct = Date.now() - statistics.buttons[i].clickTimestamp;
281     statistics.buttons[i].avgClickHold =
282         (statistics.buttons[i].avgClickHold * statistics.buttons[i].clicks +

```

```

283         ct) /
284         (statistics.buttons[i].clicks + 1);
285         statistics.buttons[i].clickTimestamp = Date.now();
286     });
287
288     // Page Load Events
289     window.addEventListener("load", () => {
290         statistics.setTime();
291
292         checkForMobile();
293
294         if (window.performance.navigation) {
295             if (window.performance.navigation === 1) {
296                 statistics.reload = true;
297             }
298
299             if (window.performance.navigation === 2) {
300                 statistics.bckFwd = true;
301             }
302         }
303
304         if (window.performance.getEntriesByType("navigation")) {
305             let navType = window.performance.getEntriesByType("navigation")[0].type;
306
307             if (navType == "reload") {
308                 statistics.reload = true;
309             }
310
311             if (navType == "back_forward") {
312                 statistics.bckFwd = true;
313             }
314         }
315
316         sendStats();
317     });
318
319     let scrollSendData = setTimeout(sendStats, 500);
320     //Scroll Events
321     window.addEventListener("scroll", () => {
322         statistics.setTime();
323         clearTimeout(scrollSendData);
324         statistics.scrollM.move(window.scrollX, window.scrollY, false);
325         scrollSendData = setTimeout(sendStats, 500);
326     });
327
328     //Unload event, plain AJAX won't work here, but sendBeacon will
329     window.addEventListener("unload", () => {
330         statistics.setTime();
331         sendStats();

```

```
332 });
```

A.2 Detector Modules

The first listing is for a set of helper functions to load data from a directory with multiple JSON files, according to a naming convention set by the Web Engineering team.

dataloader.py

```
1
2 import os
3 import json
4
5 def sortDir(lof):
6     sof = []
7     for filo in lof:
8         if not (".json" in filo):
9             continue
10        if len(sof) == 0:
11            sof.append(filo)
12        elif len(sof) == 1:
13            date = filo.split('-')[1]
14            date = date.split('_')
15            day = int(date[0])
16            mon = int(date[1])
17            d2 = sof[0].split('-')[1]
18            d2 = d2.split('_')
19            dd2 = int(d2[0])
20            md2 = int(d2[1])
21
22            if md2 < mon or (mon == md2 and dd2 > day):
23                sof.insert(0, filo)
24            else:
25                sof.append(filo)
26        else:
27            date = filo.split('-')[1]
28            date = date.split('_')
29            day = int(date[0])
30            mon = int(date[1])
31            i = 0
32            for i in range(0, len(sof)):
33                d2 = sof[i].split('-')[1]
34                d2 = d2.split('_')
35                dd2 = int(d2[0])
36                md2 = int(d2[1])
```



```

37         if md2 > mon or (md2 == mon and dd2 > day):
38             break
39         sof.insert(i, filo)
40     return sof
41
42 def loadJSONFromDirS(dir):
43     data = []
44     lof = os.listdir(dir)
45     sof = sortDir(lof)
46
47     for filo in sof:
48         if ".json" in filo:
49             f = open(dir + filo, "r")
50             data += json.loads(f.read())
51
52     return data
53
54 def loadJSONFromDir(dir):
55     data = []
56     lof = os.listdir(dir)
57     for filo in lof:
58         if ".json" in filo:
59             f = open(dir + filo, "r")
60             data += json.loads(f.read())
61
62     return data

```

The next listings are for the session class, that defines how a session is reconstructed, and for the associated preprocessing module.

session.py

```

1 import time
2
3 import numpy as np
4
5 from dateutil import parser
6
7 # Maybe Session should keep track of no. of events
8
9 class Session:
10
11     def __init__(self, point):
12
13         if "sid" in point:
14             self.key = point["sid"]
15         else:

```

```

16     self.key = point["ip"] + point["ua"]
17
18     if point["page"] == "":
19         point["page"] = "https://jscrambler.com"
20
21     self.lp = point["page"]
22     self.pc = 1
23
24     # Total Time Script has been running
25     self.totalTime = point["timeElapsed"]
26     # Total no. of focus shifts
27     self.totalFS = point["fs"]
28     # Total no. of Alphanumeric Characters
29     self.totalAlphas = point["alphanumKeys"]
30     # Total no. of Deletes
31     self.totalDels = point["delKeys"]
32     # Other keys
33     self.totalOthers = point["otherKeys"]
34
35     self.totalKeyDowns = point["strikes"]
36
37     rat = self.totalAlphas + self.totalDels + self.totalOthers
38     if (rat < 1):
39         rat = 1
40
41     self.alphaRat = self.totalAlphas/rat
42     self.delRat = self.totalDels/rat
43     self.otherRat = self.totalOthers/rat
44
45     self.totalKeyUps = point["keyPresses"]
46
47     self.strikeStdDev = point["strikeStdDev"]
48     self.strikeEntropy = point["strikeEntropy"]
49     self.strikeSpeed = point["strikeSpeed"]
50     self.strikeAcel = point["strikeAcel"]
51     self.avgFlightTime = point["avgFlightTime"]
52     self.avgKeyHold = point["avgKeyHold"]
53
54     self.totalCuts = point["cuts"]
55     self.totalCpys = point["cpys"]
56     self.totalPastes = point["pastes"]
57
58     self.totalTouches = point["touch"]["touches"]
59     self.totalClick0 = point["buttons"]["0"]["clicks"]
60     self.totalClick1 = point["buttons"]["1"]["clicks"]
61     self.totalClick2 = point["buttons"]["2"]["clicks"]
62
63     try:
64         self.totalClick3 = point["buttons"]["3"]["clicks"]

```

```

65         self.totalClick4 = point["buttons"]["4"]["clicks"]
66     except KeyError:
67         self.totalClick3 = 0
68         self.totalClick4 = 0
69
70
71     self.lastDate = point["date"]
72
73     self.lastMouseX = point["mouseM"]["totalX"]
74     self.lastMouseY = point["mouseM"]["totalY"]
75     self.lastMouseSX = point["mouseM"]["speedX"]
76     self.lastMouseSY = point["mouseM"]["speedY"]
77     self.lastMouseAX = point["mouseM"]["acelX"]
78     self.lastMouseAY = point["mouseM"]["acelY"]
79     self.lastMouseJX = point["mouseM"]["jerkX"]
80     self.lastMouseJY = point["mouseM"]["jerkY"]
81     self.lastC0Hold = point["buttons"]["0"]["avgClickHold"]
82     self.lastC1Hold = point["buttons"]["1"]["avgClickHold"]
83     self.lastC2Hold = point["buttons"]["2"]["avgClickHold"]
84     try:
85         self.lastC3Hold = point["buttons"]["3"]["avgClickHold"]
86         self.lastC4Hold = point["buttons"]["4"]["avgClickHold"]
87     except KeyError:
88         self.lastC3Hold = 0
89         self.lastC4Hold = 0
90
91     self.lastTouchX = point["touchM"]["totalX"]
92     self.lastTouchY = point["touchM"]["totalY"]
93     self.lastTouchSX = point["touchM"]["speedX"]
94     self.lastTouchSY = point["touchM"]["speedY"]
95     self.lastTouchAX = point["touchM"]["acelX"]
96     self.lastTouchAY = point["touchM"]["acelY"]
97     self.lastTouchJX = point["touchM"]["jerkX"]
98     self.lastTouchJY = point["touchM"]["jerkY"]
99     self.lastTouchHold = point["touch"]["avgTouchHold"]
100
101     self.lastScrollX = point["scrollM"]["totalX"]
102     self.lastScrollY = point["scrollM"]["totalY"]
103     self.lastScrollSX = point["scrollM"]["speedX"]
104     self.lastScrollSY = point["scrollM"]["speedY"]
105     self.lastScrollAX = point["scrollM"]["acelX"]
106     self.lastScrollAY = point["scrollM"]["acelY"]
107     self.lastScrollJX = point["scrollM"]["jerkX"]
108     self.lastScrollJY = point["scrollM"]["jerkY"]
109
110
111     self.lastClickPresh = point["avgClickPresh"]
112     self.lastTouchPresh = point["avgTouchPresh"]
113     self.lastPointerPresh = point["avgPointerPresh"]

```

```

114
115     self.mobile = point["mobile"]
116     self.reload = point["reload"]
117     self.bckFwd = point["bckFwd"]
118
119
120
121
122     self.lastPacks = {point["page"]:point}
123
124 def isInTimeOut(self, point):
125     dt1 = parser.parse(self.lastDate)
126     dt2 = parser.parse(point["date"])
127
128     dt1_ts = time.mktime(dt1.timetuple())
129     dt2_ts = time.mktime(dt2.timetuple())
130
131     return (dt2_ts > dt1_ts and (int(dt2_ts - dt1_ts)/60) <= 30)
132
133 def isSameSession(self, point):
134     if "sid" in point:
135         key = point["sid"]
136     else:
137         key = point["ip"] + point["ua"]
138
139     return (key == self.key and self.isInTimeOut(point))
140
141 def addPoint(self, point):
142     #Need to update vars... not gonna mess with advanced stats rn -> entropy and
        movement variables
143     lastTime = 0
144     lastFs = 0
145     lastAlphas = 0
146     lastDels = 0
147     lastOthers = 0
148     lastCuts = 0
149     lastCpys = 0
150     lastPastes = 0
151     lastKUs = 0
152     lastKDs = 0
153     lastMobile = False
154     lastReload = False
155     lastBckFwd = False
156     lastTouches = 0
157     lastC0 = 0
158     lastC1 = 0
159     lastC2 = 0
160     lastC3 = 0
161     lastC4 = 0

```

```

162
163     self.pc += 1
164     if point["page"] == "":
165         point["page"] = self.lp
166
167     if point["page"] in self.lastPackts:
168         self.lp = point["page"]
169         lastTime = self.lastPackts[point["page"]]["timeElapsed"]
170         lastFs = self.lastPackts[point["page"]]["fs"]
171         lastAlphas = self.lastPackts[point["page"]]["alphanumKeys"]
172         lastDels = self.lastPackts[point["page"]]["delKeys"]
173         lastOthers = self.lastPackts[point["page"]]["otherKeys"]
174         lastCuts = self.lastPackts[point["page"]]["cuts"]
175         lastCpys = self.lastPackts[point["page"]]["cpys"]
176         lastPastes = self.lastPackts[point["page"]]["pastes"]
177         lastKUs = self.lastPackts[point["page"]]["keyPresses"]
178         lastKDs = self.lastPackts[point["page"]]["strikes"]
179         lastMobile = self.lastPackts[point["page"]]["mobile"]
180         lastReload = self.lastPackts[point["page"]]["reload"]
181         lastBckFwd = self.lastPackts[point["page"]]["bckFwd"]
182         lastTouches = self.lastPackts[point["page"]]["touch"]["touches"]
183         lastC0 = self.lastPackts[point["page"]]["buttons"]["0"]["clicks"]
184         lastC1 = self.lastPackts[point["page"]]["buttons"]["1"]["clicks"]
185         lastC2 = self.lastPackts[point["page"]]["buttons"]["2"]["clicks"]
186         try:
187             lastC3 = point["buttons"]["3"]["clicks"]
188             lastC4 = point["buttons"]["4"]["clicks"]
189         except KeyError:
190             lastC3 = 0
191             lastC4 = 0
192
193     # Accumulated vars
194
195     # Total Time Script has been running
196     self.totalTime += point["timeElapsed"] - lastTime
197     # Total no. of focus shifts
198     self.totalFS += point["fs"] - lastFs
199     # Total no. of Alphanumeric Characters
200     self.totalAlphas += point["alphanumKeys"] - lastAlphas
201     # Total no. of Deletes
202     self.totalDels += point["delKeys"] - lastDels
203     # Other keys
204     self.totalOthers += point["otherKeys"] - lastOthers
205
206     rat = self.totalAlphas + self.totalDels + self.totalOthers
207     if (rat < 1):
208         rat = 1
209
210     self.alphaRat = self.totalAlphas/rat

```

```

211 self.delRat = self.totalDels/rat
212 self.otherRat = self.totalOthers/rat
213
214
215 self.totalCuts += point["cuts"] - lastCuts
216 self.totalCpys += point["cpys"] - lastCpys
217 self.totalPastes += point["pastes"] - lastPastes
218 self.totalKeyDowns += point["strikes"] - lastKDs
219 self.totalKeyUps += point["keyPresses"] - lastKUs
220
221 self.totalTouches += point["touch"]["touches"] - lastTouches
222
223 self.totalClick0 += point["buttons"]["0"]["clicks"] - lastC0
224 self.totalClick1 += point["buttons"]["1"]["clicks"] - lastC1
225 self.totalClick2 += point["buttons"]["2"]["clicks"] - lastC2
226 try:
227     self.totalClick3 += point["buttons"]["3"]["clicks"] - lastC3
228     self.totalClick4 += point["buttons"]["4"]["clicks"] - lastC4
229 except KeyError:
230     self.totalClick3 += 0
231     self.totalClick4 += 0
232
233
234 # Need to mess w/ these stats still maybe...
235 self.strikeStdDev = point["strikeStdDev"]
236 self.strikeEntropy = point["strikeEntropy"]
237 self.avgFlightTime = point["avgFlightTime"]
238 self.avgKeyHold = point["avgKeyHold"]
239
240 # Instant Vars
241 self.lastDate = point["date"]
242 # self.lastMouseMove = point["mouseM"]
243 # self.lastTouchMove = point["touchM"]
244 # self.lastScrollMove = point["scrollM"]
245
246 self.lastMouseX = point["mouseM"]["totalX"]
247 self.lastMouseY = point["mouseM"]["totalY"]
248 self.lastMouseSX = point["mouseM"]["speedX"]
249 self.lastMouseSY = point["mouseM"]["speedY"]
250 self.lastMouseAX = point["mouseM"]["acelX"]
251 self.lastMouseAY = point["mouseM"]["acelY"]
252 self.lastMouseJX = point["mouseM"]["jerkX"]
253 self.lastMouseJY = point["mouseM"]["jerkY"]
254
255 self.lastTouchX = point["touchM"]["totalX"]
256 self.lastTouchY = point["touchM"]["totalY"]
257 self.lastTouchSX = point["touchM"]["speedX"]
258 self.lastTouchSY = point["touchM"]["speedY"]
259 self.lastTouchAX = point["touchM"]["acelX"]

```

```

260     self.lastTouchAY = point["touchM"]["acelY"]
261     self.lastTouchJX = point["touchM"]["jerkX"]
262     self.lastTouchJY = point["touchM"]["jerkY"]
263     self.lastTouchHold = point["touch"]["avgTouchHold"]
264
265     self.lastScrollX = point["scrollM"]["totalX"]
266     self.lastScrollY = point["scrollM"]["totalY"]
267     self.lastScrollSX = point["scrollM"]["speedX"]
268     self.lastScrollSY = point["scrollM"]["speedY"]
269     self.lastScrollAX = point["scrollM"]["acelX"]
270     self.lastScrollAY = point["scrollM"]["acelY"]
271     self.lastScrollJX = point["scrollM"]["jerkX"]
272     self.lastScrollJY = point["scrollM"]["jerkY"]
273
274     self.lastC0Hold = point["buttons"]["0"]["avgClickHold"]
275     self.lastC1Hold = point["buttons"]["1"]["avgClickHold"]
276     self.lastC2Hold = point["buttons"]["2"]["avgClickHold"]
277     try:
278         self.lastC3Hold = point["buttons"]["3"]["avgClickHold"]
279         self.lastC4Hold = point["buttons"]["4"]["avgClickHold"]
280     except KeyError:
281         self.lastC3Hold = 0
282         self.lastC4Hold = 0
283
284
285     self.lastClickPresh = point["avgClickPresh"]
286     self.lastTouchPresh = point["avgTouchPresh"]
287     self.lastPointerPresh = point["avgPointerPresh"]
288     self.strikeSpeed = point["strikeSpeed"]
289     self.strikeAcel = point["strikeAcel"]
290
291     # Boolean Vars
292     self.mobile |= point["mobile"]
293     self.reload |= point["reload"]
294     self.bckFwd |= point["bckFwd"]
295
296     self.lastPacks[point["page"]] = point
297
298     def toNumpyArray(self):
299         #arr = np.array([self.totalTime, self.totalFS])
300         a = []
301         for key in self.__dict__.keys():
302             if key == "mobile" or key == "reload" or key == "bckFwd":
303                 a.append(int(self.__dict__[key]))
304             elif key != "key" and key != "lastDate" and key != "lastPacks" and key != "
                 lp":
305                 a.append(self.__dict__[key])
306
307         b = np.array(a, dtype=float)

```

```
308     b[np.isnan(b)] = 0
309     return b
310
311     #When we save data to file we gon convert it to numpy arrays and store them in a
    csv file for l8r retrieval
```

preprocessor.py

```
1 import sys
2 import time
3 import joblib
4
5 import numpy as np
6
7 from dateutil import parser
8 from session import Session
9 from dataloader import loadJSONFromDirS, loadJSONFromDir
10
11 c = {}
12
13
14 datasetDir = "crawlers/" + "#d0/"
15
16 datacsv = "crawlers/crawlers.csv"
17
18 data = loadJSONFromDir(datasetDir)
19
20
21 createDataSet = False
22 if ("create" in sys.argv):
23     createDataSet = True
24
25 if createDataSet == True:
26     seshCount = 0
27     npArray = []
28     for point in data:
29         key = ""
30         if "page" not in point:
31             point["page"] = ""
32         if "sid" in point:
33             key = point["sid"]
34         else:
35             key = point["ip"] + point["ua"]
36
37         if not (key in c):
38             c[key] = Session(point)
```



```

39     npArray.append(c[key].toNumpyArray())
40     seshCount += 1
41
42     else:
43         if (c[key].isSameSession(point)):
44             c[key].addPoint(point)
45             npArray.append(c[key].toNumpyArray())
46
47         else:
48             sesh = Session(point)
49             seshCount += 1
50             c[key] = sesh
51             npArray.append(c[key].toNumpyArray())
52
53     np.random.shuffle(npArray)
54     np.savetxt(datacsv, npArray, delimiter=", ")
55
56     print("Session count: " + str(seshCount))
57     print(len(npArray))
58
59 else:
60     contents = np.loadtxt(datacsv, delimiter=", ")
61     print(contents)

```

The code for training the detectors can be seen below:

detector.py

```

1
2 import re
3 import sys
4 import joblib
5 import time
6 import numpy as np
7
8
9 from session import Session
10 from dataloader import loadJSONFromDir
11 from pyod.models.auto_encoder import AutoEncoder
12 from pyod.models.iforest import IForest
13 from pyod.models.lof import LOF
14 from sklearn.preprocessing import StandardScaler
15
16 reStr = ".*Bot.*"
17
18 reStr2 = "LinguineBot"
19

```

```

20 rng = np.random.RandomState(42)
21
22 X_humans = np.loadtxt("humans/humans.csv", delimiter=", ")
23 np.random.shuffle(X_humans)
24
25 X_humans = X_humans[0:522]
26 X_bots = np.loadtxt("crawlers/crawlers.csv", delimiter=", ")
27
28 # scaler = joblib.load("Standard.scaler")
29 # X_train = scaler.transform(X_train)
30 # X_humans = scaler.transform(X_humans)
31 # X_bots = scaler.transform(X_bots)
32
33 #joblib.dump(scaler, "Standard.scaler")
34
35 test = False
36
37 if ("test" in sys.argv and not "train" in sys.argv):
38     test = True
39
40 train = False
41 if ("train" in sys.argv and not "test" in sys.argv):
42     train = True
43
44 if train:
45
46     print("Training")
47     # Generate data
48     X_train = np.loadtxt("d0/d0train.csv", delimiter=", ")
49     # fit the model 69 estims got better as of n w/ max_samples = 2048 IsoForest
50     # , loss='binary_crossentropy'
51     # for AE the best model so far has hidden_neurons=[29, 14, 14, 29]
52     #clf = AutoEncoder(contamination=0.25, batch_size=2048, epochs=65, random_state=
53         rng).fit(X_train)
54     clf = IForest(max_samples=256, verbose=1, random_state=rng, n_estimators=75,
55         n_jobs=-1, contamination=0.1).fit(X_train)
56     #clf = LOF(contamination=0.25, n_jobs=-1).fit(X_train)
57     joblib.dump(clf, "detectorIFTest3.pkl")
58     print("Training Complete")
59     print("Testing")
60     total = 0
61     pp = 0
62     pn = 0
63     tp = 0
64     tn = 0
65     fp = 0
66     fn = 0
67
68     y_pred_test = clf.predict(X_humans)

```

```
67     for i in range(0, len(y_pred_test)):
68         total += 1
69         if y_pred_test[i] == 1:
70             fp += 1
71             pp += 1
72         else:
73             tn += 1
74             pn += 1
75
76     y_pred_test = clf.predict(X_bots)
77     for i in range(0, len(y_pred_test)):
78         total += 1
79         if y_pred_test[i] == 1:
80             tp += 1
81             pp += 1
82         else:
83             fn += 1
84             pn += 1
85
86
87     print("Total: " + str(total))
88     print("PP: " + str(pp))
89     print("TP: " + str(tp))
90     print("FP: " + str(fp))
91     print("PN: " + str(pn))
92     print("TN: " + str(tn))
93     print("FN: " + str(fn))
94     print("ACC: " + str((tp + tn)/total))
95     print("F1: " + str((tp*2)/(2*tp + fp + fn)))
96
97
98 elif test:
99     clf = joblib.load("detectorIFTTest2.pkl")
100
101     total = 0
102     pp = 0
103     pn = 0
104     tp = 0
105     tn = 0
106     fp = 0
107     fn = 0
108
109     y_pred_test = clf.predict(X_humans)
110     for i in range(0, len(y_pred_test)):
111         total += 1
112         if y_pred_test[i] == 1:
113             fp += 1
114             pp += 1
115         else:
```

```

116         tn += 1
117         pn += 1
118
119     y_pred_test = clf.predict(X_bots)
120     for i in range(0, len(y_pred_test)):
121         total += 1
122         if y_pred_test[i] == 1:
123             tp += 1
124             pp += 1
125         else:
126             fn += 1
127             pn += 1
128
129
130     print("Total: " + str(total))
131     print("PP: " + str(pp))
132     print("TP: " + str(tp))
133     print("FP: " + str(fp))
134     print("PN: " + str(pn))
135     print("TN: " + str(tn))
136     print("FN: " + str(fn))
137     print("ACC: " + str((tp + tn)/total))
138     print("F1: " + str((tp*2)/(2*tp + fp + fn)))

```

extractor.py

This file is responsible for extracting either Bots marked by a predetermined UA, or Humans of a given GID cookie value.

```

1 import re
2 import sys
3 import json
4 from dataloader import loadJSONFromDirS
5
6 uaRe = "(SpagetBot|RavioliFunghiBot|PizzaBot|RavioliPestoBot|InterBot|InterBot2)$"
7
8 datasetDir = "dtest/"
9
10 lf = ""
11
12 if ("crawlers" in sys.argv):
13     lf = "crawlers"
14 elif ("humans" in sys.argv):
15     lf = "humans"
16
17

```

```

18 data = loadJSONFromDirS(datasetDir)
19 if lf == "crawlers":
20     botData = []
21
22     for point in data:
23         if re.search(uaRe, point["ua"]):
24             botData.append(point)
25
26     df = open( datasetDir + "crawlers.json", "w")
27     json.dump(botData, df, indent=4)
28
29 elif lf == "humans":
30     idFile = open("ids.csv", "r")
31     ids = idFile.read().split(",")
32     idFile.close()
33     usedIds = []
34     hoomandata = []
35     for point in data:
36         if "sid" in point:
37             for gid in ids:
38                 if point["sid"] == gid:
39                     hoomandata.append(point)
40                 if not gid in usedIds:
41                     usedIds.append(gid)
42     print(usedIds)
43     df = open( datasetDir + "humans.json", "w")
44     json.dump(hoomandata, df, indent=4)

```

Interaction Flow Extractor

The process of extracting interaction flows was split in two scripts. These must be run in order, as the first extracts interactions from all sessions, and the second counts the occurrences of a set of interactions in all other sets.

interactionExtractor.py

```

1 import re
2 import sys
3 import json
4
5 from itertools import groupby
6 from session import Session
7 from dataloader import loadJSONFromDir
8
9 reStr = "(C|c)rawler|(P|p)asser|Bot).*"
10 datasetDir = "d0/"

```

```

11 data = loadJSONFromDir(datasetDir)
12
13 c = {}
14 seshCount = 0
15
16 interSeqs = {}
17
18 sequenceFile = open("seq2.txt", "w")
19
20 for point in data:
21     if re.search(reStr, point["ua"]):
22         continue
23     key = ""
24     if "page" not in point:
25         continue
26     if "sid" in point:
27         key = point["sid"]
28     else:
29         key = point["ip"] + point["ua"]
30
31     if not (key in c):
32         c[key] = Session(point)
33         interSeqs[key] = "Nav:" + point["page"] + ", "
34         seshCount += 1
35
36     else:
37         if (c[key].isSameSession(point)):
38             actions = ""
39             if (point["page"] != c[key].lp):
40                 actions += "Nav:" + point["page"] + " "
41             else:
42                 if (point["mouseM"]["totalX"] != c[key].lastMouseX or point["mouseM"]["totalY"] != c[key].lastMouseY):
43                     actions += "MoveMouse "
44                 if (point["scrollM"]["totalX"] != c[key].lastScrollX or point["scrollM"]["totalY"] != c[key].lastScrollY):
45                     actions += "MoveScroll "
46                 if (point["touchM"]["totalX"] != c[key].lastTouchX or point["touchM"]["totalY"] != c[key].lastTouchY):
47                     actions += "MoveTouch "
48
49             lAs = c[key].totalAlphas
50
51             lDs = c[key].totalDels
52
53             lc0 = c[key].totalClick0
54             lc1 = c[key].totalClick1
55             lc2 = c[key].totalClick2
56             c[key].addPoint(point)

```

```

57
58     if (c[key].totalAlphas > 1As):
59         actions += "Type "
60     if (c[key].totalDels > 1Ds):
61         actions += "DEL "
62
63     if (lc0 > c[key].totalClick0):
64         actions += "Click0 "
65     if (lc1 > c[key].totalClick1):
66         actions += "Click1 "
67     if (lc2 > c[key].totalClick2):
68         actions += "Click2 "
69
70     if not (not actions) and actions != " ":
71         actions += point["page"]
72         al = actions.split(',')
73         finalActs = [i[0] for i in groupby(al)]
74         actions = ",".join(finalActs)
75         interSeqs[key] += ", " + actions
76
77     else:
78         sesh = Session(point)
79         seshCount += 1
80         c[key] = sesh
81         # pagesB4 = interSeqs[key].split(",")
82         # pagesAft = []
83         # for i in range(0, len(pagesB4) - 1):
84         #     if pagesB4[i] != pagesB4[i + 1]:
85         #         pagesAft.append(pagesB4[i])
86
87         # pages = " -> ".join(pagesAft)
88
89         #if not (not pages):
90             tw = interSeqs[key].replace(", ", ", ")
91             sequenceFile.write(tw + "\n")
92
93         interSeqs[key] = "Nav:" + point["page"] + ", "

```

interactionCounter.py

```

1 import json
2
3 flows = []
4
5 fcFile = open("seqcount1.json", "w")
6

```

```

7 flowFile = open("seq2.txt", "r")
8 flows = flowFile.readlines()
9
10 flowCounter = {}
11
12 for i in range(0, len(flows)):
13     flow = flows[i].strip()
14     if flow in flowCounter:
15         continue
16     else:
17         count = 0
18         for j in range(i, len(flows)):
19             if flow in flows[j]:
20                 count += 1
21
22         if count > 99:
23             flowCounter[flow] = count
24
25 json.dump(flowCounter, fcFile)

```

The code listing for the experiments ran can be seen in these next sections.

expl.py

```

1 # https://scikit-learn.org/stable/modules/generated/sklearn.metrics.RocCurveDisplay
  .html#sklearn.metrics.RocCurveDisplay.from_predictions
2
3 import joblib
4 import numpy as np
5 import matplotlib.pyplot as plt
6
7 from session import Session
8 from dataloader import loadJSONFromDir
9
10 from sklearn.metrics import f1_score
11 from sklearn.metrics import recall_score
12 from sklearn.metrics import accuracy_score
13 from sklearn.metrics import precision_score
14 from sklearn.metrics import balanced_accuracy_score
15
16 from sklearn.metrics import RocCurveDisplay
17 from sklearn.metrics import ConfusionMatrixDisplay
18 from sklearn.metrics import PrecisionRecallDisplay
19
20 X_humans = np.loadtxt("humans/humans.csv", delimiter=", ")
21 np.random.shuffle(X_humans)
22 X_humans = X_humans[0:522]

```



```

23
24 X_bots = np.loadtxt("crawlers/crawlers.csv", delimiter=", ")
25
26 Y_humans = np.zeros(522)
27 Y_bots = np.ones(174)
28
29 X_total = np.concatenate((X_humans, X_bots))
30 Y_total = np.concatenate((Y_humans, Y_bots))
31
32 #scaler = joblib.load("Standard.scaler")
33 #X_total = scaler.transform(X_total)
34
35 #https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix
    .html#sklearn.metrics.confusion_matrix
36
37 clf = joblib.load("detectorAE.pkl")
38
39 Y_test = clf.predict(X_total)
40 #print(Y_test[0:250])
41
42 print("Not accounting for class imbalance")
43 print("A:" + str(accuracy_score(Y_total, Y_test)))
44 print("P:" + str(precision_score(Y_total, Y_test)))
45 print("R:" + str(recall_score(Y_total, Y_test)))
46 print("F:" + str(f1_score(Y_total, Y_test)))
47
48 print("Accounting for class imbalance w/ minority of 25%")
49 print("B:" + str(balanced_accuracy_score(Y_total, Y_test)))
50 print("P:" + str(precision_score(Y_total, Y_test, average="macro")))
51 print("R:" + str(recall_score(Y_total, Y_test, average="macro")))
52 print("F:" + str(f1_score(Y_total, Y_test, average="macro")))
53
54
55 #Apresentar dados com e sem macro
56 #ConfusionMatrixDisplay.from_predictions([0, 1, 1, 0], [1, 0, 0, 1], cmap="binary")
57 RocCurveDisplay.from_predictions(Y_total, Y_test, name="AE")
58 ConfusionMatrixDisplay.from_predictions(Y_total, Y_test, cmap="binary")
59 PrecisionRecallDisplay.from_predictions(Y_total, Y_test, name="AE")
60 plt.show()

```

exp2.py

```

1 import sys
2 import time
3 import joblib
4

```

```

5 import numpy as np
6
7 from dateutil import parser
8 from session import Session
9 from dataloader import loadJSONFromDirS, loadJSONFromDir
10
11 c = {}
12 d = {}
13 crawlerKS = []
14 found = []
15
16 clf = joblib.load("detectorIF.pkl")
17
18 datasetDir = "crawlers/"
19
20 data = loadJSONFromDir(datasetDir)
21
22 sc = 0
23 for point in data:
24     key = ""
25     if "page" not in point:
26         point["page"] = ""
27     if "sid" in point:
28         key = point["sid"]
29     else:
30         key = point["ip"] + point["ua"]
31
32     if not (key in c):
33         crawlerKS.append(key)
34         c[key] = Session(point)
35         if clf.predict(c[key].toNumpyArray().reshape(1, -1)) == 1 and c[key].pc > 1:
36             d[key] = c[key].pc
37             found.append(point['ua'] + "PACKET COUNT: " + str(c[key].pc))
38
39     else:
40         if (c[key].isSameSession(point)):
41             c[key].addPoint(point)
42             if clf.predict(c[key].toNumpyArray().reshape(1, -1)) == 1 and not (key in d)
43                 and c[key].pc > 1:
44                 d[key] = c[key].pc
45                 found.append(point['ua'] + "PACKET COUNT: " + str(c[key].pc))
46
47         else:
48             sesh = Session(point)
49             c[key] = sesh
50             if clf.predict(c[key].toNumpyArray().reshape(1, -1)) == 1 and c[key].pc > 1:
51                 d[key] = c[key].pc
52                 found.append(point['ua'] + "PACKET COUNT: " + str(c[key].pc))

```

```

53 #print("Crawler Keys")
54 #print(c)
55
56 missed = []
57 for k in crawlerKS:
58     if not (k in d):
59         missed.append(k)
60
61
62 c = {}
63 d = {}
64 crawlerKS = []
65
66 datasetDir = "humans/"
67
68 data = loadJSONFromDir(datasetDir)
69
70 for point in data:
71     key = ""
72     if "page" not in point:
73         point["page"] = ""
74     if "sid" in point:
75         key = point["sid"]
76     else:
77         key = point["ip"] + point["ua"]
78
79     if not (key in c):
80         crawlerKS.append(key)
81         c[key] = Session(point)
82         if clf.predict(c[key].toNumpyArray().reshape(1, -1)) == 1 and c[key].pc > 1:
83             d[key] = c[key].pc
84             found.append(key + "PACKET COUNT: " + str(c[key].pc))
85
86     else:
87         if (c[key].isSameSession(point)):
88             c[key].addPoint(point)
89             if clf.predict(c[key].toNumpyArray().reshape(1, -1)) == 1 and (not (key in d
90 )) and c[key].pc > 1:
91                 d[key] = c[key].pc
92                 found.append(key + "PACKET COUNT: " + str(c[key].pc))
93
94         else:
95             sesh = Session(point)
96             if (key in d):
97                 d.pop(key)
98             c[key] = sesh
99             if clf.predict(c[key].toNumpyArray().reshape(1, -1)) == 1 and c[key].pc > 1:
100                 d[key] = c[key].pc
101                 found.append(key + "PACKET COUNT: " + str(c[key].pc))

```

```
101
102 correct = []
103 for k in crawlerKS:
104     if not (k in d):
105         correct.append(k)
106
107
108 print(found)
```

A.3 Bots Developed

rawFocusedCrawler.py

A crawler that uses the basic Selenium functions to scrape the web page. Can move mouse, but trajectory is defined by Selenium internally.

```
1 import time
2 from selenium import webdriver
3
4 from selenium.webdriver.support.ui import WebDriverWait
5 from selenium.webdriver.common.by import By
6 from selenium.webdriver import ActionChains
7 from selenium.webdriver.common.keys import Keys
8
9 url = "https://jscrambler.pt"
10
11
12 options = webdriver.FirefoxOptions()
13 options.set_preference("general.useragent.override", "RavioliFunghiBot")
14
15 driver = webdriver.Firefox(options=options)
16 driver.get(url)
17
18 actions = ActionChains(driver)
19
20 cookies = WebDriverWait(driver, 3).until(lambda d: d.find_element(By.ID, "
    CybotCookiebotDialogBodyLevelButtonLevelOptinAllowAll"))
21 products = WebDriverWait(driver, 3).until(lambda d: d.find_element(By.ID, "products
    "))
22
23 productsToView = []
24
25
26 actions.move_to_element(cookies)
27 actions.click()
28 actions.perform()
```

```
29 actions.reset_actions()
30
31 actions.move_to_element(products)
32 actions.perform()
33 actions.reset_actions()
34
35
36 productLinks = WebDriverWait(driver, 3).until(lambda d: d.find_elements(By.
    CLASS_NAME, "linkWrapper--7sh4o"))
37 for p in productLinks:
38     span = WebDriverWait(driver, 3).until(lambda d: p.find_element(By.TAG_NAME, "span
        "))
39     productsToView.append(span.text)
40
41 for p in productsToView:
42     link = WebDriverWait(driver, 10).until(lambda d: d.find_element(By.XPATH, "//*[
        contains(text(),' " + p + "')]))
43     actions.move_to_element(link)
44     actions.click()
45     actions.perform()
46     actions.reset_actions()
47     time.sleep(1)
48     footer = WebDriverWait(driver, 3).until(lambda d: d.find_element(By.TAG_NAME, "
        footer"))
49     driver.execute_script("arguments[0].scrollIntoView()", footer)
50
51     driver.back()
52     products = WebDriverWait(driver, 3).until(lambda d: d.find_element(By.ID, "
        products"))
53     actions.move_to_element(products)
54     actions.perform()
55     actions.reset_actions()
56     time.sleep(1)
57
58 res = WebDriverWait(driver, 3).until(lambda d: d.find_element(By.ID, "resources"))
59 actions.move_to_element(res)
60 actions.click()
61 actions.perform()
62 actions.reset_actions()
63 footer = WebDriverWait(driver, 3).until(lambda d: d.find_element(By.TAG_NAME, "
        footer"))
64 driver.execute_script("arguments[0].scrollIntoView()", footer)
65 time.sleep(1)
66
67 par = WebDriverWait(driver, 3).until(lambda d: d.find_element(By.ID, "partners"))
68 actions.move_to_element(par)
69 actions.click()
70 actions.perform()
71 actions.reset_actions()
```

```
72 footer = WebDriverWait(driver, 3).until(lambda d: d.find_element(By.TAG_NAME, "
    footer"))
73 driver.execute_script("arguments[0].scrollIntoView()", footer)
74 time.sleep(1)
75
76 time.sleep(3)
77 driver.quit()
```

rawPasswordTryer.py

A bot that simulates a password cracker using only basic Selenium functionality.

```
1 import time
2
3 from selenium import webdriver
4
5 from selenium.webdriver.support.ui import WebDriverWait
6 from selenium.webdriver.common.by import By
7 from selenium.webdriver import ActionChains
8 from selenium.webdriver.common.keys import Keys
9
10 import random, string
11
12 def randomword(length):
13     letters = string.ascii_lowercase
14     return ''.join(random.choice(letters) for i in range(length))
15
16
17
18 url = "https://jscrambler.com/login"
19
20
21 options = webdriver.FirefoxOptions()
22 options.set_preference("general.useragent.override", "RavioliPestoBot")
23
24 driver = webdriver.Firefox(options=options)
25 driver.get(url)
26
27 cookies = WebDriverWait(driver, 3).until(lambda d: d.find_element(By.ID, "
    CybotCookiebotDialogBodyLevelButtonLevelOptinAllowAll"))
28 cookies.click()
29
30 uname = WebDriverWait(driver, 3).until(lambda d: d.find_element(By.ID, "username"))
31 passwd = WebDriverWait(driver, 3).until(lambda d: d.find_element(By.ID, "password")
    )
32
33 unameStr = "fake@email.com"
```

```

34 uname.send_keys(unameStr)
35
36 for i in range(10):
37     passwdStr = randomword(10)
38     passwd.clear()
39     passwd.send_keys(passwdStr)
40     WebDriverWait(driver, 3).until(lambda d: d.find_element(By.ID, "formSubmitButton"
41         )).click()
42     time.sleep(2)
43
44
45 driver.quit()

```

windmouse.py

Code adapted from [Land, 2021], to be integrated with Selenium to generate human like, mouse trajectories.

```

1 import numpy as np
2 sqrt3 = np.sqrt(3)
3 sqrt5 = np.sqrt(5)
4
5 def windmouse(start_x, start_y, dest_x, dest_y, G_0=9, W_0=3, M_0=15, D_0=12,
6     move_mouse=lambda x,y: None):
7     '''
8     WindMouse algorithm. Calls the move_mouse kwarg with each new step.
9     Released under the terms of the GPLv3 license.
10    G_0 - magnitude of the gravitational force
11    W_0 - magnitude of the wind force fluctuations
12    M_0 - maximum step size (velocity clip threshold)
13    D_0 - distance where wind behavior changes from random to damped
14    '''
15    current_x, current_y = start_x, start_y
16    v_x = v_y = W_x = W_y = 0
17    while (dist:=np.hypot(dest_x-start_x, dest_y-start_y)) >= 1:
18        W_mag = min(W_0, dist)
19        if dist >= D_0:
20            W_x = W_x/sqrt3 + (2*np.random.random()-1)*W_mag/sqrt5
21            W_y = W_y/sqrt3 + (2*np.random.random()-1)*W_mag/sqrt5
22        else:
23            W_x /= sqrt3
24            W_y /= sqrt3
25            if M_0 < 3:
26                M_0 = np.random.random()*3 + 3
27            else:
28                M_0 /= sqrt5

```

```

28     v_x += W_x + G_0*(dest_x-start_x)/dist
29     v_y += W_y + G_0*(dest_y-start_y)/dist
30     v_mag = np.hypot(v_x, v_y)
31     if v_mag > M_0:
32         v_clip = M_0/2 + np.random.random()*M_0/2
33         v_x = (v_x/v_mag) * v_clip
34         v_y = (v_y/v_mag) * v_clip
35     start_x += v_x
36     start_y += v_y
37     move_x = int(np.round(start_x))
38     move_y = int(np.round(start_y))
39     if current_x != move_x or current_y != move_y:
40         #This should wait for the mouse polling interval
41         move_mouse(current_x:=move_x, current_y:=move_y)
42     return current_x, current_y

```

three.py

Module containing the functions to simulate human behaviour on a web page.

```

1  import time
2  import random
3
4  import windmouse
5
6  from selenium import webdriver
7  from selenium.webdriver.support.ui import WebDriverWait
8  from selenium.webdriver.common.by import By
9  from selenium.webdriver import ActionChains
10 from selenium.webdriver.common.keys import Keys
11
12 options = webdriver.FirefoxOptions()
13 options.set_preference("general.useragent.override", "InterBot2")
14
15 driver = webdriver.Firefox(options=options)
16
17 path = []
18 mouse = 0
19 scroll = 0
20 actions = ActionChains(driver, duration=5)
21
22 def mimicHumanType(chain, str):
23     for char in str:
24         ttw = random.randrange(1, 100)/100 #need to tune this probably
25         chain.pause(ttw)
26         chain.send_keys(char)
27

```



```

28 def mimicHumanPath(xoffset, yoffset):
29     if xoffset < 0: xoffset = 0
30
31     if xoffset > int(driver.get_window_size().get("width")): xoffset = int(driver.
        get_window_size().get("width")) - 1
32
33     if yoffset < 0: yoffset = 0
34
35     if yoffset > int(driver.get_window_size().get("height")): yoffset = int(driver.
        get_window_size().get("height")) - 1
36
37     path.append([xoffset, yoffset])
38
39 def mimicHumanMouseMove(chain, x_to, y_to):
40     path.clear()
41     mouX = 0
42     mouY = 0
43     try:
44         (mouX, mouY) = getMouse()
45     except Exception:
46         injectMouse()
47         (mouX, mouY) = getMouse()
48     windmouse.windmouse(mouX, mouY, x_to, y_to, move_mouse=mimicHumanPath)
49     # print(mouX)
50     # print(mouY)
51     for point in path:
52         #print("In loop")
53         offX = point[0] - mouX
54         if mouX + offX < 0:
55             offX = 0
56         offY = point[1] - mouY
57         if mouY + offY < 0:
58             offY = 0
59         # print(str(mouX) + ":" + str(mouY))
60         # print(str(offX) + ":" + str(offY))
61         chain.move_by_offset(offX, offY)
62         (mouX, mouY) = point
63
64 def mimicHumanMouseMoveToElem(element):
65     path.clear()
66     x_to = int(element.location.get("x"))
67     y_to = int(element.location.get("y"))
68
69     # if x_to == 0: x_to = 20
70     # if x_to >= int(driver.get_window_size().get("width")): x_to = int(driver.
        get_window_size().get("width")) - 20
71
72     # if y_to == 0: y_to = 20

```

```

73 # if y_to >= int(driver.get_window_size().get("height")): y_to = int(driver.
    get_window_size().get("height")) - 20
74 print(element.rect)
75 print("Where to: " + str(x_to) + ":" + str(y_to))
76 mimicHumanMouseMove(actions, x_to, y_to)
77 actions.move_to_element(element)
78
79 # def setMouse(x, y):
80 #     driver.execute_script("""
81 #         const element = document.getElementById()
82 #         """)
83
84 # To use the windmouse code, we need the current mouse position, so we inject this
    hacky div to keep it
85 # https://stackoverflow.com/questions/15510882/selenium-get-coordinates-or-
    dimensions-of-element-with-python -Z this gets the coords to where we wanna go
86 def injectMouse():
87     driver.execute_script("""
88         const element = document.createElement('div');
89         element.id = "mymouse"
90         document.body.appendChild(element);
91         onmousemove = (e) => {
92             element.innerHTML = "" + e.clientX + ":" + e.clientY;
93             console.log(element.innerHTML);
94         };
95         onclick = (e) => {
96             console.log("AAAA" + e.clientX + ":" + e.clientY);
97         };
98         """)
99     act = ActionChains(driver)
100     act.move_by_offset(driver.get_window_size().get("width")/2, driver.
        get_window_size().get("height")/2)
101     act.perform()
102     del act
103
104 def getMouse():
105     mouse = WebDriverWait(driver, 3).until(lambda d: d.find_element(By.ID, "mymouse")
        )
106     return (int(mouse.text.split(":")[0]), int(mouse.text.split(":")[1]))
107
108 def injectScroll():
109     driver.execute_script("""
110         const scrollElem = document.createElement('div');
111         scrollElem.id = "myscroll";
112         scrollElem.innerHTML = "" + document.body.scrollTop + ":" + document.body.
            scrollLeft;
113         document.body.appendChild(scrollElem);
114         onscroll = (e) => { scrollElem.innerHTML = "" + document.body.scrollTop + ":" +
            document.body.scrollLeft; }

```

```

115     "")
116
117 def getScroll():
118     scroll = WebDriverWait(driver, 3).until(lambda d: d.find_element(By.ID, "myscroll"))
119     return (int(scroll.text.split(":")[0]), int(scroll.text.split(":")[1]))
120
121 def microScroll(x, y):
122     driver.execute_script("""scrollTo({
123     top:"" + str(y) + "",
124     left:"" + str(x) + "",
125     behavior: 'smooth'
126     })""")
127
128 def mimicHumanScroll(element):
129     x_to = int(element.location.get("x"))
130     y_to = int(element.location.get("y"))
131     x = 0
132     y = 0
133     try:
134         (x, y) = getScroll()
135     except Exception:
136         injectScroll()
137         (x, y) = getScroll()
138
139     while x != x_to or y != y_to:
140         mult_x = 0
141         mult_y = 0
142         add_x = 0
143         add_y = 0
144         if abs(x_to - x) > 400:
145             add_x = random.randrange(150, 400)
146         else:
147             add_x = random.randrange(2, 20)
148
149         if abs(y_to - y) > 400:
150             add_y = random.randrange(150, 400)
151         else:
152             add_y = random.randrange(2, 20)
153         if x < x_to:
154             mult_x = 1
155         if x > x_to:
156             mult_x = -1
157         if y < y_to:
158             mult_y = 1
159         if y > y_to:
160             mult_y = -1
161
162         if abs(x_to - x) < add_x:

```

```
163         add_x = abs(x_to - x)
164
165         if abs(y_to - y) < add_y:
166             add_y = abs(y_to - y)
167
168         x = x + (mult_x*add_x)
169         y = y + (mult_y*add_y)
170         microScroll(x, y)
171         time.sleep(random.randrange(0, 3)/15)
```

cookedFocusedCrawler.py

Advanced, stealthier scraper that uses the three.py module.

```
1 import three
2 from selenium.common.exceptions import MoveTargetOutOfBoundsException
3 from selenium.webdriver.support.ui import WebDriverWait
4 from selenium.webdriver.common.by import By
5 from selenium.webdriver.common.keys import Keys
6
7 from selenium.webdriver import ActionChains
8
9 import time
10
11 url = "https://jscrambler.pt"
12
13 three.driver.get(url)
14
15 print("Accepting cookies")
16 cookies = WebDriverWait(three.driver, 3).until(lambda d: d.find_element(By.ID, "
    CybotCookiebotDialogBodyLevelButtonLevelOptinAllowAll"))
17 three.mimicHumanMouseMoveToElem(cookies)
18 three.actions.click()
19 three.actions.perform()
20 three.actions.reset_actions()
21
22
23 print("Move to Products, hover")
24 products = WebDriverWait(three.driver, 3).until(lambda d: d.find_element(By.ID, "
    products"))
25
26 (mX, mY) = three.getMouse()
27 three.actions.move_by_offset(mX, mY)
28 three.mimicHumanMouseMoveToElem(products)
29 three.actions.perform()
30 time.sleep(3)
31 three.actions.reset_actions()
```

```
32
33 print("Go to resources")
34 (mX, mY) = three.getMouse()
35 three.actions.move_by_offset(mX, mY)
36
37 res = WebDriverWait(three.driver, 3).until(lambda d: d.find_element(By.ID, "
    resources"))
38 three.mimicHumanMouseMoveToElem(res)
39 three.actions.click()
40 three.actions.perform()
41 three.actions.reset_actions()
42 footer = WebDriverWait(three.driver, 3).until(lambda d: d.find_element(By.TAG_NAME,
    "footer"))
43 three.mimicHumanScroll(footer)
44 (mX, mY) = three.getMouse()
45 print((mX, mY))
46 three.actions.reset_actions()
47 three.actions.move_by_offset(mX, mY)
48 three.actions.reset_actions()
49 time.sleep(2)
50
51 print("Go to partners")
52 print(three.actions)
53 three.driver.back()
54 three.actions.reset_actions()
55 par = WebDriverWait(three.driver, 3).until(lambda d: d.find_element(By.ID, "
    partners"))
56
57 print("Set up mouse")
58 three.injectMouse()
59 (mX, mY) = three.getMouse()
60 print((mX, mY))
61 three.actions.reset_actions()
62 three.actions.move_by_offset(mX, mY)
63
64 print(three.actions)
65 print(par.location)
66 three.mimicHumanMouseMoveToElem(par)
67 three.actions.click()
68 three.actions.perform()
69 three.actions.reset_actions()
70 footer = WebDriverWait(three.driver, 3).until(lambda d: d.find_element(By.TAG_NAME,
    "footer"))
71 three.mimicHumanScroll(footer)
72 three.actions.reset_actions()
73 time.sleep(1)
74
75
76 time.sleep(3)
```

```
77 three.driver.quit()
```

cookedPasswordTryer.py

Password cracker using the advanced mimicking functions of three.py.

```
1 import three
2 from selenium.webdriver.support.ui import WebDriverWait
3 from selenium.webdriver.common.by import By
4 from selenium.webdriver.common.keys import Keys
5
6 import time
7
8 import random, string
9
10 def randomword(length):
11     letters = string.ascii_lowercase
12     return ''.join(random.choice(letters) for i in range(length))
13
14 url = "https://jscrambler.pt"
15
16 three.driver.get(url)
17
18 print("Accepting Cookies")
19 cookies = WebDriverWait(three.driver, 3).until(lambda d: d.find_element(By.ID, "
    CybotCookiebotDialogBodyLevelButtonLevelOptinAllowAll"))
20 three.mimicHumanMouseMoveToElem(cookies)
21 three.actions.click()
22 three.actions.perform()
23 three.actions.reset_actions()
24 time.sleep(3)
25
26 (mX, mY) = three.getMouse()
27 three.actions.move_by_offset(mX, mY)
28 print("Clicking Login")
29 login = WebDriverWait(three.driver, 3).until(lambda d: d.find_element(By.LINK_TEXT,
    "Login"))
30 three.mimicHumanMouseMoveToElem(login)
31 three.actions.click()
32 three.actions.perform()
33 three.actions.reset_actions()
34 time.sleep(2)
35
36 uname = WebDriverWait(three.driver, 3).until(lambda d: d.find_element(By.ID, "
    username"))
37 passwd = WebDriverWait(three.driver, 3).until(lambda d: d.find_element(By.ID, "
    password"))
```

```
38
39 btn = WebDriverWait(three.driver, 3).until(lambda d: d.find_element(By.ID, "
    formSubmitButton"))
40
41 (mX, mY) = three.getMouse()
42 three.actions.move_by_offset(mX, mY)
43 print("Typing Uname")
44 unameStr = "fake@email.com"
45 three.mimicHumanMouseMoveToElem(uname)
46 three.actions.click()
47 three.mimicHumanType(three.actions, unameStr)
48 three.actions.perform()
49 three.actions.reset_actions()
50
51 for i in range(5):
52
53     (mX, mY) = three.getMouse()
54     three.actions.move_by_offset(mX, mY)
55
56     passwdStr = randomword(i+10)
57     three.mimicHumanMouseMoveToElem(passwd)
58     three.actions.click()
59     three.mimicHumanType(three.actions, passwdStr)
60     three.actions.perform()
61     three.actions.reset_actions()
62
63     (mX, mY) = three.getMouse()
64     three.actions.move_by_offset(mX, mY)
65     three.mimicHumanMouseMoveToElem(btn)
66     three.actions.click()
67     three.actions.perform()
68     three.actions.reset_actions()
69
70     (mX, mY) = three.getMouse()
71     three.actions.move_by_offset(mX, mY)
72     three.mimicHumanMouseMoveToElem(passwd)
73     three.actions.click()
74     three.actions.perform()
75     three.actions.reset_actions()
76
77     while passwd.get_attribute("value") != "":
78         passwd.send_keys(Keys.BACK_SPACE)
79         j = random.randrange(1, 5)/100
80         time.sleep(j)
81
82     three.actions.reset_actions()
83
84 time.sleep(15)
85 three.driver.quit()
```
