MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Střasák František**	Personal ID number:	**420189**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Computer Science**		
Study program:	**Open Informatics**		
Branch of study:	**Artificial Intelligence**		

II. Master’s thesis details

Master’s thesis title in English:

Should I click on a link? Machine Learning to Protect from Cyber Attacks on the Web

Master’s thesis title in Czech:

Strojové učení k ochraně před kybernetickými útoky na webu

Guidelines:

1. Review the state-of-the art methods for detection malicious websites and links.
2. Find available datasets and also create own datasets for this problem.
3. Propose and implement new features and methods for detection malicious websites and links with special attention to Deep learning methods.
4. Experimentally evaluate the proposed solution on real malicious and normal website and attacks on links datasets.
5. Create a web service for the general public to test if some URL is malicious or not.
6. Critically analyze the results and propose further extensions of the solution with respect to its applicability.

Bibliography / sources:

[1] Graph Neural Networks: A Review of Methods and Applications; Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, Maosong Sun; 2019
[2] Framework for Malicious HTML File Detection; Samuel Hess, School of Electrical and Computer Engineering, University of Arizona; 2017
[3] Malicious web content detection by machine learning; Yung-Tsung Hou, Yimeng Chang, Tsuhan Chen , Chi-Sung Laih , Chia-Mei Chen; 2010
[4] Learning detectors of malicious web requests for intrusion detection in network traffic; Lukas Machlica, Karel Bartos, Michal Sofka; 2017
[5] Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering; Michaël Defferrard Xavier Bresson Pierre Vandergheynst; 2017

Name and workplace of master’s thesis supervisor:

Ing. Sebastián García, Ph.D., Artificial Intelligence Center, FEE

Name and workplace of second master’s thesis supervisor or consultant:

Date of master’s thesis assignment: **03.07.2019** Deadline for master's thesis submission: **22.05.2020**

Assignment valid until: **19.02.2021**

Ing. Sebastián García, Ph.D. Supervisor’s signature	Head of department’s signature	prof. Mgr. Petr Páta, Ph.D. Dean’s signature

© ČVUT v Praze, Design: ČVUT v Praze, VIC

III. Assignment receipt

The student acknowledges that the master’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master’s thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature

CVUT-CZ-ZDP-2015.1 © ČVUT v Praze, Design: ČVUT v Praze, VIC

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science and Engineering

Master's thesis

Should I click on a link? Machine Learning to Protect from Cyber Attacks on the Web

Bc. František Střasák

Supervisor: Ing. Sebastián García, Ph.D.

Study Programme: Open Informatics

Field of Study: Artificial Intelligence

May 22, 2020

Aknowledgements

I would like to express my very great appreciation to Ing. Sebastián García, Ph.D for the patience and for his valuable and constructive suggestions during the development of this thesis. I would also like to extend my thanks to all members of Stratosphere team for the help with writing this thesis. I am particularly grateful for the assistance given by Johannes Gilger for consultation about urlscan.io and Angelo Dell'Aera for consultation about Thug tool. Finally, I wish to thank to Bára, my parents and rest of the family for their support and encouragement throughout my study.

Declaration

I declare that I elaborated this thesis on my own and that I mentioned all the information sources and literature that have been used in accordance with the Guideline for adhering to ethical principles in the course of elaborating an academic final thesis.

In Prague on May 22, 2020 ....

Abstract

The detection of unsafe websites poses a challenging task for our security community because their attacking techniques are varied, advanced and dangerous. There are many types of unsafe websites that can infect user's devices or steal their sensitive data. The most prevalent representative type of unsafe websites are the *evil twin websites* that use phishing techniques to steal sensitive data and credentials from users. Evil twin websites are clone websites imitating other real websites to trick users into using them. Therefore, users judging the authenticity of a website by its look, can be defrauded by inputting sensitive information in the evil twin website. To detect these unsafe websites, previous studies have mainly used blacklists, but they constant updates when a new URL appears. This results in the approach not protecting from the new and current threats. Another common solution is to detect the website by analyzing the URL string, which may shows satisfying results under certain conditions. However, the complexity of domain names and URL parameters makes this approach to have errors also. Since websites offer much more information than only a URL, this thesis proposes novel methods to detect unsafe and evil twin websites based on the analysis of the behavior, content, and structure of websites. The structure refers to the HTML structure, the content and the behavior refer a large group of features extracted from the urlscan.io service that provides a complex description of websites. To fulfil its goal of better detecting unsafe websites, this thesis is mainly separated in two parts. The first part focuses on the detection of unsafe websites in general by using different set of features. The second part of this thesis specifically concentrates on the detection of evil twin websites. For both problems we created and publish our own datasets that can be useful for the whole community. This thesis presents evidence that features from the content, behaviour and structure of websites play an essential role for detecting cyber attacks on the websites. The results show that our models are able to separate between unsafe and legitimate websites with an accuracy of 92.69% and between evil twin websites and legitimate websites with an accuracy of 95.28%. Detecting unsafe websites is a hard topic because they keep evolving, but we believe that this thesis improves the research to detect this threat.

Keywords: Unsafe Websites, Evil Twin Websites, Machine Leaning, Phishing

Abstrakt

Detekce nebezpečných webových stránek dnes představuje velkou výzvu, neboť techniky, které jsou v útocích využívány, jsou velmi rozmanité, pokročilé a nebezpečné. Nebezpečná webová stránka může infikovat uživatelovo zařízení či ukrást jeho citlivá data. Nejrozšířenějším zástupcem nebezpečných stránek jsou tzv. *evil twin* webové stránky, které používají phishingové praktiky ke krádeži citlivých dat. Evil twin webové stránky se snaží co nejvěrněji napodobit vzhled reálné webové stránky, zmást uživatele a přesvědčit ho k zadání citlivých údajů, těmi jsou například přihlašovací údaje. Uživatelé, kteří ověřují autenticitu webové stránky pouze podle vzhledu, mohou být tedy velmi jednoduše podvedeni. Častou technikou k detekci nebezpečných stránek jsou blacklisty, jež obsahují list nebezpečných URL. Problém ale nastává, když se objeví nová URL, která v blacklistu není obsažena, a tím pádem nemůže být detekována. Možným řešením tohoto problému je detekce založená na analýze URL, při jejíž použití bylo v minulosti během několika výzkumů dosaženo uspokojivých výsledků. Webová stránka ale obsahuje více informací než pouze URL. V této práci jsou předkládány nové metody pro detekci nebezpečných a evil twin stránek, jež jsou založeny na analýze chování, obsahu a struktuře webové stránky. Data o chování a obsahu webové stránky jsou získána z analýzy vytvořené pomocí urlscan.io. Tato analýza ukazuje komplexní popis webové stránky v mnoha směrech. Data o struktuře webové stránky jsou brána ze zdrojového kódu HTML. První část této práce je věnována obecně detekci nebezpečných stránek a druhá část je zaměřena pouze na detekci evil twin webových stránek. Pro oba problémy byly vytvořeny datasety, které jsou veřejně přístupné a mohou být použity pro další výzkum. Z výsledků výzkumu této diplomové práce vyplývá, že data založená na obsahu, chování a struktuře webové stránky hrají důležitou roli při detekci kybernetických útoků. Na základě metod této diplomové práce bylo dosaženo přesnosti 92.69% pro detekci nebezpečných stránek a 95.28% pro detekci evil twin stránek.

Klíčová slova: Nebezpečná webová stránka, Evil Twin, Strojové učení, Phishing

Překlad titulu: Strojové učení k ochraně před kybernetickými útoky na webu

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the growth of Internet usage, websites have become part of human life. They are used every day in countless activities facilitating human interaction with the rest of the world. Along with this trend, attackers also create websites with intent to defraud and attack users, they are consider unsafe websites. Unsafe websites comprise a wide variety of websites having malicious content or behavior and therefore, their detection poses a challenging task. According to the Google Safe Browsing category of the Google Transparency Report [1], unsafe websites can be broadly divided into two categories: malware websites and phishing websites.

- **Malware websites** use techniques to install malicious software onto user's devices. This malicious software can be installed without the user's knowledge, but sometimes the user downloads this malicious software because the user thinks it is legitimate.

- **Phishing websites** use techniques to steal user's credentials such as emails, passwords, credit card numbers and other sensitive data. We classify them into two classes: evil twin websites and scam websites. Evil twin websites imitate real websites to steal sensitive data from users. Scam websites offer fake products to users with the aim to steal their sensitive data.

The detection of malware websites and phishing websites is considered a difficult task, because (i) their attacking techniques evolve, (ii) their behaviour is very diverse for different websites, and (iii) normal websites are very complex, which makes the differentiation more difficult [1, 2].

A frequently used solution for detecting unsafe websites is through blacklists. Blacklists rely on a database containing an updated list of malicious URLs. However, this approach fails when a new URL or a variant of an existing malicious URL appears [3]. Another used approach is based on the analysis of the string of the URL. The idea is to split the URL in several parts, extract features from it and then use machine learning algorithms to classify it as malicious or not. State-of-the-art methods achieve satisfying results [4, 5, 6]. However, attackers try to avoid the detection by creating new URLs with properties similar to legitimate URLs.

In contrast to previous approaches based on URL analysis or blacklists, this research proposes novel methods based on the analysis of the behavior, content, and structure of websites. The behaviour and content data are taken from the urlscan.io service [7] that provides a complex analysis of a website. The structure data are taken from the HTML source code of a website. We hypothesize that these data can provide more informative description of unsafe websites and thus achieve satisfying results in this area.

This thesis concentrates on two main areas:

- Detection of unsafe websites
- Detection of evil twin websites

To fulfil these goals, we not only analyze unsafe websites and evil twin websites, but we created our own datasets, and we built an online free service which provides users the possibility to submit any URL and learn if the respective website is safe to visit. Our solution, called <https://shouldiclick.org> is not only useful for all users, but more importantly it is important for people at risk, such as journalists, and human right defenders, that can not afford any other protection.

The first part of this thesis focuses on detecting unsafe websites. It started with the creation of the Unsafe Website Dataset (UWD) that contains two classes: unsafe and legitimate samples. Each sample from the dataset contains the website URL and its analysis from urlscan.io. Using extracted features from the urlscan.io analysis, our machine learning algorithms achieved a detection accuracy of 92.69%.

The second part of this thesis focuses on the more specific problem of detecting evil twin websites. A core part of this experiment was the creation of the Civilsphere Evil Twin Website Dataset (CETWD) containing legitimate and evil twin samples. Each sample from the dataset contains images, JavaScript code, a screenshot of the webpage, the analysis of urlscan.io and the HTML Document Object Model (DOM). This dataset was generated and analysed with the necessary diligence in order to ensure clean data for the machine learning algorithms. In practise it meant that we spent a lot of time finding invalid samples using various techniques described in Section 5.1. The experiment with evil twin websites is split into two phases. The first phase focuses on detection using extracted features from the analysis of urlscan.io service. The second phase concentrates on the detection using the HTML content represented as a graph structure. The method based on the analysis of urlscan.io achieved a testing accuracy of 95.28% and the method based on HTML structure achieved a testing accuracy of 90%.

The contributions of this thesis are:

- Design, creation and publication of new datasets for unsafe websites and evil twin websites containing also legitimate websites.
- Publication of the set of most important features for detection of both unsafe and evil twin websites.
- Creation and publication of the <https://www.shouldiclick.org/> service, using one of the detection methods proposed in this thesis. The goal of the Should I Click service is to allow users to check if a website is safe to access or not.

One of the initial goals of this thesis was also to analyze and detect websites that *directly attack* the user with exploits. These websites can be called *exploiting websites* or *malware websites*. However, during the first phases of this research, while capturing the dataset, we found that it was *extremely* hard to find any working exploiting website because they are alive for days at a time. Therefore, this research line was temporally abandoned. This finding tells us that finding exploiting websites is very difficult. Nevertheless, as part of this thesis we provide a description of method we implemented using the Thug tool [8] that is used in the discovery of exploiting websites.

Since reproducibility and sharing with the community is an important way to advance science, all the code developed for this thesis is available in this GitHub repository [9]

The rest of the thesis is organized as follows. Chapter 2 discusses differences between types of unsafe websites and provides an overview of this problem. Chapter 3 contains an overview of related works. Chapter 4 describes our approach for the detection of unsafe websites including generation and analysis of the UWD dataset and machine learning experiments with it. Chapter 5 describes and analyses the CETD dataset and shows machine learning experiments with this dataset. Chapter 6 discusses exploiting websites and the difficulty to collect a meaningful dataset. Chapter 7 is dedicated to the Should I Click web service, and Chapter 8 presents the conclusions.

Chapter 2

Background

Our research is focused on the detection of unsafe websites and evil twin websites. This chapter explains each one of these threats, and describes the basic categories of unsafe websites together with their differences.

2.1 Unsafe websites

Unsafe website is a general term for all websites with malicious content or behavior. Their techniques are different, but all of them share the same goal: to attack users. Figure 2.1 shows the main categories of unsafe websites based on the Google Safe Browsing category of the Google Transparency Report [1]. The main two categories of unsafe websites are phishing websites and malware websites. Phishing websites are subdivided into evil twin websites and scam websites. Google's latest Transparency Report [1] shows that in the past few years, the amount of phishing websites detected per week by Google Safe Browsing is significantly higher than the amount of malware websites. This trend, that was reversed a decade ago, is shown in Figure 2.2.

2.1.1 Phishing Websites

The goal of phishing websites is to steal user credentials such as emails, passwords, credit card numbers and other sensitive data. Phishing websites use several techniques to achieve their goal, but the main idea is to exploit the lack of attention or education of the user. The great majority of phishing attacks, including targeted ones, start with a link inside an email, chat or SMS pointing to a phishing website. When a user does not have enough time or knowledge to check this link and web page carefully, they can become a potential victim of an attack. According to the Data Breach Investigations Report in 2019 [10], phishing websites stand behind 32 percent of all security breaches.

Phishing websites are classified into two classes: Evil twin websites and Scam websites. Evil twin websites are described separately in Section 2.2.

Scam websites use phishing practices to offer fake products to users with the aim to steal their credentials. Examples of scams are unexpected wins, ways to get money or offers of very cheap products. In most cases all this happens under time pressure. For instance, "You

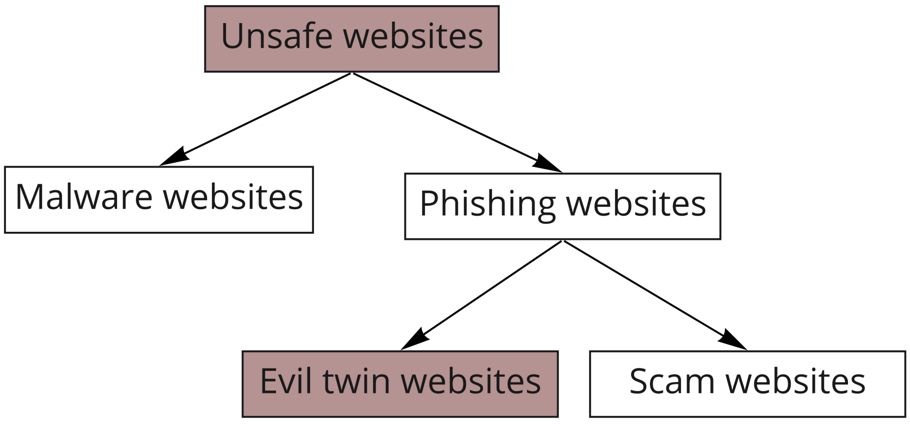


Figure 2.1: Basic categories of unsafe websites and their relationships. This thesis is focused only on the detection of unsafe websites and evil twin websites. Based on the Google Safe Browsing category of the Google Transparency Report [1]

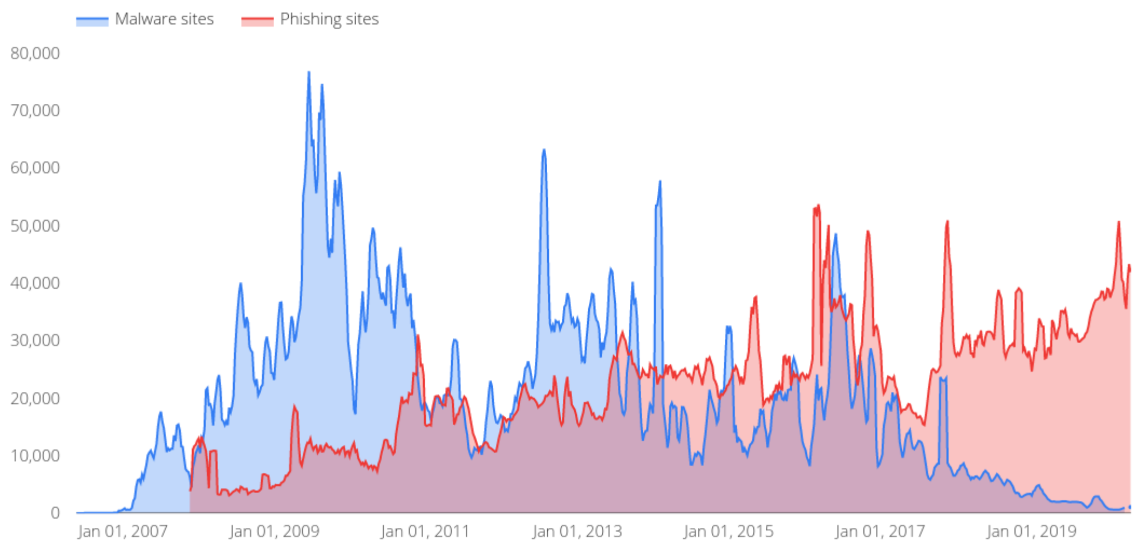
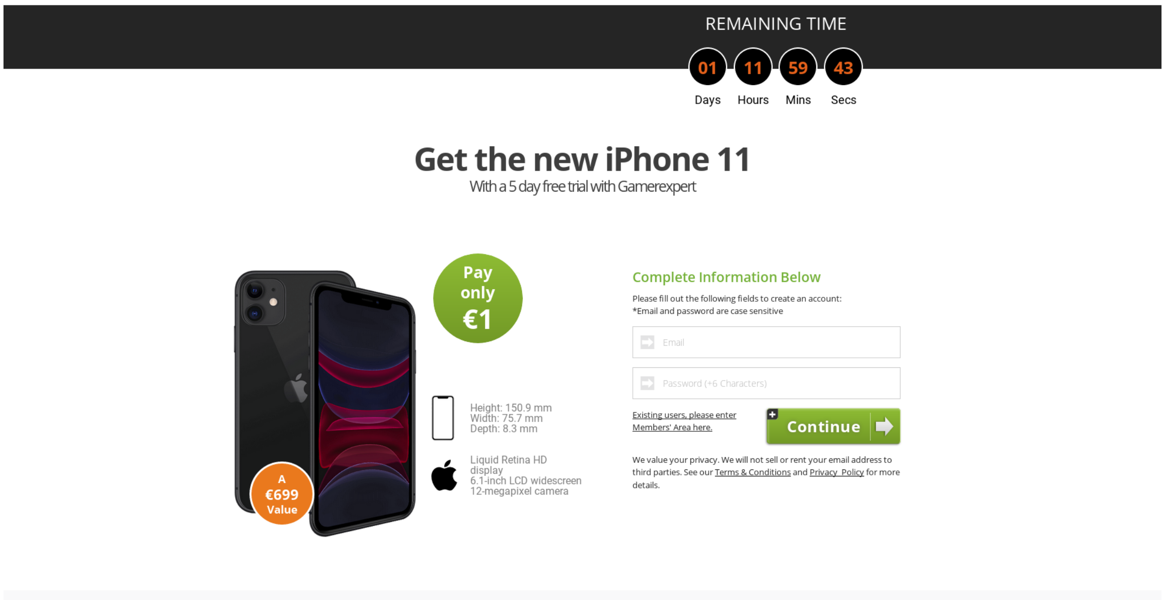


Figure 2.2: Amount of phishing and malware websites detected per week by Google Safe Browsing [1]. Data obtained on May 10th, 2020.

won a new iPhone7 and you have only 60 seconds to fill this information to get this phone". An example of a scam web page is shown in Figure 2.3, offering an iPhone 11 for €1. Scam websites would also be in the scope of this thesis as a separate problem of detection, however, we were not able to find a satisfying amount of them to build a dataset.

2.1.2 Malware Websites

The goal of malware websites is to infect user's devices. Malware websites contain malicious code that installs malicious software onto user's devices. The malicious software can



REMAINING TIME

01 11 59 43

Days Hours Mins Secs

Get the new iPhone 11

With a 5 day free trial with Gamerexpert

Pay only €1

A €699 Value

Height: 150.9 mm
Width: 75.7 mm
Depth: 8.3 mm

Liquid Retina HD display
6.1-inch LCD widescreen
12-megapixel camera

Complete Information Below

Please fill out the following fields to create an account:
*Email and password are case sensitive

Email

Password (+6 Characters)

Existing users, please enter Members' Area here.

Continue

We value your privacy. We will not sell or rent your email address to third parties. See our Terms & Conditions and Privacy Policy for more details.

Figure 2.3: An example of a scam web page offering iPhone 11.

be delivered to a user's device in two ways:

- By exploiting a vulnerability on the web browser and installing malware without the user consent.
- By luring the user to install a program that is malicious but looks legitimate.

Despite the improvements on the security of browsers there are still many security vulnerabilities. In 2018, the Common Vulnerability and Exposure list [11] and the National Vulnerability Database [12] published a database of 14,760 known security vulnerabilities, which is twice as many as were reported in 2016. According to review from Malware bytes [13], the Internet Explorer browser is still exploited in 2019 in a number of drive-by download campaigns. In our experience hunting of malware websites is very difficult but they still pose a big risk for users.

It should also be emphasised that many malware websites are compromised websites. It means that legitimate websites were compromised by attackers and admins of these compromised websites often have no idea about it. According to Google's Transparency Report [1], the amount of attacking websites created specifically for malicious activity is less than the amount of legitimate websites compromised and converted into malicious websites. This difference is shown in Figure 2.4.

2.2 Evil Twin Websites

One of the most prevalent types of phishing websites are evil twin websites. Evil twin websites are clones imitating real websites that are often indistinguishable from the real websites. They use phishing practices to steal emails, passwords, credit card numbers and

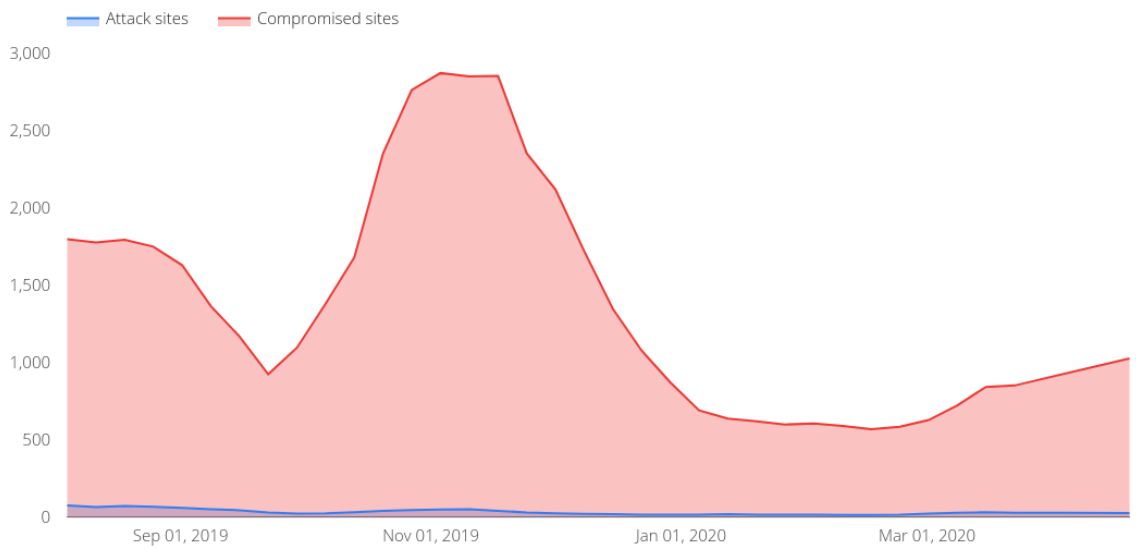


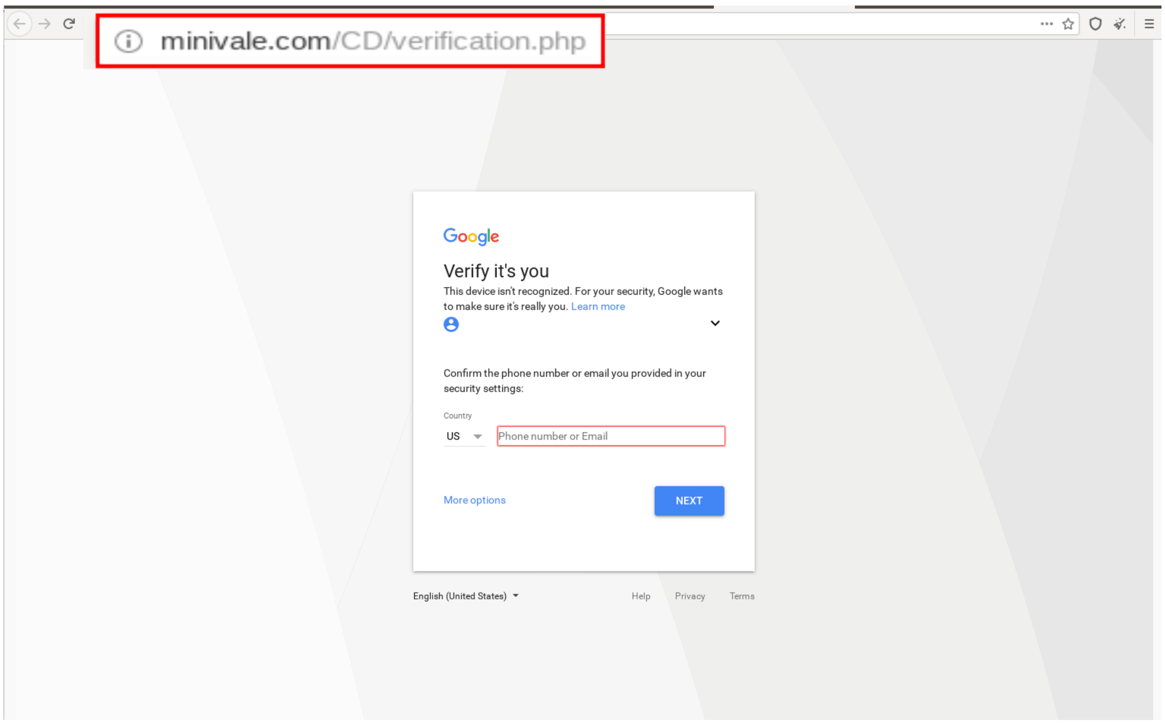
Figure 2.4: Amount of attacking websites created on purpose and amount of attacking websites that were legitimate websites compromised by attackers. Information from the detected by Google Transparency ReportSafe Browsing [1]. Data obtained on May 10th, 2020.

other sensitive data. The effectiveness of evil twin websites is based on the user's inattention, because humans tend to judge the authenticity of a web page by how it looks instead of the URL of the web page. An example of an evil twin web page is shown in Figure 2.5, where the attacker tries to imitate the Google login web page. What can be clearly seen in this figure, is that no difference is visible by the human eye except the URL of the web page.

Through manual analysis of evil twin websites we detected several common features:

- Fake links on the web page with no action after clicking on them. They are often used with phrases such as "Terms", "Data Policy", "Sign Up", "Cookies Policy", "About" and countless others. These phrases are also used for links pointing to different websites (different domain), which is not so common in legitimate websites. For example, if a user clicks on a link with text "Sign up", the expectation is that the web page redirects the user to the same website with the different web page.
- The web page and the images on it have poor resolution. This is not common in legitimate websites.
- The structure and content of the HTML code on the website is more amateur in contrast to legitimate websites.

Our analysis shows that evil twin websites are often poor in quality, since they are created with urgency and they have a short lifespan.



minivale.com/CD/verification.php

Google

Verify it's you

This device isn't recognized. For your security, Google wants to make sure it's really you. Learn more

Confirm the phone number or email you provided in your security settings:

Country

US

Phone number or Email

More options

NEXT

English (United States)

Help Privacy Terms

Figure 2.5: An example of an evil twin web page imitating the Google login form.

Chapter 3

Related Work

This chapter discusses recent work related to the detection of unsafe websites and evil twin websites. It is important to note that many research works use the term malicious websites instead of unsafe websites and do not distinguish evil twin websites as a separate category of phishing websites, but it does not affect the discussion in this chapter.

A classical technique for detecting unsafe websites is through blacklists. Blacklists contain a list of updated URLs that are labeled as unsafe. When a new URL is checked and the URL is present in the blacklist, the URL is marked as unsafe. Blacklists are hard to maintain, and new unsafe URLs are easily missed. The research work "An Empirical Analysis of Phishing Blacklists" [3] studies the effectiveness of phishing blacklists. Authors showed that the false positive rate of blacklists is excellent, but they also show that blacklists fail for new malicious URLs. Despite these limitations, blacklists are still widely used today.

The next common solution is applying machine learning on extracted features from a URL. The advantage of machine learning algorithms is the ability to generalize and be able to recognize new unsafe URLs unlike blacklisting solutions. The research work "Using Lexical Features for Malicious URL Detection – A Machine Learning Approach" [4] extracts static lexical features from the URL string and shows that purely lexical approaches can be used for the detection. Approaches based on deep learning techniques do not need feature extraction and domain knowledge. The research work "Malicious URL Detection using Deep Learning" [5] compares several deep learning methods based on CNN and RNN with very satisfying results. Their best models achieved around 93-98% malicious URL detection rate with false positive rate of 0.001. Another work "Detecting Phishing Websites through Deep Reinforcement Learning" [6] uses deep reinforcement learning providing robust, dynamic, and self-adaptive solutions with interesting results.

The next area of research is focused on detection by content of a web page. The research work "A Deep Learning Approach to Fast, Format-Agnostic Detection of Malicious Web Content" [14] proposes a deep learning approach to detecting malicious HTML content. The input representation is a sequence of tokens from HTML documents. They bring satisfying results with performance of 97.5% detection at a 0.1% false positive rate. The research work "Malicious HTML File Prediction: A Detection and Classification Perspective with Noisy Data" [15] proposes a framework for detection of malicious HTML files. It is based on static features extracted from HTML. They analyze the HTML file and show common techniques used by creators of malicious websites.

The next research area is focused on classification of JavaScript code. The most common practice to hide malicious JavaScript code is to use obfuscation techniques. The analysis from “The power of obfuscation techniques in malicious JavaScript code: A measurement study” [16] shows that JavaScript obfuscation is a common practice to evade detections. Moreover they show that many popular anti-virus software can be effectively evaded by the obfuscation techniques. The research work “A Machine Learning Approach to Malicious JavaScript Detection using Fixed Length Vector Representation” [17] came with the idea to detect malicious JavaScript code contents using Doc2Vec, which is a neural network based on context information of texts. Doc2Vec is also able to detect several obfuscating techniques.

Another perspective of detecting unsafe websites is shown in “Textual and Visual Content-Based Anti-Phishing: A Bayesian Approach” [18], where authors introduce a solution to defend protected websites by measuring similarities between the protected web page and suspicious web pages. They focus on phishing websites and their solution is based on texts and images from the web page.

Several researchers were also focused on the detection of phishing websites by favicon images. For example, research work “Favicon - a clue to phishing sites detection” [19] suggest detection and recognition of favicons to identify the brand sites and determine if a website is legitimate or phishing. Next research based on favicon is “Phishdentity: Leverage Website Favicon to Offset Polymorphic Phishing Website” [20]. Authors proposed a method called Phishdentity utilizing Google search-by-image API to identify phishing websites.

Chapter 4

Detection of Unsafe Websites

The first goal of this thesis is to detect unsafe websites by analyzing their content and behavioral characteristics. The first step of this process is to explain the generation of the dataset, then to analyze its features, to propose a machine learning method to detect unsafe websites and to evaluate the results.

4.1 Dataset of Unsafe Websites

One of the main contributions of this thesis is the generation and publication of the Unsafe Website Dataset (UWD). The UWD dataset was created to train machine learning methods to detect any type of unsafe website and we made it publicly available for the research community [21].

The UWD dataset contains two target labels as classes, *unsafe websites* and *legitimate websites*. The unsafe websites include different types of websites that are considered harmful or attacking the users. In contrast to other datasets containing only URLs of websites [22, 23, 24], each data sample in the UWD dataset contains complex analysis and description of the website. One of the major drawbacks of the UWD dataset is that it contains both phishing and malware websites and, therefore, the samples are coming from different populations.

4.1.1 Finding Suitable Data

To generate the UWD dataset it was necessary to find suitable URLs for unsafe and legitimate websites. The Alexa top 1 million sites [25] were used as the source for the legitimate websites. For unsafe websites, we found several blacklists containing a wide spectrum of phishing and malware websites [23].

In order to increase the certainty that the unsafe URLs from the blacklists were really malicious, each unsafe URL was submitted to VirusTotal [26]. VirusTotal returned information if this URL was marked as unsafe. If at least one detection engine from VirusTotal marked the URL as unsafe, the URL was labeled as unsafe in our dataset as well.

The same process was performed with the legitimate URLs from the Alexa top 1 million sites list, because it is known that the list may not contain only legitimate sites. Any legitimate URLs that were marked as malicious by VirusTotal detection engines were not

	# Samples
Unsafe Websites, (Positive samples)	10,194
Legitimate Website, (Negative samples)	10,194
Total Number of Data Samples	**20,388**

Table 4.1: The UWD dataset contains 20,388 data samples with the same amount of positive (unsafe) and negative (legitimate) data samples.

added to our dataset. After this process of cleaning the URLs by VirusTotal, we gathered 10,194 unsafe URLs and 10,194 legitimate URLs.

4.1.2 Generation of the UWD Dataset

Once the URLs of legitimate and unsafe websites were collected, the next step was to use the urlscan.io service to generate the features for all the URLs.

The urlscan.io service [7] is a free service to scan and analyse websites. It provides a very complex analysis of a website and collects information including a screenshot of a web page, its HTML DOM, domains linked in the website, IP addresses related with the website, JavaScript variables, cookies, TLS states and more [7]. Urlscan.io uses the Google Chrome browser in headless mode to browse the submitted URLs and obtain the information. The entire analysis is provided in the main webpage and also in one JSON output.

In total, 20,388 data samples were collected with the same amount of positive (unsafe websites) and negative (legitimate websites) samples as shown in Table 4.1. Each sample in the UWD dataset contains URL, urlscan.io analysis and positive or negative label.

4.1.3 Exploration of the UWD Dataset

Before creating data models for machine learning algorithms, we investigated a few aspects of the UWD dataset. First we explored the usage of HTTPS certificates, because today HTTPS is a common practise for the vast majority of websites. In particular, we wanted to know how legitimate and unsafe websites differ in this aspect. Second, we focused on the content downloaded from the web pages.

4.1.3.1 Usage of HTTPS per Class in UWD Dataset

We found only 1,130 unsafe websites having HTTPS certificates and 6,782 legitimate websites having HTTPS certificates. Figure 4.1 shows that these values constitute more than 11% unsafe websites with the HTTPS certificate and more than 66% legitimate websites with the HTTPS certificate. As far as invalid HTTPS certificates (e.g. unknown issuer, expired time validity, etc.), we found only 3 unsafe websites with invalid certificates and in the case of legitimate we found no invalid certificate.

Although generating certificates on a website is a very easy process due to the Lets Encrypt service [27], the creators of malicious websites still do not use HTTPS certificates extensively. This suggests that HTTPS can play an essential role for the detection of unsafe websites.

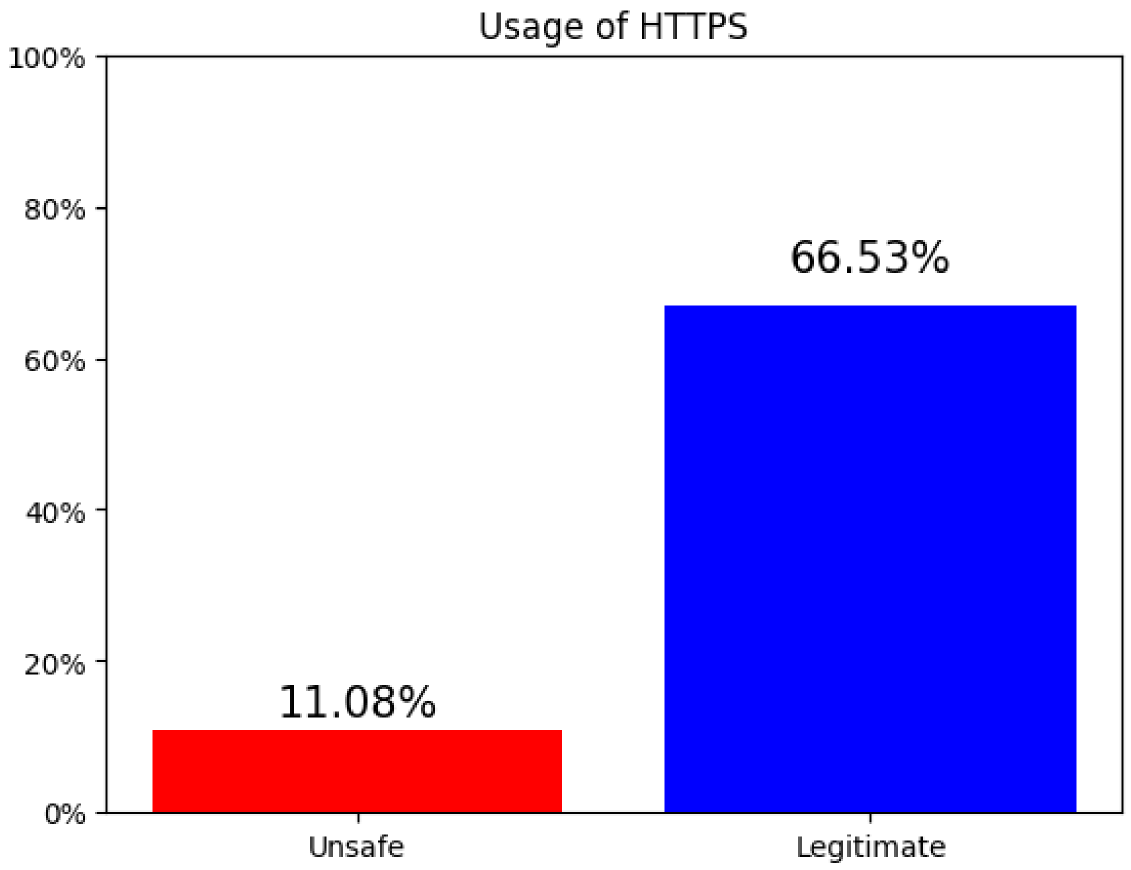


Figure 4.1: Usage of HTTPS by unsafe and legitimate websites in the UWD dataset.

4.1.3.2 Size of Downloaded Content per Class in UWD Dataset

When a web page is loaded, content is often downloaded from different sources. For example, CSS styles, JavaScript libraries, images and others. The size of all downloaded content was taken from each data sample in the dataset and the mean, standard deviation were computed from all these values. The results show that legitimate websites download, in average, more content from external sources than unsafe websites, but the results are not conclusive due to the high standard deviation. The following is a comparison of the size of downloaded files done by unsafe websites and legitimate websites:

- Unsafe websites
  - Mean of downloaded content: 1.39 MB
  - Standard deviation of downloaded content: 3.15 MB
- Legitimate websites
  - Mean of downloaded content: 3.98 MB
  - Standard deviation of downloaded content: 4.12 MB

These results also show that the size of the downloaded content of web pages is highly varied.

4.2 Feature Extraction

In order to describe all extracted features used for later machine learning experiments, first we will describe the available data obtained from urlscan.io. The urlscan.io analysis results presented by urlscan.io is split into several sections where each section describes a different aspect of the website. Since the analysis contains a large amount of data, only the important sections for our work are described here. To see all the data from the urlscan.io analysis, please visit the urlscan.io [7]. The urlscan.io analysis contains the following sections:

- Request section: The request section contains information about the communication between the browser (from urlscan.io) and the analyzed website. Each request has its response from the website. The first request is a request to the server of the submitted URL. The rest of the requests and responses depend on the web page. When a web page is loading, it often requires other resources to work and be displayed correctly. For example, CSS styles files, JavaScript libraries or images, which can be located in the same web page or in other servers. Each request and response is stored and shown.
- Cookies section: This section contains all HTTP cookies sent from and to the website. Each cookie contains information such as domain, time, validity, size, etc.
- Console section: This section contains console messages with their text message, URL, type and other information.
- Links section: This section contains all links from the web page with their texts. This information is taken from anchor tags.
- Globals section: This is a list of all JavaScript global variables used by a web page. It includes the name of the global variable and its type (function, object, number, etc.).
- Stats section: This section contains statistics about the usage of protocols, TLS, domains, IPs, etc.
- Lists section: The list section contains a list of IP addresses, countries, ASNs, domains, servers, URLs, sub domains and certificates.

All these sections contain numerical, categorical and text data. We manually extracted two feature sets from the urlscan.io analysis: feature-set-1 and feature-set-2. The feature-set-1 was created in 2018 in the beginning of our research and then we found out that the feature-set-1 does not contain all available information from the urlscan.io analysis. So we decided to create the feature-set-2 with additional information. The feature-set-1 contains 233 features based mainly on numerical data. The feature-set-2 contains 310 features based on numerical and categorical data. Due to the high number of features in both feature sets, they are detailed in sections A.1 and A.2 in the appendix.

	# Rows	# Positive rows	# Negative rows	# Features
UWD-feature-set-1$^{\text{Train}}$	16,310	8,155	8,155	233
UWD-feature-set-1$^{\text{Test}}$	4,078	2,039	2,039	233
UWD-feature-set-1	**20,388**	**10,194**	**10,194**	**233**
UWD-feature-set-2$^{\text{Train}}$	16,310	8,155	8,155	310
UWD-feature-set-2$^{\text{Test}}$	4,078	2,039	2,039	310
UWD-feature-set-2	**20,388**	**10,194**	**10,194**	**310**

Table 4.2: The data model UWD-feature-set-1 is split into the training and testing parts. The same holds for the UWD-feature-set-2.

4.3 Data Models

With data models we refer to matrices of values where each row is identified as a data sample from UWD dataset and the columns are the feature values. Also each row has its own label, either positive or negative sample.

The data models UWD-feature-set-1 and UWD-feature-set-2 were computed from the feature-set-1 and feature-set-2, respectively. Both data models contain the same amount of rows as a number of samples in the UWD dataset. Then both data models UWD-feature-set-1 and UWD-feature-set-2 were split into training and testing data for later experiments. The idea behind this is to keep the testing data for final evaluation and use the training data for finding and tuning the best methods. With a ratio of 80:20, the training part contained 16,310 samples and the testing part contained 4,078 samples for both data models. It is important to note that the training and testing data (rows) in the data model UWD-feature-set-1 come from the same samples of the MWD dataset as the training and testing data in the data model UWD-feature-set-2. Also the number of positive and negative samples is balanced for the training and testing data in both data models. The detailed overview is shown in Table 4.2.

4.3.1 Visualisation of Data Models

High dimensional data is difficult to interpret for humans. As analysts, to have a better idea of how the data looks like, it is possible to use dimensionality reduction methods that can convert high dimensional data into a lower dimension and provide a visualization in a low-dimensional space of two or three dimensions. By reducing the dimension in a way that preserves as much of the structure of the data as possible, we can get a visual representation of the data. This representation can provide us an intuition about complexity of the data.

In this thesis we decided to use a technique called the Uniform Manifold Approximation and Projection (UMAP) [28] due to its good performance and scaling. UMAP is a nonlinear dimensionality reduction technique that is created from a theoretical framework based in Riemannian geometry and algebraic topology. In contrast to other methods, such as PCA, UMAP is optimized using a randomly initiated stochastic gradient descent. Therefore, each run of UMAP produces slightly different results.

A visualisation of the UWD-feature-set-1 using UMAP is shown in Figure 4.2. The figure contains many different groups of samples for both classes. Many of them are mixed together and many single data points are spread through the whole figure. We can clearly see that positive and negative samples are not linearly separable and it can pose a difficult classification task.

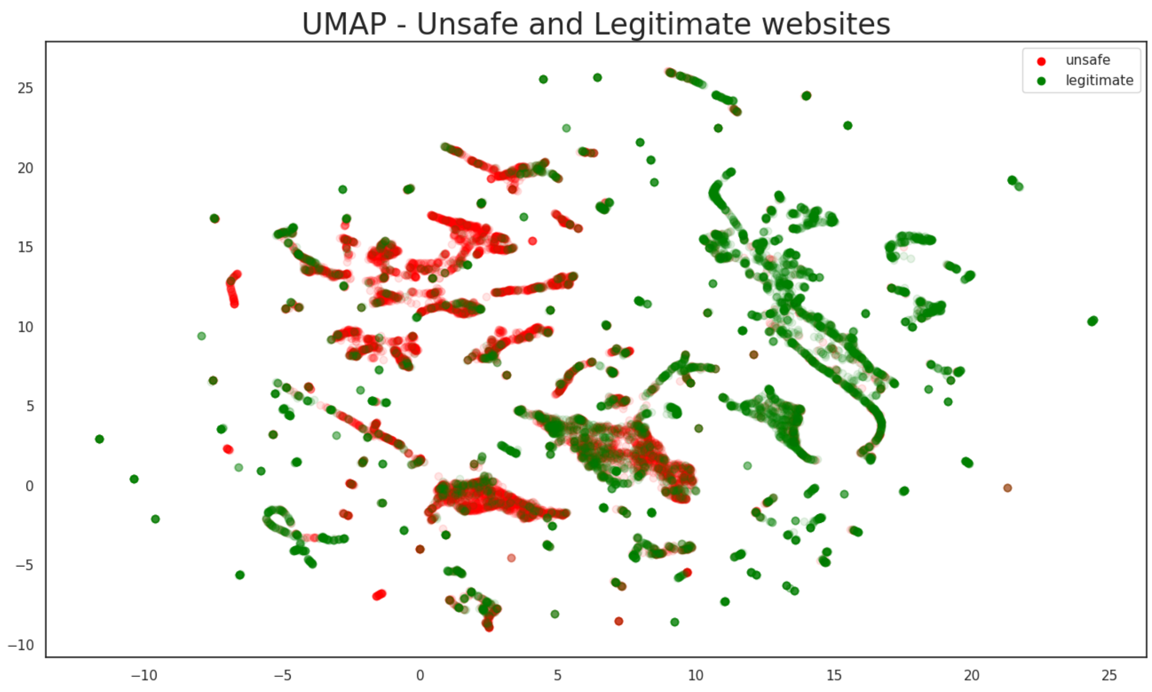


Figure 4.2: UMAP representation of the Data model using the UWD-feature-set-1 reduced from 233 dimensions to 2 dimensions. Green dots are legitimate websites and red dots are unsafe websites.

The visualisation of the UWD-feature-set-2 is shown in Figure 4.3. In this case, there are two groups of negative samples in the middle of the figure. The positive samples are spread across the whole image without noticeable grouping. The concentration of negative samples close to each other could mean that negative samples are similar in our feature space in contrast to positive samples. Even in this case, data are not linearly separable.

To compare both UMAP projections, it seems the UWD-feature-set-2, shown in Figure 4.3, has a better ability to separate positive and negative samples, because their clusters are more clear. Therefore, we expect better classification results for the UWD-feature-set-2 than UWD-feature-set-1.

4.4 Methods

4.4.1 Machine Learning Algorithms

Experiments in this chapter are based on the Random forest and XGBoost algorithms. One of the reasons why both both algorithms were chosen is because they provide an importance score for each feature. They were implemented with the sklearn library [29].

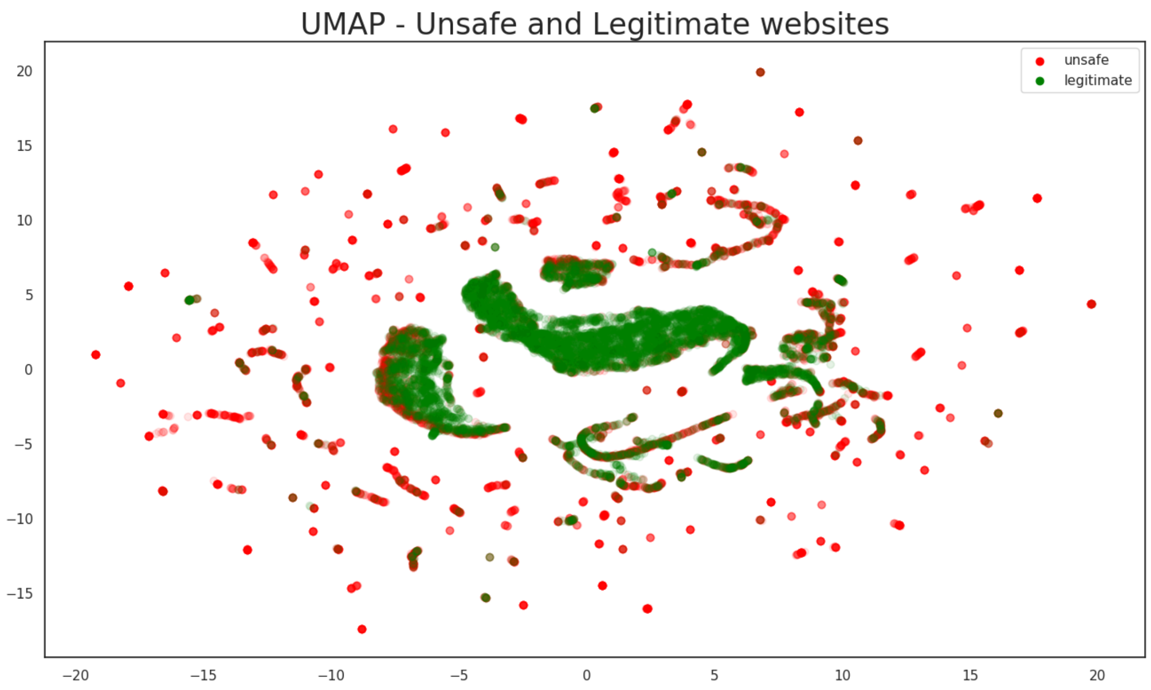


Figure 4.3: UMAP representation of the data model using the UWD-feature-set-2, reduced from 310 dimensions to 2 dimensions. Green dots are legitimate websites, red dots are malicious websites.

The random forest algorithm is a combination of tree predictors where each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest [30].

During tuning the hyper-parameters were chosen by cross-validated grid-search [31], the following hyper-parameters were picked:

- **n_estimators:** the number of trees in the forest
- **max_depth:** the maximum depth of the tree
- **min_samples_leaf:** The minimum number of samples required to be at a leaf node

XGBoost is an implementation of gradient boosted decision trees designed for speed and performance [32]. During tuning the hyper-parameters by cross-validated grid-search [31], the following hyper-parameters were picked:

- **n_estimators:** Number of gradient boosted trees
- **max_depth:** Maximum tree depth for base learners
- **min_child_weight:** Minimum sum of instance weight(hessian) needed in a child

4.4.2 Evaluation Metrics

A choice of which evaluation metric to use depends on various factors and it is a critical aspect of later evaluation of our machine learning algorithms. One of the factors is the proportion of positive and negative samples in the dataset. The balanced data is a more ideal case and many real world problems contain unbalanced data which involves using suitable metric and techniques. In this thesis all used datasets are balanced. Therefore, we compute following metrics from the confusion matrix:

- Accuracy: (TP + TN) / (P + N)
- False positive rate (FPR): FP / (FP + TN)
- False negative rate (FNR): FN / (FN + TP)

The next technique used in the experiments is the ROC curve. A ROC curve is a graph showing the true positive rate (TPR) on the y-axis and the false positive rate (FPR) on the x-axis at various threshold settings. A ROC curve shows the performance of a classification model at all classification thresholds. Based on the ROC curve, the Area Under Curve (AUC) measures the entire area under the entire ROC curve. The AUC metric tells if a model is capable of distinguishing between classes accurately. If the AUC is high, that means a model is good at prediction. The implementation of ROC curves that we used comes from the sklearn library [29].

In order to evaluate results we use learning curves. The learning curves are graphical representations of the validation score and training score with an increasing amount of training samples. Learning curves shows us how much we benefit from adding more training data. The learning curve consists of the validation curve and the training curve where the vertical axis is the accuracy of detection and the horizontal axis is the amount of the train data. The implementation of learning curves was used from sklearn library [29].

4.4.3 Feature Selection with Feature Importance

In our experiments, we used the feature selection with feature importance method (FSFI) [33] for selecting the most important features. The FSFI method is able to find the most important features using iterative training by selecting a subset of features. The FSFI method is shown in the Algorithm 1.

The algorithm starts on the ninth line of the pseudocode by creating a new model m trained with data X_{train} and labels y_{train}. Then the testing data X_{test} is used to evaluate the model with y_{test} as labels, getting an accuracy of the model with all available features. In line 14 starts the search for the best subset of features. First the threshold θ is chosen and X_{train} and X_{test} are transformed into $X^{'}_{train}$ and $X^{'}_{test}$ by the function Δ in such a way that all columns of the matrix of features with lower importance than the threshold θ are removed from the matrix. After the subset of best features is computed, in line 17 a new model $m^{'}$ is created using the transformed training data $X^{'}_{train}$ and labels y_{train}. This new model is evaluated by $X^{'}_{test}$ and a new accuracy is computed. If this new accuracy is greater than acc_{best}, this new accuracy and the current threshold θ are saved. The output of this algorithm is the list of the most important features.

Algorithm 1 FSFI

```
1: **Helpful functions**
2: $\Phi(X, y) \leftarrow$ Function to train the parameters of the model using the inputs $X$ and $y$. It returns the trained model.
3: $\Omega(m) \leftarrow$ Function that returns the feature importance list of the trained model $m$.
4: $\Delta(X, \theta, I) \leftarrow$ Function that returns the subset of features of $X$ that have their feature importance $I^i$ greater or equal to $\theta$.
5: $\sigma(X, m) \leftarrow$ Function to make predictions over a set of samples $X$ using the model $m$. It returns the list predictions.
6: $A(y_{true}, y_{pred}) \leftarrow$ Function that returns the accuracy computed using ground truth and predicted labels.
7: $\beta(t, L) \leftarrow$ Function that returns values from list $L$ greater or equal to threshold $t$.
8: procedure FSFI($X_{train}, y_{train}, X_{test}, y_{test}$)
9:     m $\leftarrow \Phi(X_{train}, y_{train})$
10:    $y_{pred} \leftarrow \sigma(X_{test}, m)$
11:    $I \leftarrow \Omega(m)$
12:    $acc_{best} \leftarrow A(y_{test}, y_{pred})$
13:    $\theta_{best} \leftarrow 0$
14:    for $\theta$ in $I$ do
15:        $X'_{train} \leftarrow \Delta(X_{train}, \theta, I)$
16:        $X'_{test} \leftarrow \Delta(X_{test}, \theta, I)$
17:        $m' \leftarrow \Phi(X'_{train}, y_{train})$
18:        $y_{pred} \leftarrow \sigma(X'_{test}, m')$
19:        $acc \leftarrow A(y_{test}, y_{pred})$
20:        if $acc > acc_{best}$ then
21:            $acc_{best} \leftarrow acc$
22:            $\theta_{best} \leftarrow \theta$
23:        end if
24:    end for
25:    return $\beta(\theta_{best}, I)$
26: end procedure
```

	Random Forest	XGBoost
n_estimators	500	500
max_depth	None	10
min_samples_leaf	1	∅
min_child_weight	∅	5

Table 4.3: The best found hyper-parameters for Random Forest and XGBoost algorithms.

	Random Forest		XGBoost	
	mean	std	mean	std
Accuracy	**0.9038**	**0.0080**	**0.9273**	**0.0044**
FPR	0.0815	0.0095	0.0645	0.0062
FNR	0.1108	0.0123	0.0809	0.0079

Table 4.4: Results for Random Forest and XGBoost algorithms with the best hyperparameters. The mean and std are computed from the 10 folds of the cross validation.

4.5 Experiments to Detect Unsafe Websites

To detect unsafe websites we propose to train and compare two algorithms: Random Forest and XGBoost. We implement and compare the performance of Random Forest and XGBoost algorithms on the data models UWD-feature-set-1 and UWD-feature-set-2. This section consists of two parts. The first part in Subsection 4.5.1 focuses on using the data model UWD-feature-set-1 to:

- train models without features selection
- select the best features
- train a model with the best features
- finally evaluate the best trained models in the testing data.

The second part in Subsection 4.5.2 repeats the process for the second data model UWD-feature-set-2.

4.5.1 Experiments using the UWD-feature-set-1

4.5.1.1 Training and Hyper-parameter Tuning

The hyper-parameter tuning with 10-fold cross validation procedure was performed on the UWD-feature-set-1$^{\text{Train}}$ data model for Random Forest and XGBoost. The best found hyper-parameters are shown in Table 4.3. The Random Forest algorithm achieved 90.38% of accuracy and XGBoost algorithm achieved 92.73% of accuracy with the selected hyper-parameters (Table 4.4).

	Random Forest		XGBoost	
Best features	147/233		107/233	
	mean	std	mean	std
Accuracy	**0.9046**	**0.0079**	**0.9257**	**0.0057**
FPR	0.0808	0.0092	0.0684	0.0077
FNR	0.1100	0.0116	0.0802	0.0095

Table 4.5: Results for the FSFI method.

4.5.1.2 Training with the Best Features

To find the most important features the FSFI method was performed. The FSFI method was run with the hyper-parameters above and it found a subset of the 147 most important features for Random Forest and a subset of the 107 most important features for XGBoost.

Then the 10-fold cross validation procedure was performed with the subsets of the features for both algorithms with the selected hyper-parameters from Table 4.3. The Random Forest algorithm achieved 90.46% of accuracy and XGBoost algorithm achieved 92.57% of accuracy (Table 4.5).

4.5.1.3 Comparison of the Trained Models

The **Random Forest** achieved better results with the selection of 147 features given by the FSFI method (Table 4.5) than with the all features (Table 4.4), because all metrics (the validation accuracy, false positive and false negative rates) were improved compared to results with all features.

Random Forest Summary:
- n_estimators: 500
- max_depth: None
- min_samples_leaf: 1
- The most important features: **147**/233
- Accuracy: 90.46%

The figure 4.4 shows the learning curve for the Random Forest with the mentioned setting. We can see clearly that the training score is still around the maximum. The validation score could be increased a little bit with more training samples. The figure 4.5 shows the ROC curve with the 10-fold cross validation. The AUC value achieved 0.9588 which means it has good measure of separability.

In the case of **XGBoost**, the subset of 107 features given by the FSFI method did not improve the results except the false negative rate. Thus, the results from table 4.4 still remain the best results.

XGBoost summary:
- n_estimators: 500
- max_depth: 10
- min_child_weight: 5
- The most important features: **All features (233)**

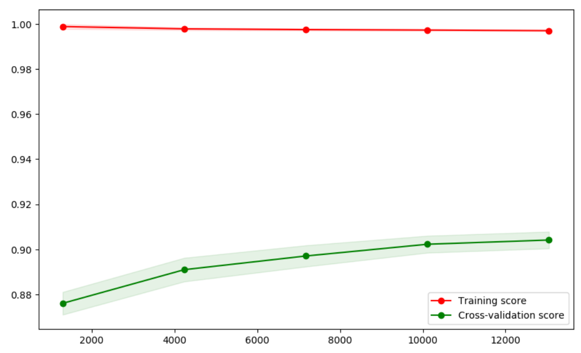


Figure 4.4: Learning curve for Random Forest with the best hyper-parameters and subset of 147 the most important features for the UWD-feature-set-1$^{\text{Train}}$.

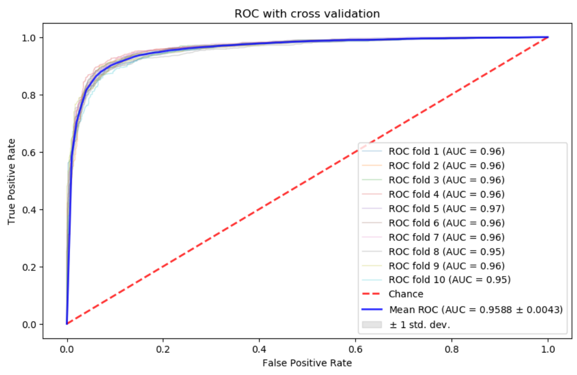


Figure 4.5: AUC ROC curve for Random Forest with the best hyper-parameters and with a subset of 147 the most important features for the UWD-feature-set-1$^{\text{Train}}$.

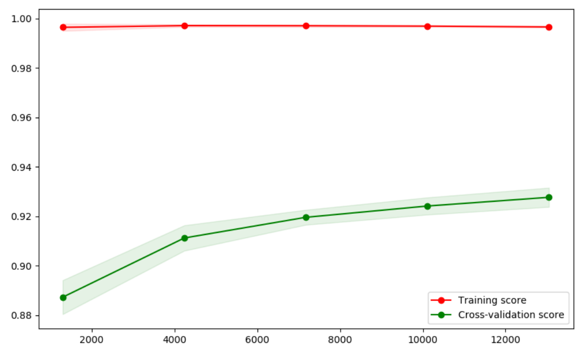


Figure 4.6: Learning curve for XGBoost with the best hyper-parameters and with the all features for the UWD-feature-set-1$^{\text{Train}}$.

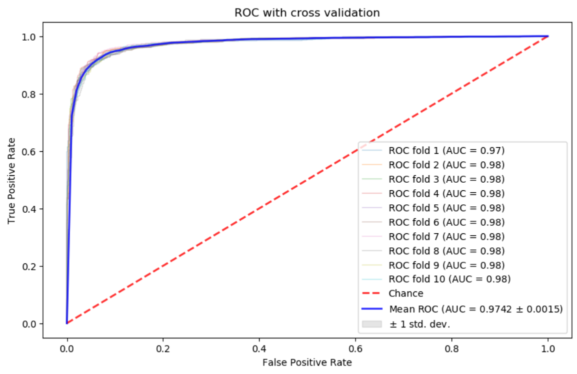


Figure 4.7: AUC ROC curve for XGBoost with the best hyper-parameters and with the all features for the UWD-feature-set-1$^{\text{Train}}$.

- Accuracy: 92.73%

Figure 4.6 shows the learning curve during training with the mentioned setting. We can see clearly that the training score is still around the maximum. In contrast to the learning curve for the Random forest, in this case the validation score could be increased more with next training samples. The Figure 4.7 shows the ROC curve with 10-fold cross validation. The AUC value achieved 0.9742 which means it also has a good measure of separability.

4.5.1.4 Evaluation on the Testing Data

The final step was to perform the Random Forest and XGBoost with the best found settings on the testing data UWD-feature-set-1$^{\text{Test}}$. The results are shown in the Table 4.6.

UWD-feature-set-1$^{\text{Test}}$	Random Forest	XGBoost
Testing accuracy	**0.9024**	**0.9269**
FPR	0.0760	0.0564
FNR	0.1192	0.0897

Table 4.6: Result of Random forest and XGBoost on UWD-feature-set-1$^{\text{Test}}$ data model.

	Random Forest	XGBoost
n_estimators	500	500
max_depth	100	10
min_samples_leaf	1	$\emptyset$
min_child_weight	$\emptyset$	10

Table 4.7: The best found hyper-parameters for both algorithms.

4.5.2 Experiments using the UWD-feature-set-2

4.5.2.1 Training with the Hyper-parameter Tuning

The hyper-parameter tuning with 10-fold cross validation procedure was performed on the UWD-feature-set-2$^{\text{Train}}$ data model for Random Forest and XGBoost algorithms. The best found hyper-parameters are shown in Table 4.7. The Random Forest algorithm achieved 90.15% of accuracy and XGBoost algorithm achieved 91.50% of accuracy with the selected hyper-parameters (Table 4.8).

4.5.2.2 Training with the Best Features

To find the most important features the FSFI method was performed. The FSFI method was run with the hyper-parameters above and it found a subset of the 222 most important features for Random Forest and a subset of the 183 most important features for XGBoost.

Then the 10-fold cross validation procedure was performed with the subsets of the features for both algorithms with the selected hyper-parameters from Table 4.7. The Random Forest algorithm achieved 90.17% of accuracy and XGBoost algorithm achieved 91.52% of accuracy (Table 4.9).

	Random Forest		XGBoost	
	mean	std	mean	std
Validation accuracy	**0.9015**	**0.0065**	**0.9150**	**0.0048**
FPR	0.1097	0.0107	0.0867	0.0108
FNR	0.0873	0.0112	0.0833	0.0094

Table 4.8: Result for Random Forest and XGBoost algorithms with best hyper-parameters. Each metric is a mean of 10 results from 10 folds from cross validation.

	Random Forest		XGBoost	
Best features	222/310		183/310	
	mean	std	mean	std
Validation accuracy	**0.9017**	**0.0050**	**0.9152**	**0.0048**
FPR	0.1108	0.0111	0.0864	0.0097
FNR	0.0858	0.0094	0.0831	0.0097

Table 4.9: Results for the FSFI method.

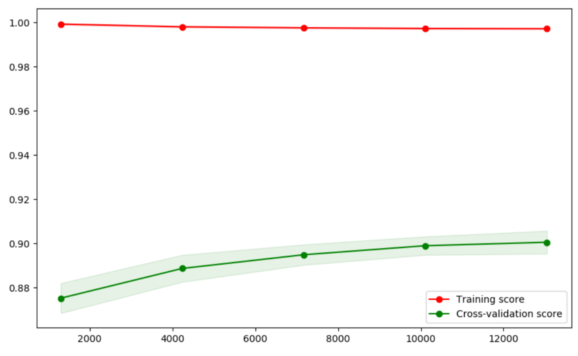


Figure 4.8: Learning curve for Random Forest with the best hyper-parameters and subset of 222 the most important features for the UWD-feature-set-2$^{\text{Train}}$.

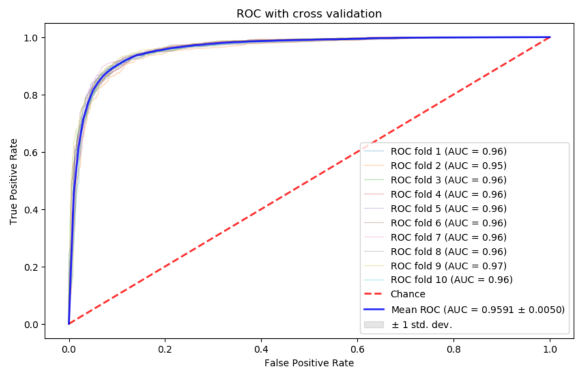


Figure 4.9: AUC ROC curve for Random Forest with the best hyper-parameters and with a subset of 222 the most important features with the all features for the UWD-feature-set-2$^{\text{Train}}$.

4.5.2.3 Comparison of the Trained Models

The **Random Forest** achieved slightly better results with the selection of 222 features given by the FSFI method (Table 4.9) than with the all features (Table 4.8), except the false positive rate. Despite the worse false positive rate the results with the subset of 222 features are still better.

Random Forest Summary:

- n_estimators: 500
- max_depth: 100
- min_samples_leaf: 1
- The most important features: **222**/310
- Accuracy: 90.17 %

The Figure 4.8 shows the learning curve during training with the mentioned setting. We can see clearly that the training score is still around the maximum. The validation score could be increased a little bit with more training samples. The Figure 4.9 shows the ROC curve with 10-fold cross validation. The AUC value achieved 0.9591 which means it has good measure of separability.

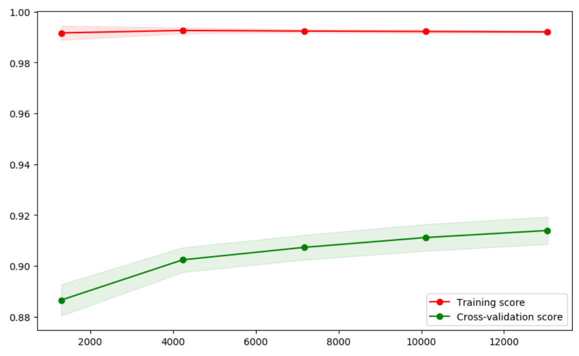


Figure 4.10: Learning curve for XGBoost with best hyper parameters and with a subset of the 183 most important features for the UWD-feature-set-2^{Train}.

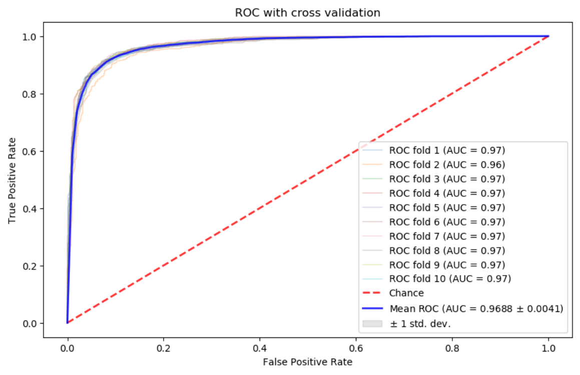


Figure 4.11: ROC for XGBoost with best hyper parameters and with a subset of the 183 most important features for the UWD-feature-set-2^{Train}.

UWD-feature-set-2^{Test}	Random Forest	XGBoost
Testing accuracy	**0.9031**	**0.9139**
FPR	0.1172	0.0927
FNR	0.0765	0.0795

Table 4.10: Result of Random forest and XGBoost on UWD-feature-set-2^{Test} data model.

The **XGBoost** algorithm achieved better results with the subset of 183 features given by the FSFI method (Table 4.9) than with all features (Table 4.8). All metrics (the validation accuracy, false positive and false negative rates) were improved compared to all features.

XGBoost summary:

- n_estimators: 500
- max_depth: 10
- min_child_weight: 10
- The most important features: **183**/310
- Accuracy: 91.52 %

The Figure 4.10 shows the learning curve with the mentioned setting. We can see clearly that the training score is still around the maximum. The validation score could be increased with more training samples. The Figure 4.11 shows the ROC curve with 10-fold cross validation. The AUC value achieved 0.9688 which means it has good measure of separability.

4.5.2.4 Evaluation on the Testing Data

The final step was to perform the Random Forest and XGBoost with the best found settings on the testing data UWD-feature-set-2^{Test}. The results are shown in the Table 4.10. The best testing accuracy was achieved with XGBoost with a value of 91.39%. In comparison Random Forest achieved 90.31%.

4.6 Analysis of Results

According to results on the testing data shown in Tables 4.6 and 4.10, the XGBoost model achieved the best accuracy for both data models UWD-feature-set-1$^{\text{Test}}$ and UWD-feature-set-2$^{\text{Test}}$. The best model for any dataset variation is XGBoost as show in Table 4.6 trained and tuned using the UWD-feature-set-1$^{\text{Train}}$ dataset because it achieved the highest accuracy with a value of 92.69% and the lowest false positive rate of 5.64% on the testing data.

These results suggest that the feature-set-1 provides a better description than the feature-set-2 for detection of unsafe websites. This fact is somewhat surprising, because the feature-set-2 contains **more** complex information from the urlscan.io analysis so we expected better results for models trained with the feature-set-2. Furthermore, the UMAP reduction method showed us also a better outcome 4.3.1 for data model UWD-feature-set-2.

To conclude this chapter, the best method found for detection of unsafe websites is the XGBoost algorithm trained and evaluated on the UWD-feature-set-1 with the feature-set-1. The complete set of features with their importance values are shown in Section A.3. As an example, the following are the top ten more important features:

- **127_secureRequests:** Total number of request that use HTTPS correctly.
- **67_ips_mean:** Mean number of IP addresses using HTTPS between the browser and web servers.
- **128_securePercentage:** Percentage of requests using HTTPS over all requests
- **18_length_linkdomains:** Total number of domains linked to this request
- **48_ips_std:** Mean number of IP addresses used between the client and the web server.
- **132_adBlocked:** Total number of advertisement trackers in the web page.
- **231_cookie_expires_now_mean:** Mean of all the *remaining time in seconds* for when all cookie will expire.
- **222_cert_valid_mean:** Mean of the validity length of all certificates used between client and webserver.
- **228_hashes_list_mean:** Total amount of cookies.
- **131_totalLinks:** Total amount of links contained in the web page

These top ten most important features also tell us that the study of HTTPS is paramount for the detection of unsafe websites given that four of the top ten features are based on HTTPS.

Chapter 5

Detection of Evil Twin Websites

The main purpose of this thesis is to develop machine learning methods to detect unsafe websites. However, there are many different types of unsafe websites with different characteristics and features. The second part of this thesis deals with a special type of unsafe websites, the *evil twin website*. These type of unsafe websites are probably the most common ones nowadays. Evil twin websites are websites designed to look like another official website with the purpose of stealing credentials.

The study of evil twin websites is done by analyzing its content, its behavioral characteristics and the structure of the website. The first step is to explain the generation of the dataset, then to analyze its features and propose a machine learning method to detect evil twin websites.

5.1 Dataset of Evil Twin Websites

In order to try to detect evil twin websites we created the Civilsphere Evil Twin Dataset (CETD), which we made publicly available for the research community [21]. The CETD dataset was created to train machine learning methods to detect evil twin websites and contains two target labels as classes, *evil twin websites* and *legitimate websites*. In contrast to the MWD dataset, the CETD dataset is very specific and its data has been verified more closely.

5.1.1 Finding Suitable Data of Evil Twin Websites

To generate the CETD dataset it was necessary to find suitable URLs for evil twin and legitimate websites.

The evil twin URLs were taken from three sources: the PhishTank database [22], the Phishing Domain Database [34] and the OpenPhish [35]. This is because evil twin websites are the primary tool in phishing attacks. All these three sources provide a list of evil twin URLs[1] being alive. However, since many evil twin websites have short life expectancy, it must be taken into account that a considerable number of these URLs may not have been alive when we obtained the data for our dataset.

[1]URLs from the mentioned sources are used for phishing attacks but their type is evil twin.

The process of finding legitimate websites was more complex. We needed to obtain the login web page of the legitimate websites, because our analysis shows that most evil twins websites are trying to copy the login web page. Therefore, the domains from the Alexa top 1 million list [25] were submitted to google search engine with “Log in” phrase. For example, for the “paypal.com” domain, the query to google search engine was “paypal.com Log in”. Then the first result from the search engine was taken and its URL was used as the final legitimate URL. For making a google search query automatically, the SP tool [36] was used. The SP is a command line tool to search startpage.com from the terminal. The startpage.com [37] is a service that provides a private search on Google search engine. The whole process of generating legitimate URLS is shown in Figure 5.1 with “paypal.com” example. At the end of the process, we gathered 25,571 evil twin URLs and 13,112 legitimate URLs.

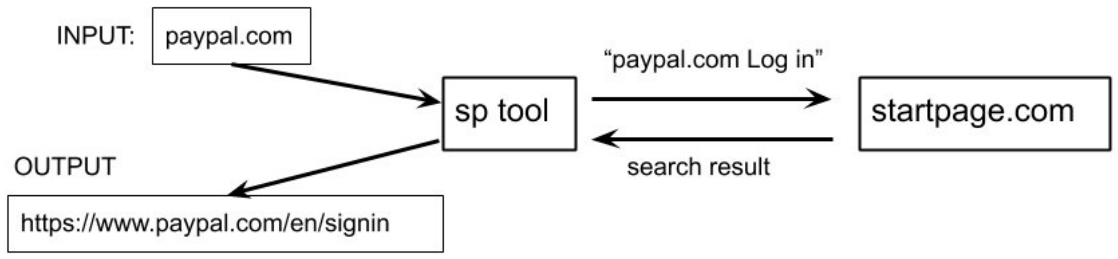


Figure 5.1: Example process of how to find the URL for the log-in page into each of the legitimate websites. First, the SP tool adds the phrase “Log in” to the domain and sends it to “startpage.com”. The site “startpage.com” returns the results from the search engine. The URL of the first result from the search engine is taken as the final legitimate URL.

5.1.2 Generation of the CETD Dataset

When the lists of evil twin and legitimate URLs were gathered, the process of generating the dataset could start. In this case, we used the urlscan.io [7] service to generate the urlscan.io analysis of a website, just like we did with the MWD dataset. However, from urlscan.io we also extracted HTML DOM for the CETD dataset. In summary, for each URL in the dataset there are two piece of data: urlscan.io analysis of a website and HTML DOM of a web page. These data are called a data sample.

Besides, for each URL from the CETD dataset we also extracted next data from the web page: a screenshot, all images and all JavaScript code. However, in this thesis, these data are not used for the experiments and they are intended for the further research. Especially the screenshot of the web page serves as proof of the website validity.

5.1.3 Dataset Cleaning

As in any other dataset, invalid samples are unfortunately included in our dataset due to several errors. One approach is to ignore the errors in case of a small number of invalid samples, because most machine learning algorithms can deal with small errors. However, during the generating of the dataset we observed that there was a significant amount of

	# Evil twin	# Legitimate
Invalid Data Samples	13,027	873
Valid Data Samples	**12,544**	**12,239**
All gathered samples	25,571	13,112

Table 5.1: The CETD dataset has 25,571 evil twin samples and 13,112 legitimate samples. After removing the invalid samples it has 12,544 evil twin samples and 12,239 legitimate samples.

invalid samples, mainly in the positive class. For this reason we decided to spend some time and effort to explore our dataset and remove invalid samples.

To clean our dataset we first checked randomly a few hundreds samples by hand to know which types of invalid samples are contained in the dataset. By looking at screenshots of web pages, we found two types of invalid samples:

- Error messages such as "Page Not Found" or "Server Not Found".
- Offers from domain providers to buy a domain of the website (parked domains).

These type of errors are expected when the domains are not valid anymore due to age, as was discussed previously.

To remove these invalid samples, we decided to use the structure of HTML DOM. The idea behind is that these invalid samples could have the same HTML structure. Our approach was following:

- For each sample construct the tree from HTML DOM
- Gather sets of samples having the same trees
- Check only one screenshot from each set
- If the checked screenshot indicates invalid sample, mark all samples from the set as invalid.

After the cleaning process, we found 13,027 invalid samples in the evil twin class and 873 invalid samples in legitimate class. These numbers show that evil twin websites have very short life expectancy in general and we had to remove more than half of all evil twin data samples (Table 5.1).

Since the amount of legitimate websites was slightly lower than the amount of evil twin websites, we collected a few more legitimate samples to have a balanced dataset. The final amount of samples in the dataset are shown in table 5.2.

5.1.4 Dataset Exploration

Before creating data models for machine learning algorithms, we investigated a few aspects of the CETD dataset. The first investigation was about the usage of HTTPS certificates, because nowadays HTTPS certificates are a common practise for the vast majority of

Positive Data Samples	12,544
Negative Data Samples	12,544
Total Number of Data Samples	25,088

Table 5.2: CETD dataset contains 25,088 samples with the same amount of positive (evil twin) and negative (legitimate) samples

websites and the most clear indicator that a site is secure and (mistakenly) trusted. Therefore we wanted to know how legitimate and evil twin websites state in this aspect. The second investigation was focused on downloaded content from the web pages. Then we also analysed the HTML DOM. So the third investigation was about total number of HTML tags in the HTML DOMs and the fourth investigation was concentrated on the number of selected HTML tags in the HTML DOM.

5.1.4.1 Use of HTTPS per Class in the CETD Dataset

We only found 5,739 evil twin websites using HTTPS certificates and 11,394 legitimate websites using HTTPS certificates. Figure 5.2 shows that it is more than 45 percent for evil twin websites and more than 90 percent for the legitimate websites. It is worth pointing out, that there were no invalid HTTPS certificates in the dataset (unknown issuer, expired time validity, etc.).

One of our main assumptions was that evil twin websites would use more HTTPS because they imitate legitimate services. However, results show that many creators of evil twin web pages still rely on the lack of user knowledge about HTTPS importance, because all users should know that all websites with login pages should use HTTPS.

5.1.4.2 Size of Downloaded Content per Class in the CETD Dataset

When a web page is loaded, usually some other content is downloaded from different sources. For example, CSS styles, JavaScript libraries, images and others. The size of all downloaded content was taken for each data sample and mean, standard deviation were computed throughout the whole dataset. The results show that the size of all downloaded data is higher for legitimate websites than in case of evil twin websites:

- Evil twin: Mean: 1.45 MB, Standard deviation: 2.58 MB
- Legitimate websites: Mean: 3.55 MB, Standard deviation: 4.13 MB

These results show that the size of the downloaded content of web pages is highly varied.

5.1.4.3 Total Amount of HTML Tags per Class in CETD Dataset

To have an idea about the amount of tags on each HTML DOM, from each HTML DOM the total number of all tags was taken. Then, for both classes we computed the mean, standard deviation, maximum and minimum. What stands out in table 5.3, is that the mean of tags for legitimate websites is four times higher than the mean for the evil twin websites.

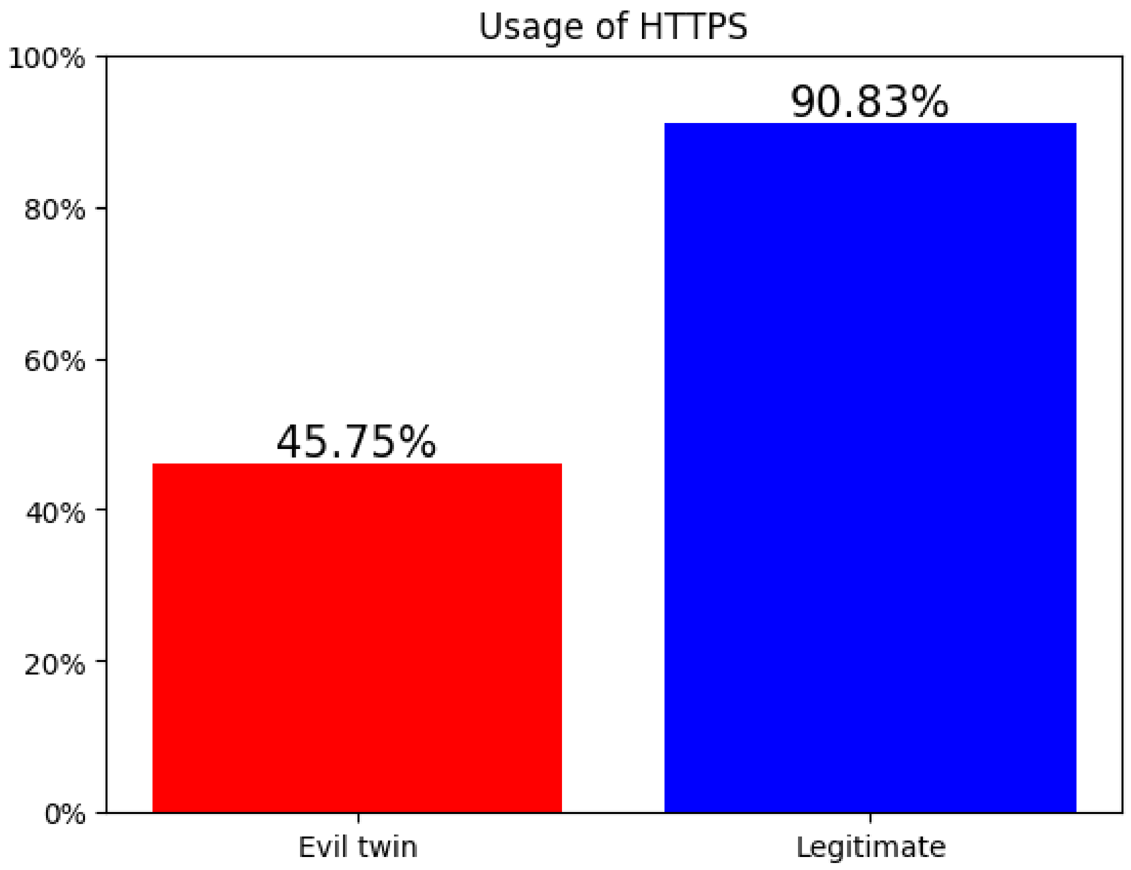


Figure 5.2: Percentage of usage of HTTPS in evil twin websites and normal websites.

Special attention shall be devoted to standard deviation of the legitimate websites, where the value is very large. This indicates that the number of tags in the legitimate websites is very varied in the CETD dataset.

5.1.4.4 Amount of HTML Tags per Class in CETD Dataset

The last investigation was about the occurrences of HTML tags in the HTML DOM. We selected the most used HTML tags for both classes. Figure 5.3 shows the means of occurrences for each selected tag per one HTML DOM. For example, we can see in the Figure that the mean of “div” tags occurrences for legitimate web pages is almost 300 while the mean of "div" tags occurrences for evil twin web pages is only 78 “div” tags. The interesting thing is, that “url” and “loc” tags are not contained in any evil twin sample, while the average legitimate web page contains 4 “URL” and “loc” tags.

	Evil twin	Legitimate
Mean	305.09	1,224.23
Standard deviation	666.25	10,786.07
Max	22,328	1,124,596
Min	3	5

Table 5.3: Statistic about total number of tags in the HTML DOM for both classes.

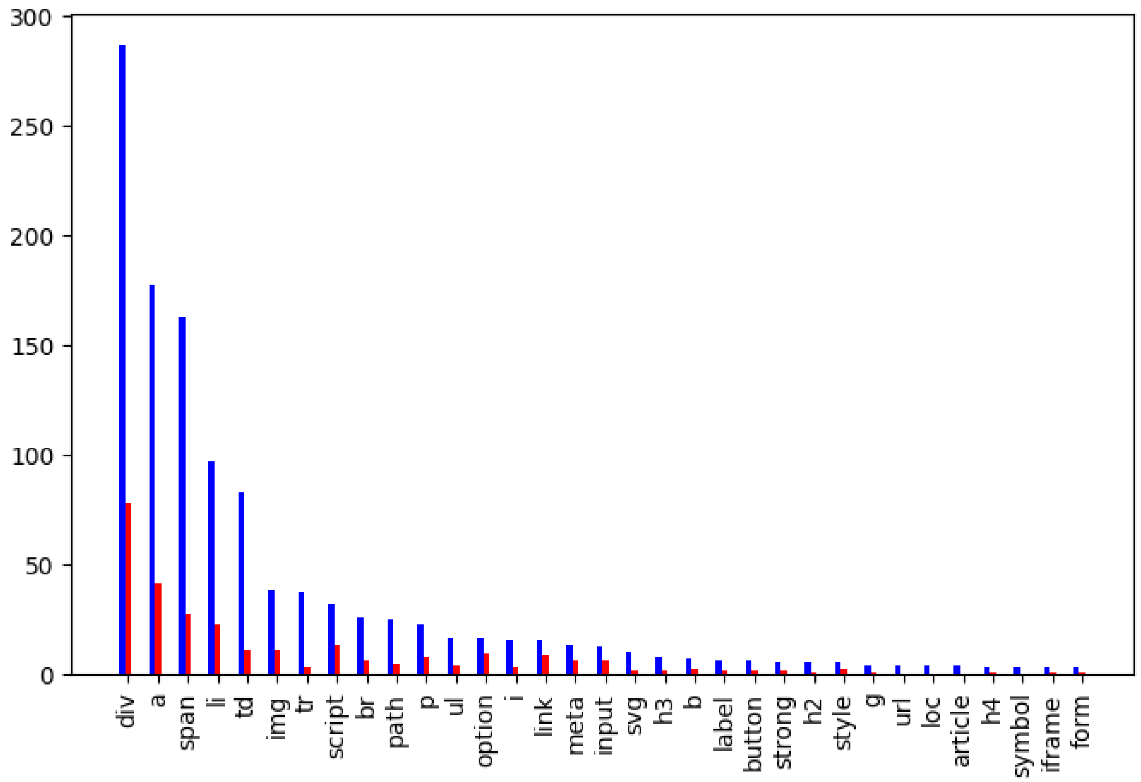


Figure 5.3: Average occurrences of HTML tags for legitimate websites (blue) and evil twin websites (red).

5.2 Data models

With data models we refer to matrices of values where each row is identified as a data sample from the CETD dataset and each column is a feature value. Also each row has its own label, either positive or negative. This chapter explains the creation of the four data models that are going to be used for the detection of evil twin websites.

The CETD-feature-set-1 and CETD-feature-set-2 are computed from the urlscan.io analysis. The CETD-DOM-1 and CETD-DOM-2 are generated from the HTML DOMs.

	# Rows	# Positive rows	# Negative rows	# Features
CETD-feature-set-1$^{\text{Train}}$	20,070	10,035	10,035	233
CETD-feature-set-1$^{\text{Test}}$	5,018	2,509	2,509	233
CETD-feature-set-1	**25,088**	**12,544**	**12,544**	**233**
CETD-feature-set-2$^{\text{Train}}$	20,070	10,035	10,035	310
CETD-feature-set-2$^{\text{Test}}$	5,018	2,509	2,509	310
CETD-feature-set-2	**25,088**	**12,544**	**12,544**	**310**

Table 5.4: The data model CETD-feature-set-1 is split into the training and testing parts. The same holds for the CETD-feature-set-2.

5.2.1 Data Models CETD-feature-set-1 and CETD-feature-set-2

We computed data models CETD-feature-set-1 and CETD-feature-set-2 from the CETD dataset by feature-set-1 and feature-set-2 explained in the section 4.2. Both data models contain the same amount of rows as a number of samples in the CETD dataset. Then both data models were split into training and testing data for the later experiments. The idea behind this is to keep the testing data for the final evaluation and use the training data for finding and tuning the best methods. With a ratio of 80:20, the training part contained 20,070 samples and the testing part 5,018 samples for both data models. It is important to note that the training and testing data in the CETD-feature-set-1 come from the same samples of the CETD dataset as the training and testing data in the CETD-feature-set-2. Also the number of positive and negative samples is balanced for the training and testing data in both data samples. The detailed overview is shown in Table 5.4.

5.2.1.1 Visualisation of Data Models CETD-feature-set-1 and CETD-feature-set-2

High dimensional data is difficult to interpret for humans. As analysis, to have a better idea of how the data looks like, we use UMAP reduction method.

Visualisation of the data model CETD-feature-set-1 by UMAP method is shown in Figure 5.4. The figure contains many different groups of samples for both classes. Many of them are mixed together and many single data points are spread through the whole figure. We can clearly see that positive and negative samples are not linearly separable and it can pose a difficult classification task.

The result of visualisation of CETD-feature-set-2 by UMAP method is shown in Figure 5.5. In the middle of the figure, there is one group of negative samples. The positive samples are rather spread across the whole figure without a noticeable group of samples. Even in this case, data are not linearly separable.

To compare both UMAP projections by human eye, it seems that features from the data model CETD-feature-set-2, shown in Figure 5.5, have better ability to separate positive and negative samples, because their group of samples are more clear. Therefore we expect better results in the experiments for the CETD-feature-set-2 than CETD-feature-set-1.

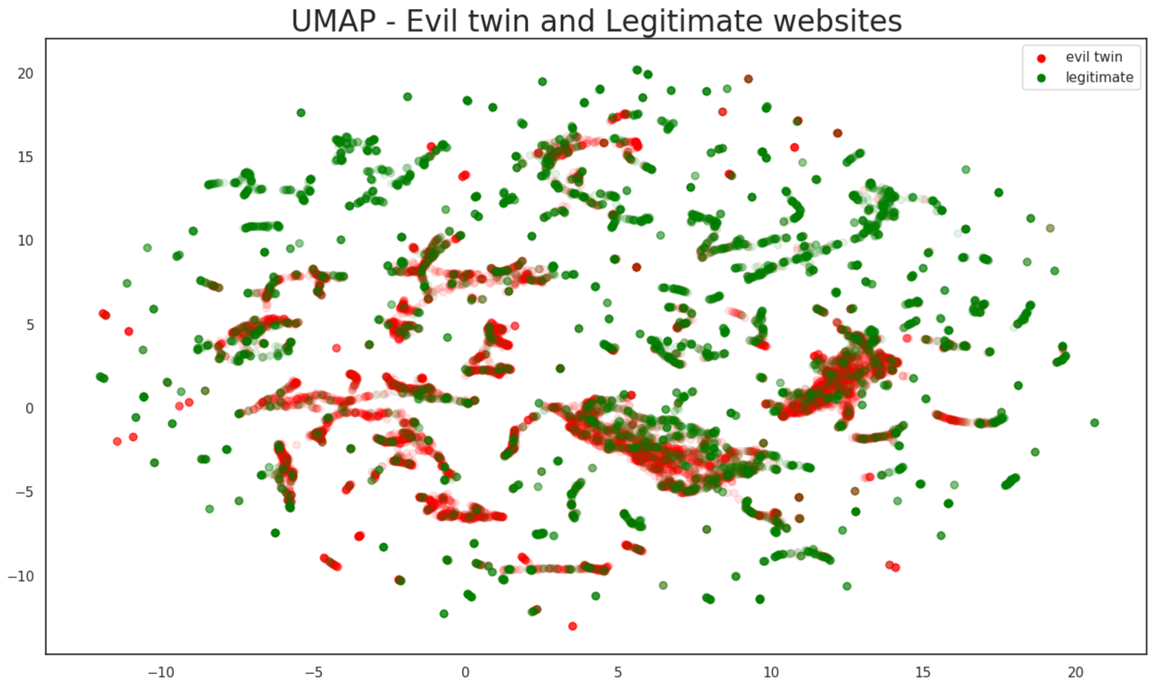


Figure 5.4: UMAP representation of data model CETD-feature-set-1, reduced from 233 dimensions to 2 dimensions. Green dots are legitimate websites, and red dots are evil twin websites.

	# Rows	# Positive rows	# Negative rows	# Columns
CETD-DOM-1	**25,088**	**12,544**	**12,544**	**257**
CETD-DOM-1$^{\text{Train}}$	16,808	8,404	8,404	257
CETD-DOM-1$^{\text{Test}}$	8,280	4,140	4,140	257

Table 5.5: The CETD-DOM-1 was split into the training and testing parts.

5.2.2 Data Model CETD-DOM-1

The data model CETD-DOM-1 was computed from the number of occurrences of selected HTML tags from each data sample in the dataset. We selected the 200 most frequently used HTML tags from both classes. Since many of them were the same, we got 257 unique HTML tags shown in A.4. The data model CETD-DOM-1 was computed by the following way: for each HTML DOM in the dataset, we computed the feature vector containing occurrences of the 257 HTML tags and this vector was normalized by the total number of HTML tags contained in the HTML DOM.

At the end, the data model CETD-DOM-1 contained 25,088 rows as the number of samples in the dataset and 257 columns as a number of selected HTML tags. All values were between 0 and 1, effectively converting it into a probability distribution. The process of computing the feature vector from the HTML DOM is shown in Figure 5.6. The last step was to split the data model CETD-DOM-1 into the training and testing parts as Table 5.5 shows.

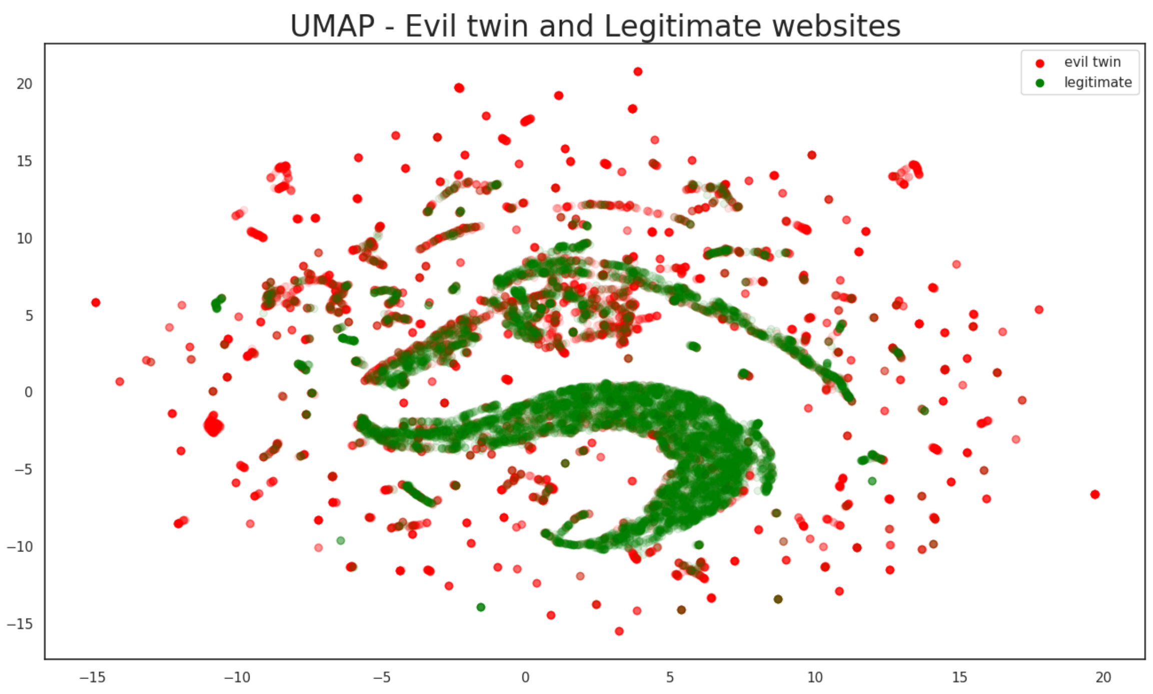
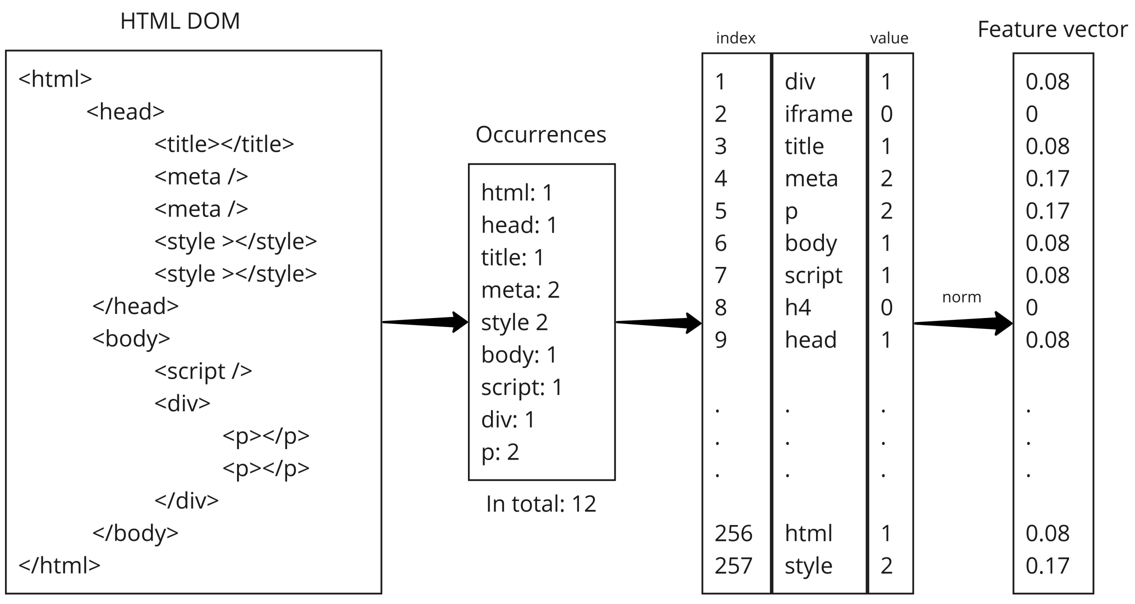


Figure 5.5: UMAP representation of the data model CETD-feature-set-2 reduced from 310 dimensions to 2 dimensions. Green dots are legitimate websites, and red dots are evil twin websites.



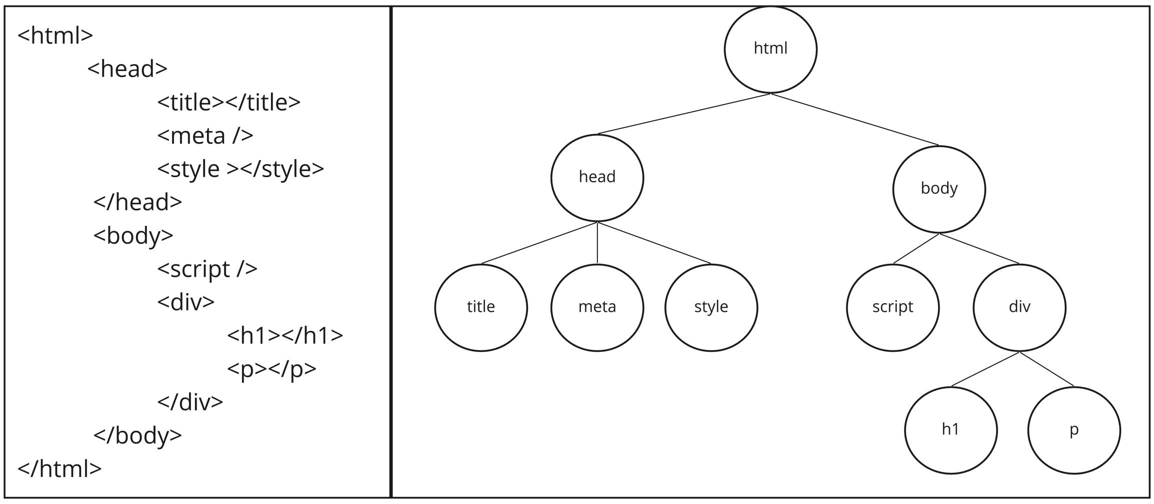
index		value	Feature vector
1	div	1	0.08
2	iframe	0	0
3	title	1	0.08
4	meta	2	0.17
5	p	2	0.17
6	body	1	0.08
7	script	1	0.08
8	h4	0	0
9	head	1	0.08
.	.	.	.
.	.	.	.
.	.	.	.
256	html	1	0.08
257	style	2	0.17

Figure 5.6: Conversion of the HTML structure, into HTML tags occurrences and into a feature vector.

5.2.3 Data Model CETD-DOM-2

The data model CETD-DOM-2 was created in a complex way based on the work *Graph classification with 2D convolutional neural networks* [38]. The authors represent a graph as a multi-channel image and then classify it with standard Convolutional Neural Networks. We applied this approach to the graph-like structure of the HTML DOM.

Each HTML DOM is transformed to a multi-channel image in the following way. Since an HTML DOM can be represented as a tree (Figure 5.7), the first step was to construct a tree from each HTML DOM. The tree was represented as a list of edges. Then the list of edges was processed by node2vec algorithm [39] to create a node embedding. Thus, each node of the tree has its own feature vector with the length of 128 features. Since node2vec is a stochastic algorithm and each run provides different results, this means that a given dimension will not be associated with the same concepts across trees in the dataset or even across several runs on the same tree. After the vectors with 128 dimensions were found, we used the PCA algorithm [40] to ensure that the embedding of all the trees in the dataset are comparable. The PCA method reduces the dimension of each node from 128 to 10 dimensions. This new matrix contains 10 columns and n rows as a number of nodes in the tree.



```
<html>
        <head>
                <title></title>
                <meta />
                <style ></style>
        </head>
        <body>
                <script />
                <div>
                        <h1></h1>
                        <p></p>
                </div>
        </body>
</html>
```

Figure 5.7: Representation of the HTML tags in an HTML structure into a graph structure.

The final step was to create histograms from the PCA node embedding matrix as described in the paper. The first histograms were taken from the first and second columns of the matrix, the second histograms were taken from third and fourth columns of the matrix, etc. The values of the columns were discretized into the bins of the histograms. An example of this process is shown in Figure 5.8.

Using this process we transform each HTML DOM into a multi-channel image with the following shape:

- Width: 55
- Height: 55
- Channels: 5

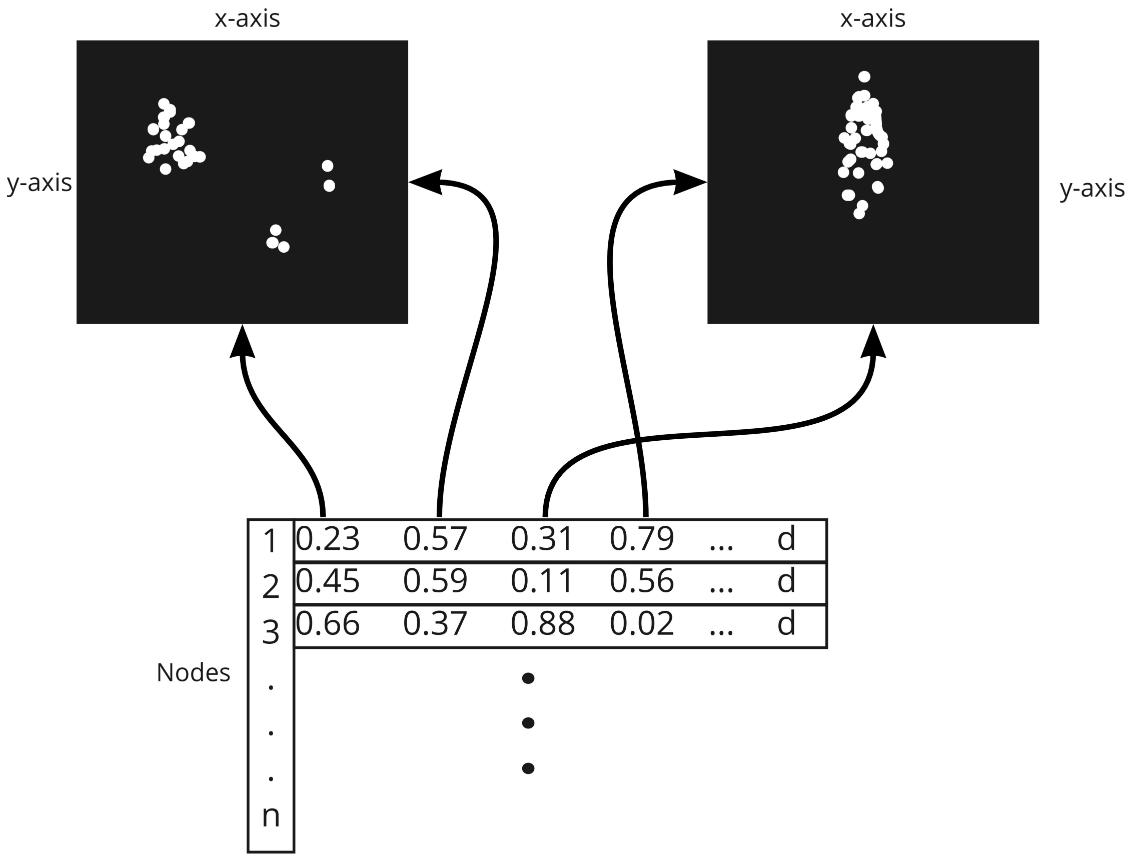

x-axis
y-axis
x-axis
y-axis
Nodes
1 0.23 0.57 0.31 0.79 ... d
2 0.45 0.59 0.11 0.56 ... d
3 0.66 0.37 0.88 0.02 ... d
n

Figure 5.8: There is the PCA node embedding matrix with n rows (nodes) and d columns. The first histogram is created from the first 2 columns of the matrix. The second histogram is created from the second 2 columns of the matrix, etc.

All these images constitute our CETD-DOM-2 data model. Figure 5.9 shows a real example of a multi-channel image from the CETD-DOM-2 data model.

After the data model CETD-DOM-2 is created it is split into training and testing datasets. The training data is used for training parameters. The testing data is used for the final evaluation (Table 5.6).

5.3 Methods

This chapter uses the same methods for the data models CETD-feature-set-1 and CETD-feature-set-2 as in the previous chapter (Section 4.4). For experiments with data models CETD-DOM-1 and CETD-DOM-2, we use Multilayer Perceptron (MLP) and Convolutional Neural Network (CNN) implemented in TensorFlow 2 [41]. The Architectures for both algorithms are described in the experiment sections.

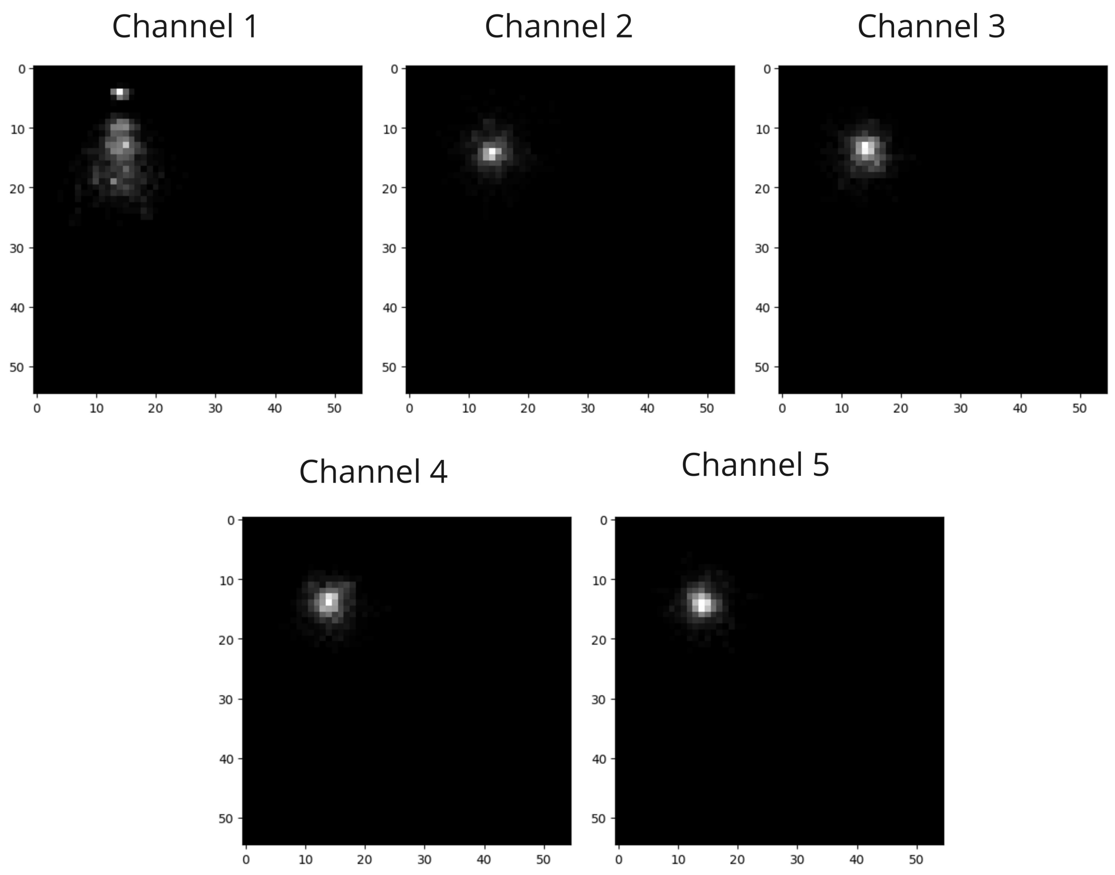


Figure 5.9: An example of an evil twin HTML DOM transformation to multi-channel image.

5.4 Experiments to Detect Evil Twin Websites

To detect evil twin websites, first we make experiments with data models CETD-feature-set-1 and CETD-feature-set-2, in sections 5.4.1 and 5.4.2 respectively. For both data models, experiments have the following structure:

- train models without features selection
- select the best features
- train a model with the best features
- finally evaluate the best trained models in the testing data.

	# Rows	# Positive rows	# Negative rows
CETD-DOM-2	**25,088**	**12,544**	**12,544**
CETD-DOM-2$^{\text{Train}}$	16,808	8,404	8,404
CETD-DOM-2$^{\text{Test}}$	8,280	4,140	4,140

Table 5.6: The CETD-DOM-2 was split into the training and testing parts.

	Random Forest	XGBoost
n_estimators	500	500
max_depth	100	10
min_samples_leaf	1	∅
min_child_weight	∅	5

Table 5.7: The best found hyper-parameters for both algorithms.

	Random Forest		XGBoost	
	mean	std	mean	std
Accuracy	**0.9322**	**0.0065**	**0.9523**	**0.0049**
FPR	0.0566	0.0044	0.0404	0.0046
FNR	0.0790	0.0118	0.0550	0.0086

Table 5.8: Results for Random Forest and XGBoost algorithms with the best hyper-parameters. The mean and std are computed from the 10 folds of the cross validation.

The sections 5.4.3 and 5.4.4 contains experiments with data models CETD-DOM-1 and CETD-DOM-2.

5.4.1 Experiments using the CETD-feature-set-1

5.4.1.1 Training with the Hyper-parameter Tuning

The hyper-parameter tuning with 10-fold cross validation procedure was performed on the CETD-feature-set-1^{Train} data model for Random Forest and XGBoost. The best found hyper-parameters are shown in Table 5.7. The Random Forest algorithm achieved 93.22% of accuracy and XGBoost algorithm achieved 95.23% of accuracy with the selected hyper-parameters (Table 5.8).

5.4.1.2 Training with the Best Features

To find the most important features the FSFI method was performed. The FSFI method was run with the hyper-parameters above and it found a subset of the 93 most important features for Random Forest and a subset of the 170 most important features for XGBoost.

Then the 10-fold cross validation procedure was performed with the subsets of the features for both algorithms with the selected hyper-parameters from Table 5.7. The Random Forest algorithm achieved 93.38% of accuracy and XGBoost algorithm achieved 95.25% of accuracy (Table 5.9).

5.4.1.3 Comparison of the Trained Models

The **Random Forest** achieved better results with the subset of 93 features given by the FSFI method (Table 5.9) than with all features (Table 5.8). All metrics (the validation accuracy, false positive and false negative rates) were improved compared to all features.

	Random Forest		XGBoost	
Best features	93/233		170/233	
	mean	std	mean	std
Accuracy	**0.9338**	**0.0051**	**0.9525**	**0.0063**
FPR	0.0542	0.0049	0.0397	0.0056
FNR	0.0781	0.0097	0.0554	0.0091

Table 5.9: Results for the FSFI method.

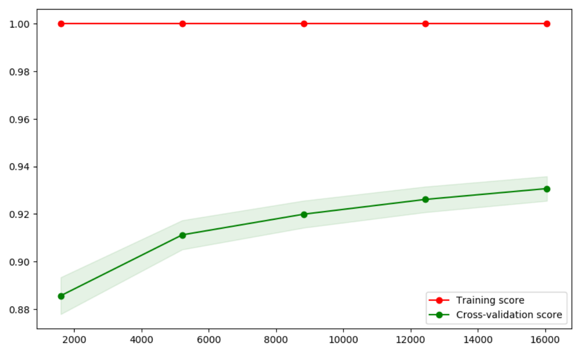


Figure 5.10: Learning curve for Random Forest with the best hyper parameters and subset of the 93 most important features for the CETD-feature-set-1$^{\text{Train}}$.

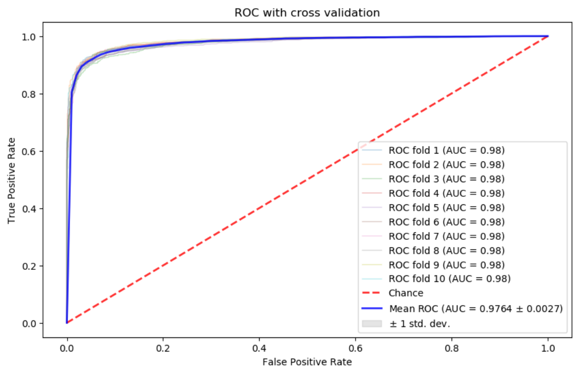


Figure 5.11: AUC ROC curve for Random Forest with the best hyper parameters and with a subset of the 93 most important features for the CETD-feature-set-1$^{\text{Train}}$.

Random Forest Summary:

- n_estimators: 500
- max_depth: 100
- min_samples_leaf: 1
- The most important features: **93**/233
- Accuracy: 93.38%

The Figure 5.10 shows the learning curve with the mentioned setting. We can see clearly that the training score is still around the maximum and the validation score could be increased with more training samples. The Figure 5.11 shows the ROC curve with 10-fold cross validation. The AUC value achieved 0.9764 which means it has a very good measure of separability.

In the case of **XGBoost**, the subset of 170 features given by the FSFI method (Table 5.9) achieved a better results than with all features (Table 5.8). The validation accuracy and false positive rate improved the results compared to all features, except the false negative rate.

XGBoost summary:

- n_estimators: 500
- max_depth: 10
- min_child_weight: 5
- The most important features: **170**/(233)

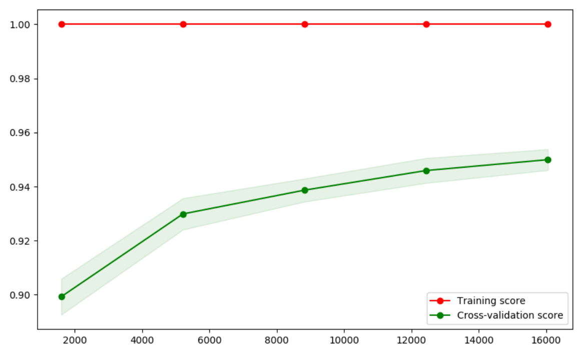


Figure 5.12: Learning curve for XGBoost with the best hyper parameters and subset of the 170 most important features for the CETD-feature-set-1$^{\text{Train}}$.

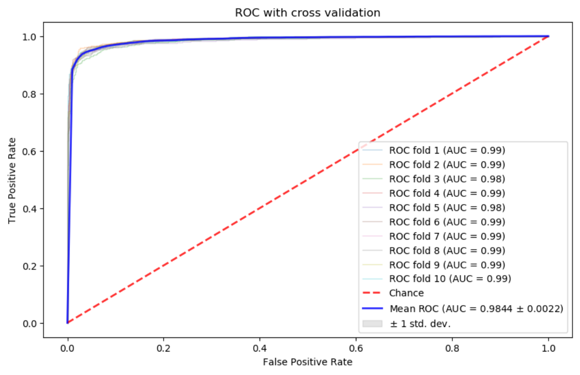


Figure 5.13: AUC ROC curve for XGBoost with the best hyper parameters and with a subset of the 170 most important features for the CETD-feature-set-1$^{\text{Train}}$.

CETD-feature-set-1$^{\text{Test}}$	Random Forest	XGBoost
Testing accuracy	**0.9350**	**0.9528**
FPR	0.0510	0.0407
FNR	0.0789	0.0538

Table 5.10: Result of Random Forest and XGBoost on CETD-feature-set-1$^{\text{Test}}$ data model.

- Accuracy: 95.25%

The Figure 5.12 shows the learning curve during training with the mentioned setting. We can see clearly that the training score is still around the maximum and the validation score could be increased with more training samples. The Figure 4.7 shows the ROC curve with 10-fold cross validation. The AUC value achieved 98.44% which means it also has a good measure of separability.

5.4.1.4 Evaluation on the Testing Data

The final step was to perform the Random Forest and XGBoost with the best found settings on the testing data CETD-feature-set-1$^{\text{Test}}$. The results are shown in the Table 5.10.

5.4.2 Experiments using the CETD-feature-set-2

5.4.2.1 Training with the Hyper-parameter Tuning

The hyper-parameter tuning with 10-fold cross validation procedure was performed on the CETD-feature-set-2$^{\text{Train}}$ data model for Random Forest and XGBoost algorithms. The best found hyper-parameters are shown in Table 5.11. The Random Forest algorithm achieved 93.80% of accuracy and XGBoost algorithm achieved 94.89% of accuracy with the selected hyper-parameters (Table 5.12).

	Random Forest	XGBoost
n_estimators	500	500
max_depth	100	10
min_samples_leaf	1	∅
min_child_weight	∅	10

Table 5.11: The best found hyper-parameters for both algorithms.

	Random Forest		XGBoost	
	mean	std	mean	std
Accuracy	**0.9380**	**0.0065**	**0.9489**	**0.0073**
FPR	0.0710	0.0094	0.0580	0.0102
FNR	0.0531	0.0098	0.0442	0.0085

Table 5.12: Result for Random Forest and XGBoost algorithm with best hyper-parameters. Each metric is a mean of 10 results from 10 folds from cross validation.

5.4.2.2 Training with the Best Features

To find the most important features the FSFI method was performed. The FSFI method was run with the hyper-parameters above and it found a subset of the 158 most important features for Random Forest and a subset of the 229 most important features for XGBoost.

Then the 10-fold cross validation procedure was performed with the subsets of the features for both algorithms with the selected hyper-parameters from Table 5.11. The Random Forest algorithm achieved 93.85% of accuracy and XGBoost algorithm achieved 94.96% of accuracy (Table 5.13).

5.4.2.3 Comparison of the Trained Models

The Random Forest achieved better results with the subset of 158 features given by the FSFI method (Table 5.13) than with all features (Table 5.12). The accuracy, false positive rate were improved but the false negative rate was not improved. Despite the worse false negative rate the results with the subset of 158 features are still better.

Random Forest Summary:

- n_estimators: 500
- max_depth: 100

	Random Forest		XGBoost	
Best features	158/310		229/310	
	mean	std	mean	std
Accuracy	**0.9385**	**0.0066**	**0.9496**	**0.0068**
FPR	0.0690	0.0096	0.0569	0.0093
FNR	0.0540	0.0097	0.0439	0.0072

Table 5.13: Results for the FSFI method.

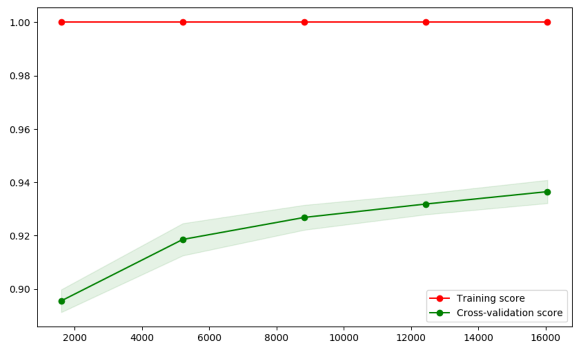


Figure 5.14: Learning curve for Random Forest with the best hyper-parameters and subset of 158 the most important features for the CETD-feature-set-2^{Train}.

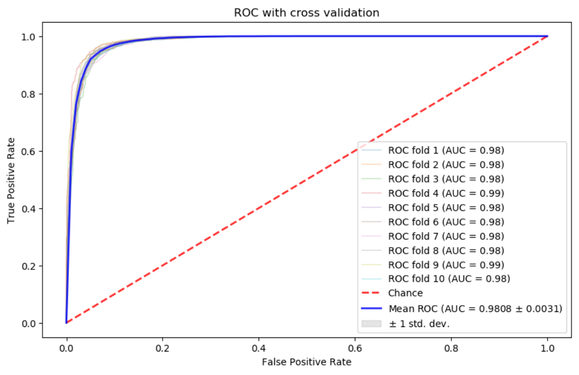


Figure 5.15: AUC ROC curve for Random Forest with the best hyper-parameters and with a subset of 158 the most important features with the all features for the CETD-feature-set-2^{Train}.

- min_samples_leaf: 1
- The most important features: **158**/310
- Accuracy: 93.85%

The Figure 5.14 shows the learning curve with the mentioned setting. We can see clearly that the training score is still around the maximum and the validation score could be increased with more training samples. The Figure 5.15 shows the ROC curve with 10-fold cross validation. The AUC value achieved 0.9808 which means it has a very good measure of separability.

The **XGBoost** algorithm achieved better results with the subset of 229 features given by the FSFI method (Table 5.13) than with all features (Table 5.12). All metrics (the validation accuracy, false positive and false negative rates) were improved compared to all features.

XGBoost summary:

- n_estimators: 500
- max_depth: 10
- min_child_weight: 10
- The most important features: **229**/310
- Accuracy: 94.96 %

The Figure 5.16 shows the learning curve with the mentioned setting. We can see clearly that the training score is still around the maximum. The validation score could be increased with more training samples. The Figure 5.17 shows the ROC curve with 10-fold cross validation. The AUC value achieved 0.9851 which means it has good measure of separability.

5.4.2.4 Evaluation on the Testing Data

The final step was to perform the Random Forest and XGBoost with the best found settings on the testing data CETD-feature-set-2^{Test}. The results are shown in the Table 5.14.

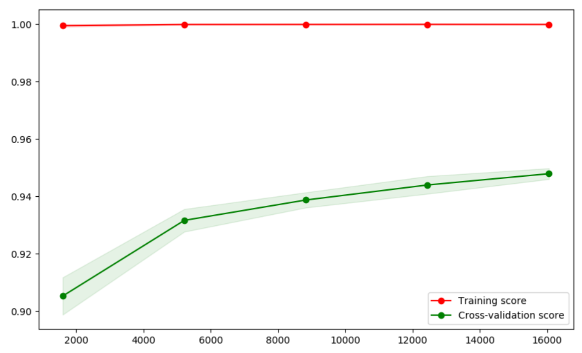


Figure 5.16: Learning curve for XGBoost with best hyper parameters and with a subset of the 229 most important features for the CETD-feature-set-2$^{\text{Train}}$.

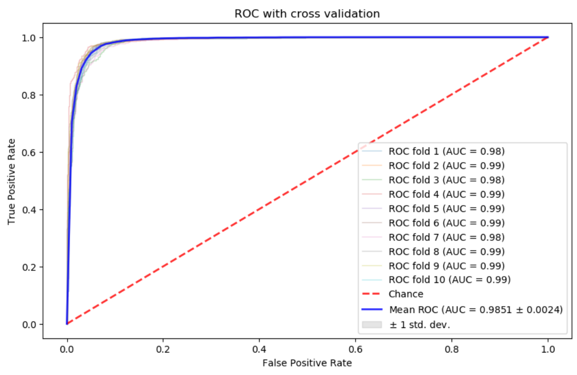


Figure 5.17: ROC for XGBoost with best hyper parameters and with a subset of the 229 most important features for the CETD-feature-set-2$^{\text{Train}}$.

CETD-feature-set-2$^{\text{Test}}$	Random Forest	XGBoost
Testing accuracy	**0.9370**	**0.9519**
FPR	0.0694	0.0542
FNR	0.0566	0.0418

Table 5.14: Result of Random forest and XGBoost on CETD-feature-set-2$^{\text{Test}}$ data model.

5.4.3 Experiments using the CETD-DOM-1

We apply two machine learning algorithms on the data model CETD-DOM-1. First we trained XGBoost algorithm on CETD-DOM-1$^{\text{Train}}$ and subsequently Multilayer perceptron.

The XGBoost was trained and turned on the CETD-DOM-1$^{\text{Train}}$ with a 10 fold cross validation procedure. The best found parameters were:

- n_estimators: 500
- max_depth: 10
- min_child_weight: 10

Then this trained model was performed on the testing data CETD-DOM-1$^{\text{Test}}$. It achieved testing accuracy of 90% with 11.84% of false positive rate and 8.16% of the false negative rate. The results are shown in the Table 5.15.

Then the multilayer perceptron was used. For the training procedure we split the CETD-DOM-1$^{\text{Train}}$ into training and validation parts with a ratio of 80:20. The training part was used only for training and the validation data was used for checking performance of the trained model. We tried multiple architecture and the best result on the validation data was achieved after 85 epochs of the training with the validation accuracy of 81.12% with the following architecture:

- An input layer $\in R^{257}$ (as a number of columns in the data model)
- A dense layer $\in R^{100}$

CETD-DOM-1$^{\text{Test}}$	XGBoost	MLP
Testing accuracy	**0.9000**	**0.7909**
FPR	0.1184	0.2360
FNR	0.0816	0.1821

Table 5.15: The results for XGBoost and MLP on CETD-DOM-1$^{\text{Test}}$ data model.

CETD-DOM-2$^{\text{Test}}$	LeNet
Testing accuracy	**0.7072**
FPR	0.1536
FNR	0.4377

Table 5.16: The results for CETD-DOM-2$^{\text{Test}}$ data model.

- A dropout layer with 0.1 rate of dropping
- A dense layer $\in R^50$
- A dense layer $\in R^10$
- A output dense layer $\in R^2$ with softmax activation function

In addition, we used L2 regularization with the value of 0.001 and label smoothing with value 0.1 and the model was trained with batch sizes of 100 samples. Then this trained model was applied on testing data CETD-DOM-1$^{\text{Test}}$. The testing accuracy was 79.09%. The results are shown in Table 5.15.

5.4.4 Experiments using the CETD-DOM-2

Since the data model CETD-DOM-2 contains multichannel images, this experiment is a computer vision task. We used two types of CNN architecture. The first architecture was our implementation based on ResNet [42] and the second architecture was our implementation of LeNet [43] proposed in the original work Graph classification with 2D convolutional neural networks [38].

First, we split the CETD-DOM-2$^{\text{Train}}$ into training and validation parts with a ratio of 80:20. The training part was used for the training of both architectures and the validation part was used for checking the performance of the model. During the training phase, we tried many variants and hyperparameters for the ResNet architecture, but the validation accuracy was not higher than 55%. The LeNet architecture worked better (Figure 5.18). With the hyper-parameters L2 regularization of 0.001 and label smoothing of 0.1, the best trained model achieved validation accuracy of 71.28%.

Then we evaluated the trained model on testing data CETD-DOM-2$^{\text{Test}}$. The testing accuracy was 70.72%, false positive rate was 15.36% and false negative rate was 43.77%. The results are shown in Table 5.16.

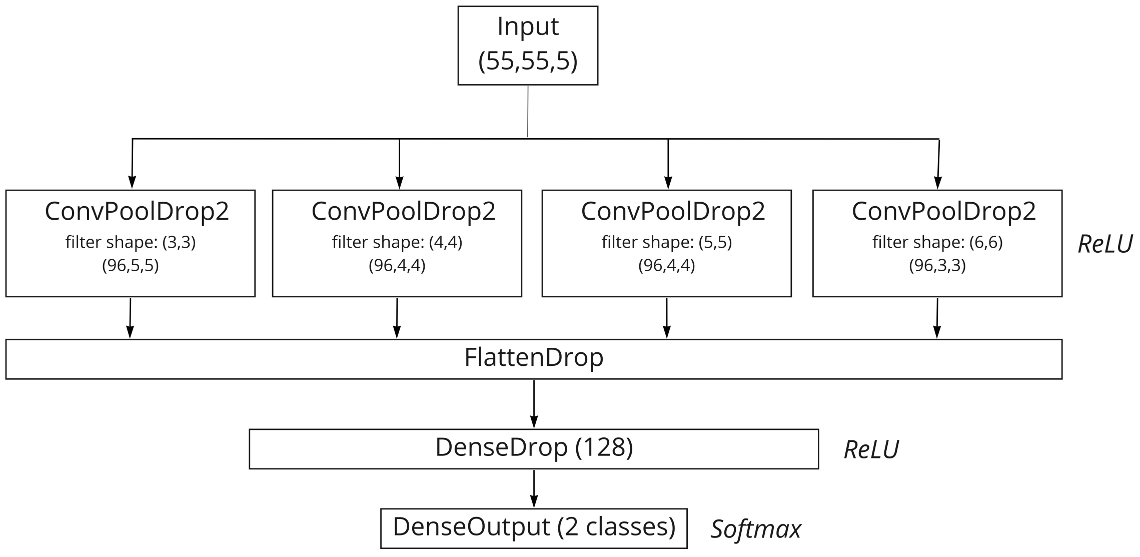


Figure 5.18: LeNet architecture - The input shape of multichannel images is (55,55,5), therefore the input layer has the same shape. Then, there are four convolutional-pooling layers (each repeated twice) in parallel. with the filter sizes of 3, 4, 5 and 6, followed by two fully connected layers. All this is flatten to and sent to the Dense layer with a drop out rate of 0.3. Finally there are output layers consisting of 2 units (negative class, positive class).

5.5 Analysis of Results

5.5.1 CETD-feature-set-1 and CETD-feature-set-2

According to results on the testing data shown in Tables 5.10 and 5.14, the XGBoost model achieved the best accuracy for both data models CETD-feature-set-1^{Test} and CETD-feature-set-2^{Test}. The best model is XGBoost as show in Table 5.10 trained and tuned using the CETD-feature-set-1^{Train} dataset because it achieved the highest accuracy with a value of 95.28% and the lowest false positive rate of 4.07% on the testing data.

These results suggest that the feature-set-1 provides a better description than the feature-set-2 for detection of evil twin websites. This fact is somewhat surprising, because the feature-set-2 contains **more** complex information from the urlscan.io service so we expected better results for models trained with the feature-set-2. Furthermore, the UMAP reduction method showed us also a better outcome 5.2.1.1 for data model CETD-feature-set-2.

5.5.2 CETD-DOM-1 and CETD-DOM-2

According to results on the testing data shown in Tables 5.15 and 5.16, the XGBoost has also the the best results. With 90% of testing accuracy, XGBoost achieved significantly better results then MLP (Tables 5.15) and LeNet (Tables 5.16). The XGBoost result shows that the number of occurrences of html tags in the HTML DOM is important aspect for detection of evil twin websites.

The bad results for the LeNet architecture can have several reasons:

- Wrong selected parameters for node2vec during generating embedding for the HTML DOM trees
- Wrong architecture of CNN
- The method transforming graphs to multichannel images is not suitable for our problem.

5.5.3 The best model for the Detection of Evil Twin Websites

To conclude this chapter, the best found method for the detection of evil twin websites is XGBoost algorithm trained and evaluated on the CETD-feature-set-1 with the best 170 features from feature-set-1. All 170 features with their importance are shown in the section A.5. Here we provide the 10 most important features in the following lines:

- **127_secureRequests:** Total number of request that use HTTPS correctly.
- **11_https_in_url:** Ratio between links on the web page having HTTPS in the URL and all links on the web page.
- **195_response_connectStart_std:** Standard deviation of list of durations, when the browser started to connect to the remote host.
- **48_ips_std:** Mean number of IP addresses used between the client and the web server.
- **97_domain_in_initiators_mean:** Mean of of domain initiators.
- **242_links_domain_in_url_mean:** Ratio of links containing the domain of the website to all links.
- **9_urls:** Total number of URLS on the web page.
- **231_cookie_expires_now_mean:** Mean of all the remaining time in seconds for when all cookie will expire.
- **131_totalLinks:** Total amount of links contained in the web page.
- **105_ips_mean:** Mean of number of IP addresses used for domain.

Chapter 6

Exploiting websites

The goal of this chapter was to create a dataset containing exploiting websites and then to propose suitable methods to detect them. Unfortunately, despite analysing thousands of potentially exploiting websites, none was found. Therefore, instead of experiments using machine learning methods, this chapter only describes our approach for hunting exploiting websites.

Before generating an exploiting website dataset, we wanted to find some method giving us proof that a website is a really an exploiting website. Therefore we used a tool called Thug. The Thug tool [8] is a Python low-interaction honey-client to detect client-side attacks on websites. The goal of Thug is to mimic the behavior of a web browser controlled by a user and be exploited by the content of the website. A core part of this tool is the Google's V8 JavaScript engine [44] for analyzing malicious JavaScript code and the Libemu library [45] to detect and emulate shellcode.

The Thug tool provides more than 40 different user agents for visiting the target website. Each user agent is a string that represents a specific version of an operating system, browser and sometimes libraries. For example:

- Windows XP - Internet Explorer 6.0
- Windows XP - Internet Explorer 6.1
- Windows 10 - Internet Explorer 11.0
- MacOS X 10.7.2 - Safari 5.1.1
- iPad, iOS 7.0.4 - Safari 7.0
- Linux - Safari 7.0

The output of Thug tool contains the analysis of the website with information about the exploits found. This information was converted into the features of our dataset.

After analyzing the features that were going to be extracted from Thug, we looked for a list of potentially exploiting websites. We found a list of URLs [24] containing over 9,000 URLs with malware, ransomware, trojans and other threats.

We ran the Thug tool with all the possible user agents in the Thug tool (which are more than 20 different user agents) to analyse all these URLs from every possible perspective. However, Thug did not find any exploit for any of these websites. This may mean that the selected websites did not contain any exploit or that Thug failed to detect them. This investigation indicates that hunting for exploiting websites may be a harder task compared to evil twin websites.

Chapter 7

Should I Click Web Service

The research methods and techniques described in this thesis may be of great value for our community, since there is a special focus on detecting websites that attack the clients. As member of the Civilsphere project I had the opportunity to implement some of these methods in a public service for our society called the **Should I Click** service and that can be found in <https://www.shouldiclick.org/>. Should I Click is a free service to check if a website is safe to access. The aim of this project is to help people at risk around the world, such as journalists, NGOs, political activists and other people at risk against targeted cyber attacks. However, its an open and public service that everyone can use. Figure 7.1 shows the main page of the service.

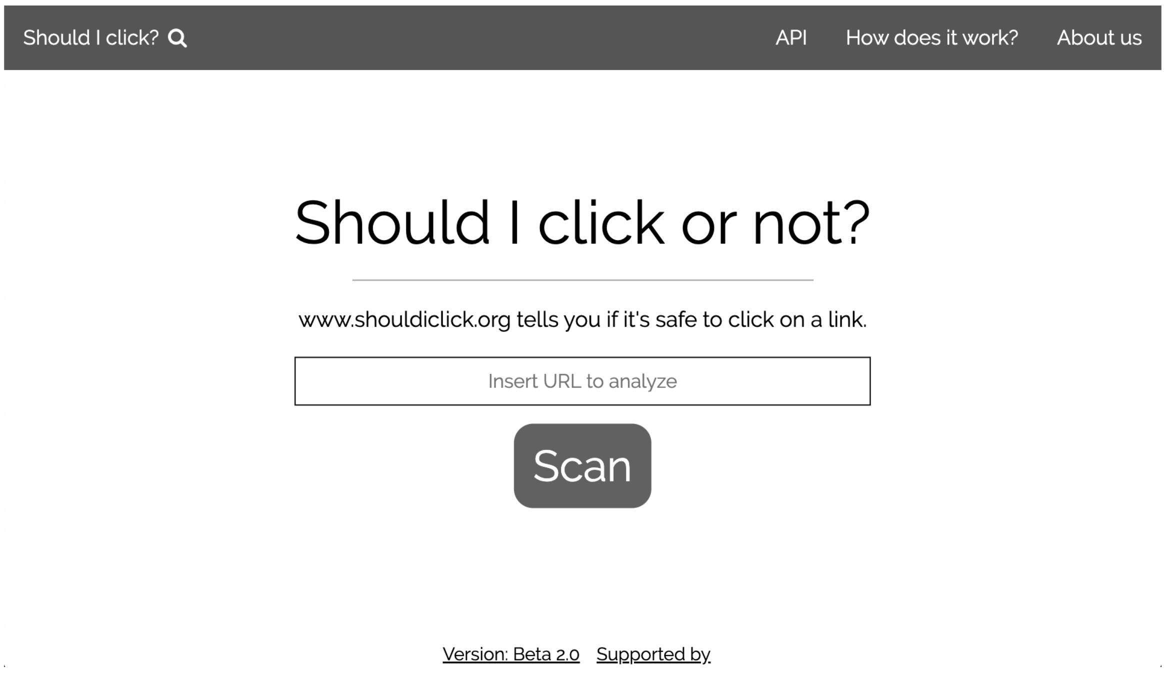


Figure 7.1: Main page of the Should I Click service that was created to implement some of the methods in this thesis for the free benefit of our community. The site is https://www.shouldiclick.org.

The idea of creating the Should I Click service was born from the problems seen in the day to day work of the Civilsphere service, which deals with helping people at risk from cyberattacks.

The first testing version of Should I Click was launched in 2018 but the official launch for the community was in February 2020. Since the main part of this thesis was done after the official launch, Should I Click contains only the variant of XGBoost model trained on the CETD-feature-set-1 (5.4.1). Future releases will incorporate all the detection models shown in this thesis.

Figure 7.2 shows the architecture of Should I Click running inside a docker container in the Czech Technical University in Prague. The web server was constructed using NGINX [46], Django [47] and uWSGI [48]. These three programs constitute a bridge between requests from clients and the python modules. When a client sends a request to analyse a URL, this request is stored in a redis database [49] and the detection module is informed about it. The detection module sends the URL to the urlscan.io API to analyse the URL and retrive a JSON file with results (the analysis takes around 30 seconds). After that, the urlscan.io JSON file is downloaded and all the features are extracted from it. The detection module predicts if this URL is safe to click or not, and stores the detection result for this URL in the redis database. The client browser keeps asking for the result every few seconds. When the detection result is ready, it is shown in client's browser.

The important part of Should I Click is that clients can provide feedback for its detection results. When a detection result is shown to a client, the client can choose if the result is right or not, and the reason. This possibility allows us to measure our performance in real websites and against the testing data.

Figure 7.3 shows an example run of the Should I Click service with the webpage `http://storecovid.blog` which is detected as suspicious and should not be clicked.

The Should I Click service has ran for 102 days since its launch in February 2020. During this time we have collected 1,529 submitted URLs and 285 feedback forms for them. Table 7.1 shows the total results based on the feedback sent by users and during the whole lifetime of the service. It can be clearly seen that the false negative rate is very high with 48.21% and the false positive rate is better with 19.08%. This means that our detection module is better in classifying legitimate websites while in case of classification of unsafe websites, our models fail often. This collection of data is very important for the future improvements in the service.

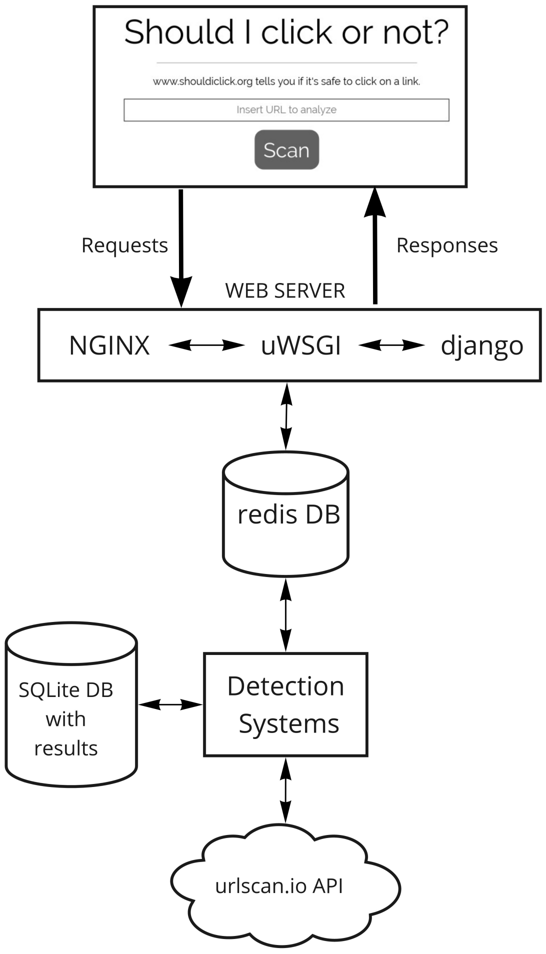
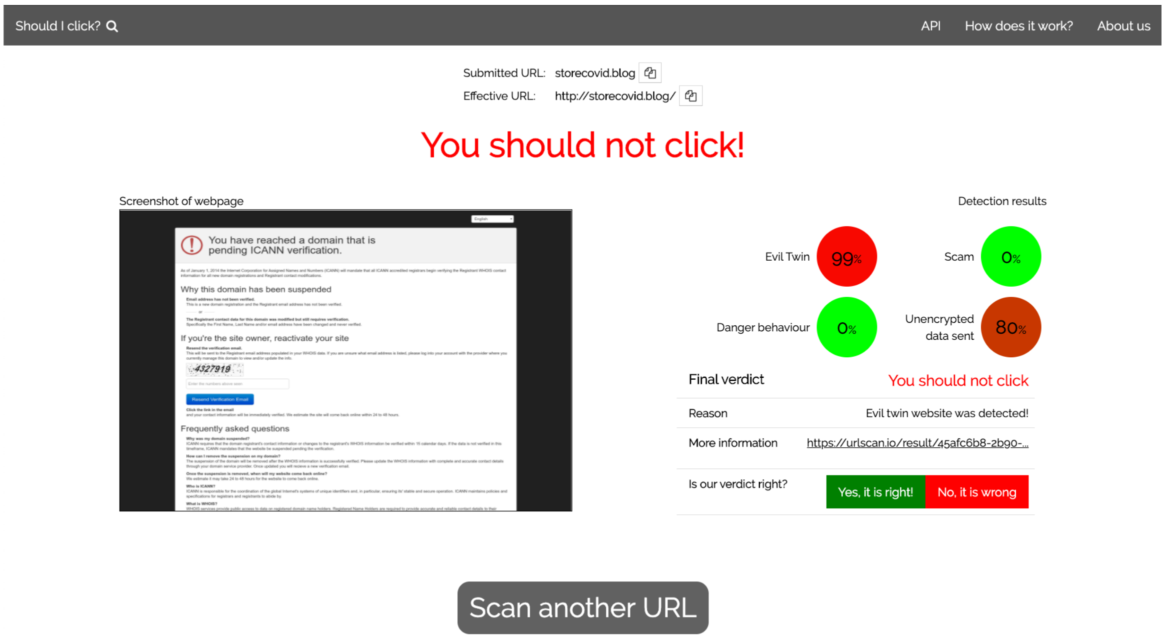


Figure 7.2: Web architecture of the implementation of the Should I Click service

All requests	1,529
Detected as unsafe websites	438
Detected as legitimate websites	1,091
All feedback	285
TN	140
FP	33
FN	54
TP	58
Accuracy	0.6947
FPR	0.1908
FNR	0.4821

Table 7.1: Statistics for Should I Click


Should I click?

API How does it work? About us

Submitted URL: storecovid.blog

Effective URL: http://storecovid.blog/

You should not click!

Screenshot of webpage

You have reached a domain that is pending ICANN verification.

Why this domain has been suspended

If you're the site owner, reactivate your site

Frequently asked questions

Detection results

Evil Twin	99%
Scam	0%
Danger behaviour	0%
Unencrypted data sent	80%

Final verdict	You should not click
Reason	Evil twin website was detected!
More information	https://urlscan.io/result/45afc6b8-2b90-...
Is our verdict right?	Yes, it is right! No, it is wrong

Scan another URL

Figure 7.3: Example analysis of a website by the Should I Click service. The web page http://storecovid.blog is detected as suspicious and suggested not to be clicked on.

Chapter 8

Conclusion

The detection of unsafe websites is one of the most important security topics nowadays given the amount of attacks, phishing and scams that are run on the Internet. To be able to have a way to determine which site is unsafe and which one is safe is a very hard problem for our industry.

The aim of our research was to analyse and detect unsafe websites with special attention to evil twin websites.

The contributions of this thesis are: This thesis publishes two balanced datasets that are publicly available [21] and they can be used for further research. The first dataset is the UWD dataset containing all types of unsafe websites, and legitimate websites. The second one is the CETD dataset containing only evil twin websites and legitimate websites. First, we proposed the detection of unsafe websites based on the UWD dataset. We used extracted features from urlscan.io analysis to describe the content and behaviour of a website. Using these features, Random forest and XGboost algorithms achieved 92.69 percent of accuracy for detection of unsafe websites. Second, we proposed the detection of evil twin websites using the CETD dataset. In this case we also used features based on urlscan.io analysis, moreover, we used the HTML DOM describing the structure of a web page. Using features extracted from urlscan.io analysis we achieved accuracy of 95.28 percent. In the case of the detection based on HTML DOM the best result was 90 percent accuracy. As a practical outcome of this research we created the Should I Click web service (https://www.shouldiclick.org/) to check whether a website is safe to visit or not.

We conclude that the content, behaviour and structure of websites is an important aspect for detection of unsafe and evil twin websites and instead of techniques relying only on URLs it is better to combine both approaches together.

The future work consists of several parts:

- Should I Click service uses only one model from chapter 5. In the future, our goal is to create an ensemble of all detection methods mentioned in this thesis.
- Should I Click uses urlscan.io web service for generating urlscan.io analysis used for extracting features. In the future, we would like to replace urlscan.io with our own solution. This would make the analysis of a website submitted by a user, faster. Currently urlscan.io analyses the web page for 30 seconds and we have to wait for it.

- This research shows us that content, behaviour and structure of a website plays an essential role for their detection. In the future, we would like to focus more on combination of HTML structure with text, images and JavaScript code of web page. We believe that this data can provide a more complex description of a website and thus better detection results in this area.

Bibliography

[1] Google transparency report. Accessed May 10th 2020. [Online]. Available: <https://transparencyreport.google.com/safe-browsing/overview>

[2] Phishing techniques. Accessed May 10th 2020. [Online]. Available: <https://www.phishing.org/phishing-techniques>

[3] S. Sheng, B. Wardman, G. Warner, L. F. Cranor, J. I. Hong, and C. Zhang, "An empirical analysis of phishing blacklists," in *CEAS 2009*, 2009.

[4] A. Joshi, L. Lloyd, P. Westin, and S. Seethapathy, "Using lexical features for malicious url detection – a machine learning approach," 2019.

[5] R. Vinayakumar, S. Sriram, K. Soman, and A. Mamoun. (2020) Malicious url detection using deep learning. [Online]. Available: <https://www.techrxiv.org/articles/Malicious_URL_Detection_using_Deep_Learning/11492622/1>

[6] M. Chatterjee and A. Namin, "Detecting phishing websites through deep reinforcement learning," in *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, vol. 2, 2019, pp. 227–232.

[7] J. Gilger. urlscan.io. Accessed May 20th 2020. [Online]. Available: <https://urlscan.io>

[8] A. Dell'Aera. Thug. Accessed May 20th 2020. [Online]. Available: <https://thug-honeyclient.readthedocs.io/en/latest/>

[9] Code for the thesis. Accessed May 10th 2020. [Online]. Available: <https://github.com/frenky-strasak/diploma_thesis_shouldiclick>

[10] Verizon. 2019 data breach investigations report. Accessed May 10th 2020. [Online]. Available: <https://enterprise.verizon.com/resources/reports/2019-data-breach-investigations-report.pdf>

[11] Common vulnerability and exposure (cve) list. Accessed May 10th 2020. [Online]. Available: <https://cve.mitre.org/>

[12] National vulnerability database (nvd). Accessed May 10th 2020. [Online]. Available: <https://nvd.nist.gov/>

[13] Malwarebytes - exploit kits: fall 2019 review. Accessed May 10th 2020. [Online]. Available: <https://blog.malwarebytes.com/exploits-and-vulnerabilities/2019/11/exploit-kits-fall-2019-review/>

[14] J. Saxe, R. Harang, C. Wild, and H. Sanders, "A deep learning approach to fast, format-agnostic detection of malicious web content," 2018.

[15] S. Hess, P. Satam, G. Ditzler, and S. Hariri, "Malicious html file prediction: A detection and classification perspective with noisy data," in *2018 IEEE/ACS 15th International Conference on Computer Systems and Applications (AICCSA)*, 2018, pp. 1–7.

[16] W. Xu, F. Zhang, and S. Zhu, "The power of obfuscation techniques in malicious javascript code: A measurement study," in *2012 7th International Conference on Malicious and Unwanted Software*, 2012, pp. 9–16.

[17] S. Ndichu, S. Ozawa, T. Misu, and K. Okada, "A machine learning approach to malicious javascript detection using fixed length vector representation," in *2018 International Joint Conference on Neural Networks (IJCNN)*, 2018, pp. 1–8.

[18] H. Zhang, G. Liu, T. W. S. Chow, and W. Liu, "Textual and visual content-based anti-phishing: A bayesian approach," *IEEE Transactions on Neural Networks*, vol. 22, no. 10, pp. 1532–1546, 2011.

[19] Guang-Gang Geng, Xiao-Dong Lee, Wei Wang, and Shian-Shyong Tseng, "Favicon - a clue to phishing sites detection," in *2013 APWG eCrime Researchers Summit*, 2013, pp. 1–10.

[20] J. C. S. Fatt, C. K. Leng, and S. S. Nah, "Phishdentity: Leverage website favicon to offset polymorphic phishing website," in *2014 Ninth International Conference on Availability, Reliability and Security*, 2014, pp. 114–119.

[21] Unsafe and evil twin website datasets. Accessed May 10th 2020. [Online]. Available: <https://github.com/stratosphereips/shouldiclick_datasets>

[22] Phishtank. Accessed May 10th 2020. [Online]. Available: <https://www.phishtank.com/>

[23] Cyber threat intelligence feeds (ctifeeds). Accessed May 10th 2020. [Online]. Available: <https://github.com/certtools/intelmq-feeds-documentation>

[24] The big list of hacked malware web sites. Accessed May 10th 2020. [Online]. Available: <https://github.com/mitchellkrogza/The-Big-List-of-Hacked-Malware-Web-Sites>

[25] Alexa top 1 million sites. Accessed May 10th 2020. [Online]. Available: <http://s3.amazonaws.com/alexa-static/top-1m.csv.zip>

[26] Virus total. Accessed May 10th 2020. [Online]. Available: <https://www.virustotal.com/>

[27] Let's encrypt. Accessed May 10th 2020. [Online]. Available: <https://letsencrypt.org/>

[28] L. McInnes, J. Healy, and J. Melville, "Umap: Uniform manifold approximation and projection for dimension reduction," 2018.

[29] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[30] L. Breiman, "Random forests," *Machine Learning*, vol. 45, pp. 5–32, 2001.

[31] Tuning the hyper-parameters of an estimator. Accessed May 10th 2020. [Online]. Available: <https://scikit-learn.org/stable/modules/grid_search.html>

[32] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 785–794. [Online]. Available: <https://doi.org/10.1145/2939672.2939785>

[33] J. Brownlee. Xgboost with python, machine learning mastery. Accessed May 10th 2020. [Online]. Available: <https://machinelearningmastery.com/feature-importance-and-feature-selection-with-xgboost-in-python/>

[34] Phishing domain database. Accessed May 10th 2020. [Online]. Available: <https://github.com/mitchellkrogza/Phishing.Database>

[35] Openphish. Accessed May 10th 2020. [Online]. Available: <https://openphish.com/>

[36] Sp tool. Accessed May 10th 2020. [Online]. Available: <https://github.com/garee/sp>

[37] Startpage. Accessed May 10th 2020. [Online]. Available: <https://www.startpage.com/>

[38] A. J.-P. Tixier, G. Nikolentzos, P. Meladianos, and M. Vazirgiannis, "Graph classification with 2d convolutional neural networks," 2017.

[39] A. Grover and J. Leskovec, "Node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 855–864. [Online]. Available: <https://doi.org/10.1145/2939672.2939754>

[40] H. Hotelling, *Analysis of a complex of statistical variables into principal components.* Baltimore: Warwick & York, 1933, 48 p. [Online]. Available: <http://hdl.handle.net/2027/wu.89097139406>

[41] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: <http://tensorflow.org/>

[42] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[43] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, November 1998.

[44] V8. Accessed May 10th 2020. [Online]. Available: <https://v8.dev/>

[45] Libemu. Accessed May 10th 2020. [Online]. Available: <http://libemu.carnivore.it/>

[46] Nginx. Accessed May 10th 2020. [Online]. Available: <https://nginx.org/>

[47] Django. Accessed May 10th 2020. [Online]. Available: <https://www.djangoproject.com/>

[48] uwsgi. Accessed May 10th 2020. [Online]. Available: <https://uwsgi-docs.readthedocs.io/en/latest/>

[49] Redis. Accessed May 10th 2020. [Online]. Available: <https://redislabs.com/>

Appendix A

Description of Features

A.1 feature-set-1 - 233 features

All these features are computed from the urlscan.io analysis. To see an example of urlscan.io analysis, go to urlscan.io website, choose some submitted URL and then there is a button called "API". If you click on it, the urlcan.io analysis will be shown in JSON format. These features are computed from this JSON output. The total number of the features is 233. The last feature has index 246, because some features were removed from the original list.

Features computed from the 'lists' JSON key. Feature numbers: 1 - 26:

- **1_number_ips**: Total length of list of 'ips' list.
- **2_number_countries**: Total length of list of 'countries' list.
- **3_number_asns**: Total length of list of 'asns' list.
- **4_mean_asns**: Mean of values in 'asns' list.
- **5_std_asns**: STD of values in 'asns' list.
- **6_number_domains**: Total length of list of 'domains' list.
- **7_diff_domains**: Total number of different TLD of the domains int 'domains' list.
- **8_servers**: Total length of list of 'server' list.
- **9_urls**: Total length of list of 'url' list.
- **10_diff_urls**: Total number of different TLD of the domains int 'url' list.
- **11_https_in_url**: Ratio between links on the web page having 'https' in the URL and all links on the web page.
- **12_domain_in_url**: Ratio of different domains in 'url list' and all domains int the 'domain' list.
- **13_image_in_url**: Number of urls in 'url' list containing '.jpeg', '.png', '.jpg', '.gif'.
- **14_sizemean_url**: Mean of length of urls in 'url' list.
- **15_sizestd_url**: STD of length of urls in 'url' list.
- **16_javascript_url**: Ratio of number of urls in 'url' list containing '.js' and all urls.
- **17_javascript_url**: Ratio of number of urls in 'url' list containing '.cookie' and all urls.
- **18_length_linkdomains**: Total length of list of 'linkDomains' list.

- **19_difftld_linkdomains**: Ratio of number of different TLD of the domains int 'url' list and all 'urls'.
- **20_numsubdomainmean_linkdomains**: Each url is split by '.' to several parts. Mean of the number of these parts is computed.
- **21_numsubdomainstd_linkdomains**: Each url is split by '.' to several parts. STD of the number of these parts is computed.
- **22_numcertificates_linkdomains**: Total numner of 'certificates' list.
- **23_certicatevalidationmean_linkdomains**: Mean of validity length of certificates.
- **24_certicatevalidationstd_linkdomains**: STD of validity length of certificates.
- **25_certicatevalidation2mean_linkdomains**: Mean of amount of seconds for each certificate remaining to expire time.
- **26_certicatevalidation2std_linkdomains**: STD of amount of seconds for each certificate remaining to expire time.

Features computed from the 'meta' JSON key. Feature numbers: 27 - 31:

- **27_diff_countries**: Number of different countries for all object with json path ['processors']['geoip']['data']
- **28_confidence_mean**: Mean of 'confidenceTotal' list.
- **29_confidence_std**: STD of 'confidenceTotal' list.
- **30_priority_mean**: Mean of 'priority' list.
- **31_priority_std**: STD of 'priority' list.

Features computed from the 'stats' JSON key. Feature numbers: 37 - 132:

- **37_count_mean**: Mean of 'count' values in 'resourceStats'.
- **38_count_std**: STD of 'count' values in 'resourceStats'.
- **39_size_mean**: Mean of 'https' values in 'resourceStats'.
- **40_size_std**: STD of 'https' values in 'resourceStats'.
- **41_ensize_mean**: Mean of 'encodedSize' values in 'resourceStats'.
- **42_ensize_std**: STD of 'encodedSize' values in 'resourceStats'.
- **43_latency_mean**: Mean of 'latency' values in 'resourceStats'.
- **44_latency_std**: STD of 'latency' values in 'resourceStats'.
- **45_compression_mean**: Mean of 'compression' values in 'resourceStats'.
- **46_compression_std**: STD of 'compression' values in 'resourceStats'.
- **47_ips_mean**: Mean of number of ips in 'ips' list.
- **48_ips_std**: STD of number of ips in 'ips' list.
- **49_countries_mean**: Mean of number of countries in 'countries' list.
- **50_countries_std**: STD of number of countries in 'countries' list.
- **51_count_mean**: Mean of 'count' values in 'protocolStats'.
- **52_count_std**: STD of 'count' values in 'protocolStats'.
- **53_size_mean**: Mean of 'https' values in 'protocolStats'.
- **54_size_std**: STD of 'https' values in 'protocolStats'.
- **55_ensize_mean**: Mean of 'encodedSize' values in 'protocolStats'.
- **56_ensize_std**: STD of 'encodedSize' values in 'protocolStats'.

- **57_ips_mean**: Mean of number of ips in 'ips' list in 'protocolStats'.
- **58_ips_std**: STD of number of ips in 'ips' list in 'protocolStats'.
- **59_countries_mean**: Mean of number of countries in 'countries' list in 'protocolStats'.
- **60_countries_std**: STD of number of countries in 'countries' list in 'protocolStats'.
- **61_count_mean**: Mean of 'count' values in 'tlsStats'.
- **62_count_std**: STD of 'count' values in 'tlsStats'.
- **63_size_mean**: Mean of 'https' values in 'tlsStats'.
- **64_size_std**: STD of 'https' values in 'tlsStats'.
- **65_ensize_mean**: Mean of 'encodedSize' values in 'tlsStats'.
- **66_ensize_std**: STD of 'encodedSize' values in 'tlsStats'.
- **67_ips_mean**: Mean of number of ips in 'ips' list in 'tlsStats'.
- **68_ips_std**: STD of number of ips in 'ips' list in 'tlsStats'.
- **69_countries_mean**: Mean of number of countries in 'countries' list in 'tlsStats'.
- **70_countries_std**: STD of number of countries in 'countries' list in 'tlsStats'.
- **71_protocols_mean**: Mean of length of protocols in 'protocols' list in 'tlsStats'.
- **72_protocols_std**: STD of length of protocols in 'protocols' list in 'tlsStats'.
- **73_count_mean**: Mean of 'count' values in 'serverStats'.
- **74_count_std**: STD of 'count' values in 'serverStats'.
- **75_size_mean**: Mean of 'https' values in 'serverStats'.
- **76_size_std**: STD of 'https' values in 'serverStats'.
- **77_ensize_mean**: Mean of 'encodedSize' values in 'serverStats'.
- **78_ensize_std**: STD of 'encodedSize' values in 'serverStats'.
- **79_ips_mean**: Mean of number of ips in 'ips' list in 'serverStats'.
- **80_ips_std**: STD of number of ips in 'ips' list in 'serverStats'.
- **81_countries_mean**: Mean of number of countries in 'countries' list in 'serverStats'.
- **82_countries_std**: STD of number of countries in 'countries' list in 'serverStats'.
- **83_count_mean**: Mean of 'count' values in 'domainStats'.
- **84_count_std**: STD of 'count' values in 'domainStats'.
- **85_size_mean**: Mean of 'https' values in 'domainStats'.
- **86_size_std**: STD of 'https' values in 'domainStats'.
- **87_ensize_mean**: Mean of 'encodedSize' values in 'domainStats'.
- **88_ensize_std**: STD of 'encodedSize' values in 'domainStats'.
- **89_ips_mean**: Mean of number of ips in 'ips' list in 'domainStats'.
- **90_ips_std**: STD of number of ips in 'ips' list in 'domainStats'.
- **91_countries_mean**: Mean of number of countries in 'countries' list in 'domainStats'.
- **92_countries_std**: STD of number of countries in 'countries' list in 'domainStats'.
- **93_redirects_mean**: Mean of number of redirects for 'redirects' values in 'domainStats'.
- **94_redirects_std**: STD of number of redirects for 'redirects' values in 'domainStats'.
- **95_initiators_mean**: Mean of number of initiators in 'initiators' list in 'domainStats'.
- **96_initiators_std**: STD of number of initiators in 'initiators' list in 'domainStats'.
- **97_domain_in_initiators_mean**: Mean of of initiators in 'initiators' list in 'domainStats' containing domain from 'domainStats'.

- **98_domain_in_initiators_std**: STD of of initiators in 'initiators' list in 'domainStats' containing domain from 'domainStats'.
- **99_count_mean**: Mean of 'count' values in 'regDomainStats'.
- **100_count_std**: STD of 'count' values in 'regDomainStats'.
- **101_size_mean**: Mean of 'https' values in 'regDomainStats'.
- **102_size_std**: STD of 'https' values in 'regDomainStats'.
- **103_ensize_mean**: Mean of 'encodedSize' values in 'regDomainStats'.
- **104_ensize_std**: STD of 'encodedSize' values in 'regDomainStats'.
- **105_ips_mean**: Mean of number of ips in 'ips' list in 'regDomainStats'.
- **106_ips_std**: STD of number of ips in 'ips' list in 'regDomainStats'.
- **107_countries_mean**: Mean of number of countries in 'countries' list in 'regDomainStats'.
- **108_countries_std**: STD of number of countries in 'countries' list in 'regDomainStats'.
- **109_redirects_mean**: Mean of number of redirects for 'redirects' values in 'regDomainStats'.
- **110_redirects_std**: STD of number of redirects for 'redirects' values in 'regDomainStats'.
- **111_subDomains_mean**: Mean of number of subDomains for 'subDomains' values in 'regDomainStats'.
- **112_subDomains_std**: STD of number of subDomains for 'subDomains' values in 'regDomainStats'.
- **113_size_mean**: Mean of 'count' values in 'ipStats'.
- **114_size_std**: STD of 'count' values in 'ipStats'.
- **115_ensize_mean**: Mean of 'https' values in 'ipStats'.
- **116_ensize_std**: STD of 'https' values in 'ipStats'.
- **117_countries_mean**: Mean of number of countries in 'countries' list in 'ipStats'.
- **118_countries_std**: STD of number of countries in 'countries' list in 'ipStats'.
- **119_redirects_mean**: Mean of number of redirects for 'redirects' values in 'ipStats'.
- **120_redirects_std**: STD of number of redirects for 'redirects' values in 'ipStats'.
- **121_ipv6_mean**: Mean of number of ipv6 for 'ipv6' values in 'ipStats'.
- **122_ipv6_std**: STD of number of ipv6 for 'ipv6' values in 'ipStats'.
- **123_ipv6_mean**: Mean of number of requests for 'requests' values in 'ipStats'.
- **124_ipv6_std**: STD of number of requests for 'requests' values in 'ipStats'.
- **125_domains_mean**: Mean of number of domains in 'domains' list in 'ipStats'.
- **126_domains_std**: STD of number of domains in 'domains' list in 'ipStats'.
- **127_secureRequests**: Total number of secure requests in 'secureRequests'.
- **128_securePercentage**: Percentage of secure requests in 'securePercentage'
- **129_IPv6Percentage**: Percentage of IPv6Percentage in 'IPv6Percentage'.
- **130_uniqCountries**: Number of uniqCountries in 'uniqCountries'.
- **131_totalLinks**: Total number of links in 'totalLinks'.
- **132_adBlocked**: Total number of advertisement trackers in 'adBlocked'.

Features computed from the 'data' JSON key. Feature numbers: 133 - 246:

- **133_len_request**: Total number of requests.

- **134_domain_in_docuurl_list_mean**: Mean of requests containing domain of the website.
- **135_domain_in_docuurl_list_std**: STD of requests containing domain of the website.
- **136_upgrade_insecure_requests_mean**: Mean of 'Upgrade-Insecure-Requests'.
- **137_upgrade_insecure_requests_std**: STD of 'Upgrade-Insecure-Requests'.
- **140_content_length_mean**: Mean of 'Content-Length' values.
- **141_content_length_std**: STD of 'Content-Length' values.
- **142_encodedDataLength_mean**: Mean of 'encodedDataLength' values.
- **143_encodedDataLength_std**: STD of 'encodedDataLength' values.
- **146_proxyStart_mean**: Mean of 'proxyStart' values.
- **147_proxyStart_std**: STD of 'proxyStart' values.
- **148_proxyEnd_mean**: Mean of 'proxyEnd' values.
- **149_proxyEnd_std**: STD of 'proxyEnd' values.
- **150_dnsStart_mean**: Mean of 'dnsStart' values.
- **151_dnsStart_std**: STD of 'dnsStart' values.
- **152_dnsEnd_mean**: Mean of 'dnsEnd' values.
- **153_dnsEnd_std**: STD of 'dnsEnd' values.
- **154_connectStart_mean**: Mean of 'connectStart' values.
- **155_connectStart_std**: STD of 'connectStart' values.
- **156_connectEnd_mean**: Mean of 'connectEnd' values.
- **157_connectEnd_std**: STD of 'connectEnd' values.
- **158_sslStart_mean**: Mean of 'sslStart' values.
- **159_sslStart_std**: STD of 'sslStart' values.
- **160_sslEnd_mean**: Mean of 'sslEnd' values.
- **161_sslEnd_std**: STD of 'sslEnd' values.
- **162_workerStart_mean**: Mean of 'workerStart' values.
- **163_workerStart_std**: STD of 'workerStart' values.
- **164_workerReady_mean**: Mean of 'workerReady' values.
- **165_workerReady_std**: STD of 'workerReady' values.
- **166_sendStart_mean**: Mean of 'sendStart' values.
- **167_sendStart_std**: STD of 'sendStart' values.
- **168_sendEnd_mean**: Mean of 'sendEnd' values.
- **169_sendEnd_std**: STD of 'sendEnd' values.
- **170_pushStart_mean**: Mean of 'pushStart' values.
- **171_pushStart_std**: STD of 'pushStart' values.
- **172_pushEnd_mean**: Mean of 'pushEnd' values.
- **173_pushEnd_std**: STD of 'pushEnd' values.
- **174_receiveHeadersEnd_mean**: Mean of 'receiveHeadersEnd' values.
- **175_receiveHeadersEnd_std**: STD of 'receiveHeadersEnd' values.
- **176_len_request_list_mean**: Mean of length of requests.
- **177_len_request_list_std**: STD of length of requests.
- **178_response_encodedDataLength_mean**: Mean of 'encodedDataLength' values in 'response'.
- **179_response_encodedDataLength_std**: STD of 'encodedDataLength' values in 'response'.

- **180_response_dataLength_mean**: Mean of 'dataLength' values in 'response'.
- **181_response_dataLength_std**: STD of 'dataLength' values in 'response'.
- **182_response_respo_encodedDataLength_mean**: Mean of 'encodedDataLength' values in ['response']['response'].
- **183_response_respo_encodedDataLength_std**: STD of 'encodedDataLength' values in ['response']['response'].
- **186_response_proxyStart_mean**: Mean of 'proxyStart' values in ['response']['response']['timing'].
- **187_response_proxyStart_std**: STD of 'proxyStart' values in ['response']['response']['timing'].
- **188_response_proxyEnd_mean**: Mean of 'proxyEnd' values.
- **189_response_proxyEnd_std**: STD of 'proxyEnd' values.
- **190_response_dnsStart_mean**: Mean of 'dnsStart' values.
- **191_response_dnsStart_std**: STD of 'dnsStart' values.
- **192_response_dnsEnd_mean**: Mean of 'dnsEnd' values.
- **193_response_dnsEnd_std**: STD of 'dnsEnd' values.
- **194_response_connectStart_mean**: Mean of 'connectStart' values.
- **195_response_connectStart_std**: STD of 'connectStart' values.
- **196_response_connectEnd_mean**: Mean of 'connectEnd' values.
- **197_response_connectEnd_std**: STD of 'connectEnd' values.
- **198_response_sslStart_mean**: Mean of 'sslStart' values.
- **199_response_sslStart_std**: STD of 'sslStart' values.
- **200_response_sslEnd_mean**: Mean of 'sslEnd' values.
- **201_response_sslEnd_std**: STD of 'sslEnd' values.
- **202_response_workerStart_mean**: Mean of 'workerStart' values.
- **203_response_workerStart_std**: STD of 'workerStart' values.
- **204_response_workerReady_mean**: Mean of 'workerReady' values.
- **205_response_workerReady_std**: STD of 'workerReady' values.
- **206_response_sendStart_mean**: Mean of 'sendStart' values.
- **207_response_sendStart_std**: STD of 'sendStart' values.
- **208_response_sendEnd_mean**: Mean of 'sendEnd' values.
- **209_response_sendEnd_std**: STD of 'sendEnd' values.
- **210_response_pushStart_mean**: Mean of 'pushStart' values.
- **211_response_pushStart_std**: STD of 'pushStart' values.
- **212_response_pushEnd_mean**: Mean of 'pushEnd' values.
- **213_response_pushEnd_std**: STD of 'pushEnd' values.
- **214_response_receiveHeadersEnd_mean**: Mean of 'receiveHeadersEnd' values.
- **215_response_receiveHeadersEnd_std**: STD of 'receiveHeadersEnd' values.
- **216_securityState_mean**: Mean of of secure requests.
- **217_securityState_std**: STD of of secure requests.
- **218_sanList_mean**: Mean of length 'sanList' list.
- **219_sanList_std**: STD of length 'sanList' list.
- **220_subject_name_in_san_list_mean**: Mean of subject names in sun list.
- **221_subject_name_in_san_list_std**: STD of subject names in sun list.
- **222_cert_valid_mean**: Mean of list of validity length of certificates.
- **223_cert_valid_std**: STD of list of validity length of certificates.
- **224_cert_valid_now_mean**: Mean of valid certificates.
- **225_cert_valid_now_std**: STD of valid certificates.

- **226_hashes_list_mean**: Mean of hashes list.
- **227_hashes_list_std**: STD of hashes list.
- **228_hashes_list_mean**: Total number of cookies.
- **231_cookie_expires_now_mean**: Mean of seconds reaming for cookie expiration.
- **232_cookie_expires_now_std**: STD of seconds reaming for cookie expiration.
- **233_cookies_sizes_mean**: Mean of sizes of cookies.
- **234_cookies_sizes_std**: STD of sizes of cookies.
- **235_cookie_http_only_mean**: Mean of cookies using only http.
- **236_cookie_http_only_std**: STD of cookies using only http.
- **237_cookie_secure_mean**: Mean of cookies using only https.
- **238_cookie_secure_std**: STD of cookies using only https.
- **239_cookie_session_mean**: Mean of cookies using session.
- **240_cookie_session_std**: STD of cookies using session.
- **241_len_links**: Total number of links of the web page.
- **242_links_domain_in_url_mean**: Mean of links containing domain of the website.
- **243_links_domain_in_url_std**: STD of links containing domain of the website.
- **244_diff_tld**: Total number of different TLD extracted from the links.
- **245_diff_tld**: Total number of global variables in JavaScript.
- **246_diff_globals**: Total number of unique global variables in JavaScript.

A.2 feature-set-2 - 310 features

All these features are computed from the urlscan.io analysis. To see an example of urlscan.io analysis, go to urlscan.io website, choose some submitted URL and then there is a button called "API". If you click on it, the urlcan.io analysis will be shown in JSON format. These features are computed from this JSON output. The feature-set-2 contains tree categories of features.

- Absolute feature - Total number of items in lists.
- Categorical feature - Total number of occurrences for a value.
- Numerical feature - Mean, Standard deviation, maximum or minimum is computed from the a list of numbers.

Each category has own table. The absolute features are shown in Table A.1, the categorical features are shown in Table A.2 and numerical features are shown in Table A.3.

The first column of each table is index of feature. The second and third columns are keys from urlscan.io analysis json. To see an example of urlscan.io analysis, take a look on the urlscan.io.

Feature Index	json key	json sub key	value
1	cookies	httpOnly	false
2	cookies	httpOnly	true
3	cookies	secure	false

4	cookies	secure	true
5	cookies	session	true
6	cookies	session	false
7	cookies	expires	expired
8	cookies	expires	valid
14	globals	type	object
15	globals	type	function
16	globals	type	string
17	globals	type	number
18	globals	type	boolean
19	globals	type	undefined
20	globals	type	symbol
22	https	securityState	insecure
23	https	securityState	None
24	https	securityState	secure
25	https	securityState	unknown
26	https	protocol	None
27	https	protocol	TLS 1.3
28	https	protocol	TLS 1.2
29	https	protocol	TLS 1.0
30	https	protocol	TLS 1.1
31	https	keyExchange	None
32	https	keyExchange	
33	https	keyExchange	ECDHERSA
34	https	keyExchange	RSA
35	https	keyExchange	ECDHEECDSA
36	https	keyExchangeGroup	None
37	https	keyExchangeGroup	P-256
38	https	keyExchangeGroup	X25519
39	https	keyExchangeGroup	P-384
40	https	cipher	None
41	https	cipher	AES256GCM
42	https	cipher	AES128GCM
43	https	cipher	CHACHA20POLY1305
44	https	cipher	AES128CBC
45	https	cipher	AES256CBC
46	https	cipher	3DESEDECBC
47	https	websitecert	websitecert
53	requests	method	GET
54	requests	method	None
55	requests	method	POST
56	requests	method	OPTIONS
57	requests	method	PUT

58	requests	method	HEAD
59	requests	method	GET.html
60	requests	method	PATCH
61	requests	method	SCRIPT
62	requests	method	DELETE
63	requests	mixedContentType	none
64	requests	mixedContentType	None
65	requests	mixedContentType	optionally-blockable
66	requests	mixedContentType	blockable
67	requests	initialPriority	VeryHigh
68	requests	initialPriority	Low
69	requests	initialPriority	None
70	requests	initialPriority	High
71	requests	initialPriority	Medium
72	requests	initialPriority	VeryLow
73	requests	referrerPolicy	no-referrer-when-downgrade
74	requests	referrerPolicy	None
75	requests	referrerPolicy	origin-when-cross-origin
76	requests	referrerPolicy	strict-origin-when-cross-origin
77	requests	referrerPolicy	origin
78	requests	referrerPolicy	no-referrer
79	requests	referrerPolicy	unsafe-url
80	requests	referrerPolicy	same-origin
81	requests	referrerPolicy	strict-origin
82	requests	Upgrade-Insecure-Requests	1
83	requests	Upgrade-Insecure-Requests	None
84	requests	Sec-Fetch-User	None
85	requests	Sec-Fetch-User	?1
86	requests	type	Document
87	requests	type	Image
88	requests	type	None
89	requests	type	Stylesheet
90	requests	type	Script
91	requests	type	Font
92	requests	type	XHR
93	requests	type	Other
94	requests	type	Fetch
95	requests	type	Media
96	requests	type	EventSource
97	requests	type	TextTrack
98	requests	hasUserGesture	false
99	requests	hasUserGesture	None
100	requests	documentURL	false

101	requests	documentURL	true
102	requests	hasdomain	true
103	requests	hasdomain	false
104	requests	hasdomain2	true
105	requests	hasdomain2	false
106	requests	notsamedocument	true
107	requests	notsamedocument	false
108	requests	fromPrefetchCache	false
109	requests	fromPrefetchCache	None
110	requests	protocol	http/1.1
111	requests	protocol	None
112	requests	protocol	h2
113	requests	protocol	data
114	requests	protocol	blob
115	requests	protocol	http/1.0
116	requests	:method	None
117	requests	:method	GET
118	requests	:method	POST
119	requests	pragma	None
120	requests	pragma	no-cache
121	requests	cache-control	None
122	requests	cache-control	no-cache
123	requests	sec-fetch-site	None
124	requests	sec-fetch-site	none
125	requests	sec-fetch-site	cross-site
126	requests	sec-fetch-site	same-origin
127	requests	sec-fetch-site	same-site
128	requests	sec-fetch-mode	None
129	requests	sec-fetch-mode	navigate
130	requests	sec-fetch-mode	nested-navigate
131	requests	sec-fetch-mode	no-cors
133	console	level	error
134	console	level	info
135	console	level	log
136	console	level	warning
137	console	level	debug
139	links	emptylinks	true
140	links	emptylinks	false
141	links	emptylinks	None
151	resourcestat	type	Image
152	resourcestat	type	Document
153	resourcestat	type	Script
154	resourcestat	type	Stylesheet

155	resourcestat	type	Font
156	resourcestat	type	XHR
157	resourcestat	type	Fetch
158	resourcestat	type	Other
159	resourcestat	type	Media
160	resourcestat	type	EventSource
161	resourcestat	type	TextTrack

Table A.2: feature-set-2 - Categorical features. All these features are computed from the lists. They represent, how many times the value occurs. For example, feature 1 in the first row, describes amount of occurrences for value "false" in the subsection "httpOnly" in section "cookies" in urlscan.io analysis.

A.3 The Most Important Features for XGBoost evaluated on UWD-feature-set-1

Feature Name	Importance
get_127_secureRequests	0.466766
get_67_ips_mean	0.123362
get_128_securePercentage	0.065410
get_18_length_linkdomains	0.035108
get_48_ips_std	0.012397
get_132_adBlocked	0.008381
get_231_cookie_expires_now_mean	0.007439
get_222_cert_valid_mean	0.005895
get_228_hashes_list_mean	0.005125
get_131_totalLinks	0.004603
get_23_certicatevalidationmean_linkdomains	0.004451
get_242_links_domain_in_url_mean	0.004327
get_232_cookie_expires_now_std	0.004315
get_111_subDomains_mean	0.003925
get_20_numsubdomainmean_linkdomains	0.003918
get_134_domain_in_docuurl_list_mean	0.003850
get_25_certicatevalidation2mean_linkdomains	0.003750
get_31_priority_std	0.003512
get_9_urls	0.003120
get_6_number_domains	0.002948
get_16_javascript_url	0.002883
get_66_ensize_std	0.002846
get_37_count_mean	0.002828
get_72_protocols_std	0.002718
get_45_compression_mean	0.002691

get_13_image_in_url	0.002670
get_15_sizestd_url	0.002650
get_215_response_receiveHeadersEnd_std	0.002600
get_84_count_std	0.002537
get_50_countries_std	0.002508
get_46_compression_std	0.002461
get_89_ips_mean	0.002445
get_30_priority_mean	0.002370
get_81_countries_mean	0.002369
get_109_redirects_mean	0.002252
get_38_count_std	0.002229
get_26_certicatevalidation2std_linkdomains	0.002165
get_238_cookie_secure_std	0.002107
get_129_IPv6Percentage	0.002102
get_80_ips_std	0.001989
get_68_ips_std	0.001933
get_119_redirects_mean	0.001920
get_157_connectEnd_std	0.001920
get_12_domain_in_url	0.001907
get_105_ips_mean	0.001894
get_99_count_mean	0.001851
get_182_response_respo_encodedDataLength_mean	0.001847
get_214_response_receiveHeadersEnd_mean	0.001845
get_7_diff_domains	0.001839
get_150_dnsStart_mean	0.001833
get_135_domain_in_docuurl_list_std	0.001821
get_133_len_request	0.001817
get_136_upgrade_insecure_requests_mean	0.001802
get_140_content_length_mean	0.001795
get_19_difftld_linkdomains	0.001784
get_166_sendStart_mean	0.001765
get_8_servers	0.001755
get_91_countries_mean	0.001747
get_175_receiveHeadersEnd_std	0.001723
get_65_ensize_mean	0.001710
get_11_https_in_url	0.001705
get_216_securityState_mean	0.001675
get_137_upgrade_insecure_requests_std	0.001643
get_142_encodedDataLength_mean	0.001628
get_174_receiveHeadersEnd_mean	0.001613
get_239_cookie_session_mean	0.001602
get_103_ensize_mean	0.001591
get_88_ensize_std	0.001589

get_179_response_encodedDataLength_std	0.001585
get_49_countries_mean	0.001567
get_57_ips_mean	0.001557
get_146_proxyStart_mean	0.001555
get_141_content_length_std	0.001538
get_87_ensize_mean	0.001534
get_177_len_request_list_std	0.001531
get_176_len_request_list_mean	0.001526
get_245_diff_tld	0.001520
get_69_countries_mean	0.001506
get_14_sizemean_url	0.001500
get_24_certicatevalidationstd_linkdomains	0.001498
get_90_ips_std	0.001487
get_70_countries_std	0.001479
get_117_countries_mean	0.001477
get_79_ips_mean	0.001457
get_95_initiators_mean	0.001425
get_10_diff_urls	0.001417
get_92_countries_std	0.001413
get_167_sendStart_std	0.001408
get_199_response_sslStart_std	0.001397
get_159_sslStart_std	0.001394
get_4_mean_asns	0.001390
get_243_links_domain_in_url_std	0.001384
get_17_javascript_url	0.001382
get_22_numcertificates_linkdomains	0.001367
get_110_redirects_std	0.001364
get_115_ensize_mean	0.001360
get_56_ensize_std	0.001337
get_21_numsubdomainstd_linkdomains	0.001310
get_62_count_std	0.001306
get_183_response_respo_encodedDataLength_std	0.001305
get_47_ips_mean	0.001305
get_220_subject_name_in_san_list_mean	0.001303
get_118_countries_std	0.001298
get_198_response_sslStart_mean	0.001298
get_152_dnsEnd_mean	0.001293
get_207_response_sendStart_std	0.001287
get_221_subject_name_in_san_list_std	0.001285
get_125_domains_mean	0.001285
get_41_ensize_mean	0.001279
get_223_cert_valid_std	0.001276
get_151_dnsStart_std	0.001274

get_61_count_mean	0.001267
get_156_connectEnd_mean	0.001257
get_51_count_mean	0.001253
get_219_sanList_std	0.001250
get_83_count_mean	0.001228
get_78_ensize_std	0.001227
get_218_sanList_mean	0.001225
get_126_domains_std	0.001222
get_153_dnsEnd_std	0.001216
get_168_sendEnd_mean	0.001215
get_100_count_std	0.001199
get_55_ensize_mean	0.001180
get_1_number_ips	0.001180
get_73_count_mean	0.001174
get_74_count_std	0.001170
get_143_encodedDataLength_std	0.001160
get_197_response_connectEnd_std	0.001157
get_217_securityState_std	0.001151
get_52_count_std	0.001147
get_246_diff_globals	0.001147
get_82_countries_std	0.001133
get_104_ensize_std	0.001128
get_106_ips_std	0.001123
get_200_response_sslEnd_mean	0.001117
get_71_protocols_mean	0.001115
get_201_response_sslEnd_std	0.001104
get_5_std_asns	0.001082
get_77_ensize_mean	0.001072
get_120_redirects_std	0.001066
get_93_redirects_mean	0.001061
get_180_response_dataLength_mean	0.001057
get_42_ensize_std	0.001035
get_154_connectStart_mean	0.001034
get_158_sslStart_mean	0.001010
get_94_redirects_std	0.001010
get_190_response_dnsStart_mean	0.001008
get_116_ensize_std	0.001000
get_181_response_dataLength_std	0.000991
get_96_initiators_std	0.000961
get_193_response_dnsEnd_std	0.000961
get_155_connectStart_std	0.000954
get_161_sslEnd_std	0.000952
get_187_response_proxyStart_std	0.000938

get_160_sslEnd_mean	0.000928
get_58_ips_std	0.000927
get_186_response_proxyStart_mean	0.000914
get_192_response_dnsEnd_mean	0.000908
get_195_response_connectStart_std	0.000885
get_206_response_sendStart_mean	0.000883
get_169_sendEnd_std	0.000862
get_191_response_dnsStart_std	0.000858
get_178_response_encodedDataLength_mean	0.000828
get_196_response_connectEnd_mean	0.000808
get_240_cookie_session_std	0.000783
get_59_countries_mean	0.000747
get_97_domain_in_initiators_mean	0.000739
get_237_cookie_secure_mean	0.000733
get_29_confidence_std	0.000694
get_236_cookie_http_only_std	0.000672
get_194_response_connectStart_mean	0.000632
get_147_proxyStart_std	0.000617
get_130_uniqCountries	0.000601
get_208_response_sendEnd_mean	0.000599
get_209_response_sendEnd_std	0.000585
get_235_cookie_http_only_mean	0.000567
get_244_diff_tld	0.000522
get_28_confidence_mean	0.000520
get_2_number_countries	0.000489
get_27_diff_countries	0.000482
get_60_countries_std	0.000462
get_225_cert_valid_now_std	0.000356
get_112_subDomains_std	0.000244
get_224_cert_valid_now_mean	0.000186
get_101_size_mean	0.000000
get_102_size_std	0.000000
get_107_countries_mean	0.000000
get_108_countries_std	0.000000
get_113_size_mean	0.000000
get_114_size_std	0.000000
get_121_ipv6_mean	0.000000
get_122_ipv6_std	0.000000
get_123_ipv6_mean	0.000000
get_124_ipv6_std	0.000000
get_148_proxyEnd_mean	0.000000
get_149_proxyEnd_std	0.000000
get_162_workerStart_mean	0.000000

get_163_workerStart_std	0.000000
get_164_workerReady_mean	0.000000
get_165_workerReady_std	0.000000
get_170_pushStart_mean	0.000000
get_171_pushStart_std	0.000000
get_172_pushEnd_mean	0.000000
get_173_pushEnd_std	0.000000
get_188_response_proxyEnd_mean	0.000000
get_189_response_proxyEnd_std	0.000000
get_202_response_workerStart_mean	0.000000
get_203_response_workerStart_std	0.000000
get_204_response_workerReady_mean	0.000000
get_205_response_workerReady_std	0.000000
get_210_response_pushStart_mean	0.000000
get_211_response_pushStart_std	0.000000
get_212_response_pushEnd_mean	0.000000
get_213_response_pushEnd_std	0.000000
get_226_hashes_list_mean	0.000000
get_227_hashes_list_std	0.000000
get_233_cookies_sizes_mean	0.000000
get_234_cookies_sizes_std	0.000000
get_241_len_links	0.000000
get_39_size_mean	0.000000
get_3_number_asns	0.000000
get_40_size_std	0.000000
get_43_latency_mean	0.000000
get_44_latency_std	0.000000
get_53_size_mean	0.000000
get_54_size_std	0.000000
get_63_size_mean	0.000000
get_64_size_std	0.000000
get_75_size_mean	0.000000
get_76_size_std	0.000000
get_85_size_mean	0.000000
get_86_size_std	0.000000
get_98_domain_in_initiators_std	0.000000

Table A.4: The feature importance for 233 features (all features) for the XGBoost algorithm with experiment UWD-feature-set-1.

A.4 257 unique HTML tags for data model CETD-DOM-1

div	a

span	li
td	img
tr	script
br	path
p	ul
option	i
link	meta
input	svg
h3	b
label	button
strong	h2
style	g
url	loc
article	h4
symbol	iframe
form	use
noscript	section
source	dd
em	time
title	tbody
table	figure
h1	header
dt	small
polygon	yatag
footer	dl
ins	h5
font	th
head	html
body	picture
nav	circle
rect	hr
h6	text
stop	aside
yt-icon	template
select	ol
yt-formatted-string	sup
defs	figcaption
fieldset	code
ytd-badge-supported-renderer	textarea
center	f
dom-repeat	area
line	abbr
cite	yt-icon-button

u	main
lineargradient	yt-img-shadow
s	paper-tooltip
util-message	noindex
video	tspan
clippath	paper-button
thead	canvas
ytd-button-renderer	pre
legend	blockquote
dom-if	ytd-channel-name
ellipse	icon
ytd-thumbnail	ytd-thumbnail-overlay-now-playing-renderer
mask	cnx
ytd-menu-renderer	field-formatter
var	ytd-thumbnail-overlay-time-status-renderer
col	jdiv
base	polyline
param	ytd-video-meta-block
desc	image
dom-module	mi
object	i-amphtml-sizer
ytd-compact-video-renderer	summary
details	optgroup
address	amp-img
mo	nobr
o:p	ytd-toggle-button-renderer
del	filter
audio	animate
fegaussianblur	paper-item
mrow	map
fecolormatrix	sub
ytd-guide-entry-renderer	identifier-formatter
c-wiz	c-data
relative-time	tt
hgroup	ytd-grid-video-renderer
pattern	wbr
ytd-expander	kbd
data-src	share-button
feoffset	amp-social-share
mat-menu	resource-url
clipboard-copy	label-with-info
l10n	details-menu
theme-icon	mark

feblend	femergenode
asside	ytd-comment-thread-renderer
ytd-comment-renderer	ytd-comment-action-buttons-renderer
iron-media-query	phoenix-ellipsis
mat-card	big
c	description
aza-menu-link	value
mn	yt-next-continuation
node	query-string
aliases	radialgradient
colgroup	phoenix-super-link
phoenix-picture	iron-iconset-svg
bdi	dfn
samp	content
li-icon	ion-col
ymaps	ion-row
sm	embed
wix-image	form:error
link-item	marquee
ui-view	caption
xx_goxgo	rabiitch
picture-media	ion-card
star-rating	menuitem
blink	frame
spa	dialog
g-dialog	g-loading-icon
g-flat-button	g-raised-button
aid-web	applenav
subnav	hero
signin	tfoot
router-outlet	nf-section
nhbk-modal	g:plusone
span1	span2
span3	span4
span5	frameset
menu	dynamic-heading
lab	rs-layer-wrap
rs-loop-wrap	rs-mask-wrap
animatetransform	src
ext:horizontal	ext:horizontallarge
ext:vertical	ext:verticallarge

Table A.5: The selcetd HTML tags for CETD-DOM-1.

Feature Index	json key	json sub key	value
13	cookies		
21	globals		
52	https		
132	requests		
138	console		
150	links		
174	resourcestat		
187	protocolstat		
200	tlsstat		
213	serverstat		
234	domainstat		
255	regdomainstat		
272	ipstat		
273	asn		
290	lists	urls	ips
291	lists	urls	countries
292	lists	urls	asns
293	lists	urls	domains
294	lists	urls	servers
295	lists	urls	urls
296	lists	urls	linkDomains
297	lists	urls	certificates
298	lists	urls	hashes
299	lists	urls	js
300	lists	urls	img
301	lists	urls	css
302	lists	urls	cookie
303	lists	urls	?
304	lists	urls	html
305	lists	urls	dll
306	lists	urls	@
307	lists	urls	//
308	lists	urls	=
309	lists	urls	-
310	lists	urls	_

Table A.1: feature-set-2 - Absolute features. These feature are length of lists except features 299-310. The features 299-310 describes how many urls contain the value in the third column. For example feature 299 represents amount of url containing ".js".

Feature Index	json key	json sub key
9,10,11,12	cookies	size
48,49,50,51	https	sanList
142,143,144,145	links	href
146,147,148,149	links	text
162,163,164,165	resourcestat	count
166,167,168,169	resourcestat	size
170,171,172,173	resourcestat	ips
175,176,177,178	protocolstat	count
179,180,181,182	protocolstat	size
183,184,185,186	protocolstat	ips
188,189,190,191	tlsstat	count
192,193,194,195	tlsstat	size
196,197,198,199	tlsstat	ips
201,202,203,204	serverstat	count
205,206,207,208	serverstat	size
209,210,211,212	serverstat	ips
214,215,216,217	domainstat	count
218,219,220,221	domainstat	size
222,223,224,225	domainstat	ips
226,227,228,229	domainstat	initiators
230,231,232,233	domainstat	redirects
235,236,237,238	regdomainstat	count
239,240,241,242	regdomainstat	size
243,244,245,246	regdomainstat	ips
247,248,249,250	regdomainstat	redirects
251,252,253,254	regdomainstat	subDomains
256,257,258,259	ipstat	requests
260,261,262,263	ipstat	size
264,265,266,267	ipstat	redirects
268,269,270,271	ipstat	domains
274,275,276,277	lists	subdomains
278,279,280,281	lists	urlpathlength
282,283,284,285	lists	urllength
286,287,288,289	lists	numbercounts

Table A.3: feature-set-2 - Numerical features. These features are the mean, standard deviation, maximum and minimum from the list of numbers. Therefore each row in this tables represents four features (the first column contains four indexes). The second and third columns represents keys in JSON urlscan.io analysis. Only the "https" json key for features 48,49,50,51 does not exist, the real JSON path is: ['response']['response']['securityDetails']['sanList'].

A.5 The Most Important Features for XGBoost evaluated on CETD-feature-set-1

Feature Name	Importance
get_127_secureRequests	0.449097
get_11_https_in_url	0.044782
get_195_response_connectStart_std	0.015201
get_48_ips_std	0.014459
get_97_domain_in_initiators_mean	0.013865
get_242_links_domain_in_url_mean	0.011719
get_9_urls	0.010392
get_231_cookie_expires_now_mean	0.009334
get_131_totalLinks	0.008810
get_105_ips_mean	0.008731
get_244_diff_tld	0.008274
get_82_countries_std	0.007027
get_12_domain_in_url	0.006454
get_243_links_domain_in_url_std	0.006403
get_222_cert_valid_mean	0.006358
get_128_securePercentage	0.006316
get_175_receiveHeadersEnd_std	0.005611
get_1_number_ips	0.005391
get_25_certicatevalidation2mean_linkdomains	0.005274
get_23_certicatevalidationmean_linkdomains	0.004935
get_38_count_std	0.004890
get_130_uniqCountries	0.004842
get_96_initiators_std	0.004785
get_19_difftld_linkdomains	0.004755
get_30_priority_mean	0.004521
get_15_sizestd_url	0.004360
get_157_connectEnd_std	0.004355
get_18_length_linkdomains	0.004289
get_245_diff_tld	0.004229
get_6_number_domains	0.004183
get_228_hashes_list_mean	0.004154
get_224_cert_valid_now_mean	0.004129
get_89_ips_mean	0.003939
get_31_priority_std	0.003919
get_111_subDomains_mean	0.003814
get_232_cookie_expires_now_std	0.003765
get_81_countries_mean	0.003730
get_193_response_dnsEnd_std	0.003720
get_135_domain_in_docuurl_list_std	0.003662
get_47_ips_mean	0.003570

get_68_ips_std	0.003434
get_182_response_respo_encodedDataLength_mean	0.003434
get_71_protocols_mean	0.003433
get_17_javascript_url	0.003382
get_237_cookie_secure_mean	0.003335
get_152_dnsEnd_mean	0.003287
get_77_ensize_mean	0.003259
get_20_numsubdomainmean_linkdomains	0.003143
get_61_count_mean	0.003137
get_150_dnsStart_mean	0.003102
get_59_countries_mean	0.003074
get_238_cookie_secure_std	0.003058
get_95_initiators_mean	0.003035
get_14_sizemean_url	0.003024
get_136_upgrade_insecure_requests_mean	0.003016
get_5_std_asns	0.002971
get_57_ips_mean	0.002853
get_104_ensize_std	0.002803
get_223_cert_valid_std	0.002796
get_52_count_std	0.002787
get_174_receiveHeadersEnd_mean	0.002771
get_79_ips_mean	0.002752
get_100_count_std	0.002747
get_183_response_respo_encodedDataLength_std	0.002696
get_4_mean_asns	0.002616
get_50_countries_std	0.002567
get_129_IPv6Percentage	0.002543
get_29_confidence_std	0.002543
get_55_ensize_mean	0.002536
get_134_domain_in_docuurl_list_mean	0.002534
get_26_certicatevalidation2std_linkdomains	0.002523
get_21_numsubdomainstd_linkdomains	0.002502
get_246_diff_globals	0.002491
get_141_content_length_std	0.002413
get_58_ips_std	0.002407
get_199_response_sslStart_std	0.002406
get_66_ensize_std	0.002387
get_235_cookie_http_only_mean	0.002365
get_87_ensize_mean	0.002356
get_13_image_in_url	0.002316
get_240_cookie_session_std	0.002280
get_8_servers	0.002273
get_91_countries_mean	0.002260

get_22_numcertificates_linkdomains	0.002253
get_156_connectEnd_mean	0.002237
get_94_redirects_std	0.002235
get_181_response_dataLength_std	0.002225
get_151_dnsStart_std	0.002225
get_133_len_request	0.002220
get_45_compression_mean	0.002219
get_176_len_request_list_mean	0.002206
get_49_countries_mean	0.002206
get_67_ips_mean	0.002202
get_167_sendStart_std	0.002197
get_83_count_mean	0.002193
get_16_javascript_url	0.002193
get_106_ips_std	0.002190
get_218_sanList_mean	0.002187
get_74_count_std	0.002156
get_56_ensize_std	0.002151
get_46_compression_std	0.002127
get_217_securityState_std	0.002092
get_27_diff_countries	0.002079
get_2_number_countries	0.002060
get_116_ensize_std	0.002050
get_65_ensize_mean	0.002031
get_110_redirects_std	0.002010
get_201_response_sslEnd_std	0.002005
get_219_sanList_std	0.001996
get_166_sendStart_mean	0.001971
get_10_diff_urls	0.001968
get_118_countries_std	0.001967
get_197_response_connectEnd_std	0.001961
get_28_confidence_mean	0.001948
get_192_response_dnsEnd_mean	0.001946
get_143_encodedDataLength_std	0.001942
get_158_sslStart_mean	0.001929
get_80_ips_std	0.001926
get_51_count_mean	0.001924
get_7_diff_domains	0.001911
get_216_securityState_mean	0.001853
get_186_response_proxyStart_mean	0.001852
get_73_count_mean	0.001849
get_99_count_mean	0.001806
get_146_proxyStart_mean	0.001792
get_103_ensize_mean	0.001787

get_42_ensize_std	0.001777
get_177_len_request_list_std	0.001769
get_62_count_std	0.001765
get_24_certicatevalidationstd_linkdomains	0.001728
get_214_response_receiveHeadersEnd_mean	0.001680
get_126_domains_std	0.001678
get_84_count_std	0.001654
get_208_response_sendEnd_mean	0.001646
get_140_content_length_mean	0.001616
get_142_encodedDataLength_mean	0.001609
get_178_response_encodedDataLength_mean	0.001576
get_88_ensize_std	0.001574
get_180_response_dataLength_mean	0.001559
get_221_subject_name_in_san_list_std	0.001553
get_37_count_mean	0.001534
get_169_sendEnd_std	0.001533
get_125_domains_mean	0.001530
get_90_ips_std	0.001529
get_206_response_sendStart_mean	0.001500
get_119_redirects_mean	0.001430
get_179_response_encodedDataLength_std	0.001421
get_200_response_sslEnd_mean	0.001396
get_109_redirects_mean	0.001379
get_78_ensize_std	0.001369
get_236_cookie_http_only_std	0.001369
get_198_response_sslStart_mean	0.001351
get_215_response_receiveHeadersEnd_std	0.001327
get_220_subject_name_in_san_list_mean	0.001322
get_207_response_sendStart_std	0.001289
get_41_ensize_mean	0.001287
get_155_connectStart_std	0.001286
get_93_redirects_mean	0.001279
get_196_response_connectEnd_mean	0.001275
get_115_ensize_mean	0.001272
get_190_response_dnsStart_mean	0.001189
get_60_countries_std	0.001184
get_69_countries_mean	0.001175
get_239_cookie_session_mean	0.001119
get_120_redirects_std	0.001070
get_160_sslEnd_mean	0.001035
get_112_subDomains_std	0.000935
get_187_response_proxyStart_std	0.000569
get_147_proxyStart_std	0.000540

get_72_protocols_std	0.000356

Table A.6: The feature importance for the subset of 170/233 features for XGBoost algorithm with experiment with the CETD-feature-set-1.

Appendix B

Content of the attachment

- textbfsource_code folder: Folder containing source code for both experiments with detection of unsafe websites and evil twin websites.