

***Machine Learning for***

***Malicious URL Detection***

David Culligan – F06861103

Department of Informatics, School of Informatics and Engineering,

Technological University Dublin

Submitted to Technological University Dublin in partial fulfilment of the requirements for the degree of

*Bachelor of Science (Honor’s)*

*in Computing in Digital Forensics*

*and Cyber Security*

Supervisor:

Mark Lane

Submission Date: 28/08/2020

Project Declaration Form

Student: David Culligan

Supervisor: Mark Lane

Course Co-Ordinator: Stephen O Shaughnessy

Module: 4th Year Thesis

Submission Date: 28/08/2020

I declare that the work contained in this report to be submitted to Technological University Dublin, in partial fulfillment of the requirement for the award of Degree of Honor’s B. Sc in Computer Science to be my own. I further declare that the work reported in this project has not been submitted and will not be submitted, either in part or full, for the award of any degree or diploma in this institute or any other institute.

I have read and fully understand the Technology University Dublin’s policy regarding plagiarism.

David Culligan

Acknowledgements

First and foremost, I would like to thank my mother and father, Catherine, and David Culligan for all the love and support they have given me throughout the years. I would not be where I am today if it were not for the both of you and I am forever grateful.

I would also like to offer my sincere gratitude to all the staff at TU Dublin Blanchardstown especially my supervisor Mark Lane for all your help with this project in what has been a difficult year, course coordinator Stephen O Shaughnessy and fellow lecturers Michael Hegarty, Mark Cummins and Arnold Hensman for all your help and guidance through my time in the university.

To all my classmates, thank you for all your help, these past four years have been some of the most memorable of my life and I feel the comradery between us all helped make it so.

I would also like to acknowledge the love and support of my brother Philip, sisters Barbara and Amanda and all my family and friends.

And finally, I would like to dedicate this body of work to my beautiful daughter Robyn. You are and always will be the best thing that has ever happened to me. You always know how to put a smile on my face, and I am so proud to be able to call you my daughter.

Daddy loves you always and forever princess.

# Table of Figures

[Figure 1: Representation of the structure of a URL 12](#_Toc49516462)

[Figure 2: Typical sequence of events in a drive-by download attack 15](#_Toc49516463)

[Figure 3: Typical Social Engineering Life Cycle Model 19](#_Toc49516464)

[Figure 4:Types of Machine Learning 21](#_Toc49516465)

[Figure 5:How Machine Learning Works 22](#_Toc49516466)

[Figure 6:How Unsupervised Machine Learning Works 23](#_Toc49516467)

[Figure 7:Reinforcement Learning 24](#_Toc49516468)

[Figure 8: Visualization of a Random Forest with two trees 24](#_Toc49516469)

[Figure 9: Evolution of XGBoost from Decision Trees 26](#_Toc49516470)

[Figure 10: How XGBoost optimizes the standard Gradient Boosting Machines algorithm 27](#_Toc49516471)

[Figure 11: Information that can be extracted from a URL 28](#_Toc49516472)

[Figure 12: Feature Representation properties 31](#_Toc49516473)

[Figure 13: DataFrame created for Test Data 32](#_Toc49516474)

[Figure 14: Regular Expression Basic Characters 32](#_Toc49516475)

[Figure 15: Sample text without the use of stopwords 33](#_Toc49516476)

[Figure 16: Data Cleanup using Re and NLTK 34](#_Toc49516477)

[Figure 17: Defining GridSearch function 35](#_Toc49516478)

[Figure 18: Feature Extraction of Model 1&2 35](#_Toc49516479)

[Figure 19: LabelEncoder 36](#_Toc49516480)

[Figure 20: Iloc position-based indexer 36](#_Toc49516481)

[Figure 21:Creating the specs for the column transformer 37](#_Toc49516482)

[Figure 22: Processing framework of malicious URL detection using a Machine Learning model 39](#_Toc49516483)

[Figure 23: Anaconda Navigator Graphical User Interface 41](#_Toc49516484)

[Figure 24: Dataset.csv information 42](#_Toc49516485)

[Figure 25: Invoking GridSearch function to find best fit model 45](#_Toc49516486)

[Figure 26: Make\_column\_transformer process 46](#_Toc49516487)

[Figure 27: Example 2x2 Confusion Matrix Structure 47](#_Toc49516488)

[Figure 28: Model 1 Test Data Performance Details 49](#_Toc49516489)

[Figure 29: Model 1 overloading computer resources on runtime 50](#_Toc49516490)

[Figure 30: PC shutdown while running Model 1 50](#_Toc49516491)

[Figure 31:Cell Output on Runtime - Model 1 - Import Libraries 53](#_Toc49516492)

[Figure 32: Cell Output on Runtime - Model 1 - Read Dataset 53](#_Toc49516493)

[Figure 33: Cell Output on Runtime - Model 1 - XGBoost Classification 54](#_Toc49516494)

[Figure 34: Cell Output on Runtime - Model 1 - Train Data Analysis - Confusion Matrix 54](#_Toc49516495)

[Figure 35: Cell Output on Runtime - Model 1 - Train Data Analysis - F1 Score 54](#_Toc49516496)

[Figure 36: Cell Output on Runtime - Model 1 - Test Data Analysis - Confusion Matrix 55](#_Toc49516497)

[Figure 37: Cell Output on Runtime - Model 1 - Test Data Analysis - F1 Score 55](#_Toc49516498)

[Figure 38: Cell Output on Runtime - Model 1 - ROC Curve for Test Data 55](#_Toc49516499)

[Figure 39: Cell Output on Runtime - Model 2 - Read Dataset 56](#_Toc49516500)

[Figure 40: Cell Output on Runtime - Model 2 - XGBoost Classification 56](#_Toc49516501)

[Figure 41: Cell Output on Runtime - Model 2 - Train Data Analysis - Plot Confusion Matrix 57](#_Toc49516502)

[Figure 42: Cell Output on Runtime - Model 2 - Train Data Analysis - F1 Score 57](#_Toc49516503)

[Figure 43: Cell Output on Runtime - Model 2 - Test Data Analysis - Plot Confusion Matrix 58](#_Toc49516504)

[Figure 44: Cell Output on Runtime - Model 2 - Test Data Analysis - F1 Score 58](#_Toc49516505)

[Figure 45: Cell Output on Runtime - Model 2 - ROC Curve for Test Data 59](#_Toc49516506)

[Figure 46: Cell Output on Runtime - Model 3 - Read Dataset 59](#_Toc49516507)

[Figure 47: Cell Output on Runtime - Model 3 - XGBoost Classification 60](#_Toc49516508)

[Figure 48: Cell Output on Runtime - Model 3 - Train Data Analysis - Plot Confusion Matrix 60](#_Toc49516509)

[Figure 49: Cell Output on Runtime - Model 3 - Train Data Analysis - F1 Score 61](#_Toc49516510)

[Figure 50: Cell Output on Runtime - Model 3 - Test Data Analysis - Plot Confusion Matrix 61](#_Toc49516511)

[Figure 51: Cell Output on Runtime - Model 3 - Test Data Analysis - F1 Score 62](#_Toc49516512)

[Figure 52: Cell Output on Runtime - Model 3 - ROC Curve for Test Data 62](#_Toc49516513)

[Figure 53: Comparing ROC Curves 64](#_Toc49516514)

# Table of Tables

[Table 1: LabelEncoder Example (before encoding) 43](#_Toc49283787)

[Table 2: LabelEncoder Example (after encoding) 43](#_Toc49283788)

[Table 3: Experiment results when changing train\_test split 51](#_Toc49283789)

Abstract

A malicious URL (website) is a widespread and severe threat to cybersecurity. This is true as one of the main concerns in the field of cybersecurity is the loss of information which ties in with malicious URLs. New malicious websites hosting unsolicited content are created on daily basis. This content can be used to form a cyber-attack and far too often people are being swindled out of money or sensitive information.

These illegitimate websites generally have a short lifespan which make with detection with conventional techniques difficult. It is therefore vitally important that we can use new methods to detect these malicious URLs and take action against the threat they pose. There are several techniques used by investigators to detect these threats. One of the more conventional techniques is known as blacklists [1].This method is used to identify malicious URLs. It is an easy method to implement and because of this is often used as a traditional method of malicious URL detection. Another technique commonly used is called the Heuristic approach. It is a more advanced system of the Blacklist method but has a drawback in that it is not suitable for all types of attacks. These traditional techniques depend on the matching of keywords and URL syntax matching when detecting malicious URLs, as a result they cannot be relied upon to effectively negotiate with more modern and ever evolving technologies and web access methods [2]. Additionally, these methods come up short when asked to detect modern URLs like short URLs.

In this paper I propose employing a machine learning model for the detection of malicious URLs . The goal of this machine learning model is to deliver a program that can learn for itself without any human interaction and to improve over time [3].

Contents

[1 Table of Figures 4](#_Toc49516515)

[2 Table of Tables 5](#_Toc49516516)

[3 Introduction 9](#_Toc49516517)

[3.1 Thesis Topic 9](#_Toc49516518)

[3.2 Thesis Research Details 10](#_Toc49516519)

[3.3 Project Goal 11](#_Toc49516520)

[4 Literature Review 11](#_Toc49516521)

[4.1 URL (Uniform Resource Locator) 11](#_Toc49516522)

[4.1.1 Structure 12](#_Toc49516523)

[4.1.2 Common Protocols 13](#_Toc49516524)

[4.2 URL Based Attacks 14](#_Toc49516525)

[4.2.1 Phishing Attack 14](#_Toc49516526)

[4.2.2 Drive by Download 14](#_Toc49516527)

[4.2.3 Bating 19](#_Toc49516528)

[4.2.4 Scareware 19](#_Toc49516529)

[4.2.5 Honeytrap 19](#_Toc49516530)

[4.2.6 Social Engineering Prevention 20](#_Toc49516531)

[4.3 Evolution of Artificial Intelligence and Machine Learning 20](#_Toc49516532)

[4.3.1 Machine Learning 20](#_Toc49516533)

[4.3.2 Classification Models 21](#_Toc49516534)

[4.3.3 Supervised Machine Learning 21](#_Toc49516535)

[4.3.4 Unsupervised Machine Learning 22](#_Toc49516536)

[4.3.5 Reinforcement Machine Learning 23](#_Toc49516537)

[4.4 Machine Learning Algorithms 24](#_Toc49516538)

[4.4.1 Logistic Regression 24](#_Toc49516539)

[4.4.2 Random Forest 24](#_Toc49516540)

[4.4.3 XGBoost 25](#_Toc49516541)

[4.5 Feature Representation 28](#_Toc49516542)

[4.5.1 Lexical Features 29](#_Toc49516543)

[4.5.2 Host Based Features 29](#_Toc49516544)

[4.5.3 Blacklist Features 30](#_Toc49516545)

[4.5.4 Content Based Features 30](#_Toc49516546)

[4.6 Libraries Used 31](#_Toc49516547)

[4.6.1 Pandas 31](#_Toc49516548)

[4.6.2 Re (Regular Expressions) 32](#_Toc49516549)

[4.6.3 Natural Language Toolkit 33](#_Toc49516550)

[4.6.4 Scikit-learn 34](#_Toc49516551)

[4.6.5 matplotlib.pyplot 37](#_Toc49516552)

[4.7 Methods of Detecting Malicious URLs 38](#_Toc49516553)

[4.7.1 Blacklist or Heuristic (Rule Based) Approach 38](#_Toc49516554)

[4.7.2 Machine Learning Approach 39](#_Toc49516555)

[5 Methodology 40](#_Toc49516556)

[5.1 Proposed Method 40](#_Toc49516557)

[5.2 Software Used 40](#_Toc49516558)

[5.2.1 Spyder IDE 40](#_Toc49516559)

[5.2.2 Anaconda 41](#_Toc49516560)

[5.2.3 Anaconda Navigator 41](#_Toc49516561)

[5.2.4 Conda 41](#_Toc49516562)

[5.2.5 Jupyter Notebook 41](#_Toc49516563)

[5.2.6 Python 3.8.3 42](#_Toc49516564)

[5.3 Project Development Phases 42](#_Toc49516565)

[5.3.1 Acquisition of Dataset 42](#_Toc49516566)

[5.3.2 XGBoost Model Code Development 43](#_Toc49516567)

[6 Training &Testing 47](#_Toc49516568)

[6.1 Test A XGBoost Model 1 49](#_Toc49516569)

[6.2 Test B XGBoost Model 2 51](#_Toc49516570)

[6.3 Test C XGBoost Model 3 52](#_Toc49516571)

[6.4 Cell Output on Runtime 53](#_Toc49516572)

[6.4.1 Model 1 53](#_Toc49516573)

[6.4.2 Model 2 56](#_Toc49516574)

[6.4.3 Model 3 59](#_Toc49516575)

[7 Analysis 63](#_Toc49516576)

[7.1 XGBoost Model 1 63](#_Toc49516577)

[7.2 XGBoost Model 2 65](#_Toc49516578)

[7.3 XGBoost Model 3 66](#_Toc49516579)

[8 Conclusion 67](#_Toc49516580)

[8.1 Future Research 67](#_Toc49516581)

[9 Bibliography 68](#_Toc49516582)

[10 Appendix 77](#_Toc49516583)

[10.1 Appendix A – XGBoost Model 1 77](#_Toc49516584)

[10.2 Appendix B – XGBoost Model 2 80](#_Toc49516585)

[10.3 Appendix C – XGBoost Model 3 83](#_Toc49516586)

# Introduction

## Thesis Topic

The role of the Internet and technology in general, in modern society cannot be understated. Over the last thirty years society has evolved to one where technology plays an essential part in people’s lives around the world. Technological advancements have made it, so people are now able to communicate with friends and loved ones, business associates or complete strangers on the other side of the world at the click of a button. As such, few can now imagine a world without it. Technology is often described as the most significant influence upon today’s society, with the internet growing at a rapid rate day by day to the extent that it has permeated into all areas of human life, whether it be our Health systems, financial institutions, various government agencies, or simply the local shop down the street, or our children uploading a video with their friends to social media. Put simply, the Internet is all encompassing in the areas in which it affects modern society.

Unfortunately, these technological advancements come hand in hand with an ever-increasing threat to their users. Cyber criminals worldwide are constantly developing new techniques to attack users. On a daily basis thousands of fraudulent websites are being created which help these attackers distribute malware to the systems of unsuspecting users or steal personal/sensitive information which can in turn lead to the theft of the victim’s money. There is an ever-increasing number of techniques that these criminals use to carry out these attacks, such as drive-by-download, bating, scareware, and various types of social engineering of which are all discussed later in this paper.

These type of attacks and the devasting affects they can have on people’s lives make the detection of malicious URLs one of the most important aspects of cyber security.

This paper will discuss the use of Machine Learning and its advantages over existing techniques in the detection of malicious URLs.

## Thesis Research Details

This Thesis will investigate, question and test information in detail relating to Machine Learning as an approach to tackle the ever-present dangers of malicious URLs. Methods and tools used in this fight such as the use of classification models to detect malicious URLs in a feature dataset will be discussed.

The thesis will mainly focus on the XGBoost algorithm, the concept behind it, its evolution from decision tree algorithms as well as the many features which have made it the go to algorithm for data scientists around the world. The various libraries used when developing the machine learning models and how the author tied it altogether will be explained.

Seen as the success of any machine learning model is dependent on the quality of the training data from which it learns, feature representation and the phases involved will also be explained.

This paper will also discuss Machine Learning, from the different classification types (i.e. reinforcement, supervised and unsupervised) and how they differ in their approach.

The structure of a URL, including common protocols will also be discussed. Attacks relating to URLs such as Drive-By-Download will be explained.

Methods like the more traditional Blacklist or Heuristic approaches as well as the Machine Learning Approach will be discussed, and a case will be made as to why Machine Learning is to be considered the future when identifying malicious URLs.

## Project Goal

The goal of this project was to build machine learning models (algorithms) capable of distinguishing between malicious and benign URLs from a datasets containing continuous and categorical information relevant to a URL.

# Literature Review

## URL (Uniform Resource Locator)

A Uniform Resource Locator or URL for short is as the name suggests a unique identifier to locate a resource on the internet [4]. URLs comprise numerous parts, including a protocol and a domain name. Their purpose is to tell a web browser how and where to retrieve a resource online for an end user. They can be accessed in a number of ways such as typing the address directly into the address bar, or clicking on a hyperlink

### Structure

The first part of a URL classifies what protocol is to be used as the principal means of access. The second part of the URL classifies the domain name where the resource is located. Optionally after the domain name a URL can also contain the following

* A path to a web page within the domain,
* A network port which is to be used to make the connection,
* A reference point within a file i.e. a named anchor in a HTML file and,
* Search parameters which can be regularly found in URLs for search results.

A close up of a logo

Description automatically generated

Figure 1: Representation of the structure of a URL

Using the URL <https://www.researchgate.net/search.Search.html?type=researcher&query=url> as an example, elements of a URL can include [4]

* **The protocol**. This is used to access a resource on the net. The resource is attained through its domain name system (DNS) name. In the example above the protocol is *https*.
* **Domain name**. This is the unique reference that signifies a specific webpage. In the example above this is represented as *researchgate.net*.
* **Port name**. The port name generally tends not to be visible within a URL, but it is needed. The default port associated with web servers is port 80.
* **Path**. The path of a URL represents a file or location on a web server. In the example above this I represented as *search*.
* **Query**. This can be found in the URL of dynamic pages and consists of a question mark followed by some parameters. A dynamic website is one that present different content by providing user interaction [5].
* **Parameters**. These are bits of information within the query string of a URL. It can consist of multiple parameters which are usually separated by ampersands (&). In the example above it is represented as *researcher&query=url*.
* **Fragment**. This is a reference to an internal page of a website and signifies a section within a given webpage.

### Common Protocols

Below is a list of some of the most used protocols today

* FTP ( File Transfer Protocol ): This is a protocol which permits the transfer of files between two machines. It works in that it allows the user on one end to connect to a server, request some files which are then retrieved by the server and displayed to the user. The protocol works like a digital language that enables a program at one end of the connection to basically communicate with a program at the other end. [6]
* SFTP ( Simple File Transfer Protocol ): this protocol perform the same action as FTP but ensures greater security by using a different technology which allows the user to authenticate and secure the connection between the user and the server. [7]
* POP3 ( Post Office Protocol ): This is an internet standard protocol which is utilized by local email software clients to retrieve emails from a remote mail server. It does this over a TCP/IP connection. Since its inception in 1984 Post Office Protocol has become one of the most commonly used protocols as it is used by practically every email client. [8]
* SMTP ( Simple Mail Transfer Protocol ): This is a communication protocol designed for the transmission of electronic mail. It is a very effective protocol in achieving what it is designed for. One drawback however is that it only allows the ending of text. [9]
* IMAP ( Internet Message Access Protocol ): This is a protocol used for accessing email on a remote server from a local client. This along with POP3 is supported by all modern email clients and web servers. By default, IMAP works on Port 143 and Port 993. [10]
* HTTP ( Hypertext Transfer Protocol ): HTTP is the protocol used in the navigation of websites. It is an underlying protocol used by the World Wide Web which defines how messages are both formatted and transmitted. [11]
* HTTPS ( Hypertext Transfer Secure ): This protocol works in a similar way to HTTP but with a bonus of a layer of protection (encryption) provided by TLS ( Transport Layer Security ) or SSL ( Secure Sockets Layer ). HTTPS ensures privacy and integrity of data between two machines when communicating over the internet.

## URL Based Attacks

There are several URL based attacks currently exploited by cyber criminals. One of the most common attack is known as Phishing.

### Phishing Attack

A phishing attack is one where the attacker tries to trick the target into providing them with some sort of data e.g. login credentials. The attack is clever in that it gets the user (target) to click on a link which will reroute them to a website which is similar to the one originally requested. When the user attempts to login to this fake site they will unknowingly present their login details to the attacker [7]. One of the many technique’s attackers use when phishing for data is to purchase a domain name similar to the site that they intend the target to use. An example of this would be were the attacker purchases instagrm.com instead of instagram.com. In doing so the attacker is taking advantage of the way the human brain does not always pick up on grammatical errors when reading familiar words, in essence it processes the as if the error does not exist in the word [7]. This in turn leads to the user not noticing that they have accessed a fake site. In 2015 the International Journal of Human – Computer Studies conducted a user study to evaluate whether “improved browser security indicators and increased awareness of phishing have led to users' improved ability to protect themselves against such attacks” [12]. The results of this study show that a user’s ability to detect phishing websites does not correlate with their technical expertise of a given device as users spend little time looking at security indicators even when given visual cues. This is backed up by the fact that participants of the study only correctly identified 53% of phishing websites even when primed to do so. This is one of the principal reasons as to why phishing attacks are so successful.

### Drive by Download

A drive by download attack explicitly signifies an attack where a malicious program / programs download to a victims device without consent [13]. A drive-by download aims to benefit from an operating system, application or web browser that encompasses security defects due to an absence of necessary updates. This type of attack leaves the users device open to a cyber-attack and is designed to penetrate the targets device for one of the following reasons [14].

1. Hijack the targets device – to construct a botnet, corrupt other devices or to simply to further infect the targets device.
2. To pry on the target’s activity – this could be done with the aim of obtaining valuable information from the user such as financial information and various user specific information (login credentials etc.).
3. Destroy data or incapacitate the targets device – This can be done by the attacker to simply cause an inconvenience to the user

[15]A close up of a map

Description automatically generated

Figure 2: Typical sequence of events in a drive-by download attack

Drive by download attacks signify a substantial threat to users of the Internet. “The ratio of web pages that contain drive-by-download attacks to benign pages has been estimated at between 0.1% and 0.6% “ [16]. Malware developers are using various techniques to manipulate browsers to dispense malicious code onto the internet. During a drive-by-download attack an automated application known as a browser exploit pack (BEP) is deployed on infected servers [17]. Due to the stealthy nature by which they operate, drive-by-download attacks infiltrate users’ devices in such a way that most victims are unaware of how they got infected. There are two principal ways that malicious drive-by-download attacks infiltrate a user’s device [14].

1. Sanctioned without being aware of the repercussions: This is where the user engages in an action such as clicking on a link leading to an infection.
2. Un-sanctioned without notification: The users device gets infected just by visiting a site. These downloads can be activated even when visiting legitimate sites.

**Sanctioned Downloads with concealed payloads:**

* Cyber-criminal builds a vector for delivery of the malware – e.g. online advertisements, genuine program downloads.
* The user interacts with the vector – This can be done by simply clicking on a malicious link.
* Malware then proceeds to install on the user’s device
* The cyber-criminal gains access to the user’s device – the malware successfully takes control of the data on the user’s device.

**Un-sanctioned without notification:** This type of download has various stages

* The cyber-criminal compromises a web page / website – This is achieved by attaching a malicious element to a security flaw within the device.
* The user activates the malicious element – This can be done by visiting the affected web page which in turn locates the user’s security flaws.
* The malicious element proceeds to download malware the device in question – This is done via the devices exploited security flaws.
* The malware accomplishes its purpose – This then grants the cyber-criminal access to the user’s device.

There are several factors responsible for the ever-increasing potential impact of drive-by-download attacks, one being the reduction in technical proficiency required by the attacker to implement such attacks.

There are a number of ways in which the owner of a website can mitigate against such attacks including [14].

1. To keep all elements relating to a website including plugins, addons up to date. This can help as updates are likely to include fixes to security flaws.
2. The use of strong passwords and usernames for administration accounts associated with the website – This is a vital technique used to mitigate against brute force attacks.
3. The removal of all unsupported elements of a website as hackers regularly exploit software which has not been recently patched.
4. The installation of appropriate web site monitoring software can help notify the owner of any potentially malicious changes to the backend code of the owner’s website.

There are also a number of ways that a user of a website can protect themselves against drive by download attacks including [14].

1. Using a non admin account for daily use and only using your devices admin account for program installations. This can help as admin privileges are required for drive by downloads to install on a user’s device without consent.
2. Regularly check operating system and browser for security updates. Up to date security patches can help prevent against such an attack as it effectively closes off any security flaws that may be exploited.
3. Use an as needed approach when populating your computer with programs and applications. This can help mitigate against drive by download attacks as it minimizes the amount of points that a system is susceptible to infection. It is also advised to uninstall any applications or programs that no longer in receipt of security updates.
4. The use of internet security specific software that provide tools for scanning websites can help mitigate against attacks . This type of software also offers the user peace of mind by keeping definitions of malware up to date to help the user spot the latest threats which could affect their device.
5. The use of an ad blocker can help minimize the user’s exposure to a drive by download attack as online advertisements are a popular way of uploading malicious code to a website.
6. Avoid adult themed sites or various types of file sharing sites as these are well known to be rampant with malicious code by which a device can become infected. It is advised to keep your web surfing to more established websites.

Another technique commonly used by cyber criminals is known as Social Engineering. This technique and the attacks that are associated with this method have developed into a severe threat for people connected to the internet worldwide [18]. The overabundance of communication tools i.e. (E-mail, Skype etc.)on the market today provide attackers with a number of new attack vectors to perform social engineering attacks.

Social Engineering is defined as “The art of manipulating people into performing actions or divulging confidential information, rather than by breaking in or using technical cracking techniques. While similar to a confidence trick or simple fraud, the term typically applies to trickery or deception for the purpose of information gathering, fraud or computer system access. In mot case the attacker never comes face-to-face with the victim. Social engineering using impersonation(e.g. to gain information over the phone or to gatecrash) is known informally as blagging. In addition to criminal purposes, social engineering has also been employed by debt collectors, skip tracers, private investigators, bounty hunters and tabloid journalists. A study by Google researchers found that 90 percent of all domains involved in distributing fake antivirus software used social engineering techniques” [19].

Social Engineering attacks can consist of one or more steps. Below I will explain four commonly used phases used by attackers when performing a social engineering attack [20].

**Investigation**

* The attacker first identifies a victim(s). This can be based on variables such as the persons position in a company. Other variables could be how accessible that person is.
* Pertinent information relating to the attack I then gathered on the target. Information such as potential security flaws and points of entry are sought out. Public information sources such as various forms of social media as well as business websites provide attackers with a wealth of information which can then be used
* Depending on the information gathered, a method(s) is selected

**Hook**

* This phase involves an initial communication with the victim. This can vary from a simple e-mail to a direct conversation and it is at this point that the attacker will attempt to earn the victims trust.
* It is also during this phase that the attacker will fabricate a story with the aim of manipulating the victim. This is done to motivate the victim to perform some sort of future action which can help the attacker perform the attack. Actions such as giving up sensitive information or permitting access are what makes social engineering attacks so threatening to organizations.

**Play**

* It is during the play phase that the attacker aims to gain a more solid foothold.
* It is also the phase that the attack is carried out
* Subject to the end goal of the attacker, disruption of services or the theft of sensitive information is also performed at this point of the attack.

**Exit**

* The aim of the attacker at this phase of the lifecycle is to bring the attack to an end without arousing suspicion.
* This is done by covering their tracks as carefully as possible
* Valuable information gained from this attack can then be used by the attacker for future attacks

**A screenshot of a cell phone

Description automatically generated**

Figure 3: Typical Social Engineering Life Cycle Model

### Bating

The purpose of this type of attack as its name suggests Is to lure a victim into a trap that subsequently takes their sensitive – personal information and or infects their device with some sort of malware [20]. A common form of baiting is known as USB baiting. This is where the attacker uses infected USB flash drives to carry out an attack. These items will invariably be left in places that the victim will be sure to see them and aim to pique curiosity in the victim.

### Scareware

Also referred to as deception software or fraud ware is the practice of flooding a victim’s device with carious false threats with the end goal of persuading the victim to download software or malware with the promise of fixing the victims device. Ironically, the installation of such software - malware only serves to infect the target device [7].

### Honeytrap

This can be describes as where the social engineer earns the trust of the victim to a level where the victim will un suspiciously open a malicious URL link sent to them from the attacker that will serve up valuable information to the attacker [7].

### Social Engineering Prevention

Luckily there are a number of policies that an organization can be put into place to optimistically prevent social engineering attacks against employees [21].

* It is of the utmost importance that management fully understand that the development and implementation of security policies and procedures should be just as important as spending money on various software security patches, security hardware etc. This is to adequately prevent against social engineering attacks [22].
* As security policy that determines what individuals can release information to the public.
* An Access Approval security policy that entails what member(s) of staff have the authority to permit access to various systems and what type of information they can give out .
* A policy that prevents the use of weak passwords as well as one that lays out the frequency that passwords must be changed.
* A policy that requires all employees to wear photo ID as well as a requirement for all visitors to wear a temporary ID card.
* Anyone on the premises should be questioned as to why they are not wearing one.

## Evolution of Artificial Intelligence and Machine Learning

“It is increasingly recognized that artificial intelligence has been touted as a new mobile. Because of the high volume of data that being generated by devices, sensors and social media users, the machine can learn to distinguish the pattern and makes a reasonably good prediction” [23]. Artificial Intelligence is a wide-ranging scientific field which can be traced back to mathematics, philosophy and computer science [24].Artificial Intelligence best describes machines or programs that have the ability to complete tasks that would traditionally be undertaken by humans. Tasks such as natural language processing are increasingly and effectively been accomplished with the use of artificial intelligence [25].

### Machine Learning

Machine can be best described as the field of scientific study that enables self-learning within computer systems without being clearly programmed to do so. It falls under the umbrella of Artificial Intelligence where algorithms have the ability to learn from given datasets. It is simply “the application of statistical models to data using computers” [24]. One of the main advantages of machine learning is the proficiency in which it can analyze data when compared to humans. This is because of the simple fact that machines are capable of processing large sets of data at a far higher level than humans.

A machine learning approach to the detection of malicious URLs address the limitations facing other more traditional methods such as the blacklist method and heuristic approach. One of the most significant drawbacks to the blacklist approach being the difficulty in maintaining a comprehensive list of URLs. Another drawback being the costs, time and manpower needed to work effectively. Machine learning as an approach addresses both issues [3].

### Classification Models

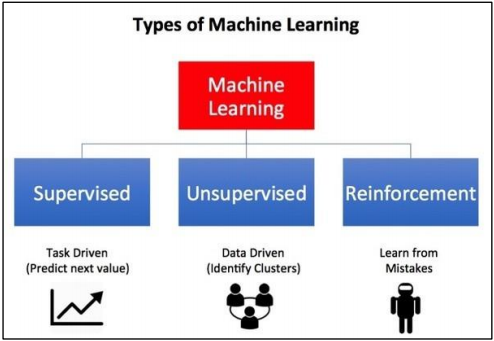
[26]

Figure 4:Types of Machine Learning

### Supervised Machine Learning

Supervised Machine Learning is where the algorithm is being trained whilst using a set of labeled data. Labeled data is essentially data that has been previously tagged with the answer that the algorithm is looking to find [27]. An example relating to this thesis would be where the URLs in a dataset already been labelled as malicious or benign. The algorithm is then trained with a percentage of said dataset and when new data is introduced to the algorithm it must then predict based on the information it has learned during the training phase whether the URLs are malicious or benign. Supervised Machine Learning uses data patterns to calculate any additional data that is introduced. The two phases involved with Supervised Machine Learning are training and testing

**Training:** The training phase can best be described as where the algorithm is learning and is where it inspects data from a training dataset. From this a classification model is built.

**Testing:** The testing phase or classifying phase is basically the model that has been developed in the training stage and its purpose is to evaluate new data presented to it.

Supervised learning is primarily used to solve two types of problems, Classification and Regression. Classification problems generally ask the model to correctly identify input data and sort this data into categories i.e. malicious and benign. Regression problems ask the model to identify real values such as weight, dollars and are looking for continuous data.

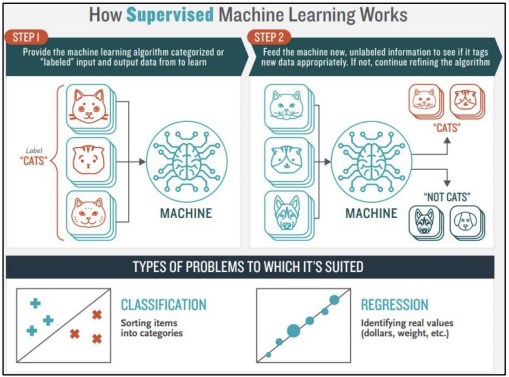


Figure 5:How Machine Learning Works

### Unsupervised Machine Learning

Unsupervised learning is a technique used when a model is given data with no clear instructions of what to do with it. The algorithm (model) is built in a way that it extracts certain features in the hope of finding a structure within the given data [27]. Simply put it is where un-labelled data is presented to the algorithm. The purpose of this type of learning is to explore datasets to find some sort of structure within [26]. The algorithm then labels the data accordingly. An example of unsupervised machine learning is where the model is given a dataset of customers with a certain set of attributes associated with each customer with the goal of identifying and clustering customers with similar attributes into groups. This works well for organizations looking to market their products and services to a specific group of customers.

[28]A screenshot of a cell phone

Description automatically generated

Figure 6:How Unsupervised Machine Learning Works

### Reinforcement Machine Learning

Reinforcement learning is different from both supervised and unsupervised learning in that the model is not told what actions to take, instead it is obliged to determine through trial and error what actions yield the best results [29]. Although it is often taught as a type of unsupervised learning because it does not rely on given examples of correct behavior through the use of predefines labels, it is different in that its purpose is to maximize a reward signal instead of trying to find a hidden structure within the given dataset [29]. Reinforcement Learning consists of three core elements, the agent, the environment, and the action. The agent or decision maker takes actions with the goal of maximizing an expected reward of a given measure of time [26]. The environment is best described as the everything that the agent interacts with during the learning process and the actions are a representation of what an agent can do during the learning process. Below is a diagram briefly explaining the Reinforcement Learning process.

A close up of a logo

Description automatically generated

Figure 7:Reinforcement Learning

## Machine Learning Algorithms

### Logistic Regression

Unlike the name suggests Logistic Regression is a supervised learning-based classification algorithm that is used to evaluate discrete values (e.g. Binary values 0/1, true/false) based on a given set of independent variables (e.g. malicious/benign). The algorithm allocates this data to a logit function which enables it to assess the probability of an event who’s values lie between 0 and 1 [30]

### Random Forest

Random Forest is one of the most widely used machine learning algorithms that can generate exceptional results without the need to tune hyper-parameters within the algorithm. Another reason for its popularity is that it can be used for classification and regression problem solving tasks which make up the majority of machine learning systems.

Random forest is a supervised machine learning algorithm that builds a group of decision trees, trains them using the bagging method (the use of a combination of learning models), and merges them together with the goal of getting a more accurate result [31].

A close up of a map

Description automatically generated

Figure 8: Visualization of a Random Forest with two trees

### XGBoost

XGBoost stands for eXtreme Gradient Boosted trees. Boosting is basically an ensemble method for enhancing the model predictions of any given learning algorithm [32]. The principle concept of boosting is to transform weak learners into strong learners sequentially. The understanding is that a model is used with multiple versions of that model chained together. Each iteration of that model and the trees within boost various attributes from previous iterations that led to misclassifications. Simply put we have multiple trees that are building on top of each other that correct the errors of the previous tree.

XGBoost is an implementation of gradient boosted decision trees that is designed specifically for performance and speed as the implementation of gradient boosting machines are normally very slow. This is because of sequential model training and results in Gradient Boosting being generally unsuitable as a scalable model. XGBoost on the other hand provides

* Parallelization of tree construction, meaning it can channel the power of multi-core processors making it a great choice when training large datasets [33].
* Distributed Computing – the training of large models using a cluster of machines [32]. Not only is XGBoost suitable for use with small to medium sized datasets, but the fact that it can run parallel over a number of threads on a cluster of machines where one machine simply cannot handle the enormity of the dataset presented to it makes it a suitable algorithm for use on large data projects.
* A wide variety of tuning parameters
* Enhanced performance when compared against other algorithms – XGBoost has consistently outperformed other algorithms when tested on a number of benchmark machine learning datasets.
* High Flexibility – Users are allowed to define evaluation criteria to suit their needs which basically leads to there being no limits to what a user can do [34].
* The ability to handle missing values – XGBoost is clever in that it self learns when encountering missing values by trying different parameters. This helps when the program comes across similar missing values in the future. The fact that it can automatically handle the missing values within the data can drastically cut the time and work of a data scientist.
* Built in Cross Validation – Makes it easier for the user to get the optimum number of boosting iterations in a single run by cross-validating at each iteration of the boosting process [34]. In layman’s terms cross validation allows the user to evaluate the performance of the XGBoost at each step of its training. This can be useful for example when the user sees no benefit from further iterations and can stop the training at a specific iteration
* Hardware Optimization – XGBoost has been designed to make effective use of a computer’s hardware resources.
* XGBoost supports incremental training which allows the user to stop training of a model, save it and pick it up a later date.
* Tree pruning – Unlike normal decision trees that stop branching when they stop seeing a benefit of doing so, XGBoost takes a different approach in that it goes very deep by default and tries to prune that tree backwards. This generally results in deeper but more optimized trees.

The chart below is a representation of the evolution of tree-based algorithms from decision trees to XGBoost

[35]A close up of a logo

Description automatically generated

Figure 9: Evolution of XGBoost from Decision Trees

When asked on Quora “*What is the difference between the R GBM (gradient boosting machine) and XGBoost (extreme gradient boosting)*” the developer Tianqi Chen responded by saying “*The name XGBoost, though, actually refers to the engineering goal to push the limit of computations resources for boosted tree algorithms. Which is the reason why many people use XGBoost*” [36]. Since its introduction as a research project at the University of Washington and presentation at SIGKDD Conference in 2016 XGBoost has exploded in the world of Machine Learning and is now the most widely used Machine Learning algorithm currently in use today.

Another feature that XGBoost uses that sets it apart from other boosted tree algorithms is known as regularized boosting. This is a form of boosting that in practice prevents overfitting, so it ensures that the model created is generalized and is not overfitted to the set of data that it has been trained on. Although XGBoost and Gradient Boosting Machines both apply the same principle methods of boosting weak learners, XGBoost is a vast improvement on the Gradient Boosting Machines (GBM) framework from which it derives through the use of system optimization and algorithmic enhancements [35] that can be seen below.

A screenshot of a cell phone

Description automatically generated

Figure 10: How XGBoost optimizes the standard Gradient Boosting Machines algorithm

XGBoost is very easy to both install and use. To install simply type pip install XGBoost from your anaconda prompt. It also offers interfaces on the command line interface for C++, which it is written in natively along with R, Julia and JVM. XGBoost is not just used for scikit\_learn and python. When using the algorithm all parameters are passed in via a dictionary.

## Feature Representation

The resultant success of any machine learning model is dependent on a number of factors. One such factor being the quality of the training data from which it learns, which in turn centers on the quality of feature representation [37]. The feature representation phases can be broken down in into two stages.

1. **Feature Extraction:** This stage is where the data scientists/researchers’ objectives are to gather as much information relevant to the project from a URL. Information such as the presence of a URL in a blacklist, host information, URL string information along with other pertinent information that can be seen in *Figure 11* below is gathered at this stage [3]. Several different types of features are deemed useful when using machine learning algorithms to detect malicious URLs. Features such as Lexical features and Host Based features amongst others can garner important information and will be discussed below.
2. **Feature Pre-processing:** The feature pre-processing stage is where the information previously collected is formatted in a way that can be understood by the machine learning algorithm that it is to be trained on. This is done by converting textual information (data) to a numerical vector.

**A screenshot of a machine

Description automatically generated**

Figure 11: Information that can be extracted from a URL

### Lexical Features

Lexical features are the textual properties of a URL and are not representative of the content of the webpage that they refer to. These properties of a URL string include the length of the hostname, the length of each individual element of the URL such as (Hostname, Primary Domain), the number of dots and other special characters within the URL etc. [37]. This method of extraction where each piece of string within a URL that is delimited by a special character (e.g. “/”, “?”, “.” etc.) is split up to form a single word from which a dictionary was formed is known as the bag-of-words model (See subheading Scikit-learn). It works in a way that if a given word was present in the dictionary it would be represented as 1, if not it would be represented as 0.

The reasoning behind using lexical features when detecting malicious URLs using machine learning is that malicious URLs are often inclined to look unusual in comparison to benign (legitimate) website URLs. Cyber criminals tend to create URLs using algorithms that can bypass blacklists. This is more difficult to do in relation to the Bag-of Words method as it can detect words within the URL that are not in the dictionary that has been created.

### Host Based Features

Host-based features are acquired from the host-name properties of the URL and explain where malicious websites (URLs) are hosted, who they are managed by and how they are administered [38]. Host-based features are an important method in detecting malicious URLs as this feature includes properties such as

1. **WHOIS properties** – Provides information such as the date the URL was registered, updated and date of expiration. Other pertinent information provided includes who the registrar and registrant are and if there are malicious domains already registered by a certain individual.
2. **IP address properties** – Includes information as to whether the IPs of the A, MX or NS records are in the same autonomous system (ASes) or prefixes as one another [38].
3. **Domain name (DNS) properties –** Contain properties such as time-to-live (TTL) values and, whether the hostname contains the keywords “client” or “server” and does the host have a pointer (PTR) record used for reverse DNS lookup that resolves an IP address to a domain or hostname [7].
4. **Blacklist membership –** Provides information as to whether the IP address is currently in a blacklist
5. **Geographic location properties –**As the name suggest it provides information relating to the geographical location of the IP address. This can be useful as certain locations around the world are deemed to be hotspots for malicious IP addresses.

Due to the difficulty in the procurement of new IPs, the features discussed above are extremely important for the detection of malicious URLs.

### Blacklist Features

The use of blacklists is commonplace as a method of URL detection. URL Blacklisting is the practice of various authorities such as Google and McAfee removing a websites URL from their respective indexes on security grounds. This is done through extensive analysis and is a basic access control mechanism that protects users from the malicious objectives of an attacker. A common blacklist used for web browsing purposes and installed by default on search engines such as Google Chrome, Safari and Firefox is Googles Safe Browsing [39].

[40] observed that despite its relative ease to execute as a detection method blacklisting has some pitfalls. One being that it suffers from a large amount of nontrivial false negatives due to the enormity of the work involved in maintain up-to-date lists. Another observation of a weakness of blacklisting was that attackers only needed to (in some cases) make minor alterations to a URL to avoid detection [41]. This was since even a minor disparity from the blacklist database could cause a malicious URL to effectively go undetected. To combat this they suggested lengthening the blacklist by obtaining new URLs based on a number of heuristics including Replacing Top-Level Domain (TLDs), IP Address Equivalence, Directory Structure Similarity, Query String substitution, and brand name equivalence [37].

### Content Based Features

Content based features are well known to be the most dangerous sort of features to acquire as they must be downloaded directly from the URL. Despite the concerns involved in the procurement of such features, the sheer amount of relevant information that can be retrieved can lead to a more accurate prediction model. Researchers often analyze content based features such as suspicious objects, the number of events, the use of functions such as exec (), eval (), search (), suspicious tags as well as the number of long strings because such features are frequently used in the distribution of malware [7].As suggested by [42] a more comprehensive analysis of content based features can facilitate in the early detection of threats.

Other feature representation types that information can be retrieved from include HTML features, JavaScript features, Visual features, and Context features. Figure illustrates the different feature representations used in Malicious URL detection and their properties.

A picture containing wooden, many, large, group

Description automatically generated

Figure 12: Feature Representation properties

## Libraries Used

### Pandas

Pandas is a python library that provides you with an amazing set of tools to do data analysis. In fact, if you are going to work with data in python whether it be data analysis, data science or machine learning, you are going to need Pandas. With pandas you can load, prepare, manipulate, model, and analyze data. As the developer of pandas [43] says, “the library provides integrated intuitive routines for performing common data manipulations and analysis on data sets”. The name pandas is derived from the term “panel data” which is an econometrics term for data sets that incorporate studies over numerous time periods for the same individuals [43]. Pandas has a number of features [44] that make it stand out from its competitors and as such make it the number one python library for data analysis. Library Highlights include

* A DataFrame object with integrated indexing that is both fast and efficient.
* Hierarchical axis indexing – provides the user with a more natural way of working with high dimensional data within a lower dimensional data structure.
* The ability to reshape and pivot data sets which provides the user with more flexibility.
* Various tools for reading in data from file formats such as .csv and text files, Excel, and SQL databases.
* In fact, when it comes to working with data your options are endless and it all revolves around a structure called a DataFrame.
* Enables the user to get their data in order very easily for whatever data analysis project they are working on.

For the purpose of this thesis pandas will be used to read in the datasets(.csv file) and create a data frame at various points within the project for illustration purposes. *Figure 13* below shows how pandas lets the user visualize Test Data results from the Confusion Matrix (See Appendix below for more details )

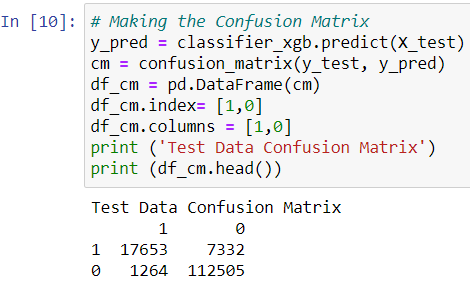


Figure 13: DataFrame created for Test Data

### Re (Regular Expressions)

Regular Expressions (RE) is a valuable tool for programmers that has many different applications such as pattern matching. Regular Expressions is essentially a programming language rooted in Python which is made available to the user through the *re* module as observed by [45]. It can be used to stipulate a given set of rules when looking to match a set of strings (e.g. e-mail addresses, plain text etc.). Regular Expressions can also be used to modify a set of strings in a number of ways.

When matching characters most letters and characters will easily match themselves. An example of this being the regular expression character string ab would match up with the string ab precisely [46]. *Figure 14* below shows a list of regular expression basics

A screenshot of a cell phone

Description automatically generated

Figure 14: Regular Expression Basic Characters

### Natural Language Toolkit

“NLTK (Natural Language Toolkit) is a leading platform for building Python programs to work with human language data” [47]. It is widely used for initial language processing and as described by its developer [48] includes a “suite of open source program modules, tutorials and problem sets”. It was developed with the intention of supporting research and teaching in areas such as artificial intelligence and machine learning.

The author of this thesis used NLTK in the development stages of the machine learning model to clean up text within the dataset through a process of converting the data (pre-processing) so that it was understandable by the computer. This was done by removing stopwords from the text in the dataset. Stopwords are commonly used words like (“the”, ”a”, ”an”) that are ignored by search engines when indexing entries for searching and when retrieving them as a result of a search query [49]. The reasoning behind the removal of stopwords is to speed up processing time. NLTK within Python has a built-in dictionary of stopwords that can be imported from nltk.corpus. *Figure 15* below shows an example of some text with and without stopwords.

A screenshot of a cell phone

Description automatically generated

Figure 15: Sample text without the use of stopwords

Another algorithm used by the author taken from NLTK is an algorithm called porter stemmer [50] which is imported from nltk.stem.porter. As described by [51] “Stemming is the process of mapping and decomposition of various forms (variants) of a word to essentially find the root word”, in effect to reduce a word to something that can used as an indexing unit in a search engine. In search of the root of a word, porter stemmer does not use a dictionary but instead uses basic analysis through a rules based algorithm [51]. Figure below, taken from the source code of Model1 (See Appendix A) shows the procedure talked about above of removing stopwords and cleaning up the text with the use of regular expression and porter stemmer in NLTK.

A screenshot of a cell phone

Description automatically generated

Figure 16: Data Cleanup using Re and NLTK

### Scikit-learn

Scikit-learn also known as sklearn is a library containing a wide range of machine learning algorithms developed for the Python programming language. As [52] observed, it is extremely popular amongst non-specialists as it focuses on ease of use whilst having minimal dependencies. Scikit-learn harnesses pythons’ high-level interactive nature to deliver state-of-the-art implementations of several machine learning algorithms. It features numerous classification, regression, and clustering algorithms such as gradient boosting (an implementation of this algorithm was used by the author in this thesis), and random forests.

Scikit-learn was used extensively by the author during the development of the machine learning models. The various algorithms taken from scikit-learn and implemented in the models are discussed below

**Train\_test\_split:** This algorithm was used to split the datasets up for the models to train and test on with 33 reserved for testing and the remaining 67% used to train the models. By withholding the data for the test, the author was assured that the models were evaluating results based on new unseen data. Typically, the features associated with the data are referred to as X and the labels as Y [53]**.** The author used the random\_state parameter to ensure that the train\_test\_split returned the same results after each execution (epoch). (See Appendix for further details).

**GridSearchCV:** GridSearchCV is a function within sklearns model\_selection package that performs hyper parameter tuning with the goal of determining the optimal values for a given model [54]. This is extremely important as the success of a model is determined by the parameter values specified to it. This author used GridSearchCV on models 2-3 (See Appendix B-C). GridSearchCV is designed to take away a lot of the tedious work involved in hyperparameter tuning by automating the process. *Figure 17* below shows how the author defined the GridSearch function, passed in a set of parameters including (estimator = model) ,which specifies to the estimator the model that we are training). This function is designed to return the best fit optimized model.

A screenshot of a cell phone

Description automatically generated

Figure 17: Defining GridSearch function

**CountVectorizer:** This algorithm is imported from sklearns feature\_extraction.text package and was used by the author in models 1 & 2 during the creation of the Bag of Words Model as can be seen below in *Figure 18*. Vectorization is required when working with machine learning algorithms as these machines cannot work directly with plain text, this leads to the need to convert the text into numerical data. This is where the Bag of Words model and CountVectorizer come into play.

As described by [55] the Bag of Words model is “simple in that it throws away all of the order information in the words and focuses on the occurrence of words in a document”. Count Vectorizer improves on this model in that it offers an easy way to both tokenize a group of text documents (or dataset) and from this build a dictionary of known words. As you can see in *Figure 18* the author has created an instance of CountVectorizer with the max\_features variable set to 1600. This builds a vocabulary that only considers the top 1600 features within the document (corpus) by term frequency [56]. CountVectorizer is then instructed to learn the vocabulary from the given document (corpus) and transform that into an array.

A screen shot of a computer

Description automatically generated

Figure 18: Feature Extraction of Model 1&2

**LabelEncoder:** When we perform classification, we typically deal with labels which can be in the form of words, numbers, or something else. When dealing with the machine learning functions within sklearn these labels are expected to be numbers. In the real-world labels are usually in the form of words, as they are human readable. In other words, a label is a word or phrase indicating what follows and where it belongs in a particular classification. We label our training data before training on our model so that mapping can be tracked.

Label encoding is the process of transforming (converting) these labels (containing words) into numerical data so it can be read by the machine learning algorithm. LabelEncoder is imported from sklearns .preprocessing package and is used to encode target values i.e. Y and not input values i.e. X [57]. As seen below in *Figure 19* the author used the fit\_transform method to fit the label encoder to the dataset and return encoded labels. The author input the target values of the dataset using the iloc indexer that locates the integers by position. The iloc indexer syntax is *data.iloc [<row selection>, <column selection>]* [58]*.*

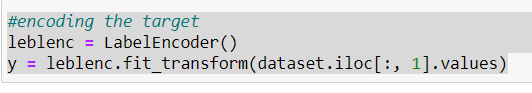
****

Figure 19: LabelEncoder

A screenshot of a cell phone

Description automatically generated

Figure 20: Iloc position-based indexer

**Confusion\_matrix:** This algorithm is imported from the sklearn.metrics package and is a table generally used to illustrate the performance of a classification model. These tables are constructed to evaluate the performance of the classifier on test data when the true values are already known. It is useful for measuring the Accuracy, Precision, Recall, Specificity and ROC Curve evaluation metrics of a model.

**Classification\_report:** This is yet another performance related visualization tool imported from the sklearn.metrics package and is often used in conjunction with the confusion matrix to evaluate a model’s performance. It is used to assess the quality of predictions from a classification model and shows the metrics Precision, Recall and F1-Score on a per class basis [59].

**Make\_column\_transformer:** Imported from the sklearn.compose module it was used to construct a ColumnTransformer from the given transformers [60]. This was used as it is necessary to first prepare raw data before fitting the machine learning model. It allows the user to apply a specific transform or a sequence of transforms to both the numerical columns and categorical columns [61] as can be seen below in *Figure 17.* The column transformer employed OneHotEncoder to encode categorical variables. StandardScaler was also used to scale the continuous data so that the distribution of it centered around 0 with a standard deviation of 1 [62]. In layman’s terms StandardScaler was used to normalize the continuous data whilst the OneHotEncoder converts categorical data into numbers.

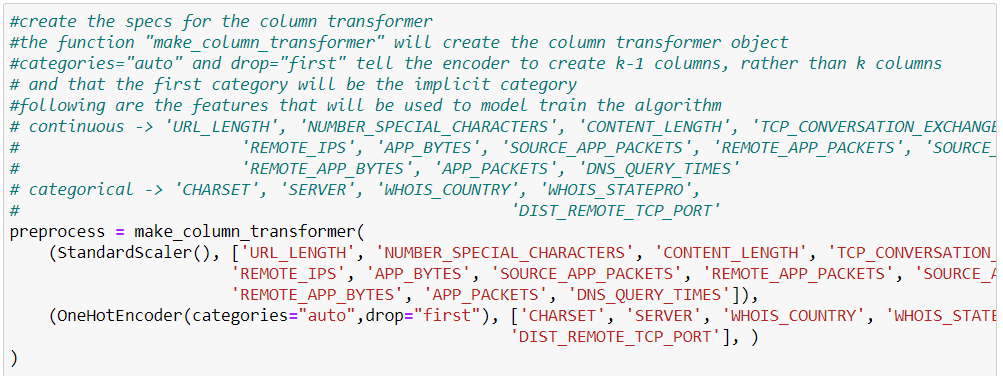


Figure 21:Creating the specs for the column transformer

### matplotlib.pyplot

“Matplotlib integrated with Python provides an interactive research and development environment with data visualization suitable for most users” [43].

## Methods of Detecting Malicious URLs

### Blacklist or Heuristic (Rule Based) Approach

A blacklist in the context of cyber security detection is a precompiled list that contains malicious URLs. This technique is very fast in its implementation as the search engine provider runs a simple query to a database when the user inputs the name of a URL. If the URL matches with one in the blacklist, a warning is then output to the user warning them of the URLs malicious nature. Blacklists have traditionally been the most popular way of detecting malicious URLs, and while this methods is fast and generally produces a low false positive rate, as discussed by [63] it as an approach does come with its limitations. One of these being the inability for a blacklist to be completely exhaustive. This is due to the resources and manpower required to achieve this. Another limitation is the fact that they tend to fail against newly generated URLs. This is a major problem as new URLs with malicious intent are created on a daily basis and are slipping through the net.

A Heuristic or Rule-based approach was developed to supplement the blacklist approach to malicious URL detection. This approach is used to identify phishing sites by way of extracting certain features common place among these type of websites [3].Some of the various heuristics of a URL are, Page Rank, Age of the Domain, IP Address, Alexa Rank, Subdomains, and DNS Record amongst others. It is these features that this type of approach analyzes when determining if a website is indeed malicious. There are two methods that use a Heuristic Based approach to detect malicious URLs, signature based, and behavior based.

A signature-based method extracts different patterns from malicious sites and classifies them like a fingerprint [3]. It is a process of establishing a unique identifier in relation to the threat (malicious website) so it can be identified in the future. The benefit of this type of method is that it can have a small error rate but a major disadvantage being that it needs additional manpower, funds, and other resources to be implemented successfully.

A behavioral based method was developed to mitigate the deficiencies that arise from signature-based methods. URLs with similar behavioral tendencies are collected, interpreted, and matched with a behavior-based signature. The main objective of this approach is “to detect the exotic and distinct malicious variants” [3] within a URL. One of the main problems with this approach is that it is very time consuming and does not include information about false positive ratios.

As already discussed the above approaches are effective in classifying whether a URL is malicious or benign, they fall short to deal with the ever-evolving techniques used by attackers today. As noted by [64] the ever-increasing amount of malicious content online requires an automated methods to detect such content. This is where a Machine Learning Approach comes into play.

### Machine Learning Approach

As already discussed above (4.3.1 Machine Learning) a Machine Learning Approach provides cyber security specialists with a technique that aims to improve on the more traditional methods of malicious URL detection. It delivers a system that can learn from experience without constant human interference. It achieves this by training on a dataset, which gives it the ability to predict if a URL is malicious or benign when future data is presented to it. The entire process by which this approach works is discussed in detail in Chapter 4. *Figure 22* below is an illustrative demonstration of this approach.

[37]A screenshot of a cell phone

Description automatically generated

Figure 22: Processing framework of malicious URL detection using a Machine Learning model

# Methodology

The objective of the research in the literature review chapter of this thesis was to establish whether the utilization of a machine learning algorithm to the task of detecting malicious URLs in a given dataset would surpass that of traditional methods such as blacklisting.

The extensive research carried out by the author was deemed to be paramount in the decision-making process of what algorithm was to be used for the purpose of this paper. There are a multitude of algorithms currently used for the purposes set out in this paper in the field of machine learning. The decision taken by the author was to use XGBoost as it is an extremely powerful machine learning algorithm particularly when it comes to precision and speed.

During the research stage carried out by the author in relation to using machine learning as a tool for malicious URL detection it was determined that a few adaptations of the XGBoost algorithm were to be applied when looking to classify a given dataset of both clean and malicious URLs.

The approach taken was to use datasets obtained from online sources of which can be found below, with a mixture of both clean and malicious URLs and to then design a machine learning model that can correctly classify this dataset.

The project on Machine Learning for malicious URL detection was developed with the programming language, using various tool and libraries such as Jupyter Notebook and Spyder IDE that were both used for code development purposed. Another valuable tool used by the author was Anaconda Navigator GUI which was used to launch applications without the need to use a command line interface.

## Proposed Method

Below is a brief run through of the proposed method to be used:

1. Acquire Datasets
2. Develop adaptations of XGBoost algorithm to suit
3. Apply algorithms to datasets for classification purposes

## Software Used

### Spyder IDE

Spyder is an extremely powerful environment written in Python specifically for Python. It is designed with data analysts in mind and offers an arrangement of tools including beautiful visualization capabilities which are beneficial when working with large amounts of data [65].

### Anaconda

Anaconda is a simple way in which a solo practitioner can implement Python/R data science and machine learning on the one machine. It has been developed with thousands of open-source packages and libraries which is why it is the words most popular Python distribution platform with over 20 million users worldwide [66]

### Anaconda Navigator

Anaconda Navigator is a graphical user interface (GUI) that lets the user to easily manage various packages, environments, and channels as well as launch applications from an easy to use interface without the need to use command line tools. . For the purpose of this thesis the author used Anaconda Navigator to launch Spyder, Jupyter Notebook and the command line tool Anaconda prompt [67] .

A screenshot of a cell phone

Description automatically generated

Figure 23: Anaconda Navigator Graphical User Interface

### Conda

Conda is an open source package management system and environment that can easily install, run and update packages and their dependencies [68]. For the purpose of this thesis the author used Conda (Anaconda3) to install XGBoost and is the run time environment for Jupyter Notebook.

### Jupyter Notebook

The Jupyter Notebook is an open web application that allows the user to create and share documents that contain live code. It has various uses which include data cleaning, numerical simulation, and machine learning. Jupyter Notebook was chosen as the primary IDE for this project for the advantages it holds over its competitors in the field of machine learning such as [69].

* The ability to easily share code with others using various tools such as e-mail, Dropbox, and GitHub.
* Jupyter Notebook being interactive in that it has the ability to export files types from your code such as HTML, images etc.
* The ease at which it can leverage big data tools from Python etc.
* Being able to then investigate that data with pandas, scikit-learn etc.

### Python 3.8.3

Python is an interpreted, high-level programming language which is considered the standard in the field of machine learning and data science in general [70]. This is attributed to its high-level interactive nature and ecosystem of scientific libraries [71]. This makes it an ideal choice for the purpose of the project at hand.

## Project Development Phases

### Acquisition of Dataset

Two datasets containing a mix of malicious and benign URLs were acquired for training and testing of the models. The first dataset (data.csv) was acquired from [72] and contained 42,464 URLs. This dataset comprised of two columns, one with the URL and the other with a label stating whether the corresponding URL was good or bad. The second dataset (dataset.csv) was acquired from the website [73] and was made up of 1,781 unique URLs(Figure ). This data set provided the author with a lot more information about the URLs than the first dataset and had 21 separate columns providing pertinent information such as URL length, number of special characters, WHOIS information, and server information.

A screenshot of a cell phone

Description automatically generated

Figure 24: Dataset.csv information

### XGBoost Model Code Development

The XGBoost models (algorithms) in this project is designed to be able to perform URL classification on the datasets provided to be able to identify whether individual URLs are malicious or benign. This was done through a process of training and testing on these datasets. The development of each model is discussed in more detail below.

#### XGBoost Model 1

The first model was developed to train and test on the (data.csv) dataset. Several libraries and algorithm taken from these libraries (packages) were imported (See Libraries Used Chapter for more details) to perform various tasks pertinent to the final goal of the model. Once all the relevant packages were imported into the model, the dataset was read in using pandas. This data (text) was then cleaned using NLTK (Natural Language Toolkit) that removed all stopwords from the data, and PorterStemmer which broke down the data to the root of each word. In the Feature Extraction phase a Bag of Words (vectorization) model was created using CountVectorizer, taking the top 1600 features into account and once transformed, was fitted to an array. The next stage was to split the dataset up for training and testing purposes. This was done using an algorithm called train\_test\_split which divided the dataset up reserving 33% for testing. The reason behind this is so the model’s performance is being tested on new unseen data. The next phase comprised of fitting the XGBClassifier to the training set with the random\_state parameter set to 42 which ensures that each time the model is run it generates the same sequence of random integers each time. This allows the model to produce consistent results. It doesn’t matter what the random\_state number is, whether it be 42 or 21, it is important that the same number is used if you want to get the same output the first time the model splits the data. When the program is run the XGBClassifier parameters are printed on output. The next step was to create a confusion matrix using pandas (DataFrame) to analyze the performance of the model. This was used to evaluate the input values against the predicted values for both the train and test split of data. A 2X2 matrix was used both times (See Testing chapter for more details). A classification report was then created with the associated arguments i.e. (y\_test, y\_pred) for both training and testing (See Testing chapter for more details). Finally, a ROC Curve (receiver operating characteristic curve) was plotted for both training and testing data. This is a form of visualization in the form of graph that shows the performance of the model by plotting the True Positive Rate and False Positive Rate.

#### XGBoost Model 2

Model 2 was developed to be an improvement on Model 1. A few changes were made which will be discussed below. These changes were designed to resolve the issues which arose when training and testing Mode1. The first change was in the feature preprocessing stage of the model were an algorithm called LabelEncoder was added (see subheading Scikit-learn above). This was used to encode labels (target values (y)) within the dataset to numerical values. An example taken from data.csv is seen below in *Tables 1*-2. As you can see when the label value repeated itself, it was assigned the same value.

|  |  |  |
| --- | --- | --- |
| Index | URL | Label |
| 1 | diaryofagameaddict.com | Bad |
| 2 | espdesign.com.au | Bad |
| 3 | iamagameaddict.com | Bad |
| 4 | en.wikipedia.org/wiki/History\_of\_IBM | Good |
| 5 | pownetwork.org/bios/m/m113.htm | Good |

Table 1: LabelEncoder Example (before encoding)

|  |  |  |
| --- | --- | --- |
| Index | URL | Label |
| 1 | diaryofagameaddict.com | 0 |
| 2 | espdesign.com.au | 0 |
| 3 | iamagameaddict.com | 0 |
| 4 | en.wikipedia.org/wiki/History\_of\_IBM | 1 |
| 5 | pownetwork.org/bios/m/m113.htm | 1 |

Table 2: LabelEncoder Example (after encoding)

The next function that was added to Model 2 is called GridSearchCV (See subheading Scikit-learn above).GridSearchCV performs hyper parameter tuning with the goal of determining the optimal values for a given model. It uses cross validation to assess models by splitting the original sample into a training and testing set. It is then divided into equal sizes with one subsamples being kept for validation purposes and the rest to train the model. This process is repeated the amount of times that is specified (in this case 3) with each of the subsamples used once for validation purposes. All possible combinations of hyperparameters are fitted to the model and each combination was scored individually with the best set of hyperparameters returned.

The function was defined with several arguments which will be discussed below.

**estimator**: input your model

**param\_grid**: define which parameters and range of values to run the cross validation on. This is to find the best hyperparameters by way of brute force i.e. (Model 2 = max\_depth and eta)

**n\_jobs**: -1 is used as standard as it utilizes all available CPUs when training the model

**scoring**: define what type of scoring. In this case accuracy was used and the function (scoring\_fit) fits it to the model. In multilabel classification, the accuracy function calculates subset accuracy, meaning the set of labels predicted by the model must equal the corresponding labels in y\_true.

The GridSearch function is then invoked to get the best fit model as can be seen in *Figure 25* . As the amount of folds was defined as 3 for each of the 9 possible combinations, a total of 27 fits were run on the model.

A screenshot of a cell phone

Description automatically generated

Figure 25: Invoking GridSearch function to find best fit model

As you can also see in the above screenshot the dataset was reduced for both training and testing purposes to avoid overloading the machine as the model is resource heavy.

As with Model 1 a classification report was printed, and a ROC Curve was plotted. I made changes in how I visualized the confusion matrix in that it was plotted to a 2x2 matrix for this model.

#### XGBoost Model 3

Model 3 was developed to detect if the URLs from a dataset are malicious or benign based on several columns of pertinent information. The model read in a different dataset (dataset.csv) to the other two models where more information was provided (i.e. WHOIS Information, URL length, Number of special characters etc.).

One of the first differences when designing Model 3 to its predecessors when reading the dataset, the model was instructed to drop down the rows which contain empty cells. This was done using the following code (dataset.dropna(inplace=True)).

In the preprocessing stage a make\_column\_transformer class was created called preprocess (*Figure 26)* to prepare raw data before fitting the model. It took in arguments where a Standard Scaler was used to normalize the continuous data (variables.) Continuous variables are numeric values (e.g. data/time) that can have an infinite number of values between any two values [74]. The remaining data (variables) within the dataset, known as categorical data, was converted into numerical data using the OneHotEncoder algorithm imported from the sklearn.preprocessing package (See Scikit-learn subheading). The fit\_transform function was called to preprocess the data with the feature dataset and replaces the values (transformation).

A screenshot of a cell phone

Description automatically generated

Figure 26: Make\_column\_transformer process

When it came to setting hyperparameter values within the parameter grid I set the gamma values (as well as max\_depth and eta like Model 2) prior to the commencement of the training process.

As with Model 2 the model then invoked the grid search function to find the best hyperparameter’s and plotted the analysis visualization graphs and matrixes.

# Training &Testing

There are four values (Precision, Recall, F1-Score and Accuracy) that are of significance when determining how well the models perform(These metrics will be discussed below). When we are talking about classification models, a confusion matrix and associated metrics is generally used to evaluate how good a model is at solving the problem put to it.

In layman’s terms a confusion matrix is a table used to gauge the performance of a classification model. For the purpose of this project the author used a 2x2 matrix to illustrate the number of accurate and inaccurate predictions made by each of the three models.

[75]A screenshot of a cell phone

Description automatically generated

Figure 27: Example 2x2 Confusion Matrix Structure

As you can see in *Figure 27* above the confusion matrix evaluates the performance of a model by showing a combination of different values associated to the predictions the model has made. Some key terms to better understand this matrix are as follows

* Positive (P): The observation is positive (e.g. URL is malicious)
* Negative (N): The observation is negative (e.g. URL is benign)
* True Positive (TP): Correctly predicted positive values
* True Negative (TN): Correctly predicted negative values
* False Positive (FP): When the actual class value is negative, but the model predicted positive. This is also known as a type 1 error
* False Negative (FN): When the actual class value is positive, but the model predicted negative. This is also known as a type 2 error

Once we have a better understanding of how a confusion matrix works we can discus some of the aforementioned metrics and how to calculate them from the confusion matrix

* Accuracy: How accurate the model was in predicting both positive and negative values. Overall accuracy of the model. This should also be as high as possible. Even with this there are concerns with the validity of a high accuracy score when evaluating a models performance as this formula assumes equal costs for both kinds of errors [76].



* Precision: From all the positive predictions made by the model, how much are actually positive.

A close up of a logo

Description automatically generated

* Recall: From all the actual positive values, how much were correctly predicted by the model. This number should be as high as possible as it answers the important question “What number of actual positives were correctly identified by the model”.

If a model has a high recall and low precision score, it submits that the model has correctly identified most positive values (low FN) but that there are also a lot of false positive predictions. If a model has a low recall and high precision score, it submits that the model has missed a lot of positive values (high FN) but that the majority of positive predictions were correctly identified (low FP) [76].

A picture containing table

Description automatically generated

* F1-Score: This is a metric that represents both Precision and Recall. It calculates the harmonic mean of both values instead of the arithmetic mean by punishing the extreme values more. Since this is the case the F1-Score of a model will always be closer to the smaller values of Precision or Recall. It is measured on a scale of 1 (perfect precision and recall) to 0.

A screenshot of a cell phone

Description automatically generated

## Test A XGBoost Model 1

The dataset (data.csv) used for training and testing Model 1 consisted of 420,464 samples. 57,484 of these were malicious URLs while the remaining 362,980 were benign. As the model was coded to split this dataset up to preserve integrity, for testing purposes the sample set comprised of 138,754 samples (18,917 malicious and 119,837 benign), totaling 33% of the total samples. As the goal of this project is to calculate how well a given model accurately identifies malicious URLs within a given dataset, the calculations relating to those numbers is what I am showing below. However, the calculations showing how well the model performed in correctly classifying benign URLs can be seen in *Figure 28* below. The same calculations are performed for Models 2 & 3.

Accuracy = (TP + TN) / (TP + TN + FP + FN) = (17653 + 112505) / (17653 + 112505 + 7332 + 1264) = 0.94

Precision = TP / (TP + FP) = 17653 / (17653 + 1264) = 0.93

Recall = TP / (TP + FN) = 17653 / (17653 + 7332) = 0.71

F1-Score = (2 \* Recall \* Precision) / (Recall + Precision) = (2 \* 0.71 \* 0.93) / (0.71 + 0.93) = 0.80

A screenshot of a cell phone

Description automatically generated

Figure 28: Model 1 Test Data Performance Details

A screenshot of a cell phone

Description automatically generated

Figure 29: Model 1 overloading computer resources on runtime

A screen shot of a computer

Description automatically generated

Figure 30: PC shutdown while running Model 1

## Test B XGBoost Model 2

When developing Model 2 as discussed above, a decision was made to reduce the size of the training and testing dataset This was due to the resource heavy nature of Model 1 during implementation, and to mitigate the out of memory error that was encountered as shown above in *Figures 29-30*. A couple of miscellaneous parameters which affect overall functionality of the model and some booster parameters that can be effective during cross validation. These are

Verbose: This designates the type of output that is to be printed when the model fits. The different values are [77].

* 0: no output generated (default)
* 1: output is generated at certain intervals
* >1: output is generated for all trees (This was chosen for Models 2 &3)

Max\_depth: This defines the maximum depth of a tree. It is advised to tune this parameter when using cross validation (cv) as it helps to control over-fitting the model as higher depth will allow the model to learn specific relations to a sample.

Eta: Also known as learning rate this parameter makes the model more robust by shrinking the weights at each step and can also be used to prevent overfitting. If the shrinkage you do at each step is 0.1 then the step weight = 0.1 [34].

Accuracy = (TP + TN) / (TP + TN + FP + FN) = (17206 + 112174) / (17206 + 112174 + 1595 + 7779) = 0.93

Precision = TP / (TP + FP) = 17206 / (17206 + 1595) = 0.92

Recall = TP / (TP + FN) = 17206 / (17206 + 7779) = 0.69

F1-Score = (2 \* Recall \* Precision) / (Recall + Precision) = (2 \* 0.69 \* 0.92) / (0.69 + 0.92) = 0.79

## Test C XGBoost Model 3

As already discussed earlier in the code development subheading a different dataset (dataset.csv) was used for training and testing Model 3. As the model was coded to split this dataset up to preserve integrity, for testing purposes (Experiment 1, See *Table 3* )the sample set comprised of 320 samples (42 malicious and 278 benign).

In the dataset associated with this model class 0 = benign and 1 = malicious.

Accuracy = (TP + TN) / (TP + TN + FP + FN) = (26 + 278) / (26 + 278 + 0 + 16) = 0.95

Precision = TP / (TP + FP) = 26 / (26 + 0) = 1

Recall = TP / (TP + FN) = 26 / (26 + 16) = 0.62

F1-Score = (2 \* Recall \* Precision) / (Recall + Precision) = (2 \* 0.62 \* 1) / (0.62 + 1) = 0.76

|  |  |  |  |
| --- | --- | --- | --- |
| Experiment | 1 (train\_test split .33) | 2 (train\_test split .15) | 3 (train\_test split .18) |
|  |  |  |  |
| Training Samples | 647 | 821 | 792 |
| Testing Samples | 320 | 146 | 175 |
| Precision | 1.00 | 0.88 | 0.82 |
| Recall | 0.62 | 0.82 | 0.78 |
| F1-Score | 0.76 | 0.85 | 0.80 |
| Accuracy | 0.95 | 0.97 | 0.96 |
|  | Optimum Parameters (GRIDSEARCH) |  |  |
| Max\_depth | 5 | 10 | 5 |
| Eta | 1 | 1 | 0.1 |
| Gamma | 0.1 | 1 | 0 |
|  |  |  |  |

Table 3: Experiment results when changing train\_test split

Experiments were conducted fitting 3 folds for each of 200 candidates, totalling 600 fits each.

## Cell Output on Runtime

### Model 1

A screenshot of a social media post

Description automatically generated

Figure 31:Cell Output on Runtime - Model 1 - Import Libraries

A screenshot of a cell phone

Description automatically generated

Figure 32: Cell Output on Runtime - Model 1 - Read Dataset

A screenshot of a cell phone

Description automatically generated

Figure 33: Cell Output on Runtime - Model 1 - XGBoost Classification

A screenshot of a cell phone

Description automatically generated

Figure 34: Cell Output on Runtime - Model 1 - Train Data Analysis - Confusion Matrix

A screenshot of a cell phone

Description automatically generated

Figure 35: Cell Output on Runtime - Model 1 - Train Data Analysis - F1 Score

A screenshot of a cell phone

Description automatically generated

Figure 36: Cell Output on Runtime - Model 1 - Test Data Analysis - Confusion Matrix

A screenshot of a cell phone

Description automatically generated

Figure 37: Cell Output on Runtime - Model 1 - Test Data Analysis - F1 Score

A close up of a map

Description automatically generated

Figure 38: Cell Output on Runtime - Model 1 - ROC Curve for Test Data

### Model 2

A screenshot of a cell phone

Description automatically generated

Figure 39: Cell Output on Runtime - Model 2 - Read Dataset

A screenshot of a cell phone

Description automatically generated

Figure 40: Cell Output on Runtime - Model 2 - XGBoost Classification

A screenshot of a cell phone

Description automatically generated

Figure 41: Cell Output on Runtime - Model 2 - Train Data Analysis - Plot Confusion Matrix

A screenshot of a cell phone

Description automatically generated

Figure 42: Cell Output on Runtime - Model 2 - Train Data Analysis - F1 Score

A screenshot of a cell phone

Description automatically generated

Figure 43: Cell Output on Runtime - Model 2 - Test Data Analysis - Plot Confusion Matrix

A screenshot of a cell phone

Description automatically generated

Figure 44: Cell Output on Runtime - Model 2 - Test Data Analysis - F1 Score

A close up of a map

Description automatically generated

Figure 45: Cell Output on Runtime - Model 2 - ROC Curve for Test Data

### Model 3

A screenshot of a cell phone

Description automatically generated

Figure 46: Cell Output on Runtime - Model 3 - Read Dataset

A screenshot of a social media post

Description automatically generated

Figure 47: Cell Output on Runtime - Model 3 - XGBoost Classification

A screenshot of a cell phone

Description automatically generated

Figure 48: Cell Output on Runtime - Model 3 - Train Data Analysis - Plot Confusion Matrix

A screenshot of a cell phone

Description automatically generated

Figure 49: Cell Output on Runtime - Model 3 - Train Data Analysis - F1 Score

A screenshot of a cell phone

Description automatically generated

Figure 50: Cell Output on Runtime - Model 3 - Test Data Analysis - Plot Confusion Matrix

A screenshot of a cell phone

Description automatically generated

Figure 51: Cell Output on Runtime - Model 3 - Test Data Analysis - F1 Score

A screenshot of a map

Description automatically generated

Figure 52: Cell Output on Runtime - Model 3 - ROC Curve for Test Data

# Analysis

## XGBoost Model 1

The author has decided not to compare these Models as different datasets were used for each one, as such It would be like comparing apples to oranges. For this reason, analysis will be done separately on each model.

In relation to Model 1 and the binary classification of URLs from the feature dataset using a bag of words model, XGBClassifier was fitted, which is an implementation of scikit-learns native API. This enabled the author to effectively use the model as a classifier within the scikit\_learn framework. When training the model, the author passed in values. The model was then used to make predictions on the test dataset and a confusion matrix, classification report and roc curve were created to visualize results and to help evaluate the performance of the model. This Model was designed specifically to work with the information provided within the dataset which is why the author used a bag of words along with natural language processing to determine whether the URLs were malicious or benign by focusing on the occurrence of words instead of the order in which they appear.

When evaluating the performance of this model as well as Models 2 & 3, from the created classification report it can be seen that it had a 94% accuracy rating which on face value suggests that the model performed extremely well in correctly predicting the URLs label classification when put up against the target values of the labels. This, however, can be misleading as accuracy equates equal weight to both types of possible outcome. In an imbalanced dataset like the ones the author used for this project where the number of benign URLs far outweighed the malicious URLs present, solely focusing on accuracy as a performance metric is not only naive but can be potentially disastrous as it doesn’t represent the true performance of this model when solving the problem put up to it. For this reason, metrics such as precision, recall and F1-score should also be considered. A ROC Curve is also a useful evaluation tool that can be used to illustrate how well the model has performed.

Precision can be explained as, from all the positive predictions made by the model, how much are actually positive. As seen in the confusion matrix above the model correctly predicted 1765 from a total of 18917 positive predictions which resulted in a 93% precision rate which is pretty good.

The Recall metric shows that from all the actual positive values, how much were correctly predicted by the model. In relation to this model Recall answers, out of all the URLs that were in fact malicious, how many did the model correctly label as malicious. This model got a recall rating of 71%. This can be considered good but there is room for improvement.

The F1-Score takes both false positives and false negatives into account and is calculated as the harmonic mean of both. This model got an F1-score of 81%. This number is generally seen to be more valuable when evaluating the performance of a model than accuracy when dealing with an unevenly distributed dataset.

A ROC Curve (Receiver Operating Curve) encapsulates the differences between the true positive and false positive rate using different probability thresholds. It is just another way of evaluating the performance of a model. It is a useful tool when comparing different models, although as already stated this will not be done here. A ROC Curve is a “plot of the false positive rate (x-axis) versus the true positive rate (y-axis) for a number of different candidate threshold values between 0.0 and 1.0. Put another way, it plots the false alarm rate versus the hit rate [78]”

The True Positive Rate is also known as Recall and as such can be calculated the same way. The False Positive Rate illustrates how often the model has predicted positive incorrectly and can be calculated using the following formula

False Positive Rate = False Positives / (False Positives + True Negatives)

Accuracy is measured by the area under the ROC Curve (AUC). These values lie between 0.5 and 1 where 0.5 indicates a bad classifier and 1 indicates a perfect classifier. A ballpark scoring system when assessing the accuracy of a model using a point system is:

* .90 – 1 = excellent
* .80 – 90 = good
* .70 - .80 = fair
* .60 - .70 = poor
* .50 - .60 = fail

This is also illustrated in *Figure 53* below.

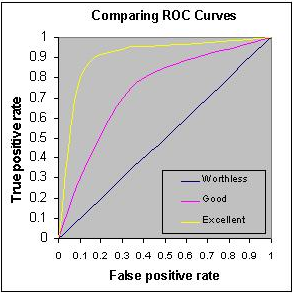


Figure 53: Comparing ROC Curves

Seen as this model scored a 0.96 of class bad indicates that this model is considered excellent as a classifier.

While the Model performed well, a few problems were encountered which if remedied could result in even better performance-based metrics. One of the problems was the sheer size of the dataset fitted to the model resulted in out of memory issues as the computers resources were unable to deal with the processing of the model.

## XGBoost Model 2

Model 2 was designed to hopefully mitigate some of the issues that arose when evaluating the performance of Model 1. The author created and invoked a GridSearch function to get the best fit model and parameters by setting a number of miscellaneous and boosting parameters that effect the functionality of the model. The dataset from which the model was to be trained on was also reduced in size to hopefully mitigate the issues discussed above. The metrics output in the classification report upon running this model were as follows

Accuracy: 0.93

Precision: 0.92

Recall: 0.69

F1-Score: 0.79

ROC Area Under Curve score: 0.95

This Model performed slightly worse than the first Model 1, but as already stated above the goal here was not to compare the Models developed in this project as they were evaluated on different datasets. It is also worth noting that with more hyperparameter tuning, it is of the opinion of the author that this model could perform far better than the results suggest. This was not possible as the computer resources did not allow for much testing due to the time involved in running tests on this model.

## XGBoost Model 3

Model 3 was developed to train on a dataset containing several continuous and categorical features which were already discussed above. This dataset had a far fewer number of unique URLs in it and a decision was also made by the author to drop a number of these URLs for consideration. The model was programmed to remove rows with cells containing blank values to get a more uniform prediction. This model also differed from the previous two in the way it preprocessed the information fitted to it . It done this by creating a column transformer and using a StandardScaler and OneHotEncoder (previously discussed) to encode continuous and categorical features within the dataset prior to training . As with Model two a GridSearch function was also created and tuned to produce the best fit model. It was decided by the author to run several experiments altering the train\_test\_split as well as other hyperparameters to see which produced the best results (See *Table 3*).

***Experiment 1: train\_test\_ split set to 33%***

Accuracy: 0.95

Precision: 1

Recall: 0.62

F1-Score: 0.76

ROC Area Under Curve score: 0.93

***Experiment 2: train\_test\_ split set to 15%***

Accuracy: 0.97

Precision: 0.88

Recall: 0.82

F1-Score: 0.85

ROC Area Under Curve score: 0.90

***Experiment 3: train\_test\_ split set to 18%***

Accuracy: 0.96

Precision: 0.82

Recall: 0.78

F1-Score: 0.80

ROC Area Under Curve score: 0.94

Upon reviewing the results of these tests and comparing them to that of the training set it is feared that the model learned rules specifically associated with the training set and that they may not have transferred well to the test set. The perfect scores output by the model on training compared with the drop off when testing suggests that the model is overfitting. Basically, it means that the model has trained so well on the training dataset that it is too well fit to that data and cannot make accurate enough predictions on new data presented to it (Test Data). This could be because of the complexity of the model and the number of features it has trained on. Mitigating this issue should result in better predictions from the Model.

# Conclusion

All three models performed well in determining whether URLs were malicious or benign, although improvements could be made of which will be discussed below. Machine Learning as a tool to be used in Identifying malicious URLs is exciting as the results shown by the algorithm used by the author in this project as well as numerous others of which have also been discussed prove that it should now be considered the standard when dealing with malicious URLs and the havoc they create around the world. The author also wants to note that creating a model specific to the problem at hand and the dataset that it is to be trained on is paramount to the success of that model

## Future Research

In relation to Models 1 & 2 and the performance issues encountered, it is suggested that by using a more powerful (higher spec) laptop, to include increased Random Access Memory (RAM) as well as a more powerful CPU, the model would produce increased performance levels due to the ability to increase the number of computations made when training and testing.

Again, in relation to the first two models it is of the opinion of the author that a technique known as under-sampling could increase the performance of these unbalanced datasets by simply removing samples from the overrepresented class (benign) [79].

For Model 3 over-sampling could be a technique to increase performance levels. This technique involves adding more samples from under- represented classes (in this case malicious) and tends to work well with smaller datasets. This can however lead to the model overfitting, a problem already encountered so a lot of consideration will be taken on what way to proceed. The issue of overfitting could be resolved by increasing the size of the dataset that the model is to be trained/tested on.

# Bibliography

|  |  |
| --- | --- |
| [1] | FirozKahn, "Detecting malicious URLs using binary classification through," *International Journal of Electrical and Computer Engineering (IJECE),* vol. 10, 2020. |
| [2] | ImmadisettiNaga, "Detection of Malicious URLs using Machine," *International Journal of Innovative Technology and Exploring Engineering (IJITEE),* vol. 8, no. 4s2, 2019. |
| [3] | Lekshmi, "DETECTING MALICIOUS URLS USING MACHINE LEARNING TECHNIQUES: A COMPARATIVE LITERATURE REVIEW," *International Research Journal of Engineering and Technology (IRJET),* vol. 06, no. 06, 2019. |
| [4] | M. Rouse, "URL (Uniform Resource Locator)," [Online]. Available: https://searchnetworking.techtarget.com/definition/URL. |
| [5] | W. A. Staff, "Static vs Dynamic Website: What Is the Difference?," 25 April 2019. [Online]. Available: https://wpamelia.com/static-vs-dynamic-website/. [Accessed 29 02 2020]. |
| [6] | DavidNield, "What Does FTP Stand for When Using a Computer?," [Online]. Available: https://smallbusiness.chron.com/ftp-stand-using-computer-70065.html. |
| [7] | MarceloFerreira, "Malicious URL Detection using Machine Learning Algorithms," Porto, 2019. |
| [8] | DebbieGrignani, "Understanding Post Office Protocol (POP3)," [Online]. Available: https://www.2brightsparks.com/resources/articles/understanding-post-office-protocol-pop3.html. |
| [9] | GeeksforGeeks, "Simple Mail Transfer Protocol (SMTP)," [Online]. Available: https://www.geeksforgeeks.org/simple-mail-transfer-protocol-smtp/. |
| [10] | SiteGround, "Email Protocols – POP3, SMTP and IMAP Tutorial," [Online]. Available: https://www.siteground.com/tutorials/email/protocols-pop3-smtp-imap/. |
| [11] | VangieBeal, "HTTP - HyperText Transfer Protocol," [Online]. Available: https://www.webopedia.com/TERM/H/HTTP.html. |
| [12] | M. Alsharnouby, "Why phishing still works," *International Journal of Human-Computer Studies,* 2015. |
| [13] | M. E. K. Kruegel, "Mitigating Drive-By Download Attacks: Challenges and Open Problems," in *iNetSec 2009 – Open Research Problems in Network Security*, Zurich, 2009. |
| [14] | Kaspersky, "What Is a Drive by Download: Everything You Need to Know," [Online]. Available: https://www.kaspersky.com/resource-center/definitions/drive-by-download. [Accessed 2020]. |
| [15] | A. Zaharia, "How Drive-by Download Attacks Work – From Disbelief to Protection," 10 November 2016. [Online]. Available: https://www.linkedin.com/pulse/how-drive-by-download-attacks-work-from-disbelief-zaharia. [Accessed July 2020]. |
| [16] | I. W. X. G. Van Lam Le, "Anatomy of Drive-by Download Attack," 201. |
| [17] | A. K. Sood, "Drive-By Download Attacks: A Comparative Study". |
| [18] | K. Krombholz, "Advanced social engineering attacks," *Journal of Information Security and Applications,* vol. 22, pp. 113-122, 2015. |
| [19] | JTF-GTMO, "File:OPSEC alert -- What is social engineering," [Online]. Available: https://commons.wikimedia.org/wiki/File:OPSEC\_alert\_--\_What\_is\_social\_engineering....jpg. |
| [20] | Imperva, "Social Engineering," [Online]. Available: https://www.imperva.com/learn/application-security/social-engineering-attack/#:~:text=Baiting,physical%20media%20to%20disperse%20malware.. [Accessed 07 2020]. |
| [21] | Sans Institute (Wendy Arthurs), "A Proactive Defence to Social," 2020. |
| [22] | S. Granger, "Social Engineering Fundamentals, Part II: Combat Strategies," 2002. |
| [23] | P. Ongsulee, "Artificial intelligence, machine learning and deep learning," in *15th International Conference on ICT and Knowledge Engineering (ICT&KE)*, Bangkok, Thailand, 2017. |
| [24] | P. S. R. A. Trishan Panch, "Artificial intelligence, machine learning and health systems," *The University of Edinburgh Journal of Global Health,* 2018. |
| [25] | N. J. Nilsson, Principles of Artificial Intelligence. |
| [26] | D. Dwivedi, "Machine Learning For Beginners," 2018. [Online]. Available: https://towardsdatascience.com/machine-learning-for-beginners-d247a9420dab. [Accessed 10 July 2020]. |
| [27] | V. Maheshwari, "Supervised and Unsupervised Learning," 2018. [Online]. Available: https://medium.com/datadriveninvestor/supervised-and-unsupervised-learning-7281050992a0. [Accessed 13 July 2020]. |
| [28] | Boozallen, "A GUIDE TO HOW MACHINES LEARN," [Online]. Available: https://www.boozallen.com/content/dam/boozallen\_site/sig/pdf/infographic/how-do-machines-learn.pdf. [Accessed July 2020]. |
| [29] | A. G. B. Richard S. Sutton, Reinforcement Learning, second edition: An Introduction, 2nd ed., MIT Press, 2018. |
| [30] | S. Ray, *Commonly used Machine Learning Algorithms (with Python and R Codes),* 2017. |
| [31] | N. Donges, *A COMPLETE GUIDE TO THE RANDOM FOREST ALGORITHM,* 2020. |
| [32] | J. D'Souza, "A Quick Guide to Boosting in ML," 2018. [Online]. Available: https://medium.com/greyatom/a-quick-guide-to-boosting-in-ml-acf7c1585cb5#:~:text=The%20term%20'Boosting'%20refers%20to,Source%3A%20Quantdare. [Accessed July 2020]. |
| [33] | M. Pathak, "Using XGBoost in Python," 2019. [Online]. Available: https://www.datacamp.com/community/tutorials/xgboost-in-python. [Accessed July 2020]. |
| [34] | A. Jain, "Complete Guide to Parameter Tuning in XGBoost with codes in Python," 2016. [Online]. Available: https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/. [Accessed July 2020]. |
| [35] | V. Morde, "XGBoost Algorithm: Long May She Reign!," 2019. [Online]. Available: https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d. [Accessed July 2020]. |
| [36] | J. Brownlee, "A Gentle Introduction to XGBoost for Applied Machine Learning," 2020. [Online]. Available: https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/. [Accessed July 2020]. |
| [37] | C. L. C. H. DOYEN SAHOO, "Malicious URL Detection using Machine Learning: A Survey," School of Information Systems, Singapore Management University, Singapore, 2019. |
| [38] | J. T. Ma, *Learning to detect malicious URLs,* San Diego: UC San Diego, 2010. |
| [39] | S. Bell, "An Analysis of Phishing Blacklists: Google Safe Browsing, OpenPhish, and PhishTank," in *Australian Computer Science Week Multiconference*, 2020. |
| [40] | M. B. J. Sushant Sinha, "Shades of grey: On the effectiveness of reputation-based “blacklists”," in *2008 3rd International Conference on Malicious and Unwanted Software (MALWARE)*, 2008. |
| [41] | P. Prakash, "PhishNet: Predictive Blacklisting to Detect Phishing Attacks," in *Infocom*, 2010. |
| [42] | D. Canali, "Prophiler: a fast filter for the large-scale detection of malicious web pages," in *20th International Conference on World Wide Web*, 2011. |
| [43] | W. McKinney, *pandas: a Foundational Python Library for Data Analysis and Statistics,* 2011. |
| [44] | Pandas, "About pandas," [Online]. Available: https://pandas.pydata.org/about/index.html. [Accessed July 2020]. |
| [45] | A. Kuchling, "Regular Expression HOWTO," [Online]. Available: https://docs.python.org/3/howto/regex.html#regex-howto. |
| [46] | T. Stubblebine, Regular Expression Pocket Reference, 2nd ed., O Reilly, 2012. |
| [47] | nltk, "Natural Language Toolkit," [Online]. Available: https://www.nltk.org/. |
| [48] | S. Bird, *NLTK: The Natural Language Toolkit,* Philadelphia, 2002. |
| [49] | GeeksforGeeks, "Removing stop words with NLTK in Python," 18 May 2020. [Online]. Available: https://www.geeksforgeeks.org/removing-stop-words-nltk-python/. |
| [50] | M. Porter, "Source code for nltk.stem.porter," [Online]. Available: https://www.nltk.org/\_modules/nltk/stem/porter.html. |
| [51] | E. U. A. S. Bonifacius Vicky Indriyono, "Portr Stemmer," *Jurnal Buana Informatika,* vol. 6, no. 4, 2015. |
| [52] | M. Brucher, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research,* 2011. |
| [53] | BitDegree, "Splitting Datasets With the Sklearn train\_test\_split Function," 21 January 2020. [Online]. Available: https://www.bitdegree.org/learn/train-test-split. |
| [54] | Krishni, "An introduction to Grid Search," 5 January 2019. [Online]. Available: https://medium.com/datadriveninvestor/an-introduction-to-grid-search-ff57adcc0998. |
| [55] | J. Brownlee, "How to Encode Text Data for Machine Learning with scikit-learn," 8 June 2020. [Online]. Available: https://machinelearningmastery.com/prepare-text-data-machine-learning-scikit-learn/. |
| [56] | scikit learn, "sklearn.feature\_extraction.text," 2020. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature\_extraction.text.CountVectorizer.html. |
| [57] | Scikit-Learn, "sklearn.preprocessing.LabelEncoder," 2020. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html. |
| [58] | S. Lynn, "Using iloc, loc, & ix to select rows and columns in Pandas DataFrames," [Online]. Available: https://www.shanelynn.ie/select-pandas-dataframe-rows-and-columns-using-iloc-loc-and-ix/#iloc-selection. |
| [59] | M. Krishnan, "Understanding the Classification report through sklearn," 7 July 2018. [Online]. Available: https://muthu.co/understanding-the-classification-report-in-sklearn/. |
| [60] | Scikit-learn, "API Reference," 2020. [Online]. Available: https://scikit-learn.org/stable/modules/classes.html. |
| [61] | J. Brownlee, "How to Use the ColumnTransformer for Data Preparation," 20 December 2019. [Online]. Available: https://machinelearningmastery.com/columntransformer-for-numerical-and-categorical-data/. |
| [62] | J. Hale, "Scale, Standardize, or Normalize with Scikit-Learn," 4 March 2019. [Online]. Available: https://towardsdatascience.com/scale-standardize-or-normalize-with-scikit-learn-6ccc7d176a02. |
| [63] | Q. P. D. S. S. C. H. Hung Le, "URLNet: Learning a URL Representation with Deep Learning for Malicious URL Detection," Singapore, 2017. |
| [64] | M. Al-Janabi, "Using supervised machine learning algorithms to detect suspicious URLs in online social networks," in *2017 IEEE/ACM International Conference on Advances in Social Network Analysis and Mining*, 2017. |
| [65] | Spyder, "Spyder," [Online]. Available: https://www.spyder-ide.org/. [Accessed 2020]. |
| [66] | Anaconda, "Individual Edition," 2020. [Online]. Available: https://www.anaconda.com/products/individual. [Accessed 03 July 2020]. |
| [67] | Anaconda, "Anaconda Navigator," 2020. [Online]. Available: https://docs.anaconda.com/anaconda/navigator/#:~:text=Anaconda%20Navigator%20is%20a%20desktop,without%20using%20command%2Dline%20commands.. [Accessed 2020]. |
| [68] | Conda, "Conda," 2020. [Online]. Available: https://conda.io/en/latest/. [Accessed 2020]. |
| [69] | Jupyter, "Jypyter," 2020. [Online]. Available: https://jupyter.org/. [Accessed July 2020]. |
| [70] | K. J. Millman, "Python for Scientists and Engineers," *Computing in Science & Engineering,* vol. 13, no. 2, 2011. |
| [71] | G. V. G. ,. M. Fabian Pedregosa, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research,* 2011. |
| [72] | faizann24, *Machine Learning and Security | Using machine learning to detect malicious URLs,* 2017. |
| [73] | C. Raerek, "Predict Malicious Websites: XGBoost," 2019. [Online]. Available: https://www.kaggle.com/craerek/predict-malicious-websites-xgboost/data. |
| [74] | Minitab, "What are categorical, discrete, and continuous variables?," [Online]. Available: https://support.minitab.com/en-us/minitab-express/1/help-and-how-to/modeling-statistics/regression/supporting-topics/basics/what-are-categorical-discrete-and-continuous-variables/#:~:text=Categorical%20variables%20contain%20a%20finite,not%20have%20a%20log. |
| [75] | T. S, "Understanding the Confusion Matrix and How to Implement it in Python," [Online]. Available: https://towardsdatascience.com/understanding-the-confusion-matrix-and-how-to-implement-it-in-python-319202e0fe4d. |
| [76] | GeeksforGeeks, "Confusion Matrix in Machine Learning," 02 February 2020. [Online]. Available: https://www.geeksforgeeks.org/confusion-matrix-machine-learning/. |
| [77] | "Complete Machine Learning Guide to Parameter Tuning in Gradient Boosting (GBM) in Python," 21 February 2016. [Online]. Available: https://www.analyticsvidhya.com/blog/2016/02/complete-guide-parameter-tuning-gradient-boosting-gbm-python/. |
| [78] | J. Brownlee, "How to Use ROC Curves and Precision-Recall Curves for Classification in Python," August 2018. [Online]. Available: https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/. |
| [79] | J. Nabi, "Machine Learning — Multiclass Classification with Imbalanced Dataset," 22 December 2018. [Online]. Available: https://towardsdatascience.com/machine-learning-multiclass-classification-with-imbalanced-data-set-29f6a177c1a. |
| [80] | V. L. W. I. G. X. S. a. K. P. Le, "Identification of potential malicious web pages," 2011. |
| [81] | T. R. (Microsoft), *What You Should Know About Drive-By Download Attacks – Part 1,* 2011. |
| [82] | RSA, "DRIVE-BY DOWNLOAD," [Online]. Available: https://www.rsa.com/content/dam/en/case-study/asoc-drive-by-download.pdf. |
| [83] | G. I. S. Blog, "Social Engineering Methods Pt 1," [Online]. Available: https://blogs.gwu.edu/gwinfosec/tag/social-engineering/#:~:text=Social%20engineering%20is%20no%20different,an%20attacker%20performs%20their%20recon.. |
| [84] | K. S. W. a. W. S. Mitnick, The Art Of Deception, Indianapolis: Wiley Publishers, 2002. |
| [85] | K. a. S. W. Mitnick, The Art of Deception, Indianapolis: Wiley Publishers, 2005. |
| [86] | F. Salahdine, "Social Engineering Attacks: A Survey," North Dakota, 2019. |
| [87] | Kiruthika\_Parthiban, "Battle between working with Spyder vs Jupyter Notebook," 7 November 2019. [Online]. Available: https://medium.com/@Kiruthika\_Parthiban/battle-between-working-with-spyder-vs-jupyter-notebook-ac457aebe93b. [Accessed July 2020]. |
| [88] | W. Koehrsen, "Interactive Controls in Jupyter Notebooks," Towards Data Science, 27 January 2019. [Online]. Available: https://towardsdatascience.com/interactive-controls-for-jupyter-notebooks-f5c94829aee6. [Accessed July 2020]. |
| [89] | D. Masoud-Rahmani, "Webpage Classification based on Compound of Using HTML Features & URL Features and Features of Sibling Pages," Tehran, 2012. |
| [90] | A. E. Nunan, "Automatic classification of cross-site scripting in web pages using document-based and URL-based features," in *IEEE Symposium on Computers and Communications (ISCC)*, 2012. |
| [91] | Q. P. D. S. S. C. H. Hung Le, "URLNet: Learning a URL Representation with Deep Learning for Malicious URL Detection," School of Information Systems, Singapore Management University, Singapore, 2017. |
| [92] | W. M. Kinney, *pandas (software),* 2008. |
| [93] | pandas, "API Reference DataFrame," 2020. [Online]. Available: https://pandas.pydata.org/docs/reference/frame.html. |
| [94] | Python, "Regular expression operations," 2020. [Online]. Available: https://docs.python.org/3/library/re.html. |
| [95] | DebuggexBeta, "Python Regex Cheatsheet," [Online]. Available: https://www.debuggex.com/cheatsheet/regex/python. |
| [96] | V. Lavrenko, *IR4.6 Stemming algorithms.* |
| [97] | J. Brownlee, "Why One-Hot Encode Data in Machine Learning?," 30 June 2017. [Online]. Available: https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/. |
| [98] | Positive\_mine, "One Hot Encoding-Method of Feature Engineering," 5 October 2019. [Online]. Available: https://medium.com/analytics-vidhya/one-hot-encoding-method-of-feature-engineering-11cc76c4b627. |
| [99] | stackoverflow, *Label encoding across multiple columns in scikit-learn,* 2014. |
| [100] | R. Shaikh, "Choosing the right Encoding method-Label vs OneHot Encoder," 9 November 2018. [Online]. [Accessed 2020]. |
| [101] | Yellowbrick, "Classification Report," 2020. [Online]. Available: https://www.scikit-yb.org/en/latest/api/classifier/classification\_report.html#quick-method. |
| [102] | E. Katz, "The Power of Grid Searching and How to Minimize Overfitting with Cross-Validation," 2017. [Online]. Available: https://www.linkedin.com/pulse/power-grid-searching-how-minimize-overfitting-erez-katz. |
| [103] | scikit-learn, "sklearn.metrics.recall\_score," 2020. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall\_score.html. |
| [104] | A. Teleron, "Cleaning Missing Values in a Pandas Dataframe," 6 November 2019. [Online]. Available: https://towardsdatascience.com/cleaning-missing-values-in-a-pandas-dataframe-a88b3d1a66bf. |
| [105] | S. P. Shruti Pankajkumar Parekh, "An approach for malicious URL detection through URL heuristic," *IJEDR,* vol. 5, no. 2, 2017. |
| [106] | C. Feeney, "Machine Learning Steganography," 2019. |
| [107] | M. Reilly, "Deep Learning Steganalysis," 2019. |

# Appendix

## Appendix A – XGBoost Model 1

# David Culligan F06861103

# Technological University Dublin 2020

**# Import Libraries**

# Libraries are discussed by the author earlier in the document

import pandas as pd

import re

from nltk.stem.porter import PorterStemmer

import nltk

nltk.download('stopwords')

from nltk.corpus import stopwords

from sklearn.model\_selection import train\_test\_split

from sklearn.feature\_extraction.text import CountVectorizer

from xgboost import XGBClassifier

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import classification\_report

import scikitplot as skplt

import matplotlib.pyplot as plt

**# Read Dataset**

**#**Read in dataset from .csv file and print the head of the document

print ('running')

dataset = pd.read\_csv('data.csv')

print(dataset.columns)

dataset.head()

**# Data Cleanup**

# cleaning the texts using nltk (natural language tool kit)

# remove general english stopwords and clean up the url

# re.sub() removes unwanted characters

# review.lower() converts capital to lowercase

# porter stemmer looking for the root of word

corpus = []

for i in range(0, dataset.shape[0]):

review = re.sub('[^a-zA-Z]', ' ', dataset['url'][i])

review = review.lower()

review = review.split()

ps = PorterStemmer()

review = [ps.stem(word) for word in review if not word in set(stopwords.words('english'))]

review = ' '.join(review)

corpus.append(review)

**# Feature Extraction**

# creating the Bag of Words model (vectorization)

# this bag of words will be the feature dataset

# top 1600 features are taken to account

#countvectorizer learns vocabulary and transform to array

cv = CountVectorizer(max\_features = 1600)

X = cv.fit\_transform(corpus).toarray()

y = dataset.iloc[:, 1].values

**# Train Test Split**

# reserve 33% of data for testing

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.33, random\_state=42)

**# XGBoost Classification**

# Fitting xgboost to the Training set

classifier\_xgb = XGBClassifier( random\_state=42)

classifier\_xgb.fit(X\_train, y\_train)

**# Train Data Analysis**

# Making the Confusion Matrix

y\_pred = classifier\_xgb.predict(X\_train)

cm = confusion\_matrix(y\_train, y\_pred)

df\_cm = pd.DataFrame(cm)

df\_cm.index= [1,0]

df\_cm.columns = [1,0]

print ('Train Data Confusion Matrix')

print (df\_cm.head())

# Higher F1 Score Means Better Model

print ('Train Data Perfomance Details')

print (classification\_report(y\_train, y\_pred))

**# Test Data Analysis**

# Making the Confusion Matrix

y\_pred = classifier\_xgb.predict(X\_test)

cm = confusion\_matrix(y\_test, y\_pred)

df\_cm = pd.DataFrame(cm)

df\_cm.index= [1,0]

df\_cm.columns = [1,0]

print ('Test Data Confusion Matrix')

print (df\_cm.head())

**# Higher F1 Score Means Better Model**

print ('Test Data Perfomance Details')

print (classification\_report(y\_test, y\_pred))

**# Plot ROC curve for test data**

y\_true = y\_test

y\_probas = classifier\_xgb.predict\_proba(X\_test)

skplt.metrics.plot\_roc(y\_true, y\_probas)

plt.show()

## Appendix B – XGBoost Model 2

# David Culligan F06861103

# Technological University Dublin 2020

**# Import Libraries**

# Libraries are discussed by the author earlier in the document

import pandas as pd

import numpy as np

import re

from nltk.stem.porter import PorterStemmer

from nltk.corpus import stopwords

from sklearn.model\_selection import train\_test\_split, GridSearchCV

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.preprocessing import LabelEncoder

from xgboost import XGBClassifier

from sklearn.metrics import plot\_confusion\_matrix

from sklearn.metrics import classification\_report

import scikitplot as skplt

import matplotlib.pyplot as plt

**# Read Dataset**

dataset = pd.read\_csv('data.csv')

dataset.head()

**# Data Cleanup**

# cleaning the texts using nltk (natural language tool kit)

# remove general english stopwords and clean up the url

corpus = []

for i in range(0, dataset.shape[0]):

review = re.sub('[^a-zA-Z]', ' ', dataset['url'][i])

review = review.lower()

review = review.split()

ps = PorterStemmer()

review = [ps.stem(word) for word in review if not word in set(stopwords.words('english'))]

review = ' '.join(review)

corpus.append(review)

**# Feature Extraction**

# creating the Bag of Words model (vectorization)

# this bag of words will be the feature dataset

# top 1600 features are taken to account

cv = CountVectorizer(max\_features = 1600)

X = cv.fit\_transform(corpus).toarray()

**#Encoding the target**

**#** fit\_transform method used to fit the label encoder to the dataset and return encoded labels.

# target values of the dataset input using the iloc indexer that locates the integers by position.Y = target

leblenc = LabelEncoder()

y = leblenc.fit\_transform(dataset.iloc[:, 1].values)

**# Train Test Split**

# reserve 33% of data for testing

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.33, random\_state=42)

**# XGBoost Classification**

# a function to define the gridsearch inorder to get the best fit model with optimized parameters

# function return the best fit optimized model after performing cross validation

# parameter\_grid requires a list of parameters and the range of values for each parameter of the specified model

def gridsearchfn(X\_train\_data, X\_test\_data, y\_train\_data, y\_test\_data,

model, param\_grid, cv=10, scoring\_fit='accuracy'):

gs = GridSearchCV(

estimator=model,

param\_grid=param\_grid,

cv=cv,

n\_jobs=-1,

scoring=scoring\_fit,

verbose=2

)

fitted\_model = gs.fit(X\_train\_data, y\_train\_data)

return fitted\_model

# define xgboost classifier with random state as 42 as model. This allowed me to get same sequence of numbers each time I split the dataset

model = XGBClassifier(random\_state=42)

# parameter grid is defined inorder to search through and get the best fit parameters

param\_grid = {

'max\_depth': [15,20,25],

'eta': [0.1, 0.2, 0.3]

}

# invoke the gridsearch function to get the best fit model and parameters

# reduce the train and test data size inorder to avoid out of memory error

# this was done because of issues when running this mode and model

model = gridsearchfn(X\_train[:10000], X\_test[:3000], y\_train[:10000], y\_test[:3000], model, param\_grid, cv=3)

print('Best Parameters : ', model.best\_params\_)

**# Train Data Analysis**

# plot the confusion matrix

plot\_confusion\_matrix(model, X\_test, y\_test)

plt.title('Test Data')

**# Higher F1 Score Means Better Model**

print ('Train Data Perfomance Details')

y\_pred = model.predict(X\_train)

print (classification\_report(y\_train, y\_pred))

**# Test Data Analysis**

# plot the confusion matrix

plot\_confusion\_matrix(model, X\_test, y\_test)

plt.title('Test Data')

**# Higher F1 Score Means Better Model**

print ('Test Data Perfomance Details')

y\_pred = model.predict(X\_test)

print (classification\_report(y\_test, y\_pred))

**# Plot ROC curve for test data**

y\_true = y\_test

y\_probas = model.predict\_proba(X\_test)

skplt.metrics.plot\_roc(y\_true, y\_probas)

plt.show()

## Appendix C – XGBoost Model 3

# David Culligan F06861103

# Technological University Dublin 2020

**# Import Libraries**

import pandas as pd

from sklearn.model\_selection import train\_test\_split, GridSearchCV

from xgboost import XGBClassifier

from sklearn.metrics import classification\_report

import scikitplot as skplt

import matplotlib.pyplot as plt

from sklearn.metrics import plot\_confusion\_matrix

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.compose import make\_column\_transformer

**# Read Dataset**

#Dataset.dropna drops down the rows that contain empty cells

dataset = pd.read\_csv('dataset.csv')

dataset.dropna(inplace=True)

dataset.head()

**# Feature Engineering**

# create feature and target dataset using iloc indexer

feature\_dataset = dataset.iloc[:, :-1]

target = dataset.iloc[:, -1]

#create the specs for the column transformer

#the function "make\_column\_transformer" will create the column transformer #object

#categories="auto" and drop="first" tell the encoder to create k-1 columns, #rather than k columns

#and that the first category will be the implicit category

#following are the features that will be used to model train the algorithm

# continuous -> 'URL\_LENGTH', 'NUMBER\_SPECIAL\_CHARACTERS','CONTENT\_LENGTH', #'TCP\_CONVERSATION\_EXCHANGE','REMOTE\_IPS','APP\_BYTES','SOURCE\_APP\_PACKETS', #'REMOTE\_APP\_PACKETS', 'SOURCE\_APP\_BYTES','REMOTE\_APP\_BYTES','APP\_PACKETS', #'DNS\_QUERY\_TIMES'

# categorical -> 'CHARSET', 'SERVER', 'WHOIS\_COUNTRY', 'WHOIS\_STATEPRO',

#'DIST\_REMOTE\_TCP\_PORT'

# StandardScaler was used to normalize the continuous data whilst the #OneHotEncoder converts categorical data into numbers

preprocess = make\_column\_transformer(

(StandardScaler(), ['URL\_LENGTH', 'NUMBER\_SPECIAL\_CHARACTERS', 'CONTENT\_LENGTH', 'TCP\_CONVERSATION\_EXCHANGE',

'REMOTE\_IPS', 'APP\_BYTES', 'SOURCE\_APP\_PACKETS', 'REMOTE\_APP\_PACKETS', 'SOURCE\_APP\_BYTES',

'REMOTE\_APP\_BYTES', 'APP\_PACKETS', 'DNS\_QUERY\_TIMES']),

(OneHotEncoder(categories="auto",drop="first"), ['CHARSET', 'SERVER', 'WHOIS\_COUNTRY', 'WHOIS\_STATEPRO',

'DIST\_REMOTE\_TCP\_PORT'], )

)

#Fit\_transform is the function that fits the transformations to the data #and then does the transformation

X = preprocess.fit\_transform(feature\_dataset)

# target dataset

y = target

**# Train Test Split**

# reserve 33% of data for testing

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.33, random\_state=42)

**# XGBoost Classification**

# a function to define the gridsearch inorder to get the best fit model #with optimized parameters

# function return the best fit optimized model

def gridsearchfn(X\_train\_data, X\_test\_data, y\_train\_data, y\_test\_data,

model, param\_grid, cv=10, scoring\_fit='accuracy'):

gs = GridSearchCV(

estimator=model,

param\_grid=param\_grid,

cv=cv,

n\_jobs=-1,

scoring=scoring\_fit,

verbose=2

)

fitted\_model = gs.fit(X\_train\_data, y\_train\_data)

return fitted\_model

# define xgboost classifier with random state as 42 as model.

model = XGBClassifier(random\_state=42)

# parameter grid is defined inorder to search through and get the best fit #parameters

param\_grid = {

'max\_depth': [5,10,15,20,25,30,35,40,45,50],

'eta': [0.001, 0.01, 0.1, 1],

'gamma': [0, 0.1 , 1, 2, 5]

}

# invoke the gridsearch function to get the best fit model and parameters

model = gridsearchfn(X\_train, X\_test, y\_train, y\_test, model, param\_grid, cv=3)

print('Best Parameters : ', model.best\_params\_)

**# Train Data Analysis**

# plot the confusion matrix

plot\_confusion\_matrix(model, X\_train, y\_train)

plt.title('Train Data')

**# Higher F1 Score Means Better Model**

y\_pred = model.predict(X\_train)

print ('Train Data Perfomance Details')

print (classification\_report(y\_train, y\_pred))

**# Test Data Analysis**

# plot the confusion matrix

plot\_confusion\_matrix(model, X\_test, y\_test)

plt.title('Test Data')

**# Higher F1 Score Means Better Model**

print ('Test Data Perfomance Details')

y\_pred = model.predict(X\_test)

print (classification\_report(y\_test, y\_pred))

**# Plot ROC curve for test data**

y\_true = y\_test

y\_probas = model.predict\_proba(X\_test)

skplt.metrics.plot\_roc(y\_true, y\_probas)

plt.show()