

DETECTION OF PHISHING WEBSITES USING MACHINE LEARNING

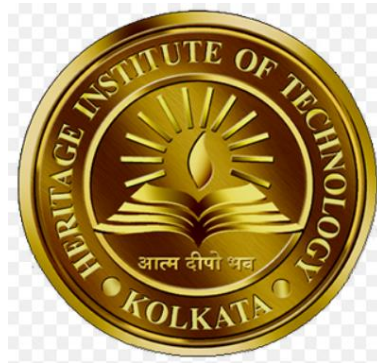
A PROJECT REPORT

In a partial fulfilment of the Requirements for the award of the degree of
BACHELOR OF COMPUTER APPLICATION

Under the guidance of
MAHENDRA DUTTA

By

**HERITAGE INSTITUTE OF TECHNOLOGY,
KOLKATA**



In association with



DECLARATION

We hereby declare that the project work being presented in the project proposal entitled “Detection of Phishing Websites using Machine Learning and Python” in partial fulfilment of the requirements for the award of the degree of BACHELOR OF COMPUTER APPLICATION at HERITAGE INSTITUTE OF TECHNOLOGY, is an authentic work carried out under the guidance of Mahendra Dutta. The matter embodied in this project work has not been submitted elsewhere for the award of any degree of our knowledge and belief.

Date: 08/07/2024

Name of the Students: SAYAN SAM

BIVASH ROY

RAJDIP MUKHERJEE

SWAPNIL DEY

ACKNOWLEDGEMENT

Success of any project depends largely on the encouragement and guidelines of many others. We take this sincere opportunity to express our gratitude to the people who have been instrumental in the successful completion of this project work. We would like to show our greatest appreciation to MAHENDRA DUTTA, Project Engineer at Ardent Computech Pvt. Ltd. We always feel motivated and encouraged every time by his valuable advice and constant inspiration; without his encouragement and guidance this project would not have materialized. Words are inadequate in offering our thanks to the other trainees, project assistants and other members at Ardent Computech Pvt. Ltd. for their encouragement and cooperation in carrying out this project work. The guidance and support received from all the members and who are contributing to this project, was vital for the success of this project.

CONTENTS

1. ABSTRACT	5
2. CHAPTER 1: INTRODUCTION	6
3. CHAPTER 2: LITTERATURE REVIEW	9
4. CHAPTER 3: SYSTEM ANALYSIS AND DESIGN	15
5. CHAPTER 4: SYSTEM IMPLEMENTATION AND RESULT	27
6. CHAPTER 5: SUMMARY, CONCLUSION, RECOMENDATIONS AND FUTURE WORKS	45
7. REFERENCE	48

ABSTRACT

Phishing attacks are a rapidly expanding threat in the cyber world, costing internet users billions of dollars each year. It is a criminal crime that involves the use of a variety of social engineering tactics to obtain sensitive information from users. Phishing techniques can be detected using a variety of types of communication, including email, instant chats, pop-up messages, and web pages. This study develops and creates a model that can predict whether a URL link is legitimate or phishing.

The data set used for the classification was sourced from an open source service called 'Phish Tank' which contain phishing URLs in multiple formats such as CSV, JSON, etc. and also from the University of New Brunswick dataset bank which has a collection of benign, spam, phishing, malware & defacement URLs. Over six (6) machine learning models and deep neural network algorithms all together are used to detect phishing URLs.

This study aims to develop a web application software that detects phishing URLs from the collection of over 5,000 URLs which are randomly picked respectively and are fragmented into 80,000 training samples & 20,000 testing samples, which are equally divided between phishing and legitimate URLs. The URL dataset is trained and tested base on some feature selection such as address bar-based features, domain-based features, and HTML & JavaScript-based features to identify legitimate and phishing URLs.

In conclusion, the study provided a model for URL classification into phishing and legitimate URLs. This would be very valuable in assisting individuals and companies in identifying phishing attacks by authenticating any link supplied to them to prove its validity.

CHAPTER 1

INTRODUCTION

1.1 Background Information

The Internet has become an important part of our lives for gathering and disseminating information, particularly through social media. According to Pamela (2021), the Internet is a network of computers containing valuable data, so there are many security mechanisms in place to protect that data, but there is a weak link: the human. When a user freely gives away their data or access to their computer, security mechanisms have a much more difficult time protecting their data and devices.

Therefore, Imperva (2021) defines social engineering (a type of attack used to steal user data, including login credentials and credit card numbers) as a type of attack that is one of the most common social engineering attacks. The attack happens when an attacker fools a victim into opening an email, instant message, or text message as if it were from a trusted source. Upon clicking the link, the recipient is fooled into believing that they've received a gift and unsuspectingly clicks a malicious link, resulting in the installation of malware, the freezing of the system as part of a ransomware attack, or the disclosure of sensitive information.

Computer security threats have increased substantially in recent years, owing to the rapid adoption of technology improvements, while simultaneously increasing the vulnerability of human exploitation. Users should know how the phishers do it, and they should also be aware of techniques to help protect themselves from becoming phished.

The strategies employed by cybercriminals are becoming more complex as technology advances. Other than phishing, there are a variety of methods for obtaining personal information from users. KnowBe4 (2021) stated the following techniques:

a) Vishing (Voice Phishing): This kind of phishing includes the phisher calling the victim to get personal information about the bank account. The most common method of phone phishing is to use a phony caller ID.

b) Smishing (SMS Phishing): Phishing via Short Message Service (SMS) is known as Smishing. It is a method of luring a target through the SMS text message service by sending a link to a phishing website.

c) Ransomware: A ransomware attack is a type of attack that prevents users from accessing a device or data unless they pay up.

d) Malvertising: Malvertising is malicious advertising that uses active scripts to download malware or push undesirable information onto your computer. The most prevalent techniques used in malvertisements are exploits in Adobe PDF and Flash.

Hence, this is a rapidly evolving threat to individuals as well as big and small corporations. Criminals now have access to industrial-strength services on the dark web, resulting in an increase in the amount of these phishing links and emails, and, more frighteningly, they are increasing in 'quality,' making them tougher to detect.

1.2 Aim of Study

This project aims to detect phishing websites using machine learning and deep neural networks by developing a web application that allows users to check if a URL is phishing or legitimate and have access to resources to help tackle phishing attacks.

1.3 Objectives of the Study

To accomplish the project's purpose, the following particular objectives have been established:

- i) dataset collection and pre-processing;
- ii) machine-learning model selection and development ;
- iii) development of a web-based application for detection;
- iv) Integration of the developed model to a web application.

1.4 Scope of the Study

This study explores data science and machine learning models that use datasets gotten from open-source platforms to analyze website links and distinguish between phishing and legitimate URL links.

The model will be integrated into a web application, allowing a user to predict if a URL link is legitimate or phishing. This online application is compatible with a variety of browsers.

CHAPTER 2

LITERATURE REVIEW

2.1 Overview of the Study

This chapter offers an insight into various important studies conducted by excellent scholars from articles, books, and other sources relevant to the detection of phishing websites. It also provides the project with a theoretical review, conceptual review, and empirical review to demonstrate understanding of the project.

2.2 Theoretical Review

According to Internet world stats ("Internet world stats usage and population statistics", 2014), the total numbers of Internet users worldwide are 2.97 billion in 2014; that is, more than 38% of the world population uses the Internet. Hackers take advantage of the insecure Internet system and can fool unaware users to fall for phishing scams. A phishing e-mail is used to defraud both individuals and financial organizations on the Internet. ("RSA Anti-Fraud Command Center", n.d.) Said the Anti-Phishing Working Group (APWG) is an international consortium that is dedicated to promoting research, education, and law enforcement to eliminate online fraud and cyber-crime. In 2012, total phishing attacks increased by 160% over 2011, signifying a record year in phishing volumes. The total phishing attacks detected in 2013 were approximately 450,000 and led to financial losses of more than 5.9 billion dollars ("RSA Anti-Fraud Command Center", n.d.). Total attack increases by 1% in 2013 as compared to 2012. The total number of phishing attacks noticed in Q1 (first quarter) of 2014 was 125,215, a 10.7 percent increase over Q4 (fourth quarter) of 2013. More than 55% of phishing websites contain the name of the target site in some form to fool users and 99.4% of phishing websites use port 80 ("Anti-Phishing Working Group (APWG) Phishing activity trends report first quarter", 7 2014). According to the APWG report in the first quarter of 2014, the second-highest number of phishing attacks ever recorded was between January and March 2014 ("Anti Phishing Working Group (APWG) Phishing activity trends report first quarter", 2014) and payment services are the most targeted industry.

During the second half of 2014, 123,972 unique phishing attacks were observed ("APWG report", 2014).

In the year 2011, total financial losses were 1.2 billion, and they rose to 5.9 billion dollars in 2013. The financial losses due to phishing attacks in 2014 and 2015 were 4.5 and 4.6, respectively, as shown in Figure 2.1 ("The RSA Current State of Cybercrime", n.d.). The growth of phishing attacks from 2005 to 2015 is shown in Figure 2.2.



Figure 2.1 Worldwide financial losses (in billion) due to phishing attacks.

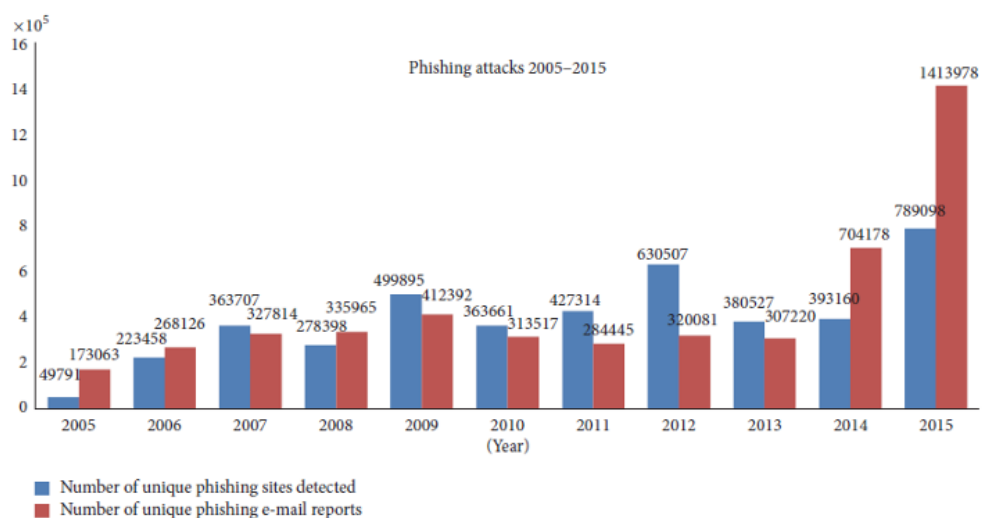


Figure 2.2 growth of phishing attacks from 2005 to 2015

2.3 Feature Extraction

Feature extraction is a process of dimensionality reduction by which an initial set of raw data is reduced to more manageable groups for processing. A characteristic of these large data sets is a large number of variables that require a lot of computing resources to process. Feature extraction is the name for methods that select and or combine variables into features, effectively reducing the amount of data that must be processed, while still accurately and completely describing the original data set.

These feature selections include:

- i)Address Bar Features
- ii)HTML and Java Script-based Features
- iii)Domain-based Features

All the listed feature selection above consists of feature extraction which are guided by rules. The feature extraction is as follows:

2.3.1 Address bar Features

(1) Using the IP Address

Rule: IF $\begin{cases} \text{If The Domain Part has an IP Address} \rightarrow \text{Phishing} \\ \text{Otherwise} \rightarrow \text{Legitimate} \end{cases}$

(2) Long URL to Hide the Suspicious Part

Rule: IF $\begin{cases} \text{URL length} < 54 \rightarrow \text{feature} = \text{Legitimate} \\ \text{else if URL length} \geq 54 \text{ and } \leq 75 \rightarrow \text{feature} = \text{Suspicious} \\ \text{otherwise} \rightarrow \text{feature} = \text{Phishing} \end{cases}$

(3) Using URL Shortening Services “Tiny-URL”

Rule: IF $\begin{cases} \text{TinyURL} \rightarrow \text{Phishing} \\ \text{Otherwise} \rightarrow \text{Legitimate} \end{cases}$

(4) URL's having "@" Symbol

Rule: IF $\begin{cases} \text{Url Having @ Symbol} \rightarrow \text{Phishing} \\ \text{Otherwise} \rightarrow \text{Legitimate} \end{cases}$

(5) Redirecting using "/"

Rule: IF $\begin{cases} \text{the position of the Last Occurrence of "/" in the URL} > 7 \rightarrow \text{Phishing} \\ \text{Otherwise} \rightarrow \text{Legitimate} \end{cases}$

(6) Adding Prefix or Suffix Separated by (-) to the Domain

Rule: IF $\begin{cases} \text{Domain Name Part Includes (-) Symbol} \rightarrow \text{Phishing} \\ \text{Otherwise} \rightarrow \text{Legitimate} \end{cases}$

(7) The Existence of "HTTPS" Token in the Domain Part of the URL

Rule: IF $\begin{cases} \text{Using HTTP Token in Domain Part of The URL} \rightarrow \text{Phishing} \\ \text{Otherwise} \rightarrow \text{Legitimate} \end{cases}$

2.3.2 Domain-based Features

(1) Age of Domain

Rule: IF $\begin{cases} \text{Age Of Domain} \geq 6 \text{ months} \rightarrow \text{Legitimate} \\ \text{Otherwise} \rightarrow \text{Phishing} \end{cases}$

(2) DNS Record

Rule: IF $\begin{cases} \text{no DNS Record For The Domain} \rightarrow \text{Phishing} \\ \text{Otherwise} \rightarrow \text{Legitimate} \end{cases}$

2.3.3 HTML and JavaScript-based Features

(1) Website Forwarding

Rule: IF $\begin{cases} \text{ofRedirect Page} \leq 1 \rightarrow \text{Legitimate} \\ \text{of Redirect Page} \geq 2 \text{ And } < 4 \rightarrow \text{Suspicious} \\ \text{Otherwise} \rightarrow \text{Phishing} \end{cases}$

(2) Status Bar Customization

Rule: IF $\begin{cases} \text{onMouseOver Changes Status Bar} \rightarrow \text{Phishing} \\ \text{It Does't Change Status Bar} \rightarrow \text{Legitimate} \end{cases}$

(3) Disabling Right Click

Rule: IF $\begin{cases} \text{Right Click Disabled} \rightarrow \text{Phishing} \\ \text{Otherwise} \rightarrow \text{Legitimate} \end{cases}$

(5) Iframe Redirection

Rule: IF $\begin{cases} \text{Using iframe} \rightarrow \text{Phishing} \\ \text{Otherwise} \rightarrow \text{Legitimate} \end{cases}$

2.4 Empirical Review

Algorithm used	Reference paper	No. of Features	Dataset	Language/Tools	Conclusion
Decision Tree (DT), Random Forest (RF), Gradient Boosting (GBM), Generalized Linear Model (GLM), Generalized Additive Model (GAM)	J. Shad and S. Sharma,	30	Not Mentioned	Python, R Language	Random Forest highest accuracy 98.4%
Support vector Machine (SVM), Naïve Bayes (NB) and Extreme Learning Machine(ELM)	Y. Sönmez, T. Tuncer, H. Gökal, and E. Avci,	30	UCI-Machine Learning Repository	MATLAB	ELM achieved 95.34% accuracy.
Natural Language Processing	T. Peng, I. Harris, and Y. Sawa,	-	Nazario Phishing Email set	Python	Proposed SEAHound provides 95% accuracy
Bayesian Network, Stochastic Descent (SGD), lazy. K. Star, Randomizable Filtered Classifier, Logistic model tree (LMT) and ID3 (iterative Dichotomiser), Multilayer	M. Karabatak and T. Mustafa	27	UCI-Machine Learning Repository	MATLAB, WEKA	Lazy. K.Star obtained 97.58% accuracy

Perception, JRip, PART, J48, Random Forest and Random Tree					
Random Forest	S. Parekh, D. Parikh, S. Kotak, and P. S. Sankhe	8	Phishtank,	RStudio	95% accuracy
Neural network model Adam, AdaDelta and SGD	K. Shima	URL length	Phishtank	Chainer	Accuracy of Adam 94.18%
Convolution neural network (CNN) and SNN long short-term memory (CNN-LSTM)	A. Vazhayil, R. Vinayakumar, and K. Soman,	-	Phishtank, OpenPhish, MalwareDomainlist, MalwareDomain	TensorFlow in conjunction with Keras	CNN-LSTM obtained 98% accuracy
Logistic Regression and Support Vector Machine (SVM)	W. Fadheel, M. Abusharkh, and I. Abdel-Qader,	19	UCI machine learning repository	Big Data	SVM accuracy 95.62%
AdaBoost, Bagging, Random Forest, and SMO	X. Zhang, Y. Zeng, X. Jin, Z. Yan, and G. Geng	11	DirectIndustry Anti-Phishing Alliance of China	Big Data	Only Semantic Features of word embedding obtained high accuracy.
C4.5 decision tree	L. MacHado and J. Gadge,	9 features and heuristic values	Phishtank Google	-	89.40%
KNN, SVM and Random Forest	A. Desai, J. Jatakia, R. Naik, and N. Raul	22	UCI-Machine Learning Repository	HTML, JavaScript, CSS, Python	Random Forest high accuracy

CHAPTER THREE

SYSTEM ANALYSIS AND DESIGN

3.1 Overview of System Analysis

This chapter describes the various process, methods, and procedures adopted by the researcher to achieve the set aim and objectives and the conceptual structure within which the research was conducted.

The methodology of any research work refers to the research approach adopted by the researcher to tackle the stated problem. Since the efficiency and maintainability of any application are solely dependent on how designs are prepared. This chapter provides detailed descriptions of methods employed to proffer solutions to the stated objectives of the research work.

According to the Merriam-Webster dictionary (11th.Ed) , system analysis is "the process of studying a procedure or business to identify its goals and purposes and create systems and procedures that will efficiently achieve them". It is also the act, process, or profession of studying an activity (such as a procedure, a business, or a physiological function) typically by mathematical means to define its goals or purposes and to discover operations and procedures for accomplishing them most efficiently. System analysis is used in every field where the development of something is done. Before planning and development, you need to understand the old systems thoroughly and use the knowledge to determine how best your new system can work.

3.4 Model Development

The model development method takes several models, tests them, and adds them to an iterative process until a model that meets the required requirements is developed. Figure 3.1 shows the steps used in the development of machine learning models using both supervised and unsupervised learning.

The following are the stages to machine learning model development for phishing detection systems:

i. Data Collection

The data used to generate the datasets on which the models are trained are gotten from different open-source platforms. The dataset collection consists of phishing and legitimate URL dataset.

The set of phishing URLs are collected from an open-source service called Phish Tank. This service provides a set of phishing URLs in multiple formats like CSV, JSON and so on that gets updated hourly. This dataset is accessible from the phishtank.com website. From this dataset, over 5000 random phishing URLs are collected to train the ML models.

The set of legitimate URLs are obtained from the open datasets of the University of New Brunswick. This dataset is accessible on the university website. This dataset has a collection of benign, spam, phishing, malware & defacement URLs. Out of all these types, the benign URL dataset is considered for this project. From this dataset, Over 5000 random legitimate URLs are collected to train the ML models.

ii. **Preprocessing**

Data preprocessing is the first and crucial step after data collection. The raw dataset obtained for phishing detection was prepared by removing redundant and irregular data and also encoded using the One-Hot Encoding technique into a useful and efficient format suitable for the machine learning model.

iii. **Exploratory data analysis**

Exploratory data analysis (EDA) technique was used on the dataset after series of data cleaning. The data visualization method was employed to analyze, explore and summarize the dataset. These visualization consist of heat-map, histograms, box plots, scatter plots, and pair plots to uncover patterns and insights within data.

iv. **Feature Extraction**

Feature Extraction aims to reduce the number of features in a dataset by creating new features from the existing ones. Thus, Website content-based features were extracted from phishing and legitimate datasets such as the Address bar-based feature which consists of 9 features, Domain-based feature which consists of 4 features, and HTML & JavaScript-based Feature which consists of 4 features. So, altogether 17 features were extracted for phishing detection.

3.2 Analysis of Existing System

The existing system of phishing detection techniques suffers low detection accuracy and high false alarm especially when different phishing approaches are introduced. Above and beyond, the most common technique used is the blacklist-based method which is inefficient in responding to emanating phishing attacks since registering a new domain has become easier, no comprehensive blacklist can ensure a perfect up to-date database for phishing detection.

3.3 Proposed System

The proposed phishing detection system utilizes machine learning models and deep neural networks. The system comprises two major parts, which are the machine learning models and a web application. These models consist of Decision Tree, Support Vector Machine, XGBooster, Multilayer Perceptions, Auto Encoder Neural Network, and Random Forest.

These models are selected after different comparison-based performances of multiple machine learning algorithms. Each of these models is trained and tested on a website content-based feature, extracted from both phishing and legitimate dataset.

Hence, the model with the highest accuracy is selected and integrated into a web application that will enable a user to predict if a URL link is phishing or legitimate.

3.3.1 Benefits of the new system

i. Will be able to differentiate between phishing(O) and legitimate(I) URLs ii.

It Will help reduce phishing data breaches for an organization iii. It Will be helpful to individuals and organizations iv. It is easy to use

3.5.1 System architecture

Architectural design is concerned with understanding how a system should be organized and designing the overall structure of that system, it shows how different components of the system work together to achieve its main objectives. It is the process for identifying sub-systems making up a system and the framework for sub-system control and communication. The diagram below represents a graphical overview of the architectural design of the proposed system.

Figure 3.1 shows the architecture view of the proposed phishing detection system such that a user enters a URL link and the link moves through different trained machine learning and deep neural network models and the best model with the highest accuracy is selected. Thus, the selected model is deployed as an API (Application Programming Interface) which is then integrated into a web application. Hence, a user interacts with the web application which is accessible across different display devices such as computers, tablets, and mobile devices.

3.5.2 Use a case diagram of the system

The Use Case diagram describes the functionality of the system as designed from the requirements, it summarizes the details of a system and the users within the system. It is a behavior diagram and visualizes the observable interactions between actors and the system under development. The Use case diagram consists of the system, the related use cases, and actors and relates to each other.

Figure 3.2 shows the Use case scenarios that a user can carry out on the phishing detection system.

v. Model Training

Model Training involves feeding Machine learning algorithms with data to help identify and learn good attributes of the dataset.

This research problem is a product of supervised learning, which falls under the classification problem. The algorithms used for phishing detection consist of supervised machine learning models (4) and deep neural network (2) which was used to train the dataset. These algorithms include Decision Tree, Random Forest, Support Vector Machines, XGBooster, Multilayer Perceptron, and Auto-encoder Neural Network. All these models were trained on the dataset. Thus, the dataset is spitted into a training and testing set. The training model consists of 80% of the dataset to enable the machine learning models to learn more about the data and be able to distinguish between phishing and legitimate URLs.

vi. Model Testing

Model Testing involves the process where the performance of a fully trained model is evaluated on a testing set.

Thus, after 80% of data has been trained, 20% of the dataset is used to evaluate the trained dataset to see the performance of the models.

vii. **Model Evaluation**

Model Evaluation involves estimating the generalization accuracy of models and deciding whether the model performs better or not.

Thus, Scikit-learn (sklearn matrices) module was used to implements several score and utility functions to measure the classification performance to properly evaluate the models deployed for phishing detection.

3.5 System Modelling

System modelling involves the process of developing an abstract model of a system, with each model presenting a different view or perspective of the system. It is the process of representing a system using various graphical notations that shows how users will interact with the system and how certain parts of the system function. The proposed system was modelled using the following diagrams:

- i. Architecture diagram
- ii. Use case diagram
- iii. Flowcharts

The proposed system will be implemented using Python Programming language along with different machine learning models and libraries such as pandas, scikit-learn, python who-is, beautiful-Soup, NumPy, seaborn, and matplotlib. Etc.

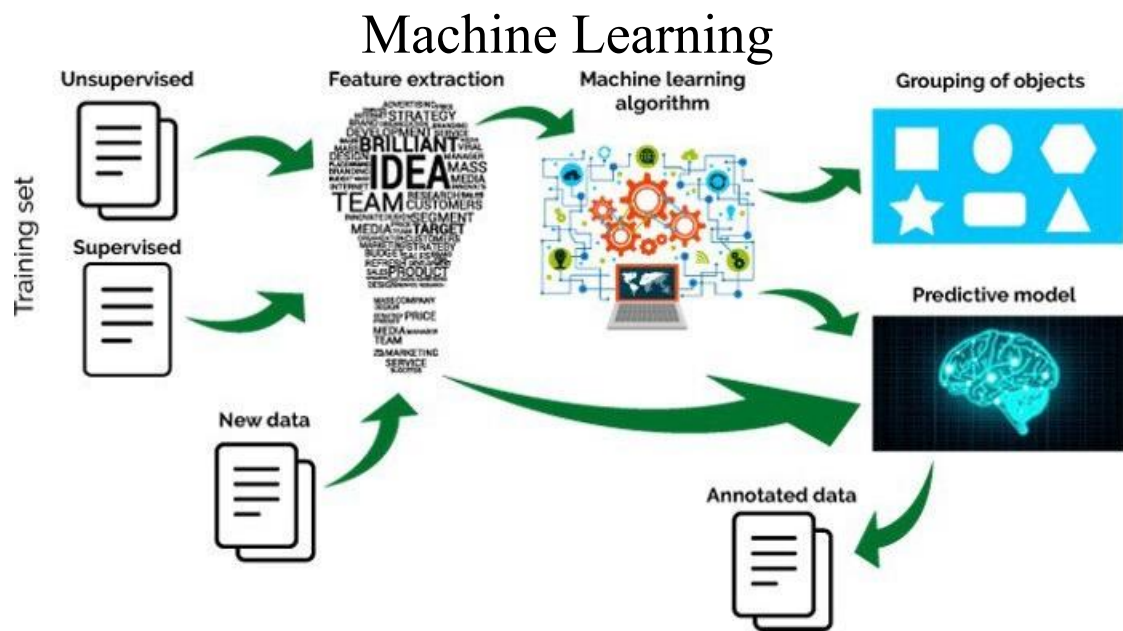


Figure 3.1 Machine Learning development process

(Source: Ayush, 2019)

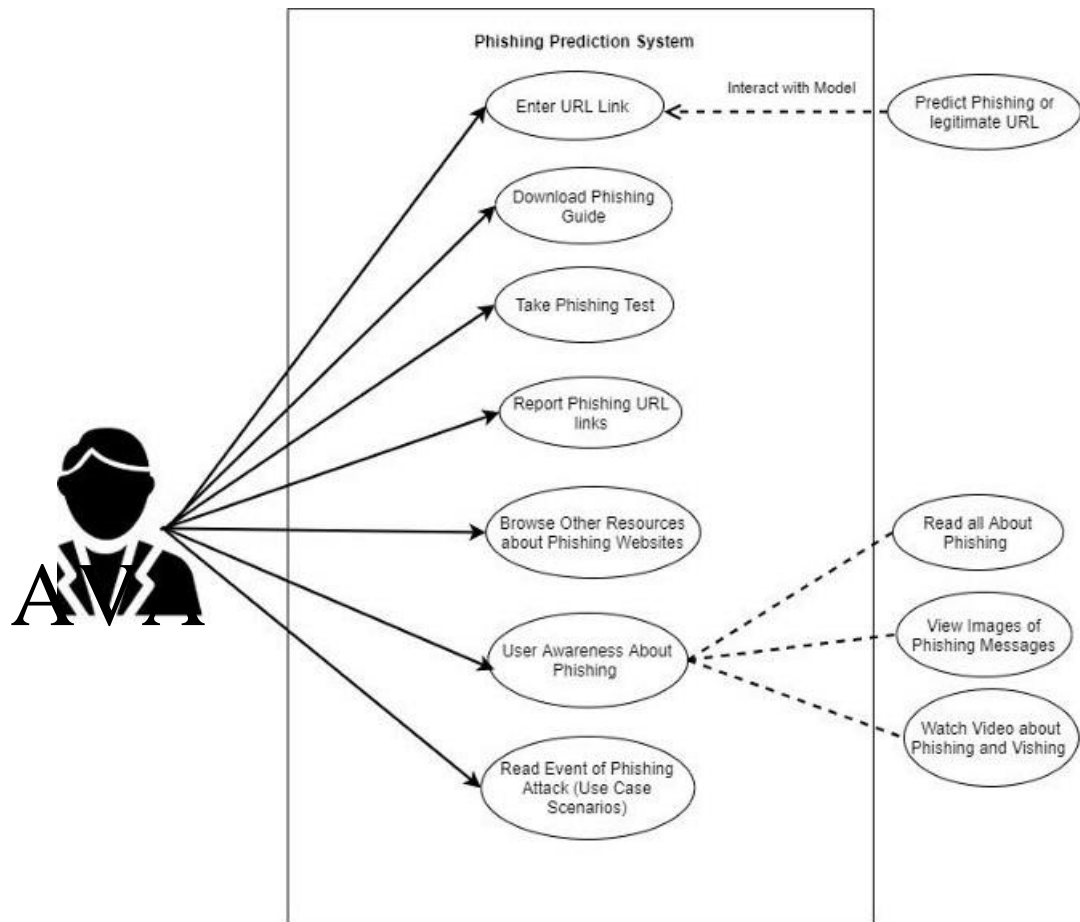


Figure 3.3 Use Case diagram for Proposed System

3.5.3 Flowchart of the system

A flowchart is a diagram that depicts a process, system, or computer algorithm. It is a graphical representation of the steps that are to be performed in a system, it shows the steps in sequential order. It is used in presenting the flow of algorithms and to communicate complex processes in clear, easy-to-understand diagrams.

Figure 3.3 shows the flow of phishing detection systems using the machine learning process.

Figure 3.4 shows the phishing detection web interface system. The user inputs a URL link and the website validates the format of the URL and then predicts if the link is phishing or legitimate.

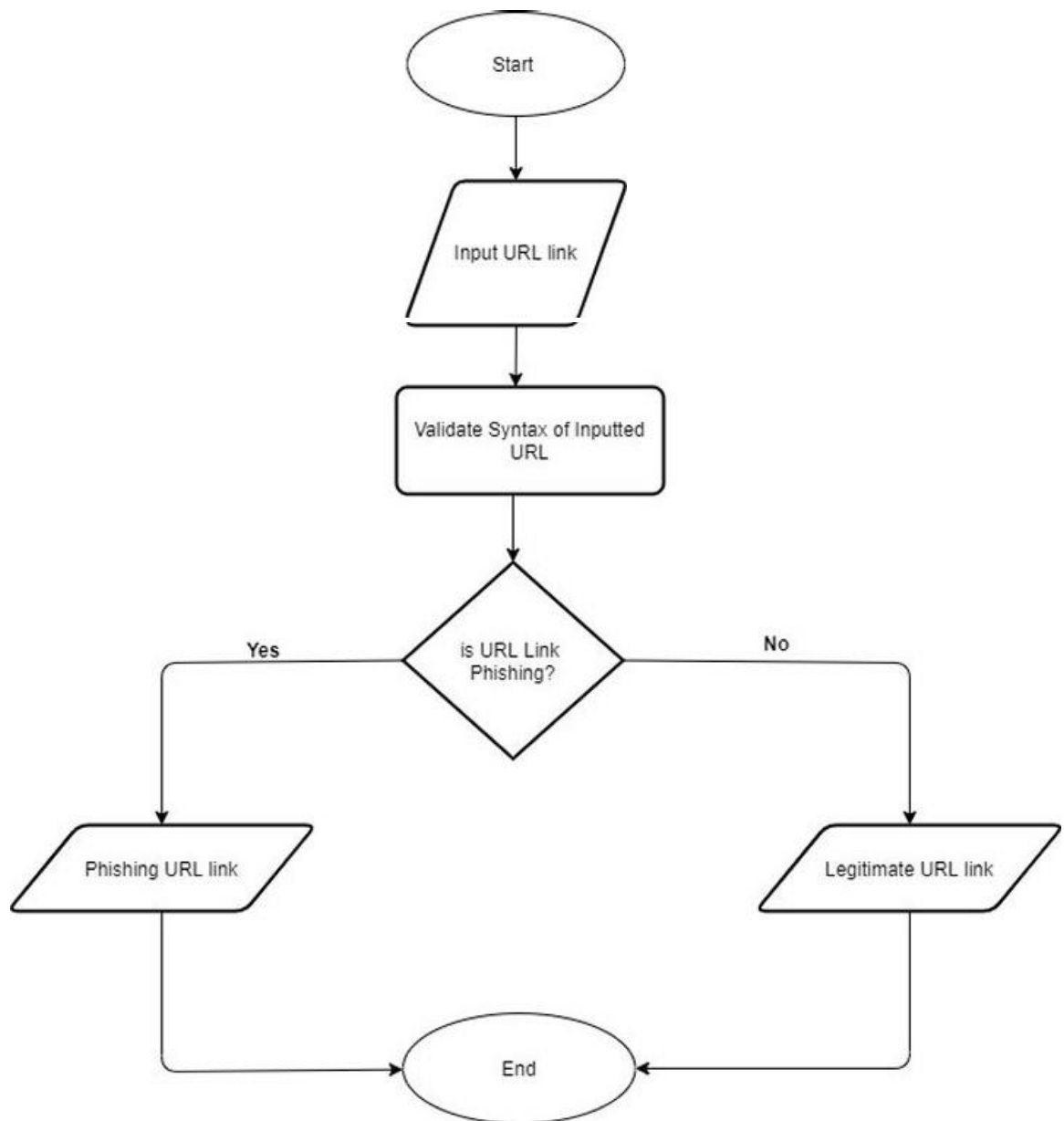


Figure 3.5 Flowchart of the web interface

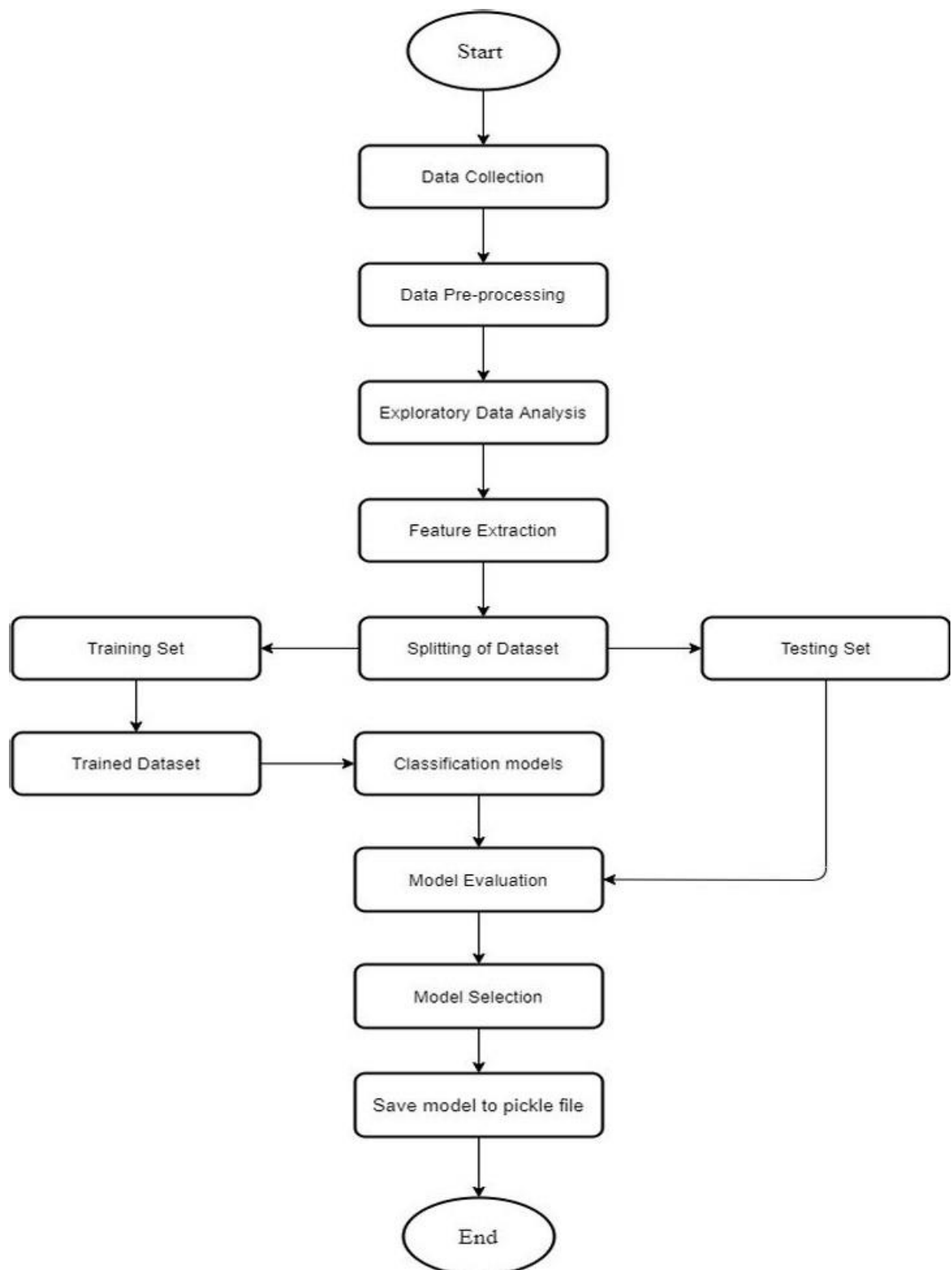


Figure 3.4 Flowchart of the proposed System

CHAPTER FOUR

SYSTEM IMPLEMENTATION AND RESULTS

This chapter deals with the implementation of multiple machine learning models for the detection of phishing website as earlier designed in the preceding chapters.

The implementation is concerned with all the activities that took place to put up the newly developed system into operation (using the approach that was stated in the methodology (e.g. architectural diagram, flowchart, uses case, etc.)) to achieve the objectives of the project to convert the theoretical design into a working system. The components of the system were also tested and evaluated.

Software requirements

The software requirements for the development of this system include:

- i. Windows Operating System (8/10)
- ii. Anaconda Navigator (Jupyter Notebook)
- iii. Web browser (Preferably Chrome)

4.1 Model Development

The model for detecting phishing URL websites was built using a python programming language with over six (6) machine learning models and deep neural network algorithms altogether and the most accurate test score on the tested 5,000 datasets were used.

4.2.1 Data collection

The dataset used for the classification was sourced from was gotten from multiple sources listed in the earlier stated methodology.

The dataset used for classifying the dataset into phishing and legitimate URLs was sourced from open-source websites, samples of which are shown below in figure 4.1 and 4.2 respectively.

	A	B	C	D	E	F	G	H	I	J
1	http://1337x.to/torrent/1048648/American-Sniper-2014-MD-ITALIAN-DVDS-264-BST-MT/									
2	http://1337x.to/torrent/1110018/Blackhat-2015-RUSSIAN-720p-WEB-DL-DD5-1-H264-RUFGT/									
3	http://1337x.to/torrent/1122940/Blackhat-2015-x264-1080p-WEB-DL-eng-nl-sub-sharky/									
4	http://1337x.to/torrent/1124395/Fast-and-Furious-7-2015-HD-TS-XVID-AC3-HQ-Hive-CM8/									
5	http://1337x.to/torrent/1145504/Avengers-Age-of-Ultron-2015-CAM-New-Audio-x264-CPG/									
6	http://1337x.to/torrent/1160078/Avengers-age-of-Ultron-2015-HQ-CAM-H264-AC3-MURD3R/									
7	http://1337x.to/torrent/294349/American-Idol-S11E04-Auditions-4-HDTV-XviD-FQM-ettv/									
8	http://189.cn/dqmh/userCenter/myOrderInfoList.do?method=listMyOrderInfo_new&isVs=no									
9	http://2gis.ru/moscow/search/%D0%9F%D0%BE%D0%B5%D1%81%D1%82%D1%8C/tab/firms/zoom/11									
10	http://abc.go.com/shows/general-hospital/episode-guide/2015-05/08-friday-may-8-2015									
11	http://abc.go.com/shows/the-muppets/video/new-abc-comedy-trailers?cid=abchp_muppets									
12	http://abcnews.go.com/US/wireStory/regulators-delays-georgia-nuclear-plant-31020059									
13	http://adultfriendfinder.com/css/live_cd/ffadult/english/0/font_face-1427390957.css									
14	http://akhbarelyom.com/news/newdetails/410322/1/%D8%A8%D9%88%D8%B6%D9%88%D8%AD.html									
15	http://allegro.pl/amadeus-quartet-haydn-string-quartets-collectors-i5207998383.html									
16	http://allegro.pl/narzedzia-i-sprzet-warsztatowy-18554?ref=simplified-category-tree									
17	http://allegro.pl/royal-string-quartet-music-for-string-quartet-cd-i5262876831.html									
18	http://allegro.pl/sexy-g-string-stringi-z-koralikami-must-have-uni-i5039087215.html									
19	http://allegro.pl/sporty-strzeleckie-i-myslistwo-13495?ref=simplified-category-tree									
20	http://allegro.pl/sporty-towarzyskie-i-rekreacja-13408?ref=simplified-category-tree									
21	http://allegro.pl/triumph-miss-sexy-allday-string-stringi-36-s-bez-i4855636396.html									
22	http://allegro.pl/triumph-stringi-exquisite-essence-string-stal-38-i5073330901.html									
23	http://allegro.pl/wyszczuplajace-body-string-pod-biust-jony-srebra-i5124700240.html									

Figure 4.1 Dataset of Phishing URLs

Source: The Dataset is collected from an open-source service called Phish-Tank. This dataset consists of 5,000 random phishing URLs which are collected to train the ML models.

	A	B	C	D	E	F	G	H	I	J
1	phish_id	url	phish_det	submissio	verified	verificatio	online	target		
2	6911546	https://jai	http://ww	2021-01-0	yes	2021-01-0	yes	Other		
3	6911545	http://poj	http://ww	2021-01-0	yes	2021-01-0	yes	Other		
4	6911536	https://sp	http://ww	2021-01-0	yes	2021-01-0	yes	Other		
5	6911494	https://hy	http://ww	2021-01-0	yes	2021-01-0	yes	Other		
6	6911483	http://sto	http://ww	2021-01-0	yes	2021-01-0	yes	Other		
7	6911482	https://oc	http://ww	2021-01-0	yes	2021-01-0	yes	Other		
8	6911481	https://tri	http://ww	2021-01-0	yes	2021-01-0	yes	Other		
9	6911480	https://jo	http://ww	2021-01-0	yes	2021-01-0	yes	Other		
10	6911467	https://su	http://ww	2021-01-0	yes	2021-01-0	yes	Other		
11	6911466	https://cri	http://ww	2021-01-0	yes	2021-01-0	yes	Other		
12	6911465	http://est	http://ww	2021-01-0	yes	2021-01-0	yes	Other		
13	6911424	https://ek	http://ww	2021-01-0	yes	2021-01-0	yes	eBay, Inc.		
14	6911414	https://ise	http://ww	2021-01-0	yes	2021-01-0	yes	Development Bank of Singa		
15	6911408	http://wir	http://ww	2021-01-0	yes	2021-01-0	yes	Other		
16	6911403	http://ma	http://ww	2021-01-0	yes	2021-01-0	yes	ABSA Bank		
17	6911400	http://ww	http://ww	2021-01-0	yes	2021-01-0	yes	Other		
18	6911398	https://bi	http://ww	2021-01-0	yes	2021-01-0	yes	Other		
19	6911395	https://fg	http://ww	2021-01-0	yes	2021-01-0	yes	Other		
20	6911394	http://fgh	http://ww	2021-01-0	yes	2021-01-0	yes	Other		
21	6911389	https://wl	http://ww	2021-01-0	yes	2021-01-0	yes	Other		
22	6911388	https://wl	http://ww	2021-01-0	yes	2021-01-0	yes	Other		
23	6911382	http://clfi	http://ww	2021-01-0	yes	2021-01-0	yes	Other		

Figure 4.2 Dataset of Legitimate URLs

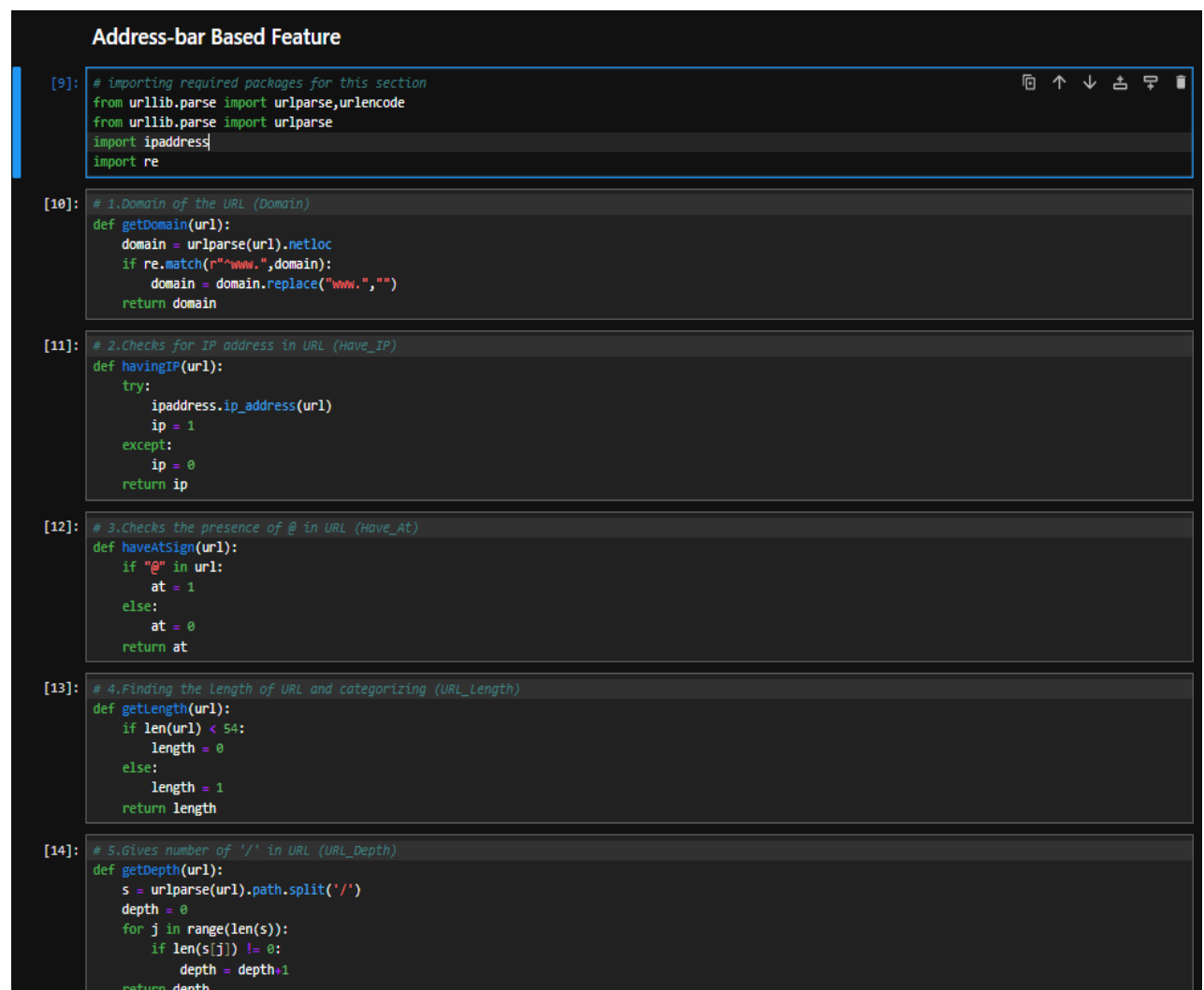
Source: The Dataset were obtained from the open datasets of the University of New Brunswick. The benign URL dataset is considered for this project. This dataset consists of 5,000 random legitimate URLs which are collected to train the ML models

4.2.2 Feature extraction on the datasets

The features extraction used on the dataset are categorized into...

- i. Address bar-based features
- ii. Domain-based features
- iii. Html & java-script based features

In figure 4.3, figure 4.4, and figure 4.5 the images show the list of code feature extraction done on the dataset while figure 4.6 shows the code computation for all the feature extraction used on the dataset.



```
Address-bar Based Feature

[9]: # importing required packages for this section
from urllib.parse import urlparse,urlencode
from urllib.parse import urlparse
import ipaddress
import re

[10]: # 1.Domain of the URL (Domain)
def getDomain(url):
    domain = urlparse(url).netloc
    if re.match(r"^www.",domain):
        domain = domain.replace("www.", "")
    return domain

[11]: # 2.Checks for IP address in URL (Have_IP)
def havingIP(url):
    try:
        ipaddress.ip_address(url)
        ip = 1
    except:
        ip = 0
    return ip

[12]: # 3.Checks the presence of @ in URL (Have_At)
def haveAtSign(url):
    if "@" in url:
        at = 1
    else:
        at = 0
    return at

[13]: # 4.Finding the Length of URL and categorizing (URL_Length)
def getLength(url):
    if len(url) < 54:
        length = 0
    else:
        length = 1
    return length

[14]: # 5.Gives number of '/' in URL (URL_Depth)
def getDepth(url):
    s = urlparse(url).path.split('/')
    depth = 0
    for j in range(len(s)):
        if len(s[j]) != 0:
            depth = depth+1
    return depth
```

```

[15]: # 6.Checking for redirection '/' in the url (Redirection)
def redirection(url):
    pos = url.rfind('/')
    if pos > 6:
        if pos > 7:
            return 1
        else:
            return 0
    else:
        return 0

[16]: # 7.Existence of "HTTPS" Token in the Domain Part of the URL (https_Domain)
def httpDomain(url):
    domain = urlparse(url).netloc
    if 'https' in domain:
        return 1
    else:
        return 0

[17]: #Listing shortening services
shortening_services = r"bit\ly|goo\gl|shorte\st|go2l\ink|x\co|ow\ly|t\co|tinyurl|tr\im|is\gd|cli\gs|" \
    r"yfrog\com|migre\me|ff\im|tiny\cc|url4\eu|twit\ac|su\pr|twurl\nl|snipurl\com|" \
    r"short\to|BudURL\com|ping\fm|post\ly|Just\as|bkite\com|snipr\com|fic\kr|loopt\us|" \
    r"doiop\com|short\ie|kl\am|wp\me|rubyurl\com|om\ly|to\ly|bit\do|t\co|lnd\in|db\tt|" \
    r"qr\ae|adf\ly|goo\gl|bitly\com|cur\lv|tinyurl\com|ow\ly|bit\ly|ity\im|q\gs|is\gd|" \
    r"po\st|bc\vc|twitthis\com|u\to|j\mp|buzurl\com|cutt\us|u\bb|yourls\org|x\co|" \
    r"prettylinkpro\com|scrnch\me|filoops\info|vztur\com|qr\net|1url\com|tweez\me|v\gd|" \
    r"tr\im|link\zip\net"

[18]: # 8. Checking for Shortening Services in URL (Tiny_URL)
def tinyURL(url):
    match=re.search(shortening_services,url)
    if match:
        return 1
    else:
        return 0

[19]: # 9.Checking for Prefix or Suffix Separated by (-) in the Domain (Prefix/Suffix)
def prefixSuffix(url):
    if '-' in urlparse(url).netloc:
        return 1 # phishing
    else:
        return 0 # Legitimate

```

Figure 4.3: Code for Address bar-based feature extraction

Domain Based Feature

```
[20]: # importing required packages for this section
import re
from bs4 import BeautifulSoup
import urllib
import urllib.request
from datetime import datetime
import whois

[ ]:

[21]: # 13.Survival time of domain: The difference between termination time and creation time (Domain_Age)
def domainAge(domain_name):
    creation_date = domain_name.creation_date
    expiration_date = domain_name.expiration_date
    if (isinstance(creation_date,str) or isinstance(expiration_date,str)):
        try:
            creation_date = datetime.strptime(creation_date,'%Y-%m-%d')
            expiration_date = datetime.strptime(expiration_date,"%Y-%m-%d")
        except:
            return 1
    if ((expiration_date is None) or (creation_date is None)):
        return 1
    elif ((type(expiration_date) is list) or (type(creation_date) is list)):
        return 1
    else:
        ageofdomain = abs((expiration_date - creation_date).days)
        if ((ageofdomain/30) < 6):
            age = 1
        else:
            age = 0
    return age

[22]: # 14.End time of domain: The difference between termination time and current time (Domain_End)
def domainEnd(domain_name):
    expiration_date = domain_name.expiration_date
    if isinstance(expiration_date,str):
        try:
            expiration_date = datetime.strptime(expiration_date,"%Y-%m-%d")
        except:
            return 1
    if (expiration_date is None):
        return 1
    elif (type(expiration_date) is list):
        return 1
    else:
        today = datetime.now()
        end = abs((expiration_date - today).days)
        if ((end/30) < 6):
            end = 1
        else:
            end = 0
    return end

[23]: # 11.DNS Record availability (DNS_Record)
def check_dns(url):
    dns = 0
    try:
        domain_name = whois.whois(urllib.parse.urlparse(url).netloc, timeout=10)
    except:
        dns = 1
    return dns
```

Figure 4.4: Code for domain-based features extraction

HTML and JavaScript based Features

```
[24]: # importing required packages for this section
import requests

[25]: # 15. IFrame Redirection (iFrame)
def iframe(response):
    if response == "":
        return 1
    else:
        if re.findall(r"<iframe>|<frameBorder>", response.text):
            return 0
        else:
            return 1

[26]: # 16.Checks the effect of mouse over on status bar (Mouse_Over)
def mouseOver(response):
    if response == "":
        return 1
    else:
        if re.findall("<script>.+onmouseover.+</script>", response.text):
            return 1
        else:
            return 0

[27]: # 17.Checks the status of the right click attribute (Right_Click)
def rightClick(response):
    if response == "":
        return 1
    else:
        if re.findall(r"event.button ?== ?2", response.text):
            return 0
        else:
            return 1

[28]: # 18.Checks the number of forwardings (Web_Forwards)
def forwarding(response):
    if response == "":
        return 1
    else:
        if len(response.history) <= 2:
            return 0
        else:
            return 1
```

Figure 4.5: Code for Html & java-script based features extraction


```
Computing URL Features

[29]: def featureExtractions(url):

    features = []
    #Address bar based features (9)
    features.append(getDomain(url))
    features.append(havingIP(url))
    features.append(haveAtSign(url))
    features.append(getLength(url))
    features.append(getDepth(url))
    features.append(redirection(url))
    features.append(httpDomain(url))
    features.append(prefixSuffix(url))
    features.append(tinyURL(url))

    #Domain based features (4)
    features.append(check_dns(url))
    #features.append(is_indexed_by_google(url))
    features.append(1 if check_dns(url) == 1 else domainAge(domain_name))
    features.append(1 if check_dns(url) == 1 else domainEnd(domain_name))

    # HTML & Javascript based features (4)
    try:
        response = requests.get(url)
    except:
        response = ""
    features.append(iframe(response))
    features.append(mouseOver(response))
    features.append(rightClick(response))
    features.append(forwarding(response))

    return features

featureExtractions('http://www.facebook.com')

[29]: ['facebook.com', 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0]
```

Figure 4.6: Code computation for all the feature extraction used dataset.

4.2.3 Data Analysis & Visualization

The image as shown in figure 4.7 shows the distribution plot of how legitimate and phishing datasets are distributed base on the features selected and how they are related to each other.

In figure 4.8 shows the plot of a correlation heat-map of the dataset. The plot shows correlation between different variables in the dataset.

In figure 4.9, figure 4.10, figure 4.11 and figure 4.12, it shows the Confusion Matrix in the model for Decision tree classifier, Random Forest classifier, Support Vector Machine and XGBoost classifier respectively.

In figure 4.13 and 4.14, it shows Auto-Encoder Neural Network accuracy and error progress with the change of epochs and Multilayer Perceptron's neuron structure.

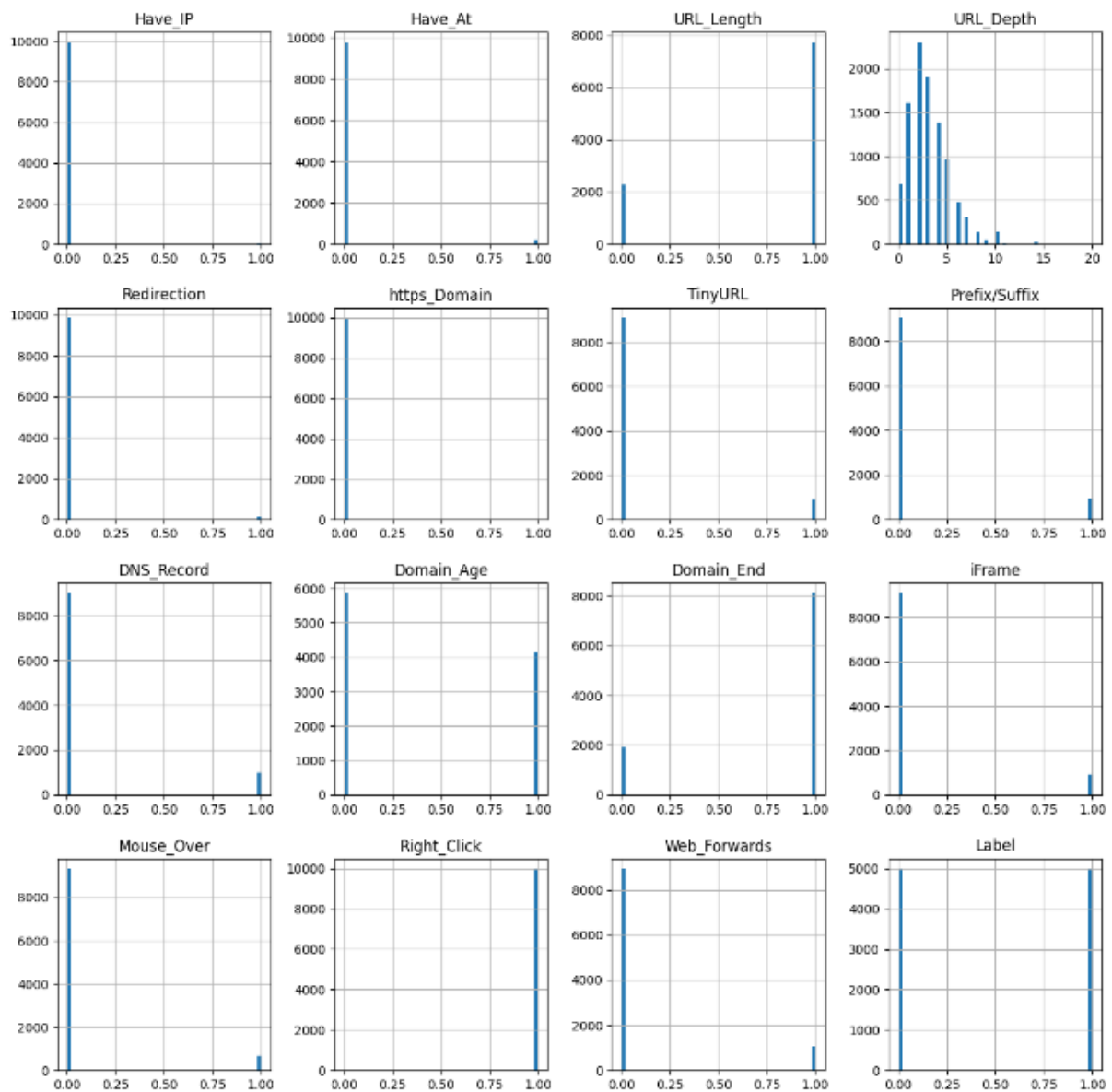


Figure 4.7: Distribution plot of dataset base on the features selected

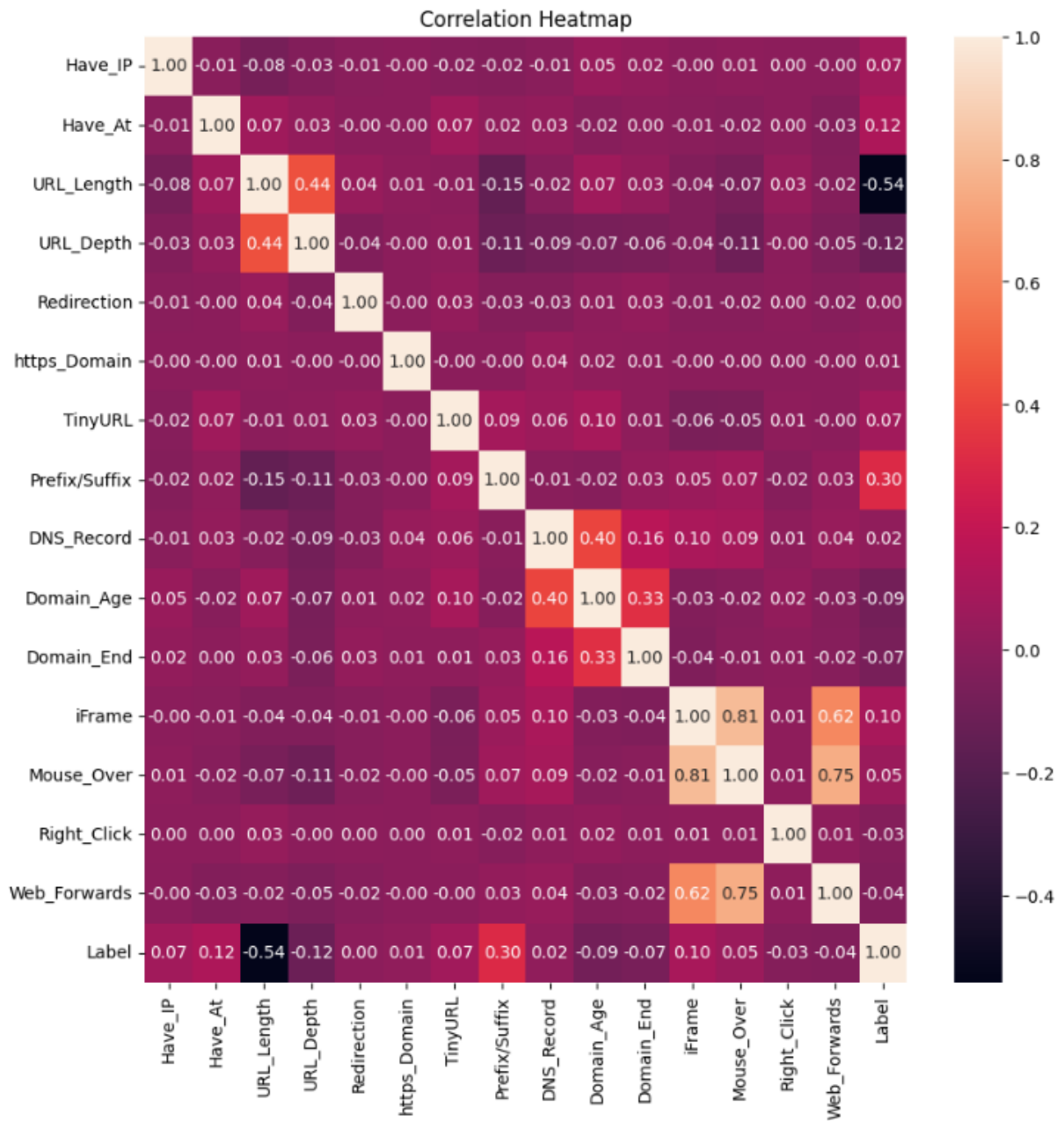


Figure 4.8: Correlation heat map of the dataset

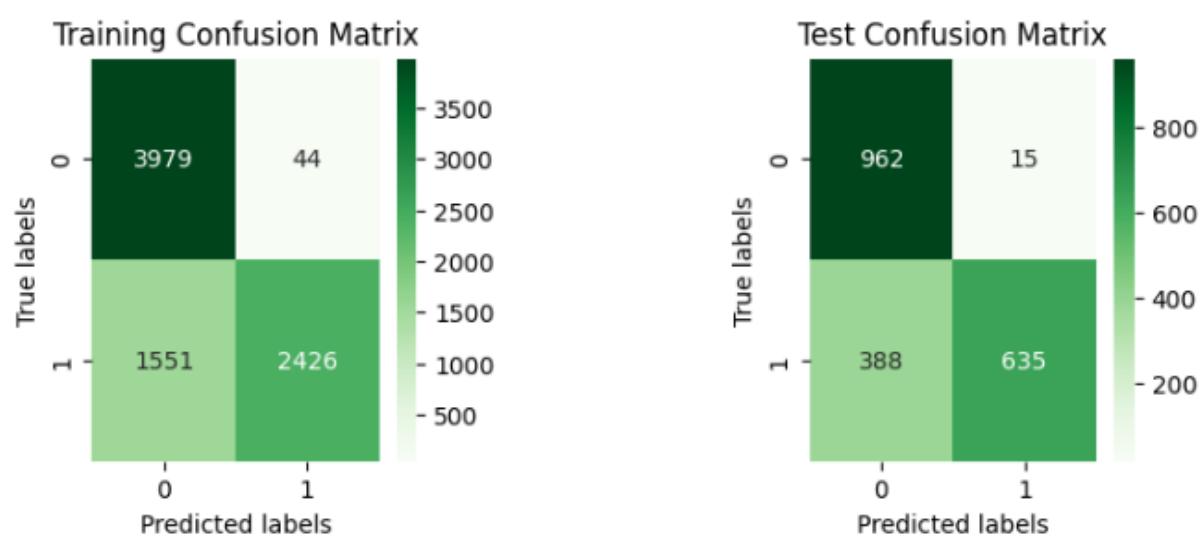


Figure 4.9: Confusion Matrix for Decision Tree classifier

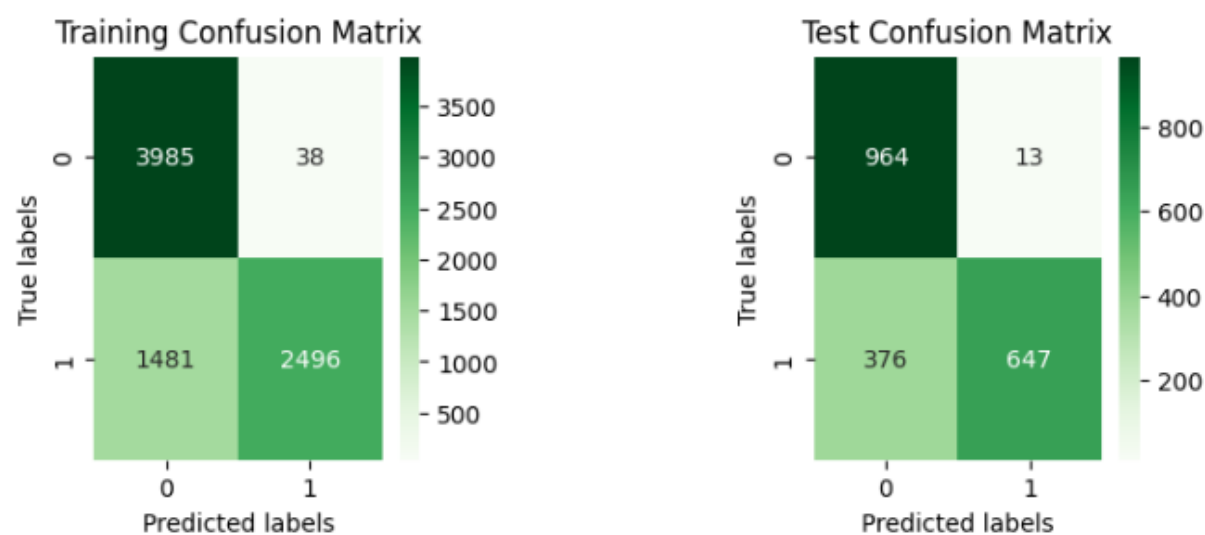


Figure 4.10: Confusion Matrix for Random Forest classifier

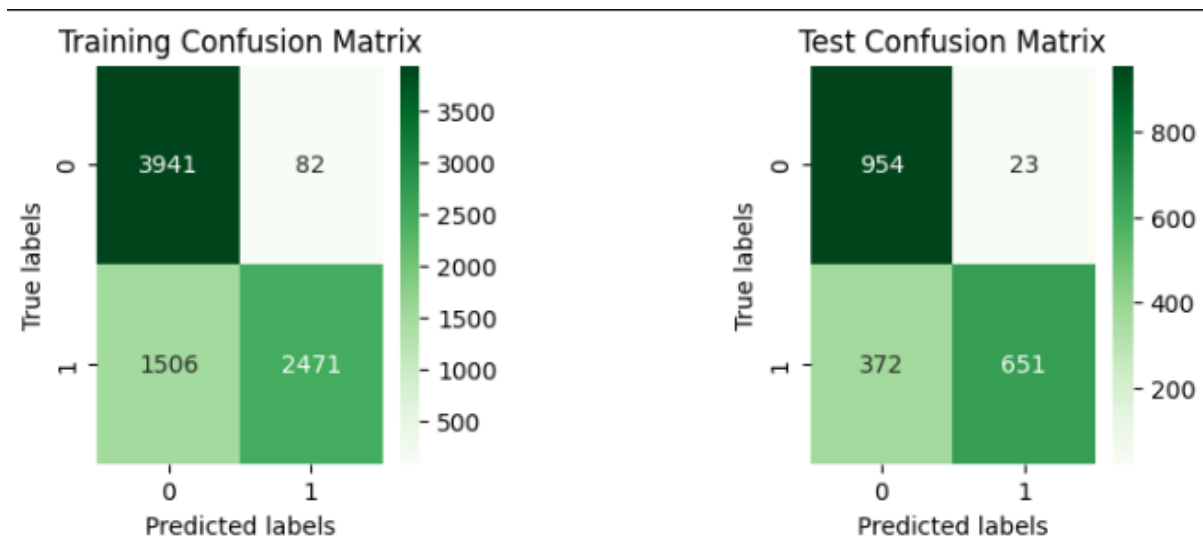


Figure 4.11: Confusion Matrix for SVM classifier

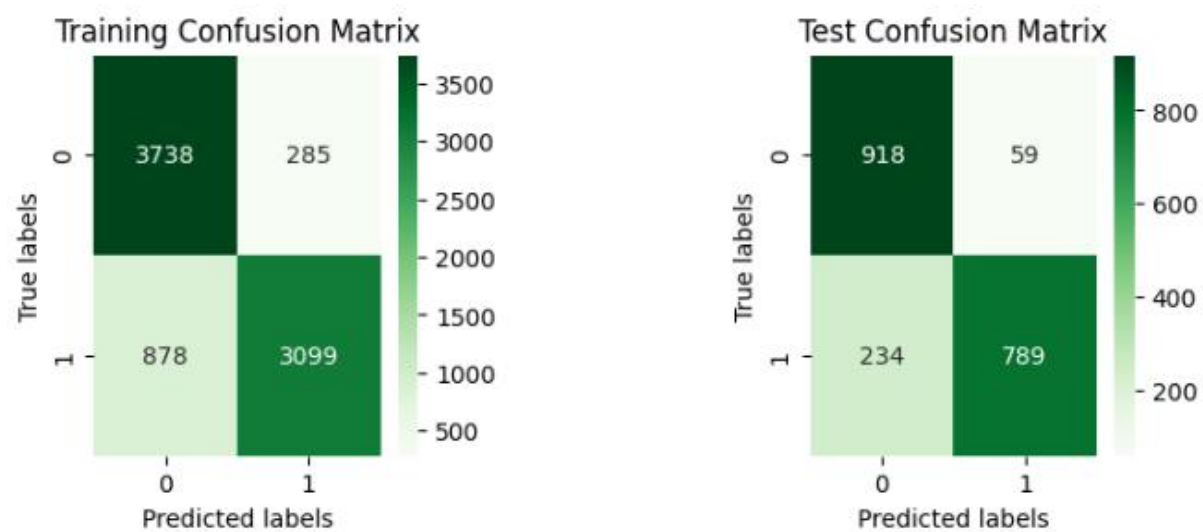


Figure 4.12: Confusion Matrix for XGBoost classifier

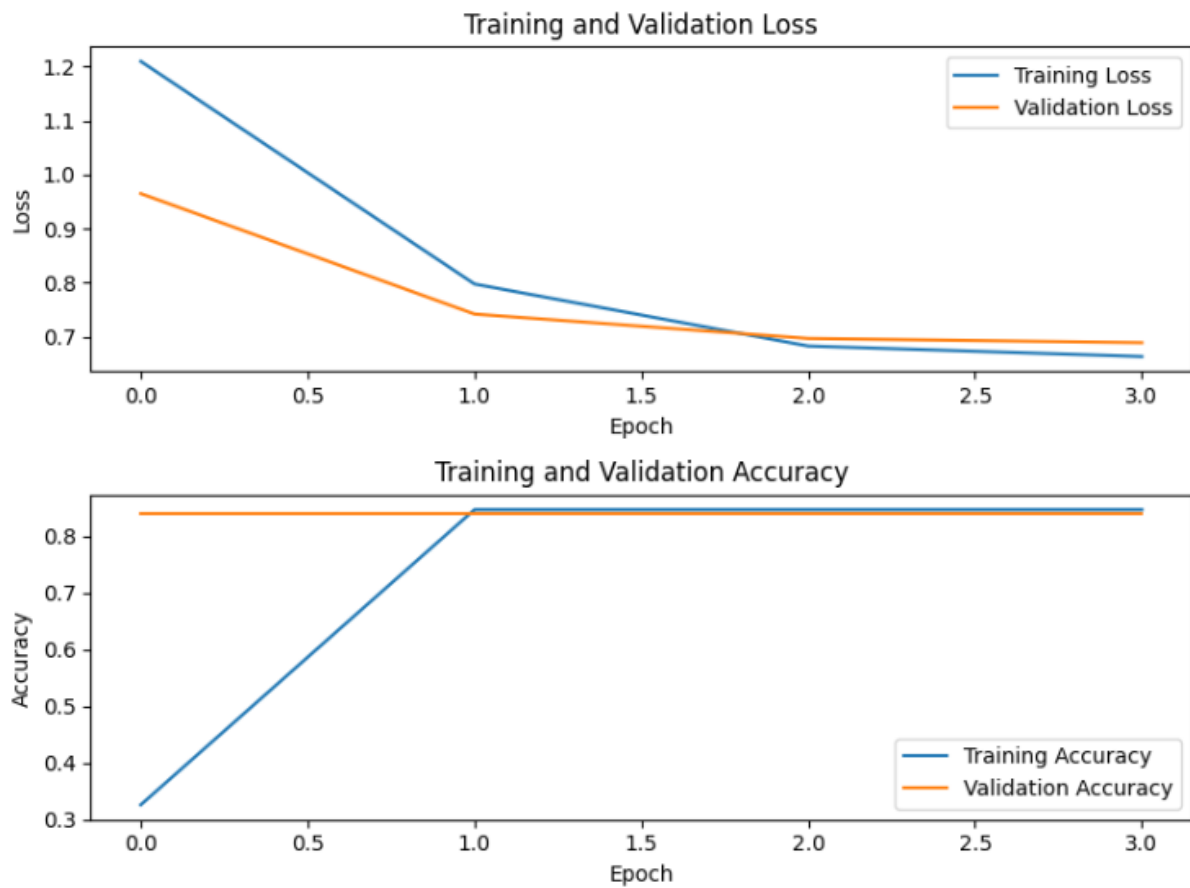


Figure 4.13: Auto-Encoder Neural Network accuracy and error progress

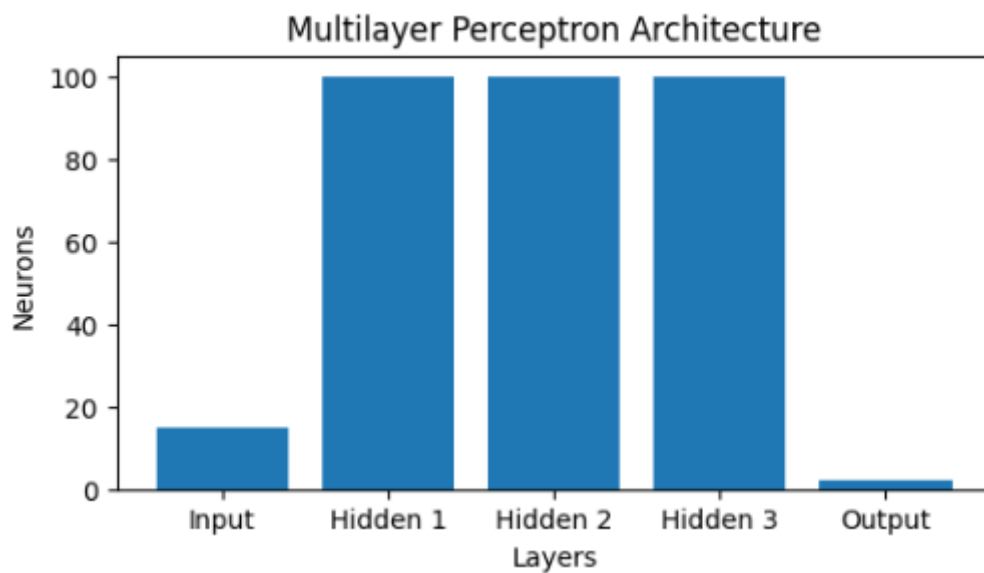


Figure 4.14: Multilayer Perceptron's neuron structure

4.2.4 Data pre-processing:

The datasets were first cleaned to remove empty entries and fill some entries by applying data pre-processing techniques and transform the data to use in the models. Figure 4.15 shows the summary of the dataset while figure 4.16 shows the number of missing values in the dataset which all appear to be zero.

```
[9]: tuna.describe()
```

	Have_IP	Have_At	URL_Length	URL_Depth	Redirection	https_Domain	TinyURL	Prefix/Suffix	DNS_Record	Domain_Age	Domain_End	
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000
mean	0.005500	0.022600	0.773400	3.072000	0.013500	0.000200	0.090300	0.093200	0.100800	0.413700	0.8099	
std	0.073961	0.148632	0.418653	2.128631	0.115408	0.014141	0.286625	0.290727	0.301079	0.492521	0.3924	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000	
25%	0.000000	0.000000	1.000000	2.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.0000	
50%	0.000000	0.000000	1.000000	3.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.0000	
75%	0.000000	0.000000	1.000000	4.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	1.0000	
max	1.000000	1.000000	1.000000	20.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.0000	

Figure 4.15: Summary of the dataset

```
[10]: #Dropping the Domain column
dfsa = tuna.drop(['Domain'], axis = 1).copy()

[11]: #checking the data for null or missing values
dfsa.isnull().sum()

[11]: Have_IP      0
Have_At      0
URL_Length   0
URL_Depth    0
Redirection   0
https_Domain 0
TinyURL      0
Prefix/Suffix 0
DNS_Record   0
Domain_Age    0
Domain_End    0
iFrame       0
Mouse_Over    0
Right_Click   0
Web_Forwards  0
Label        0
dtype: int64
```

Figure 4.16: Number of missing values in the dataset

4.2.5 Phishing detection model

The based methodology stated that the proposed system utilizes machine learning models and deep neural networks. These models consist of Decision Tree, Support Vector Machine, XGBoost, Multilayer Perceptions, Auto Encoder Neural Network, and Random Forest.

The models determine whether a website URL is phishing or legitimate. The models help give a 2-class prediction (legitimate (0) and phishing (1)). In the model development process, over six (6) machine learning models and deep neural network algorithms all together were used to detect phishing URLs using Jupyter notebook IDE with packages such as pandas, Beautiful Soup, who-is, urllib, etc.

Here are the models, their accuracy was tested using sklearn matrices with an accuracy score and their matrices are shown in figure 4.13. The XGBoost model had the highest performance score of 85.4%, the Multilayer Perceptions model had an accuracy of 83.6%, the Decision Tree model had an accuracy of 79.8%, the Random Forest model had an accuracy of 80.6%, the Support Vector Machine model had an accuracy of 80.2%, and the Auto Encoder Neural Network model had an accuracy of 84%

Figure 4.17, 4.18, 4.19 and 4.20 show the code of above-mentioned algorithms.

Autoencoder Neural Network

```
[18]: #importing required packages
import keras
from keras.layers import Input, Dense
from keras import regularizers
import tensorflow as tf
from keras.models import Model
from sklearn import metrics
from tensorflow.keras.callbacks import EarlyStopping

[19]: #building autoencoder model

input_dim = X_train.shape[1]
encoding_dim = input_dim

input_layer = Input(shape=(input_dim,))
encoder = Dense(encoding_dim, activation='relu',
                activity_regularizer=regularizers.l1(10e-4))(input_layer)
encoder = Dense(int(encoding_dim - 2), activation='relu')(encoder)
code = Dense(int(encoding_dim - 4), activation='relu')(encoder)

# Decoder
decoder = Dense(int(encoding_dim - 2), activation='relu')(code)
decoder = Dense(encoding_dim, activation='relu')(decoder)
output_layer = Dense(input_dim, activation='sigmoid')(decoder)
autoencoder = Model(inputs=input_layer, outputs=output_layer)
autoencoder.summary()
```

Model: "functional_1"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 15)	0
dense (Dense)	(None, 15)	240
dense_1 (Dense)	(None, 13)	208
dense_2 (Dense)	(None, 11)	154
dense_3 (Dense)	(None, 13)	156
dense_4 (Dense)	(None, 15)	210
dense_5 (Dense)	(None, 15)	240

Total params: 1,208 (4.72 KB)
Trainable params: 1,208 (4.72 KB)
Non-trainable params: 0 (0.00 B)


```
[20]: #compiling the model
autoencoder.compile(optimizer='adam',
                    loss='mean_squared_error',
                    metrics=['accuracy'])

# Set up EarlyStopping callback
early_stopping = EarlyStopping(
    monitor='val_accuracy',      # Monitor validation accuracy
    patience=3,                 # Number of epochs with no improvement after which training will be stopped
    mode='max',                 # We want to maximize the validation accuracy
    restore_best_weights=True   # Restore model weights from the epoch with the best value of the monitored quantity
)

#Training the model
history = autoencoder.fit(X_train, X_train, epochs=10, batch_size=64, shuffle=True, validation_split=0.2, callbacks=[early_stopping]) # Include the Ear
```

```
Epoch 1/10
100/100 — 3s 4ms/step - accuracy: 0.1268 - loss: 1.3791 - val_accuracy: 0.8406 - val_loss: 0.9648
Epoch 2/10
100/100 — 0s 2ms/step - accuracy: 0.8483 - loss: 0.8731 - val_accuracy: 0.8406 - val_loss: 0.7423
Epoch 3/10
100/100 — 0s 2ms/step - accuracy: 0.8497 - loss: 0.6846 - val_accuracy: 0.8406 - val_loss: 0.6972
Epoch 4/10
100/100 — 0s 2ms/step - accuracy: 0.8450 - loss: 0.6531 - val_accuracy: 0.8406 - val_loss: 0.6895
```

Figure 4.17: Auto-Encoder Neural Network

XGBoost Classifier

```
[24]: #XGBoost Classification model
from xgboost import XGBClassifier

# instantiate the model
xgb = XGBClassifier(learning_rate=0.4,max_depth=7)
#fit the model
xgb.fit(X_train, y_train)
```

```
[24]: XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.4, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=7, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, random_state=None, ...)
```

```
[25]: #predicting the target value from the model for the samples
y_test_xgb = xgb.predict(X_test)
y_train_xgb = xgb.predict(X_train)
```

```
[26]: #computing the accuracy of the model performance
from sklearn.metrics import accuracy_score
acc_train_xgb = accuracy_score(y_train,y_train_xgb)
acc_test_xgb = accuracy_score(y_test,y_test_xgb)

print("XGBoost: Accuracy on training Data: {:.3f}".format(acc_train_xgb))
print("XGBoost : Accuracy on test Data: {:.3f}".format(acc_test_xgb))

XGBoost: Accuracy on training Data: 0.855
XGBoost : Accuracy on test Data: 0.854
```

Figure 4.18: XGBoost Classifier

Decision Tree Algorithm

```
[29]: # Decision Tree model
from sklearn.tree import DecisionTreeClassifier

# instantiate the model
tree = DecisionTreeClassifier(max_depth = 5)
# fit the model
tree.fit(X_train, y_train)

[29]: ▾ DecisionTreeClassifier
DecisionTreeClassifier(max_depth=5)

[30]: #predicting the target value from the model for the samples
y_test_tree = tree.predict(X_test)
y_train_tree = tree.predict(X_train)

[31]: #computing the accuracy of the model performance
acc_train_tree = accuracy_score(y_train,y_train_tree)
acc_test_tree = accuracy_score(y_test,y_test_tree)

print("Decision Tree: Accuracy on training Data: {:.3f}".format(acc_train_tree))
print("Decision Tree: Accuracy on test Data: {:.3f}".format(acc_test_tree))

Decision Tree: Accuracy on training Data: 0.801
Decision Tree: Accuracy on test Data: 0.798
```

Figure 4.19: Decision Tree Classifier

Random Forest Classifier

```
[35]: # Random Forest model
from sklearn.ensemble import RandomForestClassifier

# instantiate the model
forest = RandomForestClassifier(n_estimators=100,max_depth=5)

# fit the model
forest.fit(X_train, y_train)

[35]: ▾ RandomForestClassifier
RandomForestClassifier(max_depth=5)

[36]: #predicting the target value from the model for the samples
y_test_forest = forest.predict(X_test)
y_train_forest = forest.predict(X_train)

[37]: #computing the accuracy of the model performance
acc_train_forest = accuracy_score(y_train,y_train_forest)
acc_test_forest = accuracy_score(y_test,y_test_forest)

print("Random forest: Accuracy on training Data: {:.3f}".format(acc_train_forest))
print("Random forest: Accuracy on test Data: {:.3f}".format(acc_test_forest))

Random forest: Accuracy on training Data: 0.810
Random forest: Accuracy on test Data: 0.805
```

Figure 4.20: Random Forest Classifier

```
Support Vector machine

[41]: #Support vector machine model
from sklearn.svm import SVC

# instantiate the model
svm = SVC(kernel='linear', C=1.0, random_state=12)
#fit the model
svm.fit(X_train, y_train)

[41]: +          SVC
      SVC(kernel='linear', random_state=12)

[42]: #predicting the target value from the model for the samples
y_test_svm = svm.predict(X_test)
y_train_svm = svm.predict(X_train)

[43]: #computing the accuracy of the model performance
acc_train_svm = accuracy_score(y_train,y_train_svm)
acc_test_svm = accuracy_score(y_test,y_test_svm)

print("SVM: Accuracy on training Data: {:.3f}".format(acc_train_svm))
print("SVM : Accuracy on test Data: {:.3f}".format(acc_test_svm))

SVM: Accuracy on training Data: 0.801
SVM : Accuracy on test Data: 0.802
```

Figure 4.21: SVM Classifier

```
Multilayer Perceptron

[46]: # Multilayer Perceptrons model
from sklearn.neural_network import MLPClassifier

# instantiate the model
mlp = MLPClassifier(alpha=0.001, hidden_layer_sizes=([100,100,100]))

# fit the model
mlp.fit(X_train, y_train)

[46]: +          MLPClassifier
      MLPClassifier(alpha=0.001, hidden_layer_sizes=[100, 100, 100])

[47]: #predicting the target value from the model for the samples
y_test_mlp = mlp.predict(X_test)
y_train_mlp = mlp.predict(X_train)

[48]: #importing packages
from sklearn.metrics import accuracy_score
#computing the accuracy of the model performance
acc_train_mlp = accuracy_score(y_train,y_train_mlp)
acc_test_mlp = accuracy_score(y_test,y_test_mlp)

print("Multilayer Perceptrons: Accuracy on training Data: {:.3f}".format(acc_train_mlp))
print("Multilayer Perceptrons: Accuracy on test Data: {:.3f}".format(acc_test_mlp))

Multilayer Perceptrons: Accuracy on training Data: 0.847
Multilayer Perceptrons: Accuracy on test Data: 0.836
```

Figure 4.22: Multilayer Perceptron

4.2.6 Performance Comparison:

Performance comparison of the models was done with the accuracy score, XGBoost model is the best performing model with an accuracy score of 85.4%.

```
[53]: #Sorting the dataframe on accuracy
results.sort_values(by=['Test Accuracy', 'Train Accuracy'], ascending=False)
```

[53]:	ML Model	Train Accuracy	Test Accuracy
1	XGBoost	0.855	0.854
0	AutoEncoder	0.845	0.840
5	Multilayer Perceptrons	0.847	0.836
3	Random Forest	0.810	0.806
4	SVM	0.802	0.802
2	Decision Tree	0.801	0.798

Figure 4.23: Performance comparison of the models

CHAPTER FIVE

SUMMARY, CONCLUSION, RECOMMENDATIONS AND FUTURE WORK

5.1 Summary

Phishing attacks are a rapidly expanding threat in the cyber world, costing internet users billions of dollars each year. It involves the use of a variety of social engineering tactics to obtain sensitive information from users. Hence, Phishing techniques can be detected using a variety of types of communication, including email, instant chats, pop-up messages, and web pages. This project was able to categorize and recognize how phishers carry out phishing attacks and the different ways in which researchers have helped to solve phishing detection. Hence, the proposed system of this project worked with different feature selection and machine learning and deep neural networks such as Decision Tree, Support Vector Machine, XGBooster, Multilayer Perceptions, Auto Encoder Neural Network, and Random Forest to identify patterns in which URL links can be detected easily. The Model with the highest accuracy based on the feature extraction algorithm used to identify phishing URL from legitimate URL links was integrated to a web application where users can input website URL links to detect if it is legitimate or phishing.

5.2 Contribution to Knowledge

This project provides a new and faster way to help users detect if a URL link is phishing or legitimate and also provide them access to educational resources about phishing attacks.

5.3 Conclusion

The system developed detects if a URL link is phishing or legitimate by using machine learning models and deep neural network algorithms. The feature extraction and the models used on the dataset helped to uniquely identify phishing URLs and also the performance accuracy of the models used. It is also surprisingly accurate at detecting the genuineness of a URL link. 5.4 Recommendation Through this project, one can know a lot about phishing attacks and how to prevent them. This project can be taken further by creating a browser extension that can be installed on any web browser to detect phishing URL Links.

5.4 Future works

Detecting phishing websites using machine learning is an exciting and challenging project. Here are some future work ideas to extend your project:

1. Improving Model Performance:

- **Hyperparameter Tuning:** Use techniques like grid search or Bayesian optimization to fine-tune model parameters for better performance.

2. Handling Imbalanced Data:

- **Sampling Techniques:** Explore techniques like SMOTE (Synthetic Minority Over-sampling Technique) to handle class imbalance and improve the model's ability to detect phishing sites accurately.

3. Deep Learning Approaches:

- **Recurrent Neural Networks (RNNs):** Investigate the use of RNNs or LSTM (Long Short-Term Memory) networks to capture sequential patterns in URLs or webpage content that may indicate phishing.
- **Convolutional Neural Networks (CNNs):** Apply CNNs to analyze image-based features of webpages (e.g., logos, page structure) to detect phishing attempts.

4. Real-time Detection and Scalability:

- **Streaming Data:** Design and implement a pipeline that can handle real-time data streams, enabling the detection of phishing websites as they are created or evolve.
- **Scalability:** Optimize your model to handle large-scale datasets efficiently, potentially using distributed computing frameworks like Apache Spark.

5. Adversarial Attacks and Robustness:

- **Adversarial Training:** Explore techniques to make your model more robust against adversarial attacks designed to evade phishing detection systems.

6. User Interface and Integration:

- **Deployment:** Develop a user-friendly interface or integrate your model into existing cybersecurity tools to assist users in identifying potential phishing threats.
- **Feedback Mechanism:** Implement mechanisms for users to provide feedback on identified phishing attempts, which can be used to further improve the model over time.

7. Cross-domain Generalization:

- **Transfer Learning:** Investigate techniques to transfer knowledge learned from one domain (e.g., phishing websites in one industry) to another, improving detection capabilities across different domains.

8. **Ethical and Legal Considerations:**

- **Bias and Fairness:** Assess and mitigate potential biases in your model, ensuring fairness in how it classifies websites from different demographics or geographic locations.
- **Privacy:** Ensure that your model respects user privacy and complies with relevant data protection regulations when handling sensitive information.

REFERENCE

Abdelhamid, N., Thabtah F., & Abdel-Jaber, H. Phishing detection: A recent intelligent machine learning comparison based on models' content and features," 2017 IEEE International Conference on Intelligence and Security Informatics (ISI), Beijing, 2017, pp. 72-77, DOI: 10.1109/ISI.2017.8004877.

Anjum N. S., Antesar M. S., & Hossain M.A. (2016). A Literature Review on Phishing Crime, Prevention Review and Investigation of Gaps. Proceedings of the 10th International Conference on Software, Knowledge, Information Management & Applications (SKIMA), Chengdu, 10.1109/SKIMA.2016.7916190.

China, 2016, pp. 9-15, DOI: Almomani, A., Gupta, B. B., Atawneh, S., Meulenberg, A., & Almomani, E. (2013). A survey of phishing email filtering techniques, Proceedings of IEEE Communications Surveys and Tutorials, vol. 15, no. 4, pp. 2070–2090.

Ashritha, J. R., Chaithra, K., Mangala, K., & Deekshitha, S. (2019). A Review Paper on Detection of Phishing Websites using Machine Learning. Proceedings of International Journal of Engineering Research & Technology (IJERT), 7, 2. Retrieved from www.ijert.org.

Anti-Phishing Working Group (APWG) Phishing activity trends report the first quarter. (2014) Retrieved from http://docs.apwg.org/reports/apwg_trends_report_q1_2014.pdf

APWG report. (2014). Retrieved from http://apwg.org/download/document/245/APWG_Global_Phishing_Report_2H_2014.pdf 110

Ayush, P. (2019). Workflow of a Machine Learning project. Retrieved from <https://towardsdatascience.com/workflow-of-a-machine-learning-project> ec1dba419b94

Camp W. (2001). Formulating and evaluating theoretical frameworks for career and technical education research. Journal of Vocational Education Research, 26(1), 4 25.

DeepAI (n.d.). About clinical psychology. Retrieved from <https://deepai.org/machine-learning-glossary-and-terms/feature-extraction>

Engne K., & Christopher K. (2005). Protecting Users Against Phishing Attacks. Proceedings of the Oxford University Press on behalf of The British Computer Society, Oxford University, 0, 2005, https://sites.cs.ucsb.edu/~chris/research/doc/cj06_phish.pdf

Retrieved from: Gandhi, V. (2017). A Theoretical Study on Different ways to identify the Phishing URL and Its Prevention Approaches: presented at International Conference on Cyber Criminology, Digital Forensics and Information Security at DRBCCC Hindu College, Chennai.

Retrieved from

https://www.researchgate.net/publication/319006943_A_Theoretical_Study_on_Different_ways_to_Identify_the_Phishing_URL_and_Its_Prevention_Approaches

Gupta, B. B., Tewari, A., Jain, A. K., & Agrawal, D. P. (2016). Fighting against phishing attacks: state of the art and future challenges, Neural Computing and Applications. Internet world stats usage and population statistics. (2014). Retrieved from <http://www.internetworldstats.com/stats.htm>.

111 Imperva. (2021). Phishing attacks. Retrieved from

<https://www.imperva.com/learn/application-security/phishing-attack-scam/> Kiruthiga, R., Akila, D. (2019, September). Phishing Websites Detection Using Machine Learning. Retrieved from <https://www.researchgate.net/publication/337049054>

Phishing Websites Detection Using Machine Learning. KnowBe4 (2021). Phishing Techniques. <https://www.phishing.org/phishing-techniques>

Retrieved from Kondeti, P. S., Konka, R. C., & Kavishree, S. (2021). Phishing Websites Detection using Machine Learning Techniques. International Research Journal of Engineering and Technology, 08(4), Page 1471-1473. Retrieved from <https://www.irjet.net/archives/V8/i4/IRJET-V8I4274.pdf>

Noel, B. (2016). Support Vector Machines: A Simple Explanation. Retrieved from <https://www.kdnuggets.com/2016/07/support-vector-machines-simple-explanation.html>

Osanloo, A., & Grant, C. (2016). Understanding, selecting, and integrating a theoretical framework in dissertation research: creating the blueprint for your “house”. Administrative issues journal: connecting education, practice and research 4(2), 7.

Peng, T., Harris, I., & Sawa, I. (2018). Detecting Phishing Attacks Using Natural Language Processing and Machine Learning. Proc. - 12th IEEE Int. Conf. Semant. Comput. ICSC 2018, vol. 2018–Janua, pp. 300–301.

112 Pamela (2021). Phishing attacks. Retrieved from

<https://www.khanacademy.org/computing/computers-and-internet/xcae6f4a7ff015e7d:online-data-security/xcae6f4a7ff015e7d:cyber-attacks/a/phishing-attacks>

Rami, M. M., Fadi, T., & Lee, M. (2015). Phishing Websites Features. Retrieved from <https://eprints.hud.ac.uk/id/eprint/24330/6/MohammadPhishing14July2015.pdf>

Rishikesh, M., & Irfan, S. (2018a). Phishing Website Detection using Machine Learning Algorithms. International Journal of Computer Applications, 23, 45.

doi:10.5120/ijca2018918026 Rishikesh, M., & Irfan, S. (2018b). Phishing Website Detection using Machine Learning Algorithms. International Journal of Computer Applications, 23, 45-46. doi:10.5120/ijca2018918026 Rahul, S. (2017). How the decision tree algorithm works. Retrieved from <https://dataaspirant.com/how-decision-tree-algorithm-works/> Rishikesh, M., & Irfan, S. (2018c). Phishing Website Detection using Machine Learning Algorithms. International Journal of Computer Applications, 23, 46-47. doi:10.5120/ijca2018918026

Saimadhu, P. (2017). How the random forest algorithm works in machine learning. Retrieved from [learning/ https://dataaspirant.com/random-forest-algorithm-machine](https://dataaspirant.com/random-forest-algorithm-machine)

Shaikh, A.N., Shabut, A.M., Hossain, M.A. (2016, December 15-17). A literature review on phishing crime, prevention review, and investigation of gaps. Paper presented at the Tenth International Conference on Software, Knowledge, 113 Information Management & Applications (SKIMA), Chengdu, China. Retrieved from <https://ieeexplore.ieee.org/document/7916190>

Shreya, G. (2020). Phishing website detection by machine learning techniques. Retrieved from <https://github.com/shreyagopal/Phishing-Website-Detection-byMachine-Learning-Techniques>

Sönmez, Y., Tuncer, T., Gökal, H., & Avci, E. (2018). Phishing web sites features classification based on extreme learning machine. 6th Int. Symp. Digit. Forensics Secure. ISDFS 2018 - Proceeding, vol. 2018–Janua, pp. 1–5. RSA Anti-Fraud Command Center (n.d.) Retrieved from <https://www.emc.com/collateral/fraud-report/rsa-online-fraud-report-012014.pdf>.

Shad, J., & Sharma, S. (2018). A Novel Machine Learning Approach to Detect Phishing Websites Jaypee Institute of Information Technology, pp. 425–430. The RSA Current State of Cybercrime. (n.d.). Retrieved from <https://www.rsa.com/en-us/perspectives/industry/online-fraud>

Tewari, A., Jain, A. K, & Gupta, B. B. (2016). A recent survey of various defense mechanisms against phishing attacks. Journal of Information Privacy and Security, vol. 12, no. 1, pp. 3–13. Joachim, S. (n.d). Missing Value Imputation (Statistics) – How to Impute Incomplete Data. Retrieved from [statistics https://statisticsglobe.com/missing-data-imputation](https://statisticsglobe.com/missing-data-imputation)

Kartik, M. (2021). Everything You Need to Know About Feature Selection in Machine Learning. Retrieved from <https://www.simplilearn.com/tutorials/machine-learning-tutorial/feature-selection-in-machine-learning>

114 Kiruthiga R., & Akila D. (2019). Phishing Website Detection using Machine Learning. International Journal of Recent Technology and Engineering, 8, 2S11. DOI: 10.35940/ijrte.B1018.0982S1119 Kyaw, S. (2020). A Guide to KNN Imputation. Retrieved from <https://medium.com/@kyawsawhtoon/a-guide-to-knn-imputation-95e2dc496e>

Kruegel, C., Kirda, E., Mutz, D., Robertson, W., & Vigna, G. (2005). Automating mimicry attacks using static binary analysis. Proceedings of the USENIX Security Symposium, pp. 161–176. Will, B. (2019). 6 Different Ways to Compensate for Missing Values In a Dataset (Data Imputation with examples). Retrieved from <https://towardsdatascience.com/6-different-ways-to-compensate-for-missing-values-data-imputation-with-examples-6022d9ca0779>

Workflow of a Machine Learning project (2019). Retrieved from <https://towardsdatascience.com/workflow-of-a-machine-learning-project> ec1dba419b94