Faculty of Electronic and Computer Technology and Engineering

DESIGN AND DEVELOPMENT OF PHISHING SITES DETECTOR USING MACHINE LEARNING

RUGENRAJ A/L SELVARAJU

Bachelor of Computer Engineering Technology (Computer Systems) with Honours

2024

DESIGN AND DEVELOPMENT OF PHISHING SITES DETECTOR USING MACHINE LEARNING

RUGENRAJ A/L SELVARAJU

A project report submitted
in partial fulfillment of the requirements for the degree of
Bachelor of Computer Engineering Technology (Computer Systems) with Honours

Faculty of Electronic and Computer Technology and Engineering

UNIVERSITI TEKNIKAL MALAYSIA MELAKA

2024

UNIVERSITI TEKNIKAL MALAYSIA MELAKA
FAKULTI TEKNOLOGI DAN KEJURUTERAAN ELEKTRONIK DAN KOMPUTER

BORANG PENGESAHAN STATUS LAPORAN
PROJEK SARJANA MUDA II

Tajuk Projek	:	Design and Development of Phishing Sites Detector Using Machine Learning
Sesi Pengajian	:	2023/2024

Saya RUGENRAJ A/L SELVARAJU mengaku membenarkan laporan Projek Sarjana Muda ini disimpan di Perpustakaan dengan syarat-syarat kegunaan seperti berikut:

1. Laporan adalah hakmilik Universiti Teknikal Malaysia Melaka.
2. Perpustakaan dibenarkan membuat salinan untuk tujuan pengajian sahaja.
3. Perpustakaan dibenarkan membuat salinan laporan ini sebagai bahan pertukaran antara institusi pengajian tinggi.
4. Sila tandakan (✓):

☐	**SULIT***	(Mengandungi maklumat yang berdarjah keselamatan atau kepentingan Malaysia seperti yang termaktub di dalam AKTA RAHSIA RASMI 1972)
☐	**TERHAD***	(Mengandungi maklumat terhad yang telah ditentukan oleh organisasi/badan di mana penyelidikan dijalankan.
✓	**TIDAK TERHAD**	

Disahkan oleh:

[illegible] [illegible]

(TANDATANGAN PENULIS) (COP DAN TANDATANGAN PENYELIA)

NOOR MOHD ARIFF BIN BRAHIM
Jurutera Pengajar
Fakulti Teknologi Dan Kejuruteraan Elektronik Dan Komputer (FTKEK)
Universiti Teknikal Malaysia Melaka (UTeM)

Alamat Tetap: 11, Kampung Haji Ismail, 35600 Sungkai, Perak.

Tarikh : 14 APRIL 2024 Tarikh : 15 APRIL 2024

*CATATAN: Jika laporan ini SULIT atau TERHAD, sila lampirkan surat daripada pihak berkuasa/organisasi berkenaan dengan menyatakan sekali tempoh laporan ini perlu dikelaskan sebagai SULIT atau TERHAD.

DECLARATION

I declare that this project report entitled “Design and Development of Phishing Sites Detector Using Machine Learning” is the result of my own research except as cited in the references. The project report has not been accepted for any degree and is not concurrently submitted in candidature of any other degree.

Signature	:	[illegible]
Student Name	:	RUGENRAJ A/L SELVARAJU
Date	:	14/02/2024

APPROVAL

I hereby declare that I have checked this project report and in my opinion, this project report is adequate in terms of scope and quality for the award of the degree of Bachelor of Computer Engineering Technology (Computer Systems) with Honours.

Signature	:	[illegible]
Supervisor Name	:	NOOR MOHD ARIFF BIN BRAHIN
Date	:	15/2/2024

DEDICATION

To my beloved mother, Mrs.Sumathi, and father, Mr.Selvaraju,
and
To dearest grandmother, Mrs.Vembu and
My my late grandfather, Mr.Thangaraju

ABSTRACT

This research project focuses on developing a phishing detection model to address the increasing threat of phishing attacks. The objective is to design and apply classification techniques to analyze phishing websites and improve accuracy. The study aims to provide users with effective tools to identify and protect themselves from phishing attempts, enhancing online security. The research utilizes a phishing dataset and applies techniques for model training and testing. Classification techniques are used to categorize websites as benign or phishing. The accuracy of these techniques is evaluated using metrics like confusion matrix, classification report, and accuracy score. The results demonstrate the effectiveness of the techniques in accurately detecting and classifying phishing websites. The developed model contributes to ongoing efforts in mitigating phishing attacks and protecting sensitive information by using a dual-model approach of each model having an accuracy of 74% for Logistic Regressiona and 64% for Gradient Boost Classifier respectively. In conclusion, this research shows that the classification techniques, when applied to the phishing dataset, yield promising results in identifying and classifying phishing websites. Overall, this project provides valuable insights into developing effective tools for combating phishing attacks and promoting online security.

ABSTRAK

Projek penyelidikan ini memberi fokus kepada pembangunan model pengesanan *phishing* untuk mengatasi ancaman peningkatan serangan *phishing*. Objektif adalah untuk merancang dan menggunakan teknik klasifikasi untuk menganalisis laman web *phishing* dan meningkatkan ketepatan. Kajian ini bertujuan untuk menyediakan pengguna dengan alat yang efektif untuk mengenal pasti dan melindungi diri daripada percubaan *phishing*, meningkatkan keselamatan dalam talian. Penyelidikan ini menggunakan dataset *phishing* dan menggunakan teknik untuk latihan dan pengujian model. Teknik-teknik klasifikasi digunakan untuk mengkategorikan laman web sebagai *phishing* dan selamat. Ketepatan teknik-teknik ini dinilai menggunakan metrik seperti matriks *confusion*, laporan klasifikasi, dan skor ketepatan. Keputusan menunjukkan keberkesanan teknik-teknik ini dalam mengesan dan mengkategorikan laman web phishing dengan tepat. Model yang dibangunkan menyumbang kepada usaha berterusan dalam meredakan serangan *phishing* dan melindungi maklumat yang sensitif menggunakan pendekatan model dwi di mana setiap model mempunyai ketepatan sebanyak 74% untuk *Logistic Regression* dan 64% untuk *Gradient Boost Classifier* masing-masing. yang mempunya . Kesimpulannya, penyelidikan ini menunjukkan bahawa teknik-teknik klasifikasi, apabila digunakan pada dataset *phishing*, memberikan hasil yang menjanjikan dalam mengenal pasti dan mengkategorikan laman web *phishing*. Secara keseluruhan, projek ini memberikan pandangan berharga dalam membangunkan alat yang efektif untuk memerangi serangan *phishing* dan mempromosikan keselamatan dalam talian.

ACKNOWLEDGEMENTS

First and foremost, I extend my deepest gratitude to my supervisor, Noor Mohd Ariff Bin Brahin, for his invaluable guidance, profound wisdom, and unwavering patience throughout this project. His insights and mentorship have been fundamental to my success.

I am equally grateful to my parents for their endless support and love. Their belief in my abilities and constant encouragement have been a pillar of strength during my studies.
I also wish to express my appreciation to my colleagues for their willingness to share thoughts and ideas about the project. The collaborative spirit and insightful contributions within our team have been greatly beneficial.

My highest appreciation also goes to my extended family and friends for their love, prayers, and moral support during my studies. Their understanding and encouragement have been a constant source of motivation.

Finally, I would like to acknowledge all the faculty members and classmates for their cooperation and assistance. A heartfelt thanks to everyone who has contributed to this project in any way, big or small, directly or indirectly. Your collective efforts have been instrumental in my academic journey.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Background

Cybersecurity is essential in safeguarding individuals and organizations from online threats, with phishing attacks posing a significant risk worldwide. Machine learning models have emerged as a promising solution for detecting and preventing such attacks. By analyzing data and identifying patterns associated with phishing URLs, these models can differentiate between legitimate and malicious links (Jang-Jaccard & Nepal, 2014). This project focuses on developing a robust phishing URL detection system using state-of-the-art machine learning algorithms. By accurately identifying and blocking malicious URLs in real-time, the system aims to protect users' sensitive information, prevent financial losses, and mitigate reputational damage caused by phishing scams. Integrating machine learning models into existing security systems enhances their effectiveness and provides an additional layer of defense against evolving phishing techniques. Ultimately, this project contributes to a safer online environment by leveraging artificial intelligence and data analysis to foster trust and protect sensitive information in the digital realm.

1.2 Addressing Cybercrimes involving malicious URLs using Artificial Intelligence (AI)

Cybercrimes involving malicious URLs, such as phishing attacks, malware distribution, and other forms of online fraud, pose a significant threat to individuals, organizations, and society as a whole. Due to the constantly developing nature of cyber threats, conventional ways of detecting and managing such risks frequently fall short. To combat cybercrimes employing harmful URLs, Artificial Intelligence (AI) has emerged as a possible approach. AI-powered systems can quickly and accurately identify dangerous URLs by analyzing enormous volumes of data in real-time. Large datasets of well-known harmful URLs and other relevant data can be used to train machine learning algorithms to spot patterns and traits common to bad URLs. Then, these algorithms can be used in real-time to scan and analyze URLs automatically, identifying those that are probably harmful for additional examination or blocking.

In conclusion, AI has demonstrated significant promise in combating harmful URL-related cybercrimes. It is an effective tool in the battle against cybercrime because of its capacity to analyze significant volumes of data in real-time, spot patterns, and adapt to emerging threats. To stay up with the changing threat landscape, organizations must be attentive and regularly upgrade their cybersecurity safeguards.

1.3 Problem Statement

In today's interconnected world, ensuring robust cybersecurity measures is paramount. However, three critical problem statements pose significant challenges that demand immediate attention. These challenges include insufficient user awareness and education, rapidly evolving cyber threats, and inadequate collaboration and information sharing. Addressing these problems is crucial for a secure digital future.

Insufficient user awareness and education remain a primary challenge in cybersecurity. Many individuals are unaware of the tactics employed by cybercriminals, making them easy targets. Comprehensive educational initiatives are necessary to raise awareness about common threats and promote safe online practices.

Rapidly evolving cyber threats present another challenge. Cybercriminals adapt techniques to exploit vulnerabilities, surpassing traditional security measures. Continuous research, proactive defense mechanisms, and adoption of advanced technologies like artificial intelligence are crucial for early detection and response.

Inadequate collaboration and information sharing hinder effective cybersecurity. Establishing frameworks for public-private partnerships and fostering international cooperation can enhance overall cybersecurity posture and enable swift responses to emerging threats.

In conclusion, integrating machine learning into cybersecurity provides a proactive solution to the challenges of user awareness, evolving threats, and collaboration. It enables early detection and mitigation of cyber attacks, enhancing overall defense capabilities and ensuring a more secure digital environment.

1.4 Project Objective

The main aim of this project is to design and develop a website that can take URLs as input and predict whether it's a good or bad URL using AI. Specifically, the objectives are as follows:

a) To design classification techniques for analyzing phishing.

b) To apply different classification techniques to phishing dataset.

c) To evaluate the accuracy of results using different methods.

1.5 Scope of Project

The scope of this project are as follows:

a) Data collection: Collecting a comprehensive dataset of known malicious URLs from public sources or using web scraping techniques. This dataset will be used to train and validate the machine learning model.

b) Model development: Develop a deep learning based model using machine learning to classify URLs as malicious or benign based on extracted features.

c) Model training and evaluation: Training the model on the collected dataset and evaluating it's performance.

d) Web development: Building a user-friendly web application to demonstrate the functionality of the model.The web application will allow users to input an URL and receive an output indicating whether the URL is malicious or benign.

e) Deployment: Deploying the trained model and the web application on a local server

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

This chapter will include a literature study that includes a definition of phishing, machine learning and other topics, as well as a critical analysis of the current issue, a proposed solution, and a conclusion. The goal of the literature review is to explain the subject, including definitions, classifications, and other elements. The purpose of this review of the literature is to give the researcher more knowledge based on the project work. The following section will go into greater detail on phishing, machine learning, classification strategies, parameter measurement and other topics. The proposed solution for the full project's issue will be presented at the conclusion of the literature review. The project outcome and output were submitted after gathering information from articles and the past, followed by a milestone.

2.2 Phishing

This section will discuss phishing, including its definition, several classifications, and different sorts of attacks. For this project, URL phishing will be the primary attack type that we concentrate on.

2.2.1 Definition

There are many definitions of “phishing” that have been put forth and studied by experts, researchers, and cybersecurity organizations. Even though the term “phishing’ lacks a set definition due to its ongoing evolution, it has been interpreted in a variety of ways depending on its usage and context. The de facto definition of phishing attacks in general is the process of deceiving the recipient into taking the attacker's desired action. As per some definitions websites are the only media that can be used to some definitions. Phishing is described as "a fraudulent activity that involves the creation of a replica of an existing web page to trick an user into submitting personal, financial, or password data" in the study (Merwe et al., 2005, p. 1). By providing the user malicious links that direct them to a phoney website, phishing is an attempt to deceive a user into giving personal information, such as bank account and credit card data. Some people claim that emails are the only attack channel. Phishing, for instance, is described as "a fraudulent attempt, typically made through email to steal your personal information" by PishTank (2006). According to Kirda and Kruegel (2005), phishing is "a form of online identity theft that aims to steal sensitive information such as online banking passwords and credit card information from users." According to certain definitions, using combined social and technical abilities is important. In the case of phishing, the APWG describes it as "a criminal mechanism employing both social engineering and technical subterfuge to steal consumers' personal identity data and financial account credentials" (APWG, 2018, p.1).

2.2.2 Impact of Phishing Attacks on Society

Phishing is one of the most organized crimes of the 21st century. Phishing attacks can have a negative effect on society in several ways, including money losses, identity theft, harm to one's reputation, and emotional suffering. Intentionally targeting their employees, phishing efforts were discovered by more than 80% of worldwide firms. Over 255 million phishing attacks were launched in 2022, according to the State of Phishing 2022 report by messaging security provider SlashNext, a startling 61% increase from 2021. The analysis also made clear that some security measures are unable to counteract these dangers because hackers frequently conduct attacks using commercial and personal messaging apps as well as well-known providers like Microsoft, Amazon Web providers, and Google. Based on IBM's 2022 cost of Data Breach Report, phishing was the second most common cause of data breaches at 16% costing an average of $4.9 million. The industry that was found to be the most vulnerable to phishing attacks in Q1 of 2022 was the financial sector at 23.6% followed by the software-as-a- Service (SAAS) sector at 20.5% and nextly the e-commerce site at 14.6% of the overall sectors which were targeted at. However, the cost of a breach caused by a successful phishing attack was the highest in the healthcare sector. In terms of individuals, millennials and Gen-Z internet users were the most likely to fall victim to phishing attacks at 23% compared to 19% of Generation X internet users which is because the number of elders and time spent by them on the internet being significantly lower compared to the younger generation (AAG IT Services, 2023). In these cases, individuals are targeted to get access to their device or steal users personal information especially bank details by mimicking a legitimate websites such as banks or social media.

2.2.3 Types of Phishing Attacks

This section will discuss about the common phishing types that are used by attackers to impersonate individuals or organizations to trick people into providing sensitive information.

2.2.3.1 Spear Phishing

In spear phishing, a specific person within an organization is targeted in an effort to obtain their login information. Before attacking, the attacker usually first learns about the victim, including their name, title, and contact information.

2.2.3.2 Search Engine Phishing

Attackers utilize search engine optimization strategies to build phony websites that show up at the top of search results for well-known phrases, such as banks in an effort to deceive visitors into providing their personal data.

2.2.3.3 Email Phishing

Attackers deceive users into clicking on links or downloading attachments that lead to phishing or malware-containing websites by sending fraudulent emails that look to be from a reliable source, such a bank or an e-commerce site.

2.2.3.4 Smishing and Vishing

In smishing (SMS phishing) and vishing (voice phishing), mobile phones take the place of email. Attackers that engage in the practise of smishing send texts with deceptive material that resembles phishing emails. Vishing refers to phone calls in which the con artist speaks directly to the victim.

2.2.3.5 Whaling

This type of spear phishing targets senior executives or those with access to important data or resources, including CFOs or CEOs.

2.2.3.6 Social Media Phishing

Attackers use phony social media identities to pose as real people or businesses in order to deceive users into disclosing sensitive information or clicking on harmful links.

2.2.3.7 Watering Hole Phishing

In a watering hole phishing attack, a hacker determines a website that a particular user demographic frequently visits. Then, in an effort to break into the network, they use it to infect the users' computers.

2.2.3.8 Malware-Based Phishing

Attackers can steal sensitive data like passwords or credit card details by infecting a user's device with malware.

2.2.4 Phishing Techniques

In this section we will discuss about the phishing techniques which refer to the different methods that the attacker uses to carry out a phishing attack.

2.2.4.1 Spoofing

Spoofing is a technique in which attackers can deceive victims into disclosing sensitive information by pretending to be a reputable source. Examples of spoofing techniques used in phishing attacks are:

- Website Spoofing

  To fool victims into providing sensitive information, attackers establish fake websites that mimic real ones.
- Caller ID Spoofing
- To appear as though they are calling from a trusted soource, attackers employ a phoney caller ID.
- Email Spoofing

- Attackers employ a bogus email address or change an email’s “From” field to make it seem as though it was sent by a reliable source.

2.2.4.2 Link Manipulation

Attackers employ link manipulation to trick users into clicking on links that look legitimate but direct them to phishing sites or malicious software. In phishing attempts, link manipulation tactics like the following are typical examples:

- Attackers employ URL shorteners to conceal a link's actual location. A link to a phishing website, for instance, could appear to be for a reputable website.
- Attackers build URLs that look similar to authentic ones but have mistakes in them. For instance, a URL similar to www.facebok.com might be used to deceive consumers into thinking they are accessing the official www.facebook.com.
- Homograph Attacks is when attackers build URLs that resemble genuine ones by using special characters or internationalized domain names (IDNs). For instance, a URL similar to "www.google.com" (notice the Latin "o" has been replaced with a Cyrillic "o") could be used to deceive users into believing they are accessing the genuine "www.google.com."

2.2.4.3 Malware Injection

Software that is intended to harm or exploit a computer system is known as malware. Attackers can steal private data from a victim's device using malware, including login passwords or financial information. Malware used in phishing attempts includes, for instance:

- Keyloggers are malicious programmes that keep track of a victim's keystrokes and can be used to steal login details and other private data.

- Trojans known as remote access tools (RATs) give an attacker access to a victim's device and sensitive data from a distance.
- Ransomware encrypts a victim's files and demands payment in exchange for the decryption key.

2.2.4.4 Mass Target

A mass target phishing assault is one in which the attacker sends a lot of phishing emails or messages to a big group of people in the hopes that at least some of the recipients will fall for the con. Since the attacker's intention is to target a large number of people, they frequently use generic language and may not tailor their message to each recipient in a mass target phishing attack.

2.3 Machine Learning

In this section we will discuss the definition of machine learning, types of machine learning and tools that we will be using in this project.

2.3.1 Definition

A branch of artificial intelligence known as "machine learning" focuses on creating algorithms and statistical models that let computers learn from data without having to be explicitly programmed (Alpaydin, E., 2010). In this branch of artificial intelligence, data and algorithms are used to replicate human learning, allowing machines to improve over time, become more accurate when classifying objects or making predictions, or uncover data-driven insights. In order to identify patterns and categorise data sets, it first uses a combination of data and algorithms. Then, it evaluates accuracy by using an error function.

Finally, it maximises the fit of the data points into the model ("What Is Machine Learning? | IBM"). The streamlined description of how machine learning operates is shown in Figure 2.1 below.

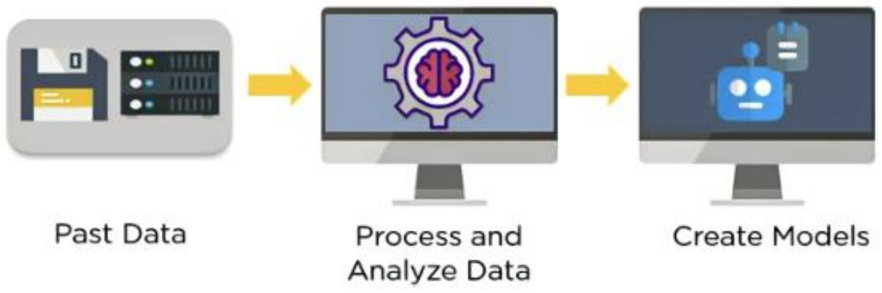


Figure 2.1: Machine Learning Work ("Types of Machine Learning | Simplilearn")

2.3.2 Types of Machine Learning

To teach a machine to learn and make predictions, detect patterns, or classify data, a lot of data must be presented to it. Supervised, unsupervised, and reinforcement learning are the three categories of machine learning.

2.3.2.1 Supervised Learning

Giving machine learning algorithms known historical input and output data allows for supervised learning to take place. The algorithm modifies the model in each step after processing each input-output pair to produce an output as near to the intended result as possible.

"Supervised learning can be used to make predictions, recognize data, or classify it."

For instance, information from tens of thousands of bank transactions could be fed into a model, with each transaction being classified as either real or fraudulent. The model will be able to identify trends that led to "fraudulent" or "not fraudulent" outputs and, with time, it will develop the ability to forecast if a particular transaction is fraudulent. Historical data, computer simulations, or labelling human data can all be used to generate input and output data. In situations where unstructured data, such as pictures, videos, audio, or text, is present, specific characteristics or categorizations may be used as output information. Data can be anticipated, recognised, and categorised using supervised learning. The visual representation of how supervised learning functions is presented in Figure 2.2 below.

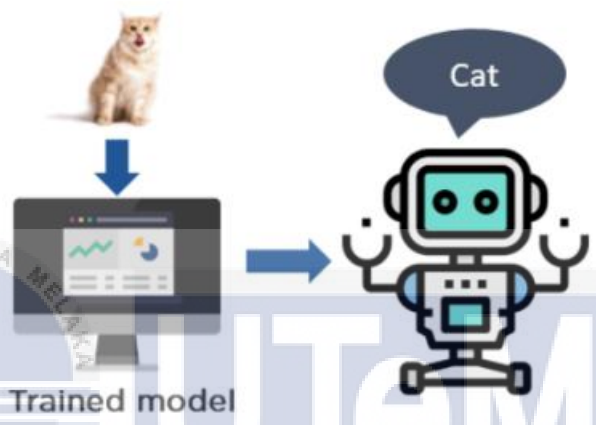


Figure 2.2: Machine Learning Work ("Types of Machine Learning | Simplilearn")

2.3.2.2 Unsupervised Learning

Contrary to supervised learning, unsupervised learning does not use the same labelled training sets and data. The computer instead looks for less obvious patterns in the data. When it comes to recognising patterns and rendering conclusions based on data, this kind of machine learning is highly helpful. An illustration of it is shown in Figure 2.3 below. Unsupervised learning methods including Hidden Markov models, k-means clustering, and Gaussian mixture models are frequently used.

Let's take the supervised learning case where you didn't know which customers had loan defaults. Instead, you would provide the computer borrower data, and it would look for trends among the borrowers before grouping them into various clusters.It is usual practise to use this kind of machine learning while creating prediction models.

Two further typical applications are clustering, which creates a model that groups objects together based on predetermined characteristics, and association, which identifies the rules that connect the clusters.

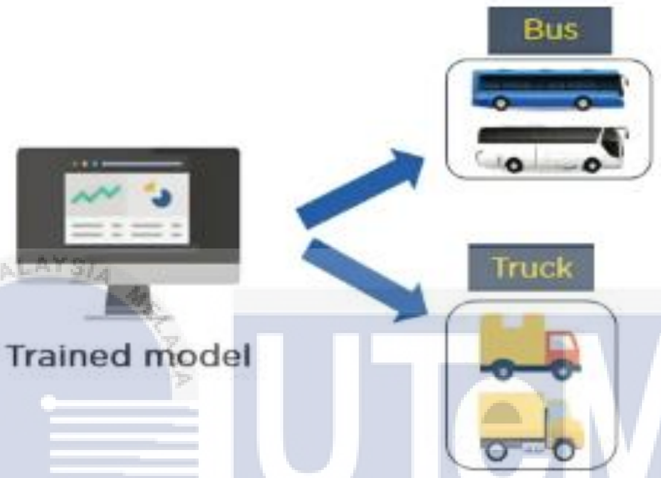


Figure 2.3: Machine Learning Work ("Types of Machine Learning | Simplilearn")

2.3.2.3 Reinforcement Learning

Reinforcement learning is the machine learning technique that most closely matches human learning. The algorithm or agent being used learns by interacting with its surroundings and getting rewards, whether positive or negative. Common algorithms include deep adversarial networks, Q-learning, and temporal differences.

As same as the bank loan client example, use a reinforcement learning system can be used to look at customer data. If the system classifies them as high-risk and they default, the algorithm benefits. If they don't default, the program gives them a negative reward. Both instances, in the end, help machine learning by increasing its awareness of the problem and its surroundings.

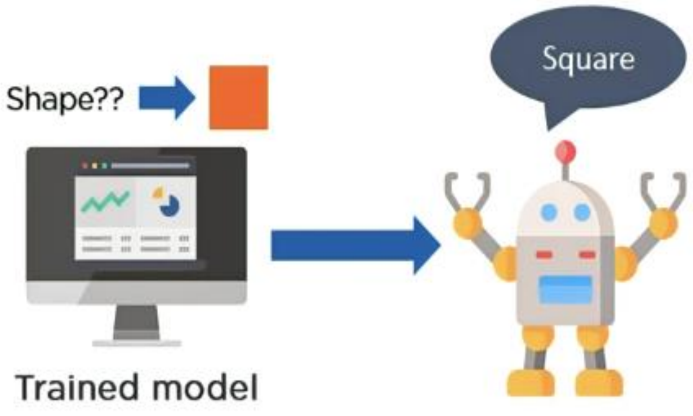


Figure 2.4: Machine Learning Work ("Types of Machine Learning | Simplilearn")

2.3.3 Machine and Deep Learning Frameworks

In this section we will discuss about what tools or frameworks will be used in this project. There were various frameworks available for machine learning to make classifications or not.

2.3.3.1 Pytorch

PyTorch is an open-source machine learning framework that is based on the Torch library. This framework is free and open-source, and it was developed by FAIR (Facebook's AI Research unit). It's a well-known machine learning framework that may be used for many different tasks, including computer vision and natural language processing. The Python interface of PyTorch is more interactive than the C++ interface ("PyTorch"). On top of PyTorch, other deep learning tools have been developed, including PyTorch Lightning, Hugging Face's Transformers, Tesla Autopilot and etc.

2.3.3.2 TensorFlow

TensorFlow is one of the most popular open-source libraries for developing deep learning and machine learning models. It was created by the Google Brain Team and provides a JS library. It is quite well-liked by machine learning aficionados, who utilise it to create various ML applications. For large-scale machine learning and deep learning projects in particular, it provides a potent library, tools, and resources for numerical computing. It makes it possible for ML developers and data scientists to quickly create and implement machine learning applications. TensorFlow offers a high-level Keras API for training and creating ML models, making it simple for users to get started with TensorFlow and machine learning ("TensorFlow").

2.3.3.3 Scikit Learn

One of the best open-source frameworks for beginning machine learning is scikit-learn. Because of its high-level wrappers, users may experiment with various methods and examine a variety of classification, clustering, and regression model. Simply by unpacking the code and following the dependencies, scikit-learn can also be a fantastic technique for the inquisitive mind to obtain more insight into the models (F. Pedregosa, G. Varoquaux, et al, 2011). The documentation for Scikit-learn is comprehensive and simple to read for both novices and experts. Scikit-learn is excellent for machine learning solutions with a constrained time and resource budget. It is solely machine learning-focused and has played a key role in popular brands' predictive solutions over the past few years.

2.3.3.4 Microsoft Azure Machine Learning

Thanks to Azure Machine Learning, data scientists and developers can create, release, and maintain high-quality models more rapidly and with greater assurance. It decreases time to value with market-leading machine learning operations (MLOps), open-source interoperability, and integrated tools. For moral machine learning applications of artificial intelligence, this trustworthy platform was developed ("Azure Machine Learning - ML as a Service | Microsoft Azure").

2.3.3.5 Jupyter Notebook

The Jupyter Notebook is an incredibly powerful tool for interactively designing and presenting data science projects. A notebook integrates code and its output in a single document that includes narrative language, mathematical equations, and other rich media. For better readability, reproducibility, and sharing of your work, it's a single page where you can run code, view the results, add reasons, formulas, and charts. Although Jupyter Notebooks support a wide range of programming languages, this post will focus on Python because it is the most popular language. Among R users, R Studio is often a more preferred solution.

2.4 Machine Learning Algorithm Selection

In this section we will discuss about the suitable types of algorithms that can be used in this project.

2.4.1 Types of the Algorithms

Machine learning techniques have been extensively investigated and used in phishing URL detection. The numerous kinds of algorithms that are accessible are illustrated here.

2.4.1.1 Decision Tree Algorithm

It ranks among the most widely used machine learning algorithms. The decision tree approach is easy to understand and apply. Starting with the best splitter among the traits that can be classified, or the tree's root, is the first task of a decision tree. Before reaching the leaf node, the algorithm keeps expanding the tree. Each internal node in a decision tree representation represents an attribute, whereas each leaf node represents a class label. Target values or classes are forecast using this training model. To calculate these nodes, the decision tree method uses the Gini index and information gain approaches ("Support Vector Machines: A Simple Explanation - KDnuggets").

2.4.1.2 Random Forest Algorithm

The random forest method, one of the most effective machine learning algorithms, is based on the concept of the decision tree algorithm. There are many decision trees in the forest, which was built using the random forest method. A huge number of trees provide high detection accuracy. The process of creating a tree employs the bootstrap methodology. Using attributes and samples from the dataset that are randomly selected, a single tree is built using the bootstrap method. The random forest method will choose the best splitter from among the randomly chosen features for categorization, much as the decision tree approach. The method also uses information gain and the Gini index to do this. This process will continue until the random forest yields on trees. The objective value is predicted by each tree in the forest, and an algorithm then determines the votes for each target forecast. The aim with the most votes is considered in the random forest algorithm's final forecast ("How the Random Forest Algorithm Works in Machine Learning - Dataaspirant").

2.4.1.3 Gradient Boost Classifier

In order to minimise a loss function, Gradient Boosting is a functional gradient algorithm that continually chooses a function that points in the direction of a weak hypothesis or negative gradient. A powerful prediction model is created by combining multiple weak learning models with a gradient boosting classifier. The theory behind this kind of hypothesis boosting is called Probability Approximately Correct Learning (PAC). This PAC learning approach looks into machine learning problems to determine their level of complexity; a comparable approach is used with hypothesis boosting. The algorithm weights the observations and instances in the training set, giving harder-to-classify examples greater weight. The system gradually adds more inexperienced students, pairing them with the hardest training cases. The majority vote method is used in AdaBoost to make predictions, and cases are categorised based on which class gets the most votes from weak learners. (GeeksforGeeks).

2.4.1.4 XG Boost Classifier

One of the most famous machine learning calculations in the planet is XGBoost. Whatever the case, whether the vaticination work is retrogression or section. The results produced by XGBoost are unquestionably superior to those of other AI calculations. In truth, it has developed the "cutting edge" machine education calculation for managing organized information ever since it began. XGBoost is a grade-appropriate variety of supporting products that has been enhanced to be exceptionally persuasive, adaptable, and portable. It carries out calculations for machine education using the Grade Boosting edge. XGBoost offers similar tree assistance (also known as GBDT or GBM) to address a variety of information understanding challenges (Aldawood and Skinner) quickly and accurately.

2.4.1.5 Logistic Regression Classifier

Logistic Regression is a classification method employed in the field of machine learning. The dependent variable is modelled using a logistic function. The dependent variable is binary, meaning it can only have two alternative outcomes (e.g., whether the URL is phishing or not). Consequently, this method is employed when handling binary data. Logistic regression employs the sigmoid function to yield the likelihood of a specific label. The Sigmoid Function is a mathematical function utilized to transform expected values into probabilities. The function can transform any real value into a value that falls between the range of 0 and 1. The logistic regression must adhere to the constraint that its value lies within the range of 0 to 1. As it is restricted to a maximum value of 1, it creates a sigmoidal curve on a graph. Here is a simple method to recognize the Sigmoid function, also known as the logistic function Doe, J. (2023, March 5).

2.5 Analysis

In this section we will discuss about the parameters that we will use to analyze the data and the dataset itself. Later, we will also discuss the metric we will be using to evaluate the performance of a model on a particular task.

2.5.1 Parameters

Parameters are an important aspect of an analysis and in this project in order to explore the effectiveness of using the algorithms different adjustments of parameters are implemented. Table 2.1 below shows the parameters of the dataset that will be used in this project. This helps us to chategorize a given population or some aspect of it.

Parameters Dataset Name	Good/Benign	Bad
(Dataset I)	✓	✓
(Dataset II)	✓	

Table 2.1: Parameters in Dataset

2.5.2 Datasets

The datasets I was obtained from Kaggle notebook which consisted of 549,346 entries. Additionally, the datasets II was retrieved from Mendeley Data which is split into 2 sets of training and testing. The Training set has 2 million URLs for each phishing and clean data. The Test set has 1 Million phish URLs and 1 million clean URLs. Both the datasets in this project are only using 2 parametersq which is good and bad urls. However, in Dataset II the good and bad urls are stored separately in 2 different files.

2.5.3 Metric

The experiment of both training and testing is repeated many times with both SVM and RF algorithms. The accuracy percentage of correct decisions among all testing samples:

$$acc = \frac{TP + TN}{TP + TN + FP + FN} \times 100\%$$

TP-True Positive counts the number of harmful URLs that have been appropriately tagged; The number of harmful URLs incorrectly labelled as safe is known as FN- False Negative; The amount of accurately labelled safe URLs is known as TN-True Negative; According to Xuan et al., the number of safe URLs incorrectly labelled as harmful is known as FP (False Positive).

Precision is the ratio of correctly classified harmful URLs (TP) to all malicious URLs classified by the classifier (TP+FP).

$$acc = \frac{TP}{TP + FP} \times 100\%$$

Recall: is the percentage of malicious URLs correctly labeled (TP) among allmalicious URLs of the testing data (TP+FN).

$$acc = \frac{TP}{TP + FN} \times 100\%$$

F1-score: is the harmonic mean of precision and recall. High F1 value means the classifier is good.

$$F1 = \frac{2 \times precision \times Recall}{precision \times Recall} \times 100\%$$

FPR (False prediction rate) is calculated as:

$$FRP = \frac{FP}{FP + TN} \times 100\%$$

Both two data subsets are utilized separately to assess the machine learning algorithm's training performance. These data subsets each have a distinct amount of data and a varied distribution of data labels, which could affect how effectively training works. The Table 2.1 shows the Confusion Matrix.

	Classified Malicious URL	Classified Safe URL
Real Malicious URLs	TP	FN
Real Safe URLs	FP	TN

Table 2.2: Confusion Matrix

2.6 Deployment of Model

There are several ways to deploy a machine learning model in a website. Given below are some common approaches.

2.6.1 FastAPi

For integrating machine learning models into a website, many people choose FastAPI. Python APIs can be built using the contemporary, high-performance web framework FastAPI. It is highly suited for deploying machine learning models because it is built for speed and effectiveness (“FastAPI”).

2.6.2 Django

Python's Django web framework offers a complete collection of tools and functionality for developing web applications. Object-Relational Mapping (ORM) system, authentication, and an admin interface are all included. Django is appropriate for bigger, more complicated projects that need a thorough framework (Django Project).

2.6.3 Node.js

You may create server-side JavaScript applications with Node.js, a well-liked JavaScript runtime environment. Express.js and Koa.js are two frameworks that can be used to build APIs to serve machine learning models. A non-blocking, event-driven architecture offered by Node.js can be helpful for handling high concurrency situations (“About | Node.js”).

2.6.4 Ruby on Rails

Rails, sometimes known as Ruby on Rails, is a Ruby-based web application framework. It adheres to the maxim of convention preceding configuration, with an emphasis on efficiency and productivity. If you prefer working with Ruby, Rails offers a simple and user-friendly framework for creating APIs (“Ruby on Rails”).

2.7 Comparison of Previous Related Projects

2.7.1 Project 1: (Ubing et al. , 2019)

It focuses on enhancing the accuracy of phishing website detection. The researchers employ a feature selection algorithm and integrate it with ensemble learning based on majority voting. They compare this approach to various classification models like Random Forest, Logistic Regression, and others. The study demonstrates that while current phishing detection technologies have an accuracy rate between 70% and 82.52%, their proposed model achieves up to 85% accuracy, outperforming existing methods. The model uses a combination of different learning models, demonstrating promising accuracy rates in experiments.

2.7.2 Project 2 : (Basit et al. , 2020)

This journal article focuses on improving phishing attack detection. It introduces a novel ensemble model combining Artificial Neural Network (ANN), K-Nearest Neighbors (KNN), Decision Tree (C4.5), and Random Forest Classifier (RFC). This model aims to enhance accuracy in detecting website phishing attacks. The study demonstrates that the ensemble of KNN and RFC achieves a detection accuracy of 87.33%. The paper emphasizes the increasing importance of effective phishing detection methods in the context of rising cyber threats, particularly during the COVID-19 pandemic.

2.7.3 Project 3 : (Vishva and Aju , 2021)

This journal presents a system for detecting phishing websites. It utilizes a novel approach that combines URL analysis and content analysis using TF-IDF values. The system employs machine learning classifiers including Logistic Regression, Random Forest, Support Vector Machine, Naive Bayes, and Stochastic Gradient Descent. The methodology achieves an accuracy of 80.68%. The study emphasizes the importance of advanced phishing detection in the context of growing cybersecurity threats.

2.7.4 Project 4 : (Al-Sarem et al. , 2021 & Ghaleb Al-Mekhlafi et al. , 2022)

This journal discusses on an improved method for detecting phishing websites using an optimized stacking ensemble model. This model combines several machine learning algorithms, including Random Forests, AdaBoost, XGBoost, Bagging, GradientBoost, and LightGBM, and optimizes them using a genetic algorithm. The study tests this approach on multiple datasets, achieving high detection accuracy and demonstrating its effectiveness over traditional methods. The paper contributes significantly to cybersecurity by offering a robust solution to combat phishing attacks.

2.7.5 Project 5 : (Taha , 2021)

The journal proposes a novel phishing detection method. It combines multiple machine learning algorithms with a weighted soft voting mechanism, enhancing the accuracy of phishing website detection. The study uses a publicly available dataset from the UCI Machine Learning Repository and achieves a high accuracy rate of 83% and an Area Under the Curve (AUC) of 78.8%. The research underscores the efficacy of using an ensemble approach with weighted voting in cybersecurity applications.

2.7.6 Summary

No.	Author reference	Classifier	Features	Issues
1	Ubing et al. (2019)	The majority voting classifier consists of several classifiers, including Gaussian Naive Bayes, Support Vector Machine, k-Nearest Neighbours, Logistic Regression, Multilayer Perceptron Neural Network, Gradient Boosting, and Random Forest classifiers.	The qualities of a URL include its lexical characteristics, content-based attributes, and external factors such as DNS information and the reputation of the web page.	• In majority voting, the individual probabilities assigned to each model are ignored. • The process of selecting features requires a thorough understanding of the domain, as the chosen features may not effectively utilise the dataset. • Content-based features need a higher amount of computational resources and are not secure when handling harmful content. • Additional processing overhead is required for external features such as DNS server-based features and web page reputation features.
2	Basit et al. (2020)	Ensemble model (combines two classifiers taking RFC as a base classifier with ANN, kNN and C4.5 algorithms)	URL lexical features, content-based features, external features such as DNS information, reputation of the web page	• The importance of the base classifiers is disregarded. • Extracting features requires time, particularly for external data, and analysing content might lead to security concerns.
3	Vishva and Aju (2021)	URL analysis: Uses default feature set and three machine	URL features and content features	• Weightage of the base classifier is not taken into

		learning classification algorithms were used such as linear regression, random forest and support vector machine for classification. Content analysis: vectorizing the text using TF-IDF, bag of words, n-grams and Naïve Bayes, linear regression, linear SVM, stochastic gradient descent and random forest classifiers are used for classification		account, and classification is based on the best classifier from both URL analysis and content analysis. • Feature extraction takes time, especially for external data, and content analysis can cause security issues.
4	Al-Sarem et al. (2021), Ghaleb Al-Mekhlafi et al. (2022)	The stacking ensemble approach utilises a combination of random forests, AdaBoost, XGBoost, Bagging, GradientBoost, and LightGBM models. The parameters of these models are optimised using genetic algorithms. The top three models have been chosen for classification.	URL features and content features	• Optimising parameters through the utilisation of genetic algorithms. • The weight of the base classifier is disregarded. • Feature extraction is a time-consuming process, particularly when dealing with external data.
5	Taha (2021)	k statistics-based weighted soft voting. (Weights were assigned to the individual classifiers using k statistics.)	URL lexical features, content-based features, external features such as DNS information, reputation of the web page	• Feature extraction takes time, especially for external data, and content analysis can cause security issues

Table 2.3 : Comparison of Past Related Projects

2.8 Summary

Overall, these projects will analyze phishing URLs using machine learning techniques. To identify and solve the problem, this chapter discusses the definition of phishing, categories, type of attack, definition machine learning, type of learning, tools that used definition techniques, classification technique, parameter, dataset, metric, and comparison of previous related works. More research and related works on phishing URLs utilizing machine learning techniques will be discussed in the following chapter.

CHAPTER 3

METHODOLOGY

3.1 Introduction

In this chapter, it presents the method that has been conducted to develop the project, to achieve the objectives. There are three main parts in this chapter, which is known as the study design, project methodology and elaboration of the process flow.

3.2 Project Workflow

In this section will discuss about the project methodology. Figure 3.1 shows the analysis methodology for phishing URL, which consists of 9 phases. During the first phase, the setup required to train and test the model is established in this case Jupyter Notebook was used. The second phase is to import the required libraries to execute blocks of codes that might require them such as for visualization and training the model. Thirdly, we will look for datasets to employ in this project and import them. The fourth phase will be to do feature engineering of extracting lexical features from raw URLs as these features will be used as the input features for training the machine learning model. The fifth phase is to do visualization for the features extracted to help get a better understanding of them to be used in this project. The sixth phase is to do data processing for cleaner data. The seventh phase is to split the dataset for testing and training. The eighth phase is to find the machine learning tool then to identify the optimum algorithm based on the dataset. Finally, in the last phase, the model will be deployed to a local server. The steps in methodology are explained in detail in the section below.

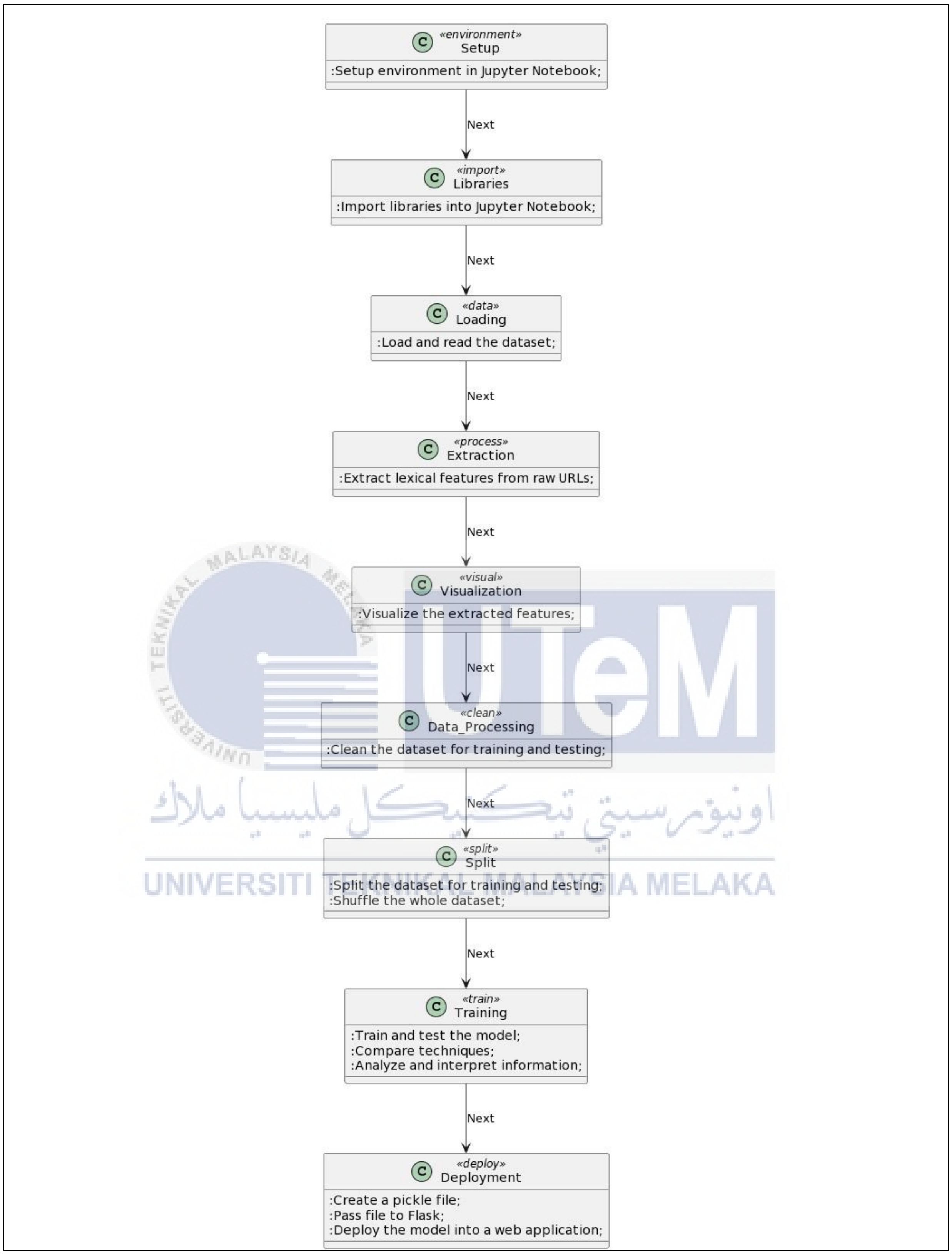

«environment» Setup
:Setup environment in Jupyter Notebook;
Next
«import» Libraries
:Import libraries into Jupyter Notebook;
Next
«data» Loading
:Load and read the dataset;
Next
«process» Extraction
:Extract lexical features from raw URLs;
Next
«visual» Visualization
:Visualize the extracted features;
Next
«clean» Data_Processing
:Clean the dataset for training and testing;
Next
«split» Split
:Split the dataset for training and testing;
:Shuffle the whole dataset;
Next
«train» Training
:Train and test the model;
:Compare techniques;
:Analyze and interpret information;
Next
«deploy» Deployment
:Create a pickle file;
:Pass file to Flask;
:Deploy the model into a web application;

Figure 3.1: Flowchart of Building the Machine Learning Model

3.2.1 Phase 1 : Setting up the Environment

Figure 3.1 shows about phase I methodology in this project which is setting up the environment. During the initial phase of this project, a machine learning tool is required to implement and execute various machine learning algorithms and techniques. It provides an environment and set of functionalities that enable users to train, test and deploy machine learning models efficiently.

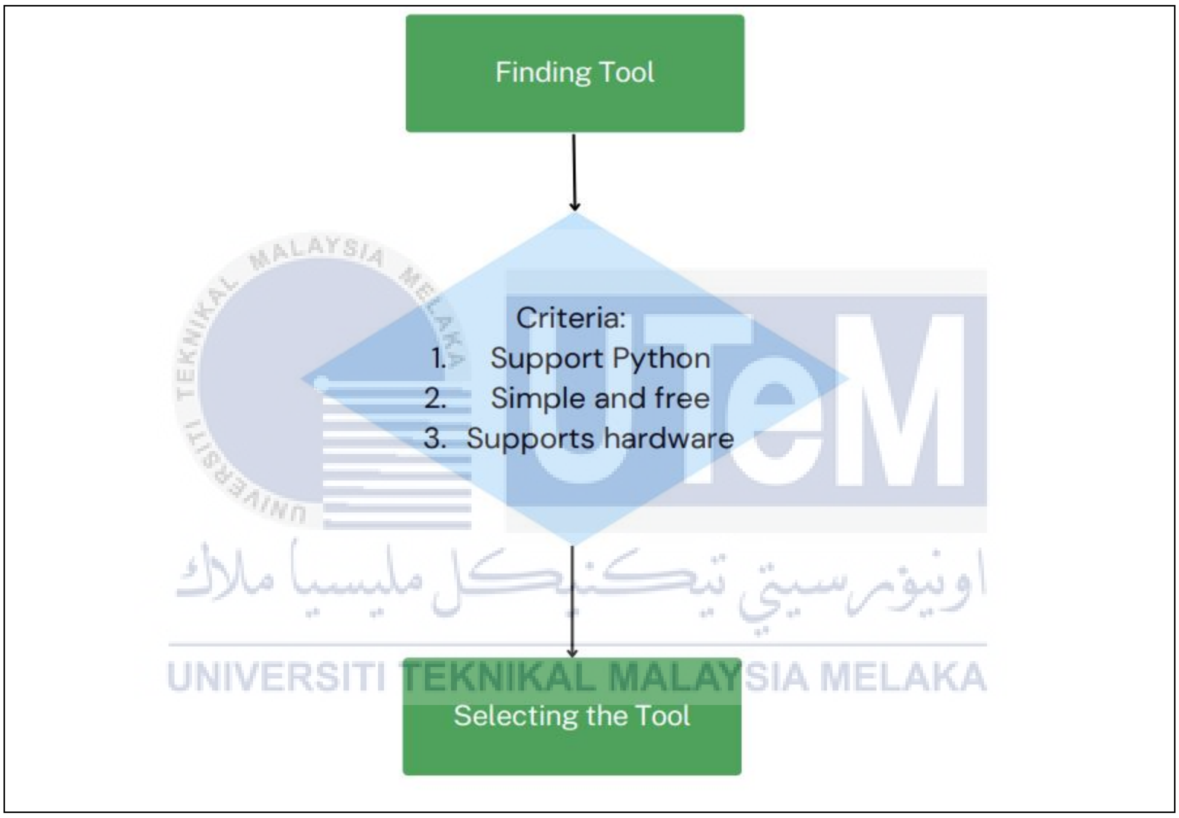
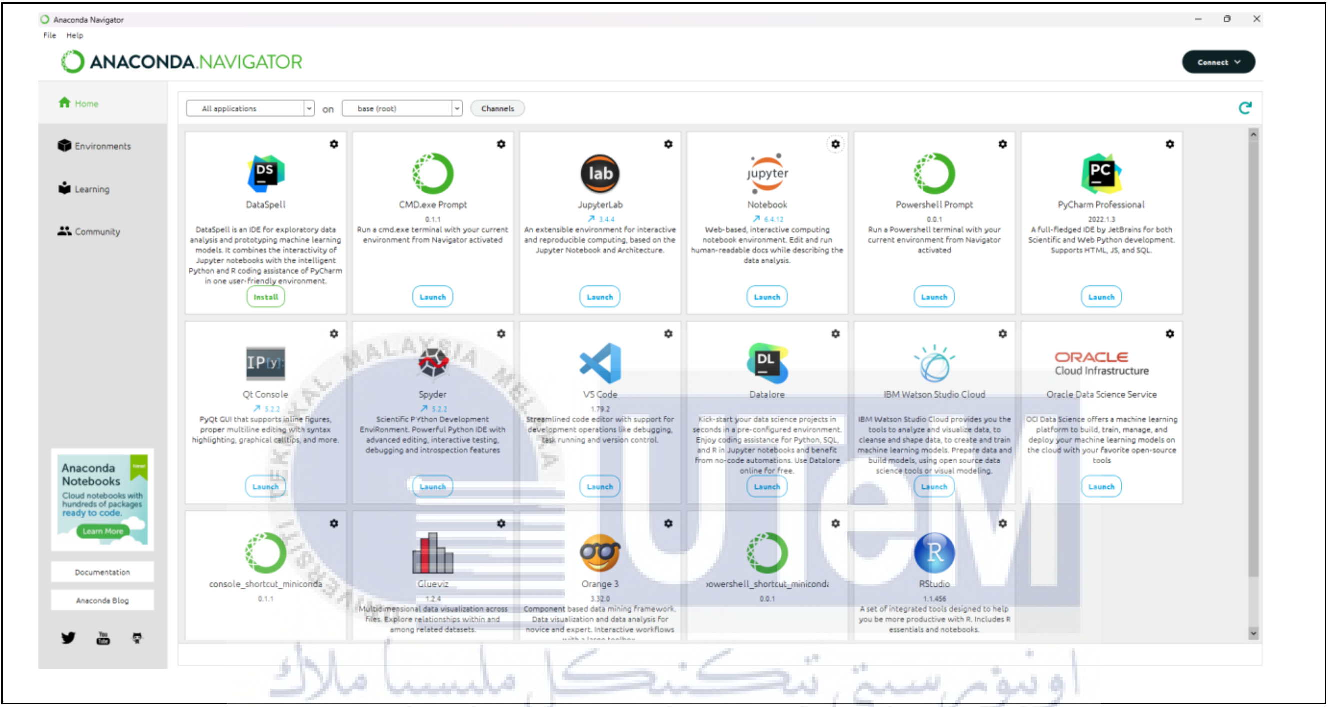


Figure 3.2: Find tool

After careful consideration and analysis, Jupyter Notebook was choosen as the machine learening framework. This is because Jupyter Notebook is a free web-based application that provides with an easy-to-use, interative data environment. It supports all programming language including Python and since it's web based application it can be run on any environment that has the minimum hardware requirements. As for this project, it will be run on a Windows 11 device with 12GB or RAM. In order to get started with it, Anaconda was installed first . It is a distributor for the Python that comes with preloaded with all the most popular libraries and tools. It lets users run Jupyter Notebook right away without the hassle of managing countless installations or worrying about dependancies and OS-specific installation issues. Figure 3.3 shows the Jupyter Notebook in the Anaconda Navigator window.



DataSpell
DataSpell is an IDE for exploratory data analysis and prototyping machine learning models. It combines the interactivity of Jupyter notebooks with the intelligent Python and R coding assistance of PyCharm in one user-friendly environment.

CMD.exe Prompt
0.1.1
Run a cmd.exe terminal with your current environment from Navigator activated

JupyterLab
3.4.4
An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Notebook
6.4.12
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

Powershell Prompt
0.0.1
Run a Powershell terminal with your current environment from Navigator activated

PyCharm Professional
2022.1.3
A full-fledged IDE by JetBrains for both Scientific and Web Python development. Supports HTML, JS, and SQL.

Qt Console
5.2.2
PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.

Spyder
5.2.2
Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

VS Code
1.79.2
Streamlined code editor with support for development operations like debugging, task running and version control.

Datalore
Kick-start your data science projects in seconds in a pre-configured environment. Enjoy coding assistance for Python, SQL, and R in Jupyter notebooks and benefit from no-code automations. Use Datalore online for free.

IBM Watson Studio Cloud
IBM Watson Studio Cloud provides you the tools to analyze and visualize data, to cleanse and shape data, to create and train machine learning models. Prepare data and build models, using open source data science tools or visual modeling.

ORACLE Cloud Infrastructure
Oracle Data Science Service
OCI Data Science offers a machine learning platform to build, train, manage, and deploy your machine learning models on the cloud with your favorite open-source tools

console_shortcut_miniconda
0.1.1

Glueviz
1.2.4
Multidimensional data visualization across files. Explore relationships within and among related datasets.

Orange 3
3.32.0
Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows

owershell_shortcut_minicond
0.0.1

RStudio
1.1.456
A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks.

Figure 3.3: Anaconda Navigator Window

After selecting or creating the desired environment, Anaconda Navigator will launch Jupyter Notebook in your web browser. Jupyter Notebook provides an interactive computing environment where you can write and execute Python code, create visualizations, and document your work using Markdown. Once Jupyter Notebook is opened, you can create a new notebook by clicking on the "New" button and selecting "Python 3" or any other available kernel that suits your project requirements. In the notebook interface, you can write and execute Python code cells as in Figure 3.4. You can also add Markdown cells for documentation and explanations. Jupyter Notebook allows you to run individual code cells or the entire notebook.

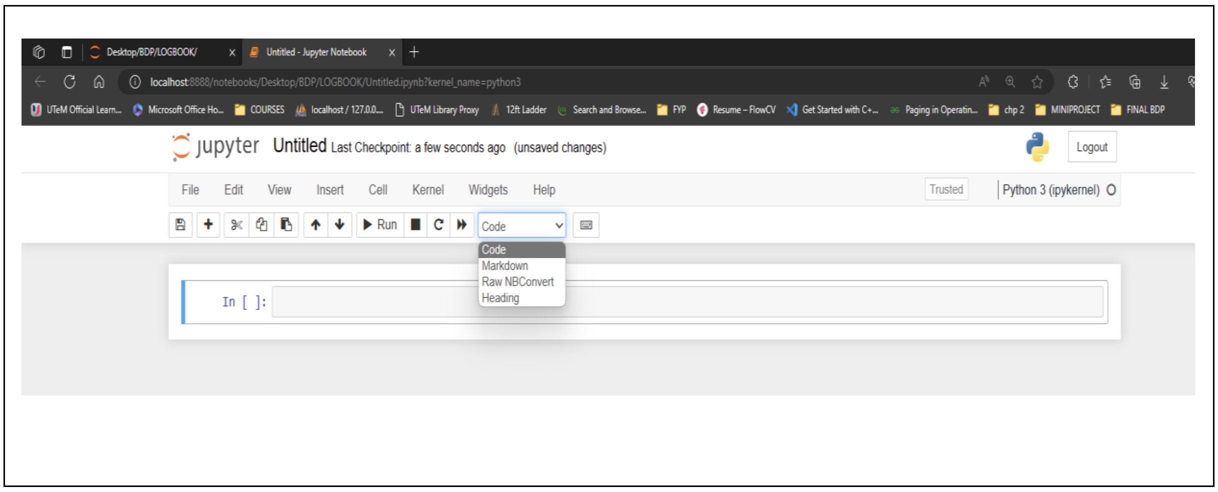
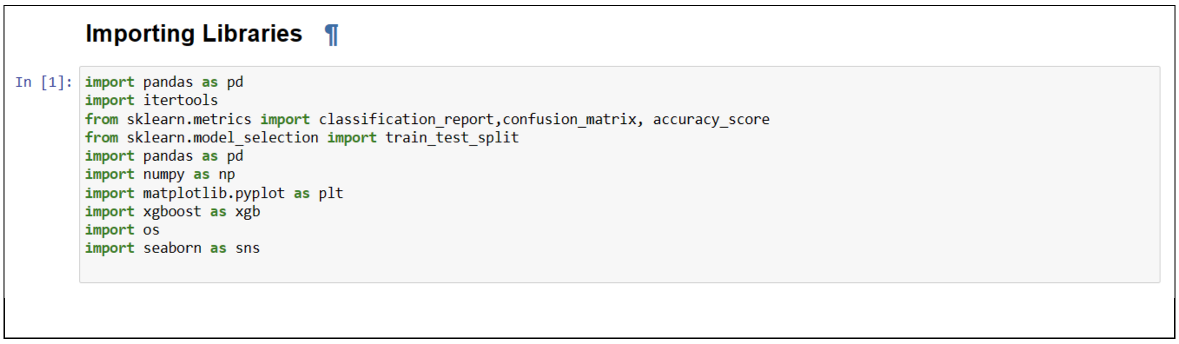

jupyter Untitled Last Checkpoint: a few seconds ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
Code
Markdown
Raw NBConvert
Heading
In [ ]:

Figure 3.4: Jupyter Notebook Environment

3.2.2 Phase 2: Importing Libraries

The libraries utilized in the code snippet in Figure 3 serve crucial roles in machine learning and data analysis tasks. These libraries provide reusable code, efficient algorithms, extended functionality, community support, integration capabilities, and security. By leveraging these libraries, developers can streamline their workflows, enhance their analyses, and build robust machine learning models with ease. More libraries can be added on while going through the rest of the process such as feature extractions and exploratory data analysis.



Importing Libraries ¶

```
In [1]: import pandas as pd
        import itertools
        from sklearn.metrics import classification_report,confusion_matrix, accuracy_score
        from sklearn.model_selection import train_test_split
        import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import xgboost as xgb
        import os
        import seaborn as sns
```

Figure 3.5: Imported Libraries

3.2.3 Phase 3: Finding and Selecting Dataset to Import

Figure 3.6 shows the phase II methodology in this project which is finding the dataset. The phishing data consists of a phishing URL gathered from earlier researches. Two number of datasets were choosen based on the criteria required. As for finding the dataset. PhishTank, Kaggle and UCI Machine Learning Repository was used. This dataset will be later split into for both training and testing. However, some level of preprocessing data might be required.

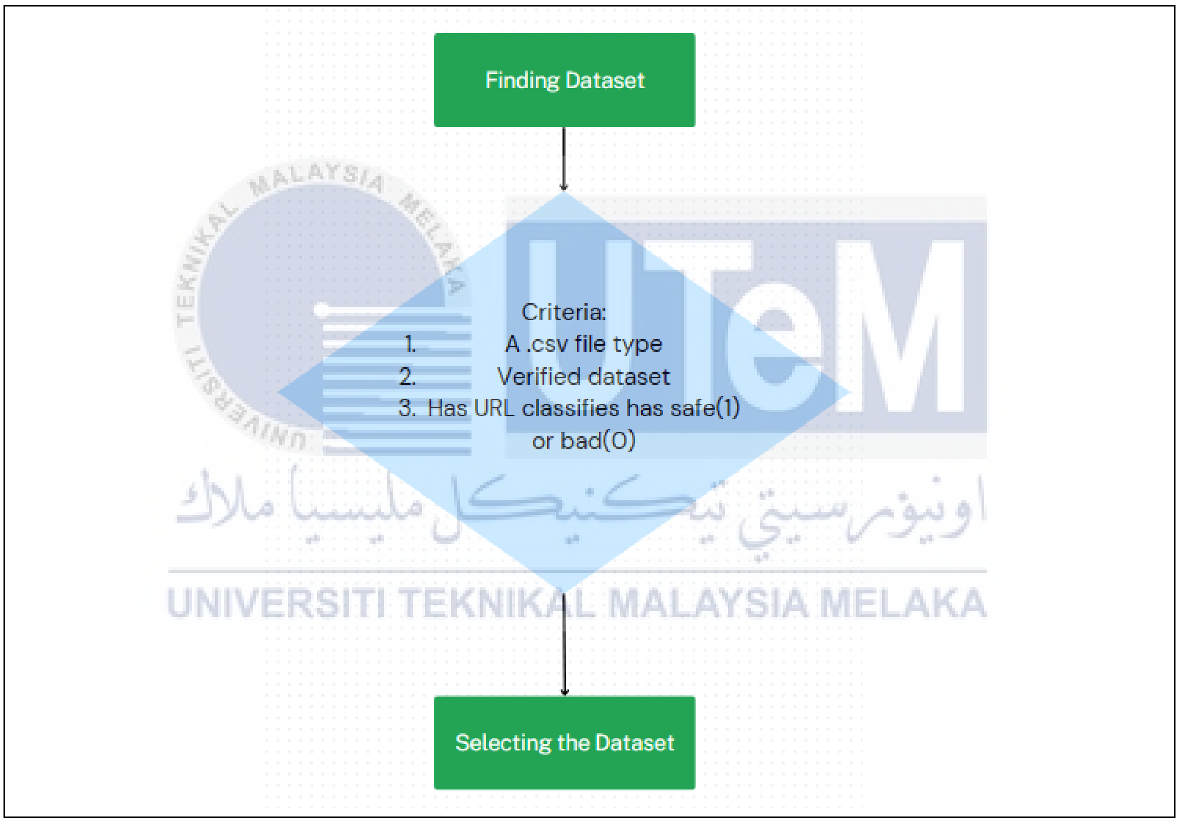
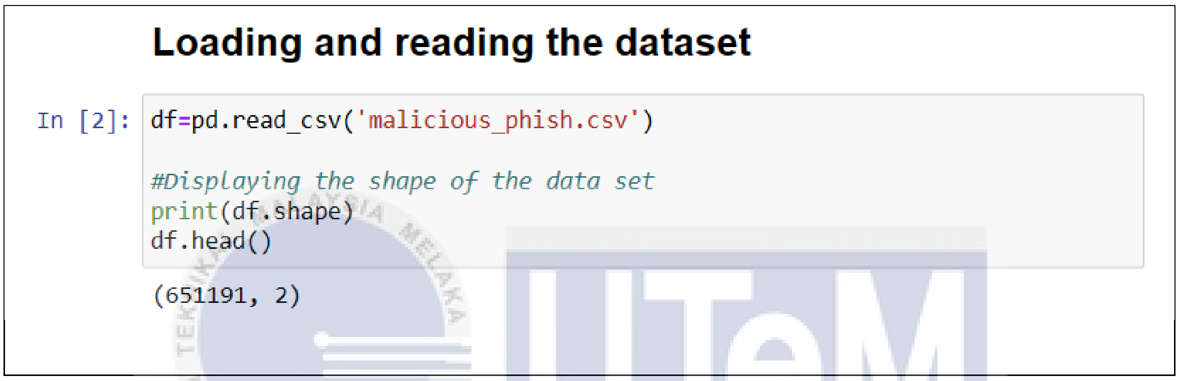


Figure 3.6: Find phishing URLs dataset

After weighing in the factors such as quality , accurasy of labels and the adequate sample size, the datasets from (Malicious URLs dataset | PhishTank) was choosen. A comprehensive dataset containing verified phishing URLs was assembled by the members of the community. To expand the dataset with more benign URLs, dataset was retrieved from personal browster history that were verified to be safe sites such as goverment and large corporation sites.

Loading and reading datasets are crucial steps in data analysis and machine learning. These steps involve retrieving data from external sources and converting it into a format that can be processed by Python code.. To load the dataset, the function of ‘read_csv()’ offered by the pandas is used to load the CSV file as in Figure 3.7.



Loading and reading the dataset

```
In [2]: df=pd.read_csv('malicious_phish.csv')

        #Displaying the shape of the data set
        print(df.shape)
        df.head()

        (651191, 2)
```

Figure 3.7: Code Snippet of Loading the Dataset

After loading the dataset, it is assigned to a variable, usually in the form of a pandas DataFrame. This tabular data structure enables easy manipulation and analysis of the data. The dataset can then be explored to understand its structure and contents. Functions and methods provided by the library allow users to view the shape of the dataset, check column names, examine sample data, and obtain basic statistics.

3.2.4 Phase 4 : Feature Extracton

After importing the datasets, the following lexical features will be extracted from raw URLs in this step and utilised as input features to train the machine learning model. The ensuing features are produced based on the Table 3.1. Each feature plays a crucial role in distinguishing between legitimate and malicious URLs. By examining these features closely, machine learning models are able to identify patterns often seen in phishing sites. The objective is to develop a robust system capable of accurately detecting and flagging potential phishing URLs, enhancing cyber security and safeguarding users from online threats. This method, which considers various elements of a URL's structure and content, ensures a high level of precision and reliability in phishing detection, offering strong protection against the myriad of deceptions found on the internet.

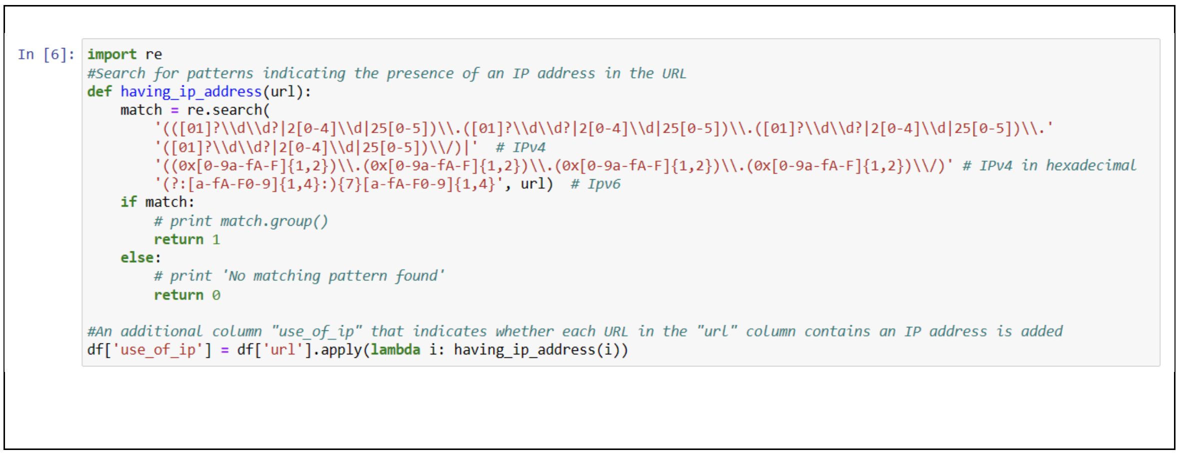
No	Feature	Description
1	IpAddress	Check if the IP address is used in the hostname of the website URL
2	Abnormal URL	Identity is frequently included in the URL of a trustworthy website.
3	GoogleIndex	Determine whether or not the URL has been indexed by Google Search Engine.
4	CountNumDot	Number of character '.' in URL
5	Count-www	Number of www in URL
6	Count_@	There exists a character '@' in URL
7	DoubleSlashInPath	There exists a slash '//' in the link path
8	UrlLength	The length of URL
9	NumDash	Number of the dash character '-'
10	NumSensitiveWords	Number of sensitive words (i.e., “secure”, “account”, “webscr”, “login”, “ebayisapi”, “sign in”, “banking”, “confirm”) in website
11	ShortURL	Determine whether a URL has been shortened using a service, such as bit.ly, goo.gl, go2l.ink, etc.
12	NoHttps	Check if there exists a HTTPS in website URL
13	CountHTTP	Check number of HTTP in URL
14	NumPercent	Number of the character '%'
15	Count(?)	Check number of times (?) has been used in URL
16	Count(-)	Check number of times (-) has been used in URL
17	Couunt(=)	Check number of times (=) has been used in URL
18	HostnameLength	Length of hostname
19	PathLength	Length of the link path
20	InitialDirectorySize	Determine how long the URL's first directory should be
21	TPD_Length	Determine length of the TLD
22	NumNumericChars	Number of the numeric character

Table 3.1: List of URL Features

Given below is few explanations of how the feature extractions works.

3.2.4.1 Search for Presence of IP Address

Initially, a search for patterns indicating the presence of an IP address in the URL is done as in Figure 3.8.



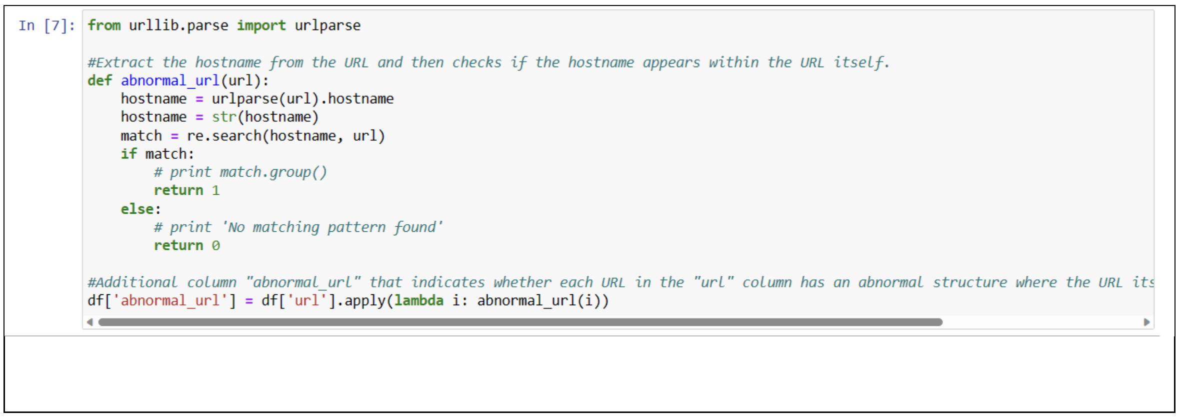
```
In [6]: import re
        #Search for patterns indicating the presence of an IP address in the URL
        def having_ip_address(url):
            match = re.search(
                '(([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.'
                '([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\/)|'  # IPv4
                '((0x[0-9a-fA-F]{1,2})\\.(0x[0-9a-fA-F]{1,2})\\.(0x[0-9a-fA-F]{1,2})\\.(0x[0-9a-fA-F]{1,2})\\/)' # IPv4 in hexadecimal
                '(?:[a-fA-F0-9]{1,4}:){7}[a-fA-F0-9]{1,4}', url)  # Ipv6
            if match:
                # print match.group()
                return 1
            else:
                # print 'No matching pattern found'
                return 0

        #An additional column "use_of_ip" that indicates whether each URL in the "url" column contains an IP address is added
        df['use_of_ip'] = df['url'].apply(lambda i: having_ip_address(i))
```

Figure 3.8: Code Snippet of Searching Presence of IP Address

The provided code snippet contains a function called having_ip_address() that checks if a given URL contains an IP address. It uses the regular expression search function from the re module to search for specific IP address patterns, including IPv4 and IPv6 formats. If a match is found, indicating the presence of an IP address, the function returns 1; otherwise, it returns 0.

3.2.4.2 Check for Hostname within URL

Nextly, an attempt to extract the hostname from the URL and then to check if the hostname appears within the URL itself is done as per in Figure 3.9.



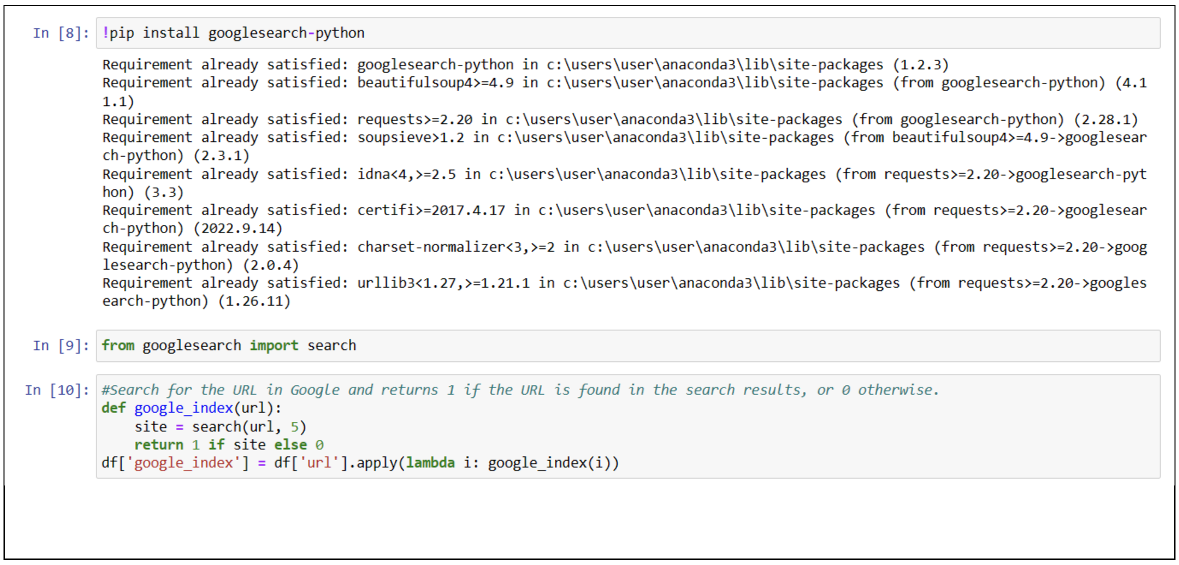
```
In [7]: from urllib.parse import urlparse

        #Extract the hostname from the URL and then checks if the hostname appears within the URL itself.
        def abnormal_url(url):
            hostname = urlparse(url).hostname
            hostname = str(hostname)
            match = re.search(hostname, url)
            if match:
                # print match.group()
                return 1
            else:
                # print 'No matching pattern found'
                return 0

        #Additional column "abnormal_url" that indicates whether each URL in the "url" column has an abnormal structure where the URL its
        df['abnormal_url'] = df['url'].apply(lambda i: abnormal_url(i))
```

Figure 3.9: Code Snippet of Checking for Hostname

The provided code snippet demonstrates a function called abnormal_url() that identifies URLs with abnormal structures where the URL itself contains its own hostname. It utilizes the urlparse() function from the urllib.parse module to extract the hostname from the given URL. The function takes a URL as input and uses urlparse(url).hostname to retrieve the hostname. It then converts the hostname to a string for further processing. Next, it applies a regular expression search using re.search() to check if the hostname appears within the URL itself. If a match is found, indicating that the URL contains its own hostname, the function returns 1. Otherwise, it returns 0. This function is applied to each URL in the DataFrame df['url'] using a lambda function.

3.2.4.3 Search for URL in Google

As in Figure 3.10 , the provided code snippet installs the googlesearch-python library and imports the search function from it. The function google_index() takes a URL as input and performs a Google search using the search() function, searching for the given URL in the top 5 search results. If the URL is found in the search results, indicating that it is indexed by Google, the function returns 1. Otherwise, it returns 0. This function is then applied to each URL in the DataFrame df['url'] using a lambda function, and the results are stored in a new column called google_index in the DataFrame.

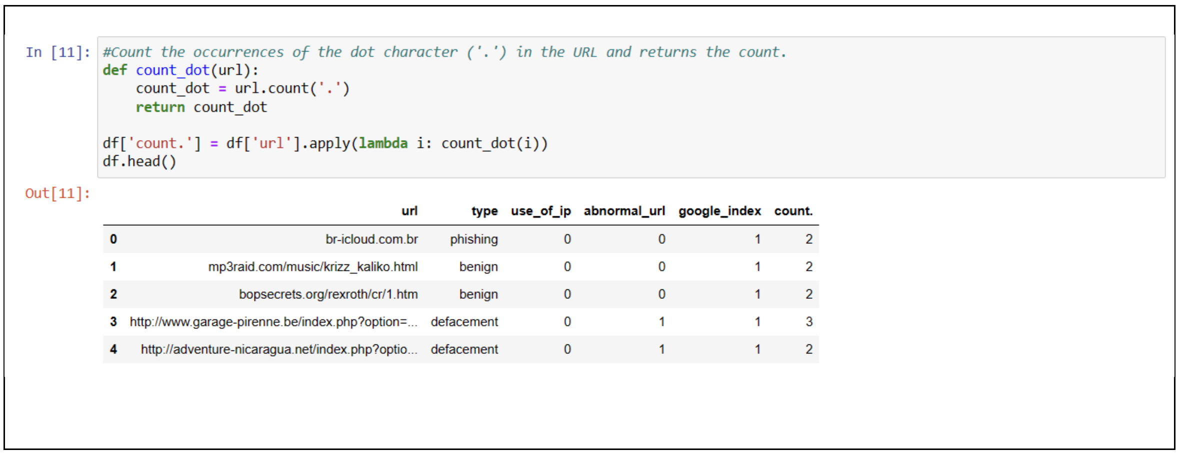


```
In [8]: !pip install googlesearch-python

        Requirement already satisfied: googlesearch-python in c:\users\user\anaconda3\lib\site-packages (1.2.3)
        Requirement already satisfied: beautifulsoup4>=4.9 in c:\users\user\anaconda3\lib\site-packages (from googlesearch-python) (4.1
        1.1)
        Requirement already satisfied: requests>=2.20 in c:\users\user\anaconda3\lib\site-packages (from googlesearch-python) (2.28.1)
        Requirement already satisfied: soupsieve>1.2 in c:\users\user\anaconda3\lib\site-packages (from beautifulsoup4>=4.9->googlesear
        ch-python) (2.3.1)
        Requirement already satisfied: idna<4,>=2.5 in c:\users\user\anaconda3\lib\site-packages (from requests>=2.20->googlesearch-pyt
        hon) (3.3)
        Requirement already satisfied: certifi>=2017.4.17 in c:\users\user\anaconda3\lib\site-packages (from requests>=2.20->googlesear
        ch-python) (2022.9.14)
        Requirement already satisfied: charset-normalizer<3,>=2 in c:\users\user\anaconda3\lib\site-packages (from requests>=2.20->goog
        lesearch-python) (2.0.4)
        Requirement already satisfied: urllib3<1.27,>=1.21.1 in c:\users\user\anaconda3\lib\site-packages (from requests>=2.20->googles
        earch-python) (1.26.11)

In [9]: from googlesearch import search

In [10]: #Search for the URL in Google and returns 1 if the URL is found in the search results, or 0 otherwise.
         def google_index(url):
             site = search(url, 5)
             return 1 if site else 0
         df['google_index'] = df['url'].apply(lambda i: google_index(i))
```

Figure 3.10: Code Snippet of Searching in Google

3.2.4.4 Count numbe of dots(.) in URL

The provided code snippet in Figure 3.11 demonstrates a function called count_dot() that counts the occurrences of the dot character ('.') in a given URL. The function uses the count() method available for strings in Python to count the number of dots in the URL.The function takes a URL as input and applies url.count('.') to count the occurrences of the dot character. It then returns the count. This function is applied to each URL in the DataFrame df['url'] using a lambda function.The code also includes an additional step where the count_dot() function is applied to each URL, and the results are stored in a new column called count. in the DataFrame.



```
In [11]: #Count the occurrences of the dot character ('.') in the URL and returns the count.
         def count_dot(url):
             count_dot = url.count('.')
             return count_dot

         df['count.'] = df['url'].apply(lambda i: count_dot(i))
         df.head()
```

```
Out[11]:
                                               url        type  use_of_ip  abnormal_url  google_index  count.
0                                   br-icloud.com.br    phishing          0             0             1       2
1                mp3raid.com/music/krizz_kaliko.html      benign          0             0             1       2
2                    bopsecrets.org/rexroth/cr/1.htm      benign          0             0             1       2
3  http://www.garage-pirenne.be/index.php?option=...  defacement          0             1             1       3
4   http://adventure-nicaragua.net/index.php?optio...  defacement          0             1             1       2
```

Figure 3.11: Count Number of dot in the URL

Similarly, the same method was used to count other lexical features by defining separate functions that extract and count specific features of interest. For example, a function can be created to count the occurrence of a specific character, count the length of the URL, count the number of digits, count the number of special characters, or count the occurrence of certain keywords. By defining and applying these functions to the URLs in the DataFrame, various lexical features can be extracted and quantified that may be informative for the analysis or machine learning model.

3.2.5 Phase 5: Data Visualization

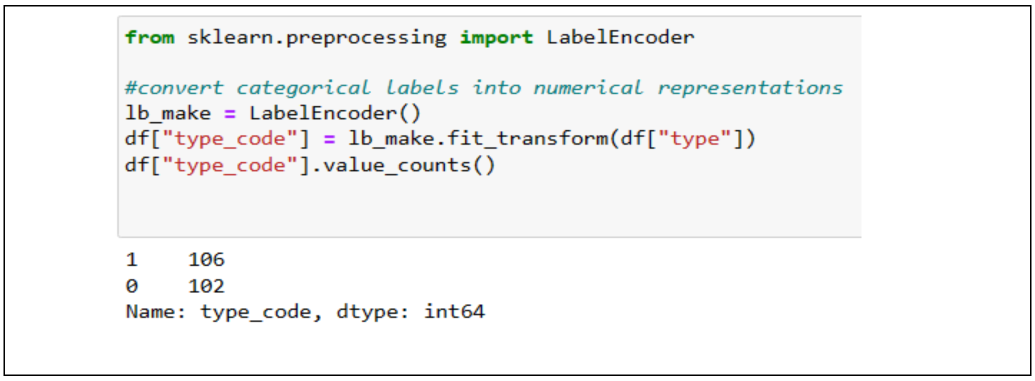
Data visualization in the context of feature extraction, as exemplified by the seaborn (sns) box plot. This is done to understand the data distribution because it help on understanding how these features vary across different groups. This can reveal patterns or anomalies that might not be apparent from raw data. By visualizing the distribution of features across different classes, one can assess how well a feature might distinguish between those classes. This helps in understanding the relevance of each feature for the classification task. It is a vital step in the data science workflow as it aids in understanding, preprocessing, and utilizing data effectively for building robust machine learning models.

3.2.6 Phase 6: Data Proprocessing

Data processing, also referred to as data preprocessing or data preparation, is a critical step that follows exploratory data analysis (EDA) in the overall data analysis workflow. It involves transforming raw data into a format suitable for machine learning algorithms or further analysis. Data processing plays a pivotal role in enhancing the quality, usability, and effectiveness of the data for modeling and analysis purposes. Feature engineering is another integral part of data processing. It involves creating new features or transforming existing ones to capture relevant information and improve model performance. Below are the two vital preprocessing methods that have been completed.

I) Target Encoding (Figure 3.12)

Target encoding is a common preprocessing step in machine learning especially for classification tasks. It enables the conversion of categorical labels into a numerical format, allowing machine learning algorithms to process the data effectively. The numerical codes assigned to each category are typically in ascending order, representing different classes or categories of the target variable. This encoding process facilitates the training and evaluation of classification models, as they generally require numeric inputs for predictions and analysis.



```
from sklearn.preprocessing import LabelEncoder

#convert categorical labels into numerical representations
lb_make = LabelEncoder()
df["type_code"] = lb_make.fit_transform(df["type"])
df["type_code"].value_counts()
```

```
1    106
0    102
Name: type_code, dtype: int64
```

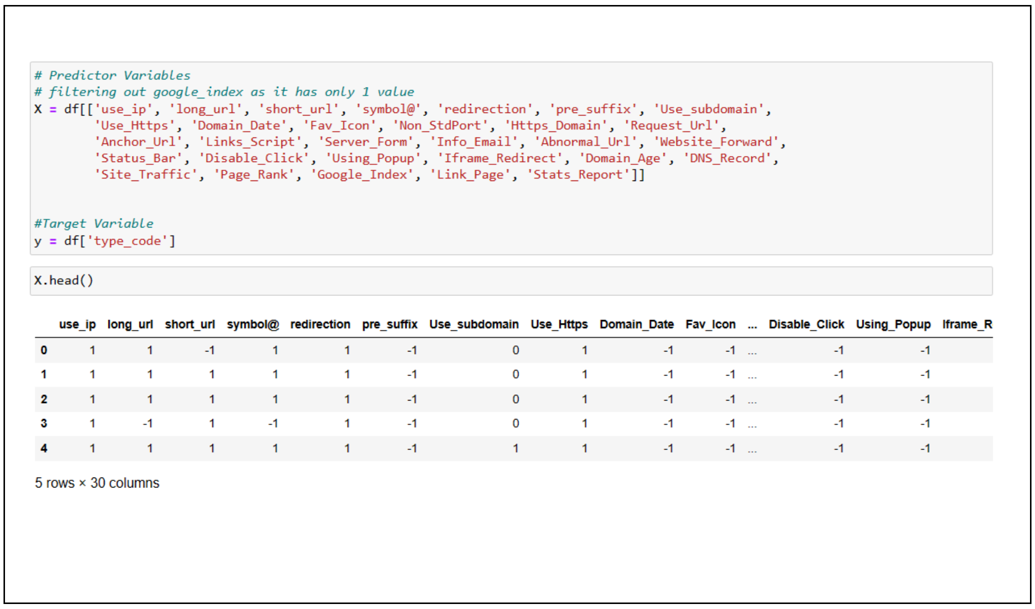
Figure 3.12: Target Encoding

The code snippet provided demonstrates the process of encoding categorical labels into numerical representations using the LabelEncoder class from the scikit-learn library. This step is commonly known as target encoding, where the target variable (in this case, "type") is encoded into numerical codes to be understood by machine learning algorithms.

First, the LabelEncoder object is instantiated as "lb_make". Then, the "fit_transform" method of the LabelEncoder object is applied to the "type" column of the DataFrame, denoted as "df['type']". This step fits the encoder to the unique categories in the "type" column and transforms those categories into corresponding numerical codes. The resulting numerical codes are assigned to a new column called "type_code" in the DataFrame.The "value_counts()" method is then used to count the occurrences of each numerical code in the "type_code" column. This provides insights into the distribution of the encoded labels, showing how many instances belong to each category.

II) Creation of Feature and Target (Figure 3.13)

This part of the code creates the predictor variables (features) and the target variable, setting the stage for model training and evaluation.



```python
# Predictor Variables
# filtering out google_index as it has only 1 value
X = df[['use_ip', 'long_url', 'short_url', 'symbol@', 'redirection', 'pre_suffix', 'Use_subdomain',
        'Use_Https', 'Domain_Date', 'Fav_Icon', 'Non_StdPort', 'Https_Domain', 'Request_Url',
        'Anchor_Url', 'Links_Script', 'Server_Form', 'Info_Email', 'Abnormal_Url', 'Website_Forward',
        'Status_Bar', 'Disable_Click', 'Using_Popup', 'Iframe_Redirect', 'Domain_Age', 'DNS_Record',
        'Site_Traffic', 'Page_Rank', 'Google_Index', 'Link_Page', 'Stats_Report']]


#Target Variable
y = df['type_code']
```

```python
X.head()
```

```
   use_ip  long_url  short_url  symbol@  redirection  pre_suffix  Use_subdomain  Use_Https  Domain_Date  Fav_Icon  ...  Disable_Click  Using_Popup  Iframe_R
0       1         1         -1        1            1          -1              0          1           -1        -1  ...             -1           -1
1       1         1          1        1            1          -1              0          1           -1        -1  ...             -1           -1
2       1         1          1        1            1          -1              0          1           -1        -1  ...             -1           -1
3       1        -1          1       -1            1          -1              0          1           -1        -1  ...             -1           -1
4       1         1          1        1            1          -1              1          1           -1        -1  ...             -1           -1

5 rows × 30 columns
```

Figure 3.13: Creation of Feature and Target

In the provided code snippet, the predictor variables and the target variable are created as part of the data preprocessing stage. The predictor variables, denoted as "X," are selected from the DataFrame "df" using double square brackets.These variables are features that will be used to predict the target variable. In this case, the selected predictor variables include features such as 'use_of_ip', 'long_url', 'Domain_Date', 'Disable_Click', 'Https_Domain', and various other features. These features are extracted from the DataFrame and stored in the "X" variable.

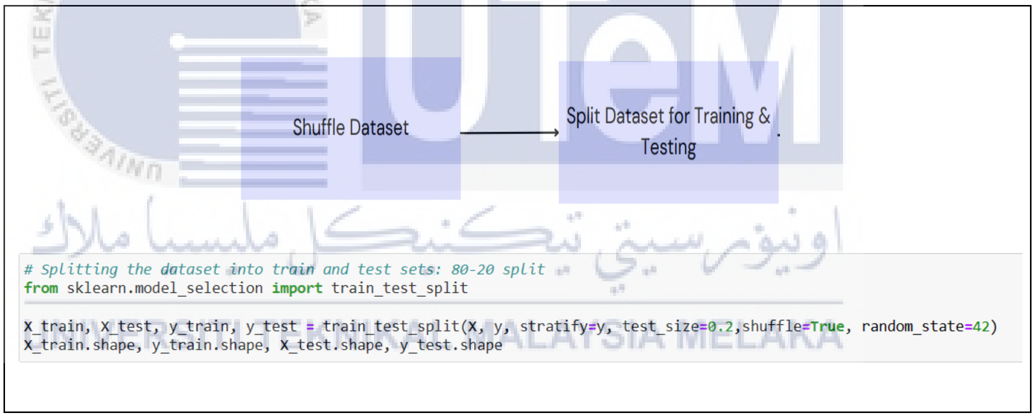
On the other hand, the target variable, denoted as "y," is extracted from the DataFrame "df" using single square brackets. In this case, the target variable is 'type_code,' which represents the encoded numerical labels for the target variable 'type.' The target variable is stored in the "y" variable.

By separating the predictor variables (features) and the target variable, the data is prepared for further analysis and modeling. The predictor variables are used as input to train the machine learning model, while the target variable serves as the ground truth or the variable to be predicted.

This separation allows for easy manipulation and processing of the data, as well as the ability to apply various machine learning algorithms to predict the target variable based on the given features.

3.2.7 Phase 7: Splitting Dataset for Training and Testing

The dataset is split into two parts which is a training set and a test set as like in Figure 3.14. This is to make sure the machine learning model generalizes the unseen data. The 'X_train' and 'y_train' represents the training data and labels. In addition, the 'X_test' and 'y_test' represent the testing data and labels. The dataset is split into a ratio of 80 to 20 for training and testing respectively.



```
# Splitting the dataset into train and test sets: 80-20 split
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.2,shuffle=True, random_state=42)
X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

Figure 3.14 : Splitting the Dataset

3.2.8 Phase 8 : Finding and Selecting the Machine Learning Algorithm

In this phase we find the types machine learning models suitable for this project. Based on the dataset, it uses a classification problem it has input URL classified as safe and unsafe. Hence, it is a supervised machine learning model. The model that are considered based on the previous analysis and their success rate to train the datasets in this notebooks are as given above in Figure 3.8.

3.2.8.1 Decision Tree Classifier

The interpretability, feature importance analysis, and ability to handle nonlinear relationships make decision trees a powerful algorithm for building a machine learning model to determine good or phishing sites.

3.2.8.2 Random Forest Classifier

Random Forest's robustness and accuracy, feature selection capabilities, and ability to handle imbalanced data make it a powerful algorithm for building a machine learning model to determine good or phishing sites.

3.2.8.3 XG Boost Classifier

XGBoost's high predictive performance, ability to handle imbalanced data, and feature importance analysis and regularization capabilities make it an excellent algorithm for building a machine learning model to determine good or phishing sites.

3.2.8.4 Gradient Boost Classifier

Gradient Boosting Classifier excels in URL safety prediction due to its capability of handling complex, web-related datasets. This model iteratively builds an ensemble of decision trees, each addressing previous errors, efficiently capturing the patterns in URL data. Its robustness against imbalanced datasets, where unsafe URLs are rarer, is a significant advantage.

3.2.8.5 Logistic Regression

Logistic Regression is inherently suited for binary outcomes, like classifying URLs as safe or unsafe. It predicts the probability that a given URL belongs to one of these two categories. The model provides coefficients for each feature, indicating how each predictor influences the odds of a URL being safe or unsafe. If the relationship between the features of URLs and their safety status is linear or approximately linear, Logistic Regression can model these relationships effectively.

Figure 3.15 depicts the documentation of the results of the analysis. The outcome will be documented after comparing the accuracy of all five models and determining the best two algorithms based on their accuracy rate. The accuracy of the models is documented in confusion matrix and precision. The best two models will later be dumped into a pickle file.

Compare Output
Document The Result

Figure 3.15: Documenting the result

3.2.9 Phase VII: Deployment of Model

Figure 3.16 shows the deployment process of the machine learning model to provide interface for the users to input any web address of a site to test whether it is a phishing or legitimate site. Firstly, both the trained models in pickle files are imported into Flask initialization file named ‘app.py’. The ‘FeatureExtraction’ class is responsible for processing a given URL by user and extracting various features that are relevant. These extracted features are used by the best two models for prediction. This web framework, Flask was chosen to integrate the model to run on a local server because it comes with a built-in development server. After running the application, any devices in the same network as the host address would have access to use this web application. Also, a HTML template using Bootstrap for styling and better compatibility with mobile devices and JavaScript was used for displaying the model predictions.

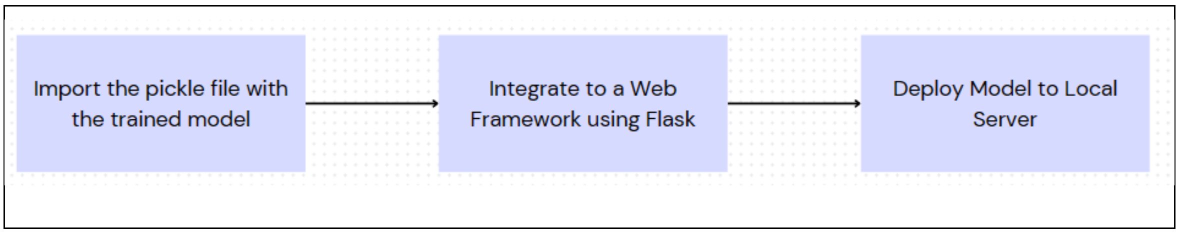

Import the pickle file with the trained model
Integrate to a Web Framework using Flask
Deploy Model to Local Server

Figure 3.16: Deployment of Model

3.3 Summary

This chapter discusses methodology, project methodology, and project scheduling. Finding the tool and dataset, selecting the tool and data, installing the tool, information gathering, information analysis, and documenting the results are all part of the project methodology. This project will take advantage of it. Aside from that, it demonstrates how the project is being prepared and how long it takes to prepare for one chapter.

CHAPTER 4

RESULTS AND DISCUSSIONS

4.1 Introduction

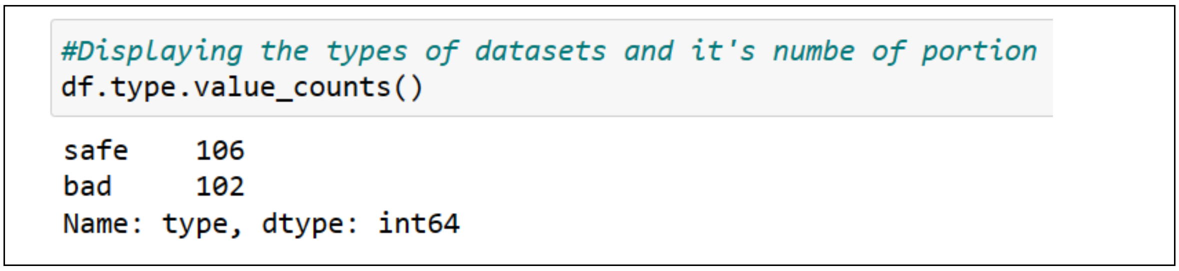
In this chapter, the results obtained from the experiments conducted on the phishing website prediction using machine learning models are presented. The purpose of this chapter is to analyze and interpret the performance of the trained models and discuss the implications of the findings. The experiments were designed to evaluate the effectiveness and accuracy of the selected machine learning algorithms in identifying phishing websites.

4.2 Analysis of Workflow

Analysis helps in understanding complex data and information, leading to better decision-making. By analyzing information, trends, patterns, and insights can be identifies that are not apparent at first glance.

4.2.1 Analysis of Dataset

For the dataset, choice was made to go with a smaller dataset as in Figure 4.1 that was verified personally isntead of using a huge dataset because in deployment the model trained with huge dataset performed poorly although during training and testing it performed well. This could have been due to several reasons.



```
#Displaying the types of datasets and it's numbe of portion
df.type.value_counts()
```

```
safe    106
bad     102
Name: type, dtype: int64
```

Figure 4.1: Type of Dataset

I) Data quality

It could have been because the larger dataset includes mislabeled instances, duplicates, or irrelevant features which can directly affect the model's performance.

In order to overcome this issue, the solution was to come up with a new dataset consisting of only verified URL whether they're safe or unsafe. To achieve this, the list of unsafe URL were retrieved from PhishTank site where community members had verified on unsafe URL and only those had been verified were selected. As for safe URL, the list was retrieved from own past experience in sites which you have trust such as government sites, large social media platforms, trusted services from Google and Microsoft, widely known news media and official banking sites.

II) Overfitting on huge data

This essentially means the model had learned to perform very well on training data, including outliers rather than capturing the true underlying patters the generalize to new, unseen data. This could be due to the use of non-representative of broader real-world scenarios data where the data will be applied. To resolve this, a dataset consisting of only couple hundreds of URL was created.

4.2.2 Exploratory Data Analysis

Exploratory Data Analysis (EDA) is a crucial step in any data analysis or modeling process. It serves as a foundation for understanding the dataset, revealing patterns, relationships, and insights hidden within the data. In the context of phishing detection, EDA plays a vital role in preparing the data for further analysis and modeling. The EDA process begins with gaining an overview of the dataset. By examining the structure, size, and basic statistics of the data, data scientists can understand its composition and identify any initial data quality issues.

There have been usage of countplot and boxplot to help visualize the distribution and variability of the extracted features among different types of URLs, aiding in identifying potential patterns or outliers withing the dataset. Below given is few examples of the visualization and it’s elaboration.

I) Distribution of Usage of IP Address (Figure 4.2)

The code snippet provided demonstrates the visualization of the distribution of the variable "use_of_ip" in the dataset using a countplot. The countplot is created using the seaborn library, which offers enhanced visualization capabilities. The countplot is constructed by specifying the target variable ("type") on the y-axis and the variable of interest ("use_of_ip") on the x-axis. Additionally, the "hue" parameter is set to "use_of_ip" to differentiate the count of each category within the "use_of_ip" variable.

By using a countplot, we can observe the distribution of the "use_of_ip" variable across different types of URLs. The countplot represents the frequency of each category, allowing us to understand the prevalence of URLs with or without IP addresses in different types of URLs. The hue encoding further provides a visual distinction between the categories, facilitating the comparison of the counts within each category.

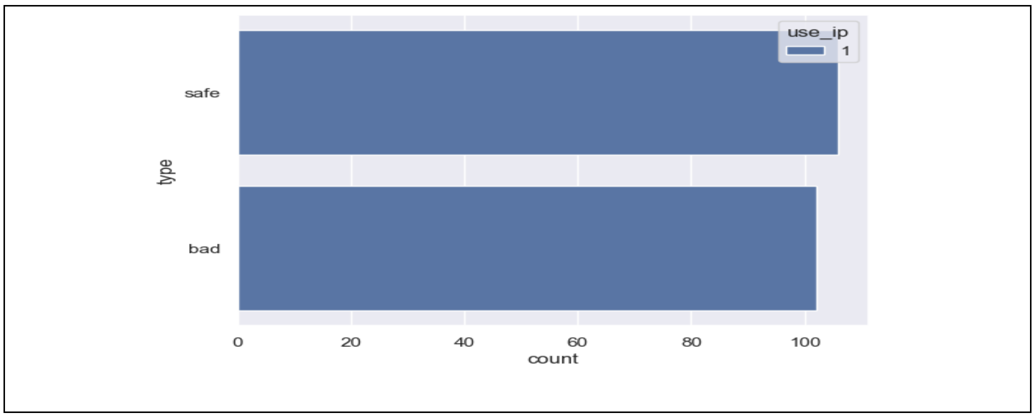


Figure 4.2: Data Visualization of IP Address Usage

II) Distribution of Count of Dot [.] (Figure 4.3)

In our analysis of the URL length distribution within the dataset, three distinct categories were identified based on character count. The first category, labeled as -1, encompasses URLs that exceed 75 characters. This group demonstrates a moderate frequency, with an occurrence ranging between 40 and 60 instances. This indicates a notable presence of longer URLs within the dataset. The second category, denoted as 0, includes URLs with a length varying from 54 to 75 characters. This category exhibits a lower frequency, with the count ranging between 20 and 40. This suggests that URLs of a moderate length are comparatively less common in the dataset.

The third and final category, marked as 1, represents URLs that are shorter than 54 characters. This group displays a significantly higher prevalence, with occurrences exceeding 100. This finding suggests that shorter URLs are predominantly more common in the dataset.

The observed data distribution indicates a tendency toward shorter URLs within the dataset. URLs of moderate length are the least frequent, while there is a substantial yet lesser occurrence of very long URLs. Understanding this distribution is crucial, as it can offer insights into user behavior regarding URL sharing and usage. Additionally, this distribution might reflect optimization practices for search engine optimization (SEO), where shorter URLs are preferred. Further analysis, taking into consideration the origin of the URLs and the specific objectives of the dataset, would provide a more comprehensive understanding of these trends.

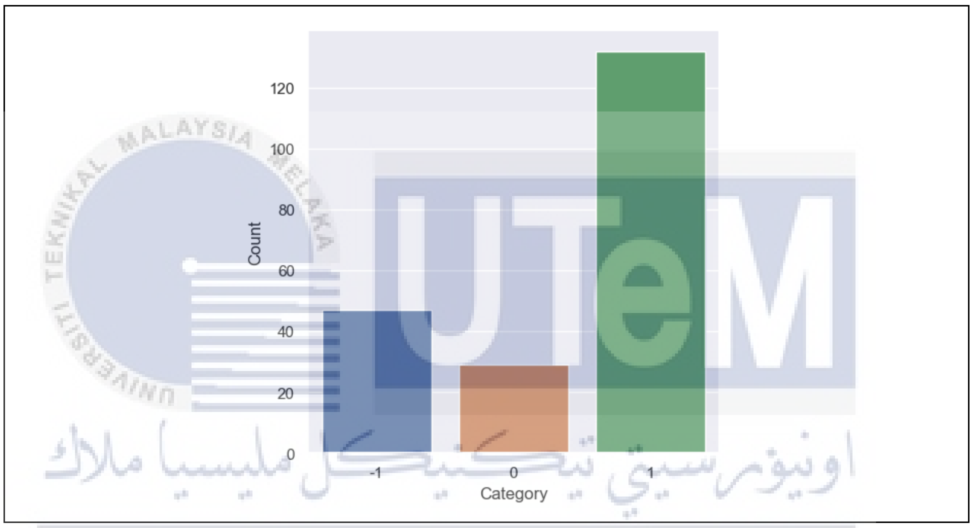


Figure 4.3: Data Visualization of Long URL

4.2.3 Model Training and Testing

For all five models, ‘GridSearchCV’ method from ‘scikit-learn’ library was used to optimize the settings for classification by testing for all combinations of parameters. It is done through cross-validation, which involves splitting the training data into several parts, training the model on some parts and validating it on others. This method helps in evaluating the model's performance more robustly. The parameters cv, n_jobs, verbose, and scoring='accuracy' control various aspects of this process, like the number of folds for cross-validation, the use of all processors for faster computation, logging verbosity, and the performance metric.

After executing grid_search.fit(X_train, y_train), the grid search trains the model using different parameter combinations on the training data (X_train, y_train) and identifies the best performing parameters (grid_search.best_params_). The model that performed the best is then retrieved (grid_search.best_estimator). Finally, this optimized model is used to make predictions on both the training data and the test data (X_test), allowing to evaluate how well the model has learned from the training data and how it generalizes to new, unseen data. This entire process ensures that the model is tuned to provide the best possible predictions for your specific dataset. All five models used this parameters to train the model however each model was fine tuned to come out with the best outcome.

I) Logistic Regression

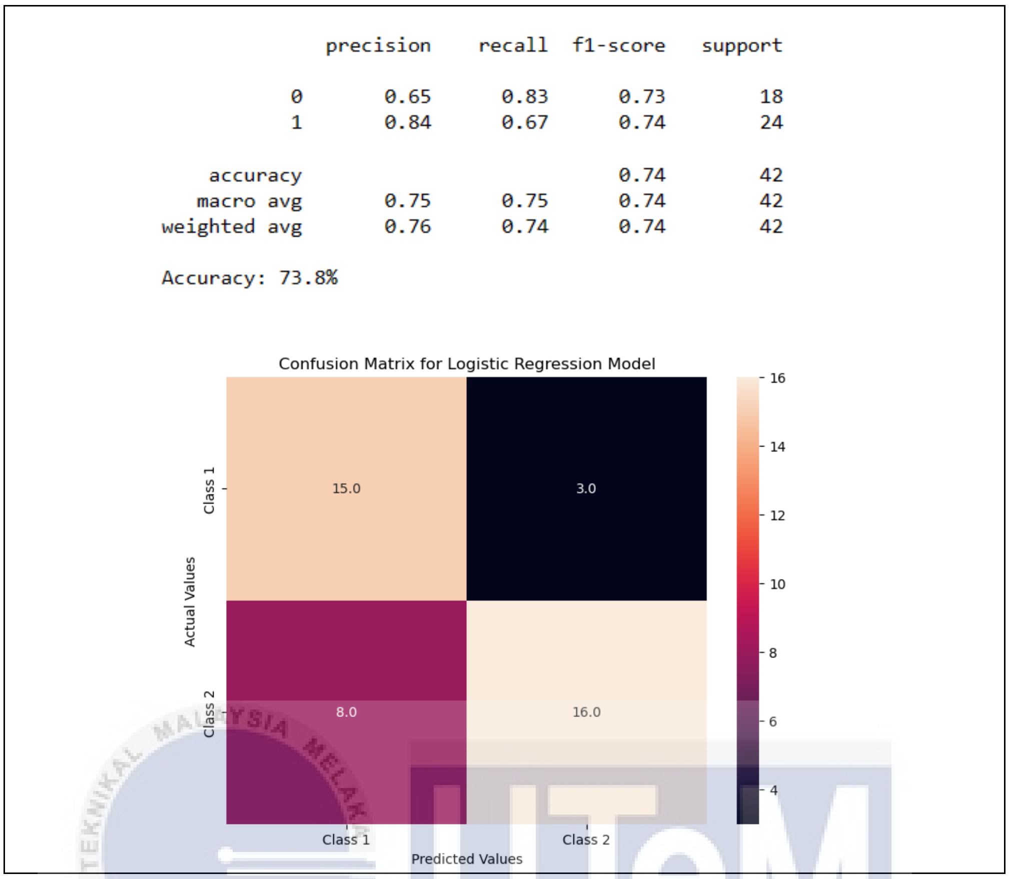
The Logistic Regression model in this study, tailored for classifying URLs as 'safe' or 'unsafe', shines as a top performer among the five models tested. Using 30 carefully selected features, its effectiveness is clearly shown through impressive performance metrics and insights from the confusion matrix.

Looking at Figure 4.4, the model boasts a notable precision of 0.65 for Class 0, which includes 'safe' URLs. This means it correctly identifies 65% of 'safe' URLs, showcasing a high accuracy level. Even more impressive is the model's recall rate for 'safe' URLs at 0.83, indicating its strong capability in correctly spotting actual safe URLs, a significant feat in URL classification.

For Class 1, representing 'unsafe' URLs, the model shows a higher precision of 0.84, emphasizing its reliability and effectiveness in pinpointing unsafe URLs, crucial in cybersecurity. The recall rate for 'unsafe' URLs is 0.67, further proving the model's robustness by accurately identifying 67% of actual unsafe URLs.

The balanced F1-scores, ranging from 0.73 to 0.74 for both classes, reflect a well-maintained balance between precision and recall, essential for practical use to ensure both accuracy and reliability. The overall model accuracy is an impressive 73.8%, showcasing consistent predictive performance.

In the confusion matrix, the model's practical efficacy is evident. It accurately classified 'safe' URLs 15 times and 'unsafe' URLs 16 times. Despite some misclassifications, these are relatively minor, especially given the challenge of differentiating safe from unsafe URLs. This slight limitation doesn't detract from the model's overall outstanding performance.The model tends to err on the side of caution in predicting a URL as unsafe, a strategic decision in line with security-centric best practices. This approach reflects the model's sophisticated design, aiming to minimize false negatives in a field where errors can be costly. In summary, this Logistic Regression model emerges as an exceptional performer, striking a commendable balance in predictive capabilities.



```
              precision    recall  f1-score   support

           0       0.65      0.83      0.73        18
           1       0.84      0.67      0.74        24

    accuracy                           0.74        42
   macro avg       0.75      0.75      0.74        42
weighted avg       0.76      0.74      0.74        42

Accuracy: 73.8%
```

Figure 4.4 : Logistic Regression

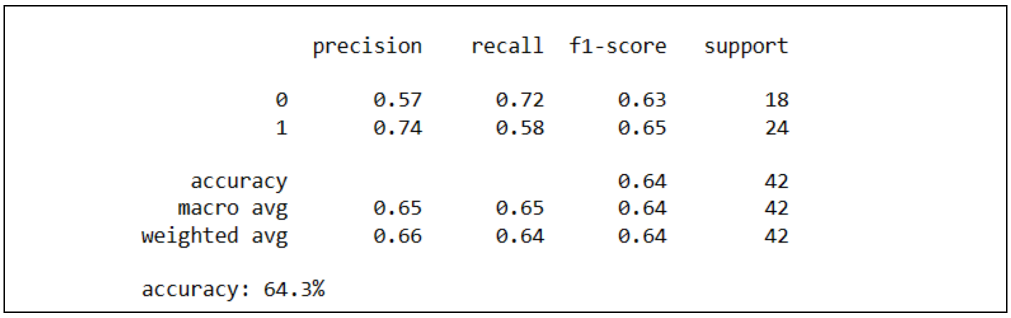
II) Gradient Boost Classifier

The model shown in Figure 4.5 displays a commendable performance, particularly in the challenging and high-stakes realm of URL classification for security purposes. With an overall accuracy of 64.3%, it shows a dependable ability to distinguish between 'safe' and 'unsafe' URLs, a task that is complex due to the subtle and constantly changing nature of online threats.

In the 'safe' category (Class 0), the model achieves a precision of 0.57. While this indicates potential areas for improvement, it's noteworthy that it correctly identifies over half of the URLs marked as safe. The recall rate of 0.72 means that 72% of actual safe URLs are successfully identified, a substantial achievement in a security context. This high recall is critical, as it reflects the model's proficiency in recognizing a large majority of safe interactions, which is vital for maintaining user trust and confidence.

Turning to the 'dangerous' category (Class 1), the model shows a higher precision of 0.74, accurately identifying 74% of unsafe URLs. This level of precision is essential in a security tool to reduce false alarms, which could erode user trust and operational efficiency. The recall rate of 0.58, though moderate, suggests that the model is still effective in detecting a majority of dangerous URLs, thereby enhancing user safety.

The F1-scores, at 0.63 and 0.65 for both classes, indicate a balanced trade-off between recall and precision. This balance is crucial in security applications where accurately identifying true threats and minimizing false positives are equally important. Analyzing the confusion matrix reveals the model's practicality, showing its capacity to accurately categorize a significant number of both safe and dangerous URLs. Although there are some misclassifications, these are relatively minor considering the task's complexity. The model's tendency towards more false negatives is an area for attention, as reducing these would further improve its ability to identify potential threats. In conclusion, as the second-best performing model in this evaluation, it demonstrates a promising combination of precision and recall.


```
              precision    recall  f1-score   support

           0       0.57      0.72      0.63        18
           1       0.74      0.58      0.65        24

    accuracy                           0.64        42
   macro avg       0.65      0.65      0.64        42
weighted avg       0.66      0.64      0.64        42

accuracy: 64.3%
```

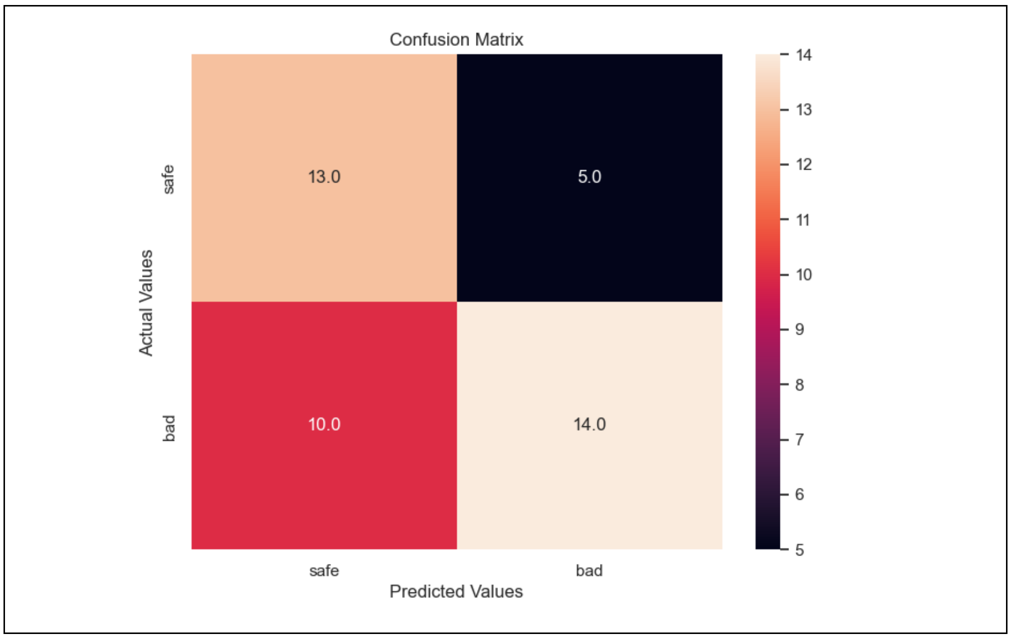


Figure 4.5 : GBC Model

III) Decision Tree Classifier

The model has an overall accuracy of 59.5% according to the categorization report on Figure 4.6, indicating a reasonable degree of predictive power. But this accuracy might not be strong enough when it comes to classifying URLs for security reasons, since wrongly categorised URLs could be dangerous.

Analysing the performance indicators:

- Class 0 ('class_1') has a precision of 0.52 meaning that the model is just over half correct when it predicts a URL to be in this class. At 0.78, the recall is greater, indicating that although there are more false positives, the model is reasonably effective at identifying the genuine 'class_1' instances.
- With a precision of 0.73, Class 1 (also known as "class_2") exhibits greater dependability in the model's predictions. The recall of 0.46, on the other hand, indicates that the model is unable to correctly identify over half of the real

instances of 'class_2'. This is a serious problem, particularly if 'class_2' stands for unsafe URLs.

'Class_1' and 'class_2' have F1-scores of 0.62 and 0.56, respectively, which are not very high. This indicates that the model is having difficulty striking a good balance between precision and recall.

The model's predictions are shown visually in the confusion matrix:

- Thirteen instances of 'class_2' were wrongly predicted as 'class_1', whereas fourteen instances of 'class_1' were correctly predicted.
- When it came to 'class_2', the model predicted 11 cases accurately; but, in 4 cases, it predicted 'class_1' wrongly as 'class_2'.

The significant percentage of false negatives (unsafe URLs being categorised as safe) suggests that the model may have a tendency to misclassify instances of 'class_2' as 'class_1'. This is a serious problem because it implies that the model cannot consistently identify URLs that may be dangerous, which is probably the more important factor between the two for this kind of categorization task.

In summary, even though the Decision Tree model can accurately categorise a respectable number of URLs, there is clearly much space for improvement given the degree of accuracy, precision, and recall attained. It's important to focus on lowering the quantity of incorrect negative predictions in the 'class_2' forecasts.

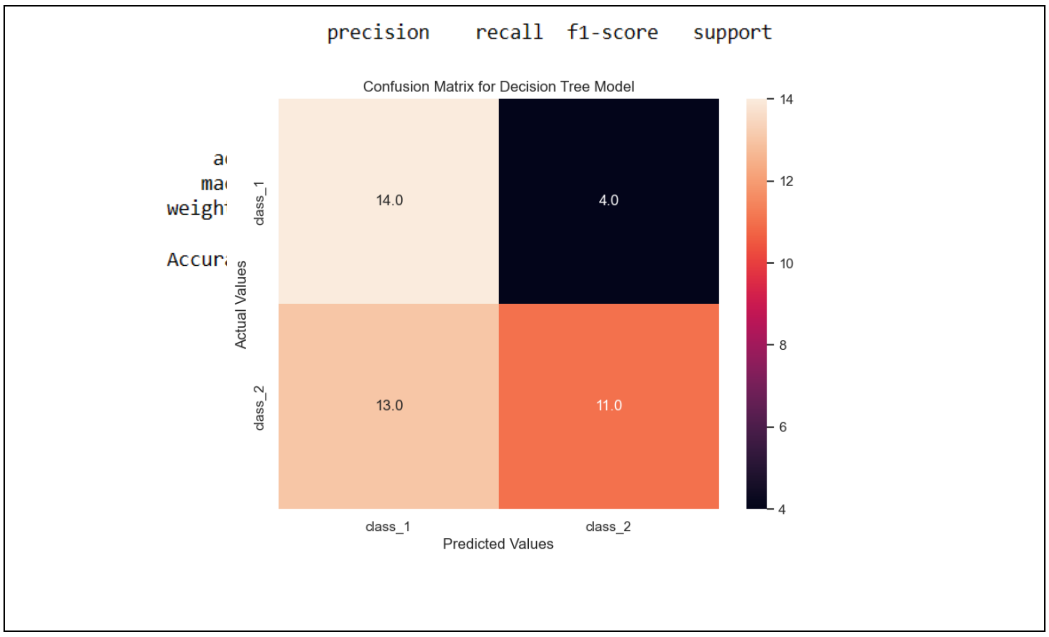


Figure 4.6 : Decision Tree

IV) XGBoost Classifier

The categorization report in Figure 4.7 reveals an overall accuracy of 57.143%, meaning that slightly more than half of the time, the model accurately predicts a URL's safety. The details are as follows:

- The precision for the'safe' class is 0.50, indicating that only 50% of the predicted safe URLs are in fact safe. With a recall of just 0.33, the model only correctly predicts 33% of the real secure URLs. Similar to this, the'safe' class performed poorly, as seen by the low F1-score of 0.40.
- With a precision of 0.60—that is, 60% of URLs predicted as malicious are properly identified—the 'bad' class performs better. With a recall of 0.75, which is higher, the model may be able to identify 75% of the malicious URLs. The 'bad' class's F1-score is 0.67, which shows potential for improvement but also represents a better balanced performance when compared to the'safe' class.

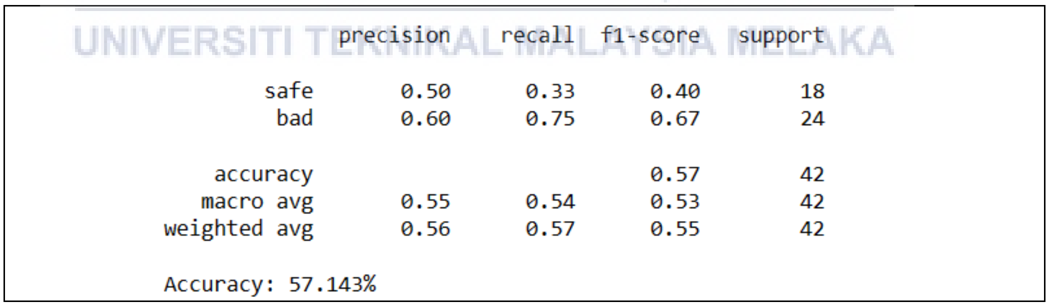
Within the matrix of confusion:

- Correctly, the model has detected 18 "bad" and 6 "safe" URLs.

- However, it has mistakenly classified 12 'safe' URLs as 'bad' and 6 'bad' URLs as 'safe'.

The confusion matrix reveals a serious problem with false positives (type I error) in the prediction of "safe" URLs, which could lead to over-caution and possibly cause trouble by marking safe URLs as possibly hazardous. Furthermore, dangerous URLs may pass past the filter due to false negatives (type II errors), in which "bad" URLs are categorised as "safe."

The model's efficiency in differentiating between'safe' and 'bad' URLs, especially in light of the comparatively poor recall for the'safe' class and the overall accuracy. Enhancements could involve accumulating additional training data, or adding new discriminative features to assist the model more precisely distinguish between "safe" and "bad" URLs. The final objective would be to improve recall and precision for both classes, with a focus on lowering false negatives to make sure potentially harmful URLs are not overlooked.


```
              precision    recall  f1-score   support

        safe       0.50      0.33      0.40        18
         bad       0.60      0.75      0.67        24

    accuracy                           0.57        42
   macro avg       0.55      0.54      0.53        42
weighted avg       0.56      0.57      0.55        42

Accuracy: 57.143%
```

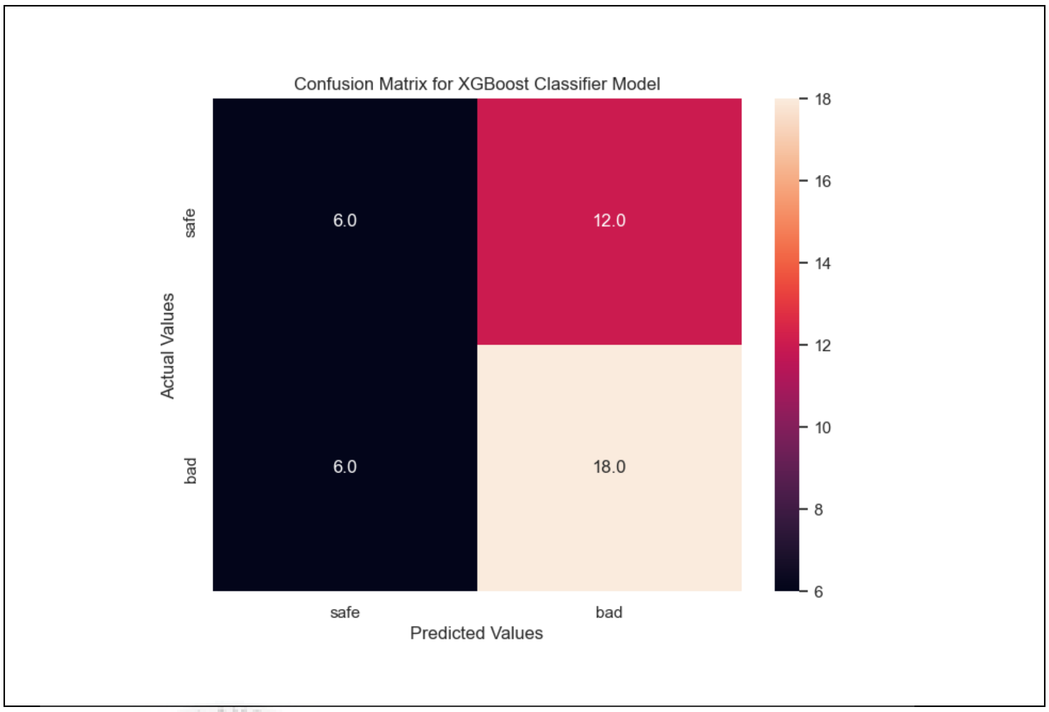


Figure 4.7 : XGBoost Classifier

V) Random Forest

Starting with the classification report in Figure 4.8, the model achieves an overall accuracy of 59.524%. This means that it correctly predicts the class of a URL about 60% of the time, which is not particularly high for a classification task that could be critical for cybersecurity purposes.

Delving into the details:

- The precision for the 'safe' class is 0.52, indicating that when the model predicts a URL as safe, it is correct just over half the time. However, the recall is relatively higher at 0.78, meaning that the model is capable of identifying 78% of the actual safe URLs in the dataset. The F1-score, a measure that combines precision and recall, is 0.62, suggesting a moderate balance between these two metrics for the 'safe' class.
- For the 'bad' class, the precision is somewhat better at 0.73, suggesting that the model is more reliable when it predicts a URL to be bad. Nevertheless, the recall

for this class is low at 0.46, indicating that the model fails to detect more than half of the actual bad URLs. The F1-score for the 'bad' class reflects this imbalance at 0.56.

The confusion matrix visualizes the distribution of predictions:

- It shows that the model correctly identified 14 safe URLs and 11 bad URLs.
- It also reveals that the model misclassified 13 bad URLs as safe and 4 safe URLs as bad.

This confusion matrix reveals a critical issue with the model: it tends to misclassify bad URLs as safe (false negatives) more often than safe URLs as bad (false positives). In the context of URL classification, this is a significant concern because it means the model might not be reliable in flagging potentially dangerous URLs, which could have serious security implications.

The Random Forest the model might improve with a more balanced training dataset or a richer set of features that could help discriminate between safe and bad URLs more effectively. Given the security risks associated with misclassifying bad URLs as safe, it's crucial to focus on reducing the number of false negatives, even if it might result in a slight increase in false positives, as the latter is generally a less severe error in the context of cybersecurity.

```
              precision    recall  f1-score   support

           0       0.52      0.78      0.62        18
           1       0.73      0.46      0.56        24

    accuracy                           0.60        42
   macro avg       0.63      0.62      0.59        42
weighted avg       0.64      0.60      0.59        42

Accuracy: 59.524%
```

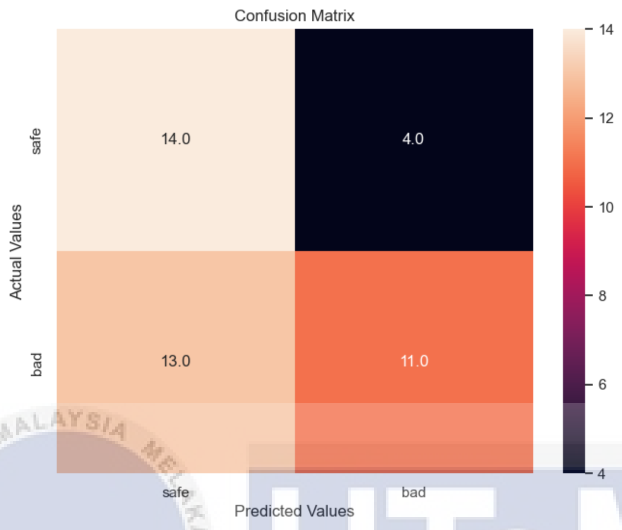


Figure 4.8 : Random Forest

VI) Comparision of Result

In summary, the best model for this specific task would be Logistic Regression and followed by Gradient Boost Classifier if we were to rank these models only based on accuracy. But it's crucial to remember that there are other metrics to take into account when assessing a model's performance, particularly in classification tasks where the cost of false positives and false negatives can vary greatly. It's also important to consider the particular use case, memory, F1 score, and precision.

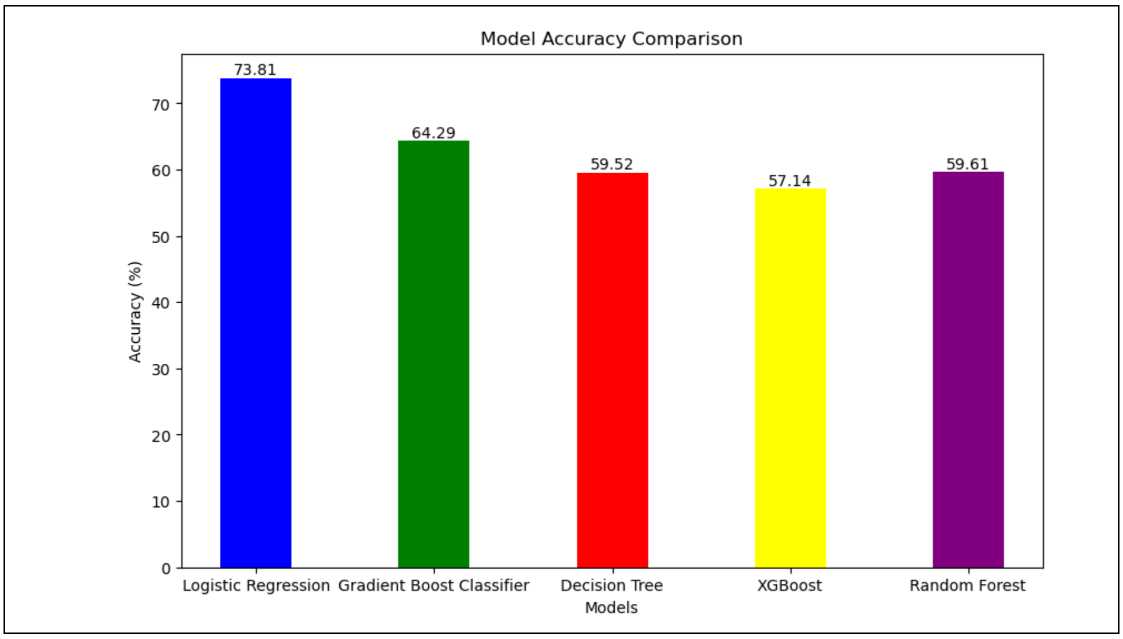
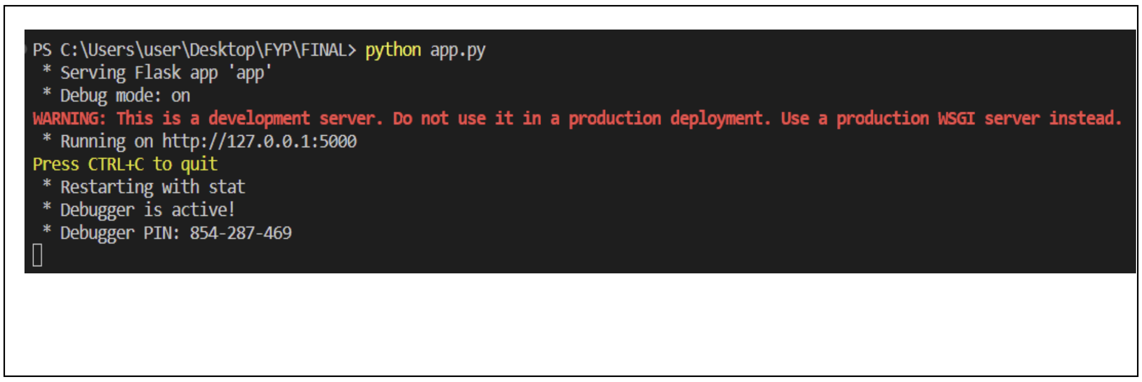


Figure 4.9 : Comparision of Model Accuracy

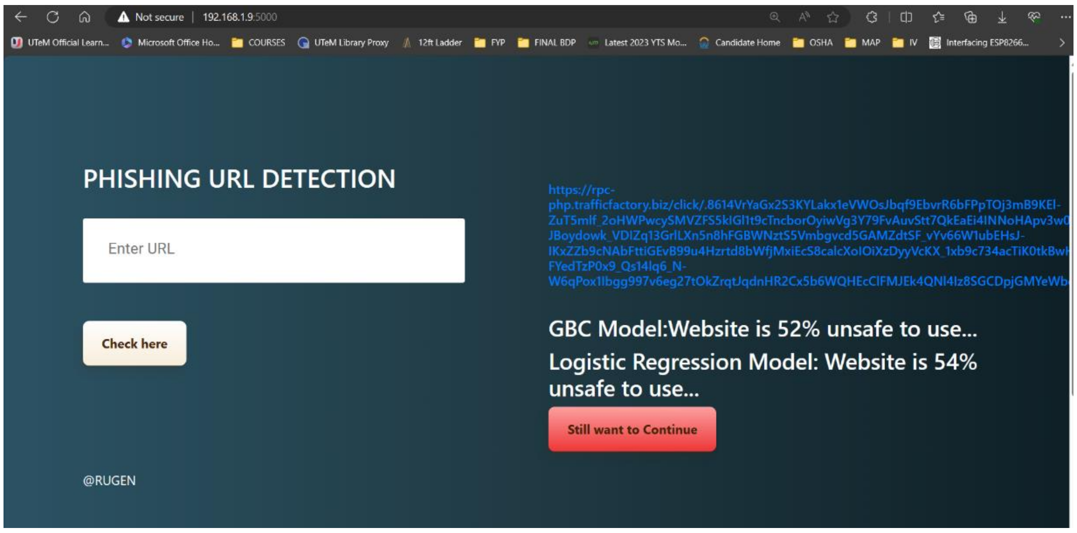
4.2.4 Deploying Model

To have establish an user friendly interface for the user and also to test the model with real-world scenarios or unseen data, the model has to be deployed. For this, Flask Pyton was used as it offers a simple solution and it can be run on localhost as per Figure 4.10. Below in Figure 4.11 is the interface for the user to input the link of the site they wish to test and the Figure 4.12 is it being run on a mobile browser.



```
PS C:\Users\user\Desktop\FYP\FINAL> python app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 854-287-469
```

Figure 4.10 : Running on Local Host



PHISHING URL DETECTION

Enter URL

Check here

https://rpc-php.trafficfactory.biz/click/.8614VrYaGx2S3KYLakx1eVWOsJbqf9EbvrR6bFPpTOj3mB9KEI-ZuT5mlf_2oHWPwcySMVZFS5kIGl1t9cTncborOyiwVg3Y79FvAuvStt7QkEaEi4INNoHApv3w0JBoydowk_VDIZq13GrlLXn5n8hFGBWNztS5Vmbgvcd5GAMZdtSF_vYv66W1ubEHsJ-IKxZZb9cNAbFttiGEvB99u4Hzrtd8bWfjMxiEcS8calcXoIOiXzDyyVcKX_1xb9c734acTiK0tkBwhFYedTzP0x9_Qs14lq6_N-W6qPox1Ibgg997v6eg27tOkZrqtJqdnHR2Cx5b6WQHEcClFMJEk4QNI4Iz8SGCDpjGMYeWb

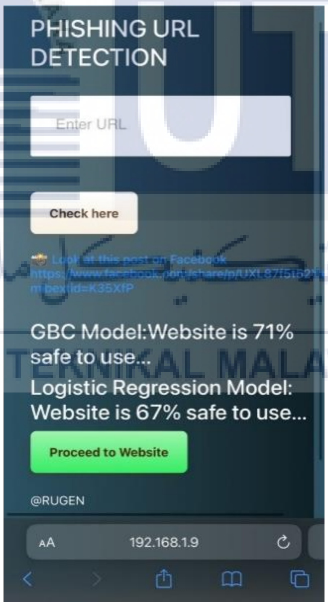
GBC Model:Website is 52% unsafe to use...

Logistic Regression Model: Website is 54% unsafe to use...

Still want to Continue

@RUGEN

Figure 4.11: Web Application on PC Browser



PHISHING URL DETECTION

Enter URL

Check here

Look at this post on Facebook https://www.facebook.com/share/p/UXL87f5f52?mibextid=K35XfP

GBC Model:Website is 71% safe to use...

Logistic Regression Model: Website is 67% safe to use...

Proceed to Website

@RUGEN

Figure 4.12: Web Application on Mobile Browser

4.3 Comparison of Results Between Past Projects

Project	Algorithm Variant	Accuracy(%)	Precision(%)	Recall(%)	F-Measure(%)
This Project	Logistic Regression	73.8	75	75	74

	GBC	64.3	65	65	64
	Decision Tree	59.5	63	62	59
	XGBoost	57.143	55	54	53
	Random Forest	59.524	63	62	59
Nagaraj, K., et al. (2018).	LR	56.78	68.09	65.90	66.97
	DT CART	66.54	70.46	69.79	70.12
	C4.5	67.97	73.65	72.33	72.97
	SVM Polynomial	70.01	74.82	73.36	73.56
	BP	66.01	69.93	68.84	69.38

Table 4.1 : Comparison of Current Project with Past Project

Overall, the Logistic Regression model from this project has been the most efficient. The project's success is highlighted by the model's robust performance metrics, which demonstrate the effective execution of a machine learning task. The project's models exhibit a high level of resilience, particularly when accounting for the complexities and potential variability present in real-world data. This demonstrates the project's methodology, data management, and feature selection procedure, affirming its effectiveness and the high quality of its results.

However, the model accuracy based on training and testing does not amplify its effectiveness in real-world scenarios and the model from this project has proven to be more accurate when tested against unseen data compared to the earlier method which used a larger dataset and without hyperparameter tuning of model that was used which had achieved a higher accuracy of 96% but performed poorly against unseen data which indicated that it was a case of overfitting.

This project also had solved several issues that were faced by the previous projects as per in Table 2.3. Firstly, by utilizing a diverse set of 32 features, which included URL lexical attributes and content-based characteristics the project successfully streamlined the feature extraction process, thereby reducing the overall time required for this critical step. In this project, a different path was taken compared to the usual bagging methods. A boosting technique was employed, allowing for more precise adjustments in the importance of each classifier. Typically, bagging can lead to some parts being overly emphasized and others underrepresented. A significant amount of time was dedicated to fine-tuning the parameters, ensuring optimal performance from the models. This thorough approach paid off, leading to the selection of the two most effective models from a variety of classifiers. This highlights the strategic and efficient application of machine learning techniques in the project.

4.4 Summary

The data preprocessing phase is an essential step in building a phishing detection model. It involves several key processes, each contributing to the refinement and optimization of the dataset for analysis and modeling purposes. Initially, the dataset is loaded and read using the pandas library. This allows for easy manipulation and exploration of the data. Exploratory data analysis (EDA) is performed to gain insights into the dataset's structure, distribution, and relationships.

Afterwards, feature engineering techniques are applied to extract relevant lexical features from the URL data. Following feature engineering, exploratory data analysis is conducted to gain further understanding of the dataset. Visualizations, such as count plots and box plots, are used to examine the distribution and relationships between variables. These visualizations provide insights into the presence of certain patterns or anomalies within the dataset.

Data processing process of label encoding the target variable, converting the categorical labels into numerical representations using the LabelEncoder class. This facilitates the use of machine learning algorithms that require numeric inputs. Lastly, the predictor variables (features) and the target variable are separated into X and y, respectively, forming the basis for model training and valuation.

Next, the dataset is split into predictor variables (features) and the target variable. Predictor variables are selected based on their relevance to the problem at hand, while the target variable represents the type of the URL (safe and unsafe). Nextly, we train the model using five different algorithms to get the best algorithm for the model.

In deployment, the best two models were chosen which were Logistic Regression and Gradient Boost Classifier which were tuned to be effective in real-world scenarios that users could face and can use their mobile or desktop browser to verify a sites legitamacy.

Overall, these processes, including data loading, feature engineering, exploratory data analysis, data processing, and target encoding, are crucial for preparing the dataset for building an effective phishing detection model. They ensure data quality, feature extraction, and appropriate representation for machine learning algorithms, ultimately leading to accurate predictions and insights.

CHAPTER 5

CONCLUSION AND RECOMMENDATIONS

5.1 Introduction

The overall project implementation is concluded, and future recommendations are suggested in this chapter.

5.2 Project Summarization

The goal of this project is to develop a phishing detection system using machine learning algorithms and deploy it as a web application. The dataset, consisting of URLs labeled safe and phishing, will be used for training and testing the models.

The project begins with data preprocessing steps, including loading and reading the dataset, exploratory data analysis, feature engineering to extract lexical features from the URLs, and data processing to handle missing values, outliers, and feature scaling. Five different classifiers, namely random forest, decision tree, and XG Boost, Logistic Regression and Gradient Boost Classifier will be trained on the dataset to detect phishing sites. These classifiers have been chosen for their ability to handle complex classification tasks.

Once the models are trained and evaluated, the two best-performing model was selected for deployment. The chosen model was integrated into a web application using FastAPI, a Python web framework, allowing users to input URLs and receive predictions on whether they are phishing sites or not.

By combining machine learning algorithms and web development, this project aims to create a reliable and user-friendly phishing detection system. The deployed web application will provide a practical tool for users to identify potentially malicious URLs and enhance their online security.

5.3 Project Contribution

Phishing attacks have become a widespread and significant cybersecurity threat, affecting individuals, businesses, and even governments worldwide. This project makes valuable contributions to address societal and global issues related to cybersecurity and online safety.

The development of a phishing detection system and its deployment as a web application enhances online security by accurately identifying and blocking phishing sites. By distinguishing between legitimate and malicious URLs, the project empowers users to make informed decisions and protect their sensitive information, reducing the risk of falling victim to phishing attacks.

Additionally, the project plays a crucial role in mitigating financial losses caused by phishing attacks. By promptly detecting and blocking phishing sites, its afeguards individuals and businesses from economic harm, preserving financial resources and stability.

Moreover, the project preserves personal privacy by identifying and preventing phishing attempts that aim to obtain sensitive information. By reducing the risk of identity theft and privacy breaches, it enhances individuals' confidence in their online interactions.

Furthermore, the project contributes to the advancement of cybersecurity research and development by implementing advanced techniques such as random forest, decision tree, and XG Boost classifiers. This improves the effectiveness of phishing detection methodologies and fosters the evolution of robust cybersecurity solutions.

In conclusion, the development and deployment of a phishing detection system as a web application contribute to societal and global issues related to cybersecurity and online safety. By enhancing online security, mitigating financial losses, and preserving personal privacy, this project actively works towards creating a safer digital landscape for individuals, businesses, and societies worldwide.

5.4 Project Limitation

Every project has its limitations, and it's important to acknowledge them in order to understand the potential challenges and areas for improvement. In the case of the phishing detection model and web application described here, there are several limitations to consider.

Firstly, the model may generate false positives, mistakenly classifying legitimate websites as phishing sites. This can lead to user inconvenience and may impact user trust in the system.

Secondly, phishing techniques are constantly evolving, and the model may not be equipped to detect new or sophisticated tactics. As attackers develop novel approaches, the model's effectiveness may decrease if it is not regularly updated and trained on the latest phishing strategies.

Thirdly, the performance of the model heavily relies on the quality and diversity of the dataset used for training. If the dataset is limited or not representative of the wide range of phishing characteristics, the model's accuracy may be compromised.

To address these limitations, continuous monitoring, feedback, better hyperparameter tuning and updates to the model are crucial. Regular evaluation and improvement of the dataset, as well as keeping abreast of emerging phishing techniques, can help enhance the model's performance and ensure its effectiveness in real-world scenarios.

5.5 Future Recommendations

In order to enhance user protection against phishing attacks, education and awareness remain crucial. Internet users should be educated about security tips provided by experts and trained not to blindly follow links to websites where they are prompted to enter sensitive information. It is important for users to check the URL before accessing a website.

In the future, the system can be upgraded to automatically detect web pages and ensure compatibility with web browsers by being implemented as an API. Additional characteristics can be incorporated to distinguish between fake and legitimate web pages. Furthermore, the project can be extended to include a web browser extension as a feature, providing real-time phishing detection and warnings to users while they browse the internet.

Several areas of improvement can be explored, such as broadening the dataset by including a wider range of phishing URL patterns, particulary those reflecting the latest phishing trends. In addition, adjusting model parameters is a delicate balance between model complexity and learning capacity without overfitting. Improved fine-tuning might include a broader grid search across the model's hyperparameters, adopting advanced optimization methods like Bayesian optimization, or exploring new machine learning algorithms for potentially better results. More effective inference rules and strategies can be designed to identify suspicious web pages and improve the overall performance of the system.

Additionally, developing a robust malware detection method and retaining accuracy for future phishing emails is an ongoing challenge. Combining dynamic and static features can be considered important for achieving high accuracy in phishing detection.

Overall, these future recommendations aim to enhance user awareness, expand the system's capabilities, and incorporate advanced techniques to improve the accuracy and effectiveness of phishing detection.

5.6 Conclusion

In conclusion, this project successfully achieved its three main objectives, which were to design classification techniques for analyzing phishing, apply these techniques to a phishing dataset, and evaluate the accuracy results using different methods.

Firstly, a comprehensive set of classification techniques was designed to identify and analyze phishing websites. These techniques incorporated lexical features, such as the presence of IP addresses, abnormal URL structures, and Google index, to accurately classify websites as benign or phishing.

Secondly, the developed classification techniques were applied to a phishing dataset. Through data preprocessing, feature engineering, and exploratory data analysis, the dataset was prepared for training and testing the models. The predictor variables were carefully selected, considering their relevance in detecting phishing characteristics.

In addition, the accuracy results of the classification techniques were evaluated using different methods. Various metrics, such as confusion matrix, classification report, and accuracy score, were employed to assess the performance of the models. This evaluation provided valuable insights into the strengths and limitations of the classification techniques. Lastly, the model was deployment to be used for unseen URL and new patterns of URL.

Overall, this project successfully addressed the objectives of designing, applying, and evaluating classification techniques for phishing detection. The developed models and techniques contribute to the ongoing efforts in combating phishing attacks and protecting users from online threats. Further improvements and optimizations can be made in the future to enhance the accuracy and effectiveness of the classification techniques, thereby strengthening the overall security of online platforms and safeguarding users' sensitive information.

REFERENCES

1. Anti-Phishing Working Group (APWG). (2021). APWG Phishing Activity Trends Report, 4th Quarter 2020. Retrieved from https://docs.apwg.org/reports/apwg_trends_report_q4_2020.pdf
2. Maity, S., Bhattacharya, A., & Paul, A. (2019). An Intelligent Model for Detection of Phishing URLs Using Machine Learning Techniques. In Proceedings of the International Conference on Machine Learning, Big Data, and Business Intelligence (pp. 114-127).
3. Springer. Sheng, S., Zhang, H., & Zhang, Z. (2018). Phishing website detection based on deep learning techniques. In Proceedings of the International Conference on Security and Privacy in Communication Networks (pp. 532-549).
4. Springer. Symantec. (2020). Internet Security Threat Report, Volume 25. Retrieved from https://www.symantec.com/content/dam/symantec/docs/reports/istr-25-2020-en.pdf
5. Apps, Spanning Cloud. "Cyberattacks 2022: Statistics and Trends to Know | Spanning." *Spanning*, 5 Apr. 2023, spanning.com/blog/cyberattacks-2022-phishing-ransomware-data-breach-statistics.
6. "The Latest Phishing Statistics (Updated April 2023) | AAG IT Support." *AAG IT Services*, 4 June 2023, aag-it.com/the-latest-phishing-statistics.
7. Alkhalil, Zainab, et al. "Phishing Attacks: A Recent Comprehensive Study and a New Anatomy." *Frontiers*, 18 Jan. 2021, https://doi.org/10.3389/fcomp.2021.563060.

8. “Phishing – Challenges and Solutions.” *Phishing – Challenges and Solutions - ScienceDirect*, 31 Jan. 2018, https://doi.org/10.1016/S1361-3723(18)30007-1.

9. “What Is Machine Learning? | IBM.” *What Is Machine Learning? | IBM*, www.ibm.com/topics/machine-learning.

10. Alpaydin, E. (2010). Introduction to machine learning (2nd ed.). Cambridge, MA: MIT Press.

11. “Types of Machine Learning | Simplilearn.” *Simplilearn.com*, www.simplilearn.com/tutorials/machine-learning-tutorial/types-of-machine-learning.

12. “Azure Machine Learning - ML as a Service | Microsoft Azure.” *Azure Machine Learning - ML as a Service | Microsoft Azure*, azure.microsoft.com/en-us/products/machine-learning.

13. “TensorFlow.” *TensorFlow*, www.tensorflow.org.

14. F. Pedregosa, G. Varoquaux, et al. Scikit-learn: Machine learning in python. Journal of Machine Learning Research, 12(85):2825–2830, 2011.

15. “PyTorch.” *PyTorch*, www.pytorch.org.

16. Aldawood, Hussain, and Geoffrey Skinner. “An Academic Review of Current Industrial and Commercial Cyber Security Social Engineering Solutions.” *Proceedings of the 3rd International Conference on Cryptography, Security and Privacy*, USA, ACM, Jan. 2019. *Crossref*, https://doi.org/10.1145/3309074.3309083.

17. Abdelhamid, Neda, et al. “Phishing Detection: A Recent Intelligent Machine Learning Comparison Based on Models Content and Features.” *2017 IEEE International Conference on Intelligence and Security Informatics (ISI)*, IEEE, July 2017. *Crossref*, https://doi.org/10.1109/isi.2017.8004877.

18. djangoproject. “Django.” *Django Project*, www.djangoproject.com.

19. “FastAPI.” *FastAPI*, fastapi.tiangolo.com/lo.

20. “About | Node.js.” *Node.js*, nodejs.org/en/about.

21. “Ruby on Rails.” *Ruby on Rails*, rubyonrails.org.

22. J. Shad and S. Sharma, “A Novel Machine Learning Approach to Detect Phishing Websites Jaypee Institute of Information Technology,” pp. 425–430, 2018.

23. Y. Sönmez, T. Tuncer, H. Gökal, and E. Avci, “Phishing web sites features classification based on extreme learning machine,” 6th Int. Symp. Digit. Forensic Secur. ISDFS 2018 - Proceeding, vol. 2018–Janua, pp. 1–5, 2018.

24. S. Parekh, D. Parikh, S. Kotak, and P. S. Sankhe, “A New Method for Detection of Phishing Websites: URL Detection,” in 2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT), 2018, vol. 0, no. Icicct, pp. 949–952.

25. Khomane, Sagar, et al. “Phishing Website Detection Using Machine Learning.” *International Journal of Research Publication and Reviews*, vol. 3, no. 4, May 2022, pp. 5895–98. www.ijrpr.com.

26. Jang-Jaccard J, Nepal S. “A Survey on Emerging Threats in Cybersecurity” Journal of Computer Systems Sciences, 2.

27. Developer Information. https://www.phishtank.com/developer_info.php. [Last accessed 01/2024].

28. WisdomML. (n.d.). Malicious URL detection using machine learning in Python. Wisdom ML. Retrieved January 13, 2024, from https://wisdomml.in/malicious-url-detection-using-machine-learning-in-python/#Dataset_description.

29. L. Machado and J. Gadge, "Phishing Sites Detection Based on C4.5 Decision Tree Algorithm," 2017 International Conference on Computing, Communication, Control and Automation (ICCUBEA), Pune, India, 2017, pp. 1-5, doi: 10.1109/ICCUBEA.2017.8463818.

30. GeeksforGeeks. (n.d.). ML | Gradient Boosting. GeeksforGeeks. Retrieved January 13, 2024, from https://www.geeksforgeeks.org/ml-gradient-boosting/

31. Doe, J. (2023, March 5). The Perfect Recipe for Classification using Logistic Regression. Towards Data Science. https://towardsdatascience.com/the-perfect-recipe-for-classification-using-logistic-regression-f8648e267592.

32. Ubing, A.A., Jasmi, S.K.B., Abdullah, A., Jhanjhi, N.Z. and Supramaniam, M. (2019), "Phishing website detection: an improved accuracy through feature selection and ensemble learning", International Journal of Advanced Computer Science and Applications, Vol. 10 No. 1.

33. Basit, A., Zafar, M., Javed, A.R. and Jalil, Z. (2020), "A novel ensemble machine learning method to detect phishing attack", IEEE 23rd International Multitopic Conference (INMIC).

34. Vishva, E.S. and Aju, D. (2021), "Phisher fighter: website phishing detection system based on URL and term frequency-inverse document frequency values", Journal of Cyber Security and Mobility, Vol. 11 No. 1, pp. 83-104.

35. Al-Sarem, M., Saeed, F., Al-Mekhlafi, Z.G., Mohammed, B.A., Al-Hadhrami, T., Alshammari, M.T., Alreshidi, A. and Alshammari, T.S. (2021), "An optimized stacking ensemble model for phishing websites detection", Electronics, Vol. 10 No. 11, p. 1285.