PHISHING WEBSITES DETECTION USING MACHINE LEARNING

A Project Report
Submitted in partial fulfillment of the
Requirements for the award of the Degree

MASTER OF COMPUTER APPLICATIONS

By

K. Sindhu priya
228W1F0056

Under the Esteemed Guidance of
Mrs. M. Prasanna Lakshmi
Assistant Professor

DEPARTMENT OF COMPUTER APPLICATIONS
V.R. SIDDHARTHA ENGINEERING COLLEGE
(Autonomous)
VIJAYAWADA-520007
ANDHRA PRADESH
2023

V.R. SIDDHARTHA ENGINEERING COLLEGE

(Autonomous)

KANURU, VIJAYAWADA-520007

DEPARTMENT OF COMPUTER APPLICATIONS

CERTIFICATE

This is to certify that the project entitled, **"Phishing Website Detection Using Machine Learning"**, is bonafied work of **K. SINDHU PRIYA** bearing Regd.No: (**228W1F0056**) submitted in partial fulfillment of the requirements for the award of degree of **MASTER OF COMPUTER APPLICATIONS** from JNTUK, Kakinada.

Internal Guide **Head of the Department**

External Examiner

Abstract

Phishing remains a significant threat, costing internet users billions of dollars annually. This nefarious practice involves the use of deceptive tactics by identity thieves to trick unsuspecting individuals into divulging their personal information. Phishers typically employ tactics such as spoofed emails and specialized phishing software to pilfer sensitive details, including usernames and passwords from financial accounts. This paper focuses on the development of a robust system for detecting phishing websites through the application of machine learning techniques. Specifically, our approach is grounded in supervised learning, and we have chosen the Random Forest algorithm due to its outstanding classification performance.

Our primary objective is to achieve the highest possible classification accuracy by carefully studying the distinguishing features of both benign and phishing URLs. Through a meticulous analysis, we aim to identify the most effective combination of these features to train our classifier. As a result of our efforts, we have achieved an impressive accuracy of 97% using the Random Forest algorithm, while the Logistic Regression method also yielded a respectable accuracy rate of 92%. We recommend the adoption of our system as a robust defense against phishing attacks, ensuring the safety and security of internet users

ACKNOWLEDGEMENT

We are extremely thankful to our project guide **Smt.M. PRASANNA LAKSHMI**, Assistant Professor, Department of Computer Applications, Velagapudi Ramakrishna Siddhartha Engineering College, JNTUK, Kakinada for her timely cooperation and valuable suggestions throughout the project. We are indebted to her for the opportunity given to work under her guidance.

We are extremely thankful to **Dr.V.Esther Jyothi**, Head of the Department, Department of Computer Applications and **Dr.A.V.Ratna Prasad, Principal, V.R.Siddhartha Engineering College, Vijayawada** for providing good environment and better infrastructure facilities.

My sincere thanks to all the teaching and non-teaching staff of Department of Computer Applications for their support throughout my project work.

Finally, I am very much indebted to my family for their moral support and encouragement to achieve higher goals.

DECLARATION

I **K. Sindhu priya** hereby declare the project report titled "**PHISHING WEBSITE DETECTION USING MACHINE LEARNING"** under esteemed supervision of Smt. **M. PRASANNA LAKSHMI is** submitted in partial fulfillment of the requirements for the award of the degree of Master of Computer Applications (MCA).

This is a record of work carried out by me and the results embodied in this project have not been reproduced or copied from any source. The results embodied in this project report have not been submitted to any other Institute or University for the award of any other degree or diploma.

K. SINDHU PRIYA
228W1F0056
II MCA, MCA Dept.,
V.R. Siddhartha Engineering College
Kanuru, Vijayawada-520007

List of Figures

S.No	Fig.No	Fig.Name	Page No
1	Fig 3.1	SYSTEM ARCHITECTURE	11
2	Fig 3.2	USE CASE DIAGRAM	12
3	Fig 3.3	SEQUENCE DIAGRAM	13
4	Fig 3.4	DATA FLOW DIAGRAM	14
5	Fig 5.1	DISPLAY WEBSITE	38
6	Fig 5.2	PREDICT THE PHISHING WEBSITE	39
7	Fig 5.3	SAFE URL	39
8	Fig 5.4	SHOWING ACCURACY	40
9	Fig 5.5	ATTACKED BY PHISHING	40
10	Fig 5.6	LOGISTIC REGRESSION RESULT	41
11	Fig 5.7	LOGISTIC REGRESSION ACCURACY	42
12	Fig 5.8	VARIABLE IMPORTANCE	43
13	Fig 5.9	RANDOM FOREST ACCURACY	43

List of Tables

S.No	Table. No	Table Name	Page No
1	TABLE 2.1	Feature List	7

INDEX

Content	**Page No.**
Abstract	**I**
Acknowledgement	**II**
Declaration	**III**
List of Figures	**1V**
List of Tables	**V**
Chapter 1: Introduction	
1.1 Background	1
1.2 Objectives	2
1.3 Purpose and Scope	
1.3.1 Purpose	2
1.3.2 Scope	3
1.4 Literature Survey	3-4
Chapter 2: System Analysis	
2.1 Existing System	5
2.2 Proposed System	5-7
2.2.1 Data Set	5
2.2.2 Features extraction	6-7
2.3 **Requirement Analysis**	8-9
2.4 **Hardware Requirements**	9
2.5 **Software Requirements**	9
Chapter 3: System Design	
3.1 Module Division	10
3.2 DFD/UML Diagrams	11-14
3.2.1. Use Case Diagram	12
3.2.2. Sequence Diagram	13
3.3.3. DFD	14

Chapter 4: Implementation and Testing

4.1 Code (Place Core segments) 15-36

4.2 Testing Approach 36-37

4.2.1 White Box Testing 36

4.2.2 Black Box System 37

Chapter 5: Results and Discussions (Output Screens) 38-43

Chapter 6: Conclusion and Future Work 44-45

Chapter 7: References 46

Chapter 1

Introduction

1.1Background

Our research encountered a significant challenge common in the field: the scarcity of reliable training datasets. This dilemma is pervasive among researchers, primarily due to the absence of a consensus within the literature regarding the definitive features that unequivocally characterize phishing websites. Consequently, constructing a comprehensive dataset that encompasses all potential features becomes a formidable task.

This article aims to address this issue by shedding light on the crucial features that have demonstrated their soundness and effectiveness in predicting phishing websites. Furthermore, we introduce new features, experimentally assign novel rules to well-established features, and update existing features. Our approach aims to contribute to the standardization of phishing dataset creation.

Phishing is a criminal tactic that combines elements of social engineering and technical manipulation to pilfer personal identity data and financial account credentials from unsuspecting individuals. Social engineering strategies involve the use of deceptive emails masquerading as legitimate communications from businesses or government agencies. These emails are designed to lead recipients to counterfeit websites that coax them into revealing sensitive financial information such as usernames and passwords.

Technical subterfuge tactics involve the installation of malicious software on individuals' computers to directly steal credentials, often intercepting online account usernames and passwords. To achieve their objectives, cybercriminals first create unauthorized replicas of genuine websites and emails, typically from financial institutions or other entities dealing with financial data. These fraudulent emails are carefully crafted to mimic the logos and slogans of legitimate companies. The inherent nature and structure of Hypertext Markup Language (HTML) make it remarkably simple to replicate images and even entire websites.

While the ease of website creation has fueled the rapid growth of the internet as a communication medium, it has also facilitated the misuse of trademarks, trade names, and corporate identifiers that consumers rely on for authentication. Phishers then disseminate these "spoofed" emails to as many recipients as possible in an attempt to entice them into their scheme. When recipients open these emails or click on embedded links, they are redirected to counterfeit websites that convincingly impersonate legitimate entities.

1.2 Objective:

The main goal of this project is to identify and alert users to potentially harmful websites created by cybercriminals with the intent of stealing their personal data. This initiative aims to empower users to engage in safe online browsing and protect their sensitive information from malicious individuals seeking to exploit their credentials for illegal purposes.

1.3Purpose and Scope

1.3.1 Purpose

Phishing websites are cleverly designed to deceive users into divulging sensitive information, such as passwords, credit card details, and Social Security numbers. These deceitful schemes are frequently orchestrated via email or messaging apps, where perpetrators send messages that convincingly mimic legitimate sources, such as banks or social media platforms. Within these messages, there is often a hyperlink leading to a phishing website that mimics the appearance of a genuine site.

What's concerning is that many smartphone users may unknowingly fall prey to phishing attacks, as they may mindlessly follow links without discerning whether they're genuine or fraudulent. The distinction between a legitimate source and a phishing attempt can be blurred in these instances.

If you happen to receive an email or instant message from an unfamiliar sender, instructing you to sign in on a website, exercise caution. This message could be a phishing email containing links to a deceptive website. These phishing websites, also known as "spoofed" sites, have a primary objective: stealing your account credentials or other confidential information by luring you into believing you're interacting with a legitimate platform. It's crucial to be vigilant, as you might even end up on a phishing site unintentionally, perhaps due to a typo in a web address.

1.3.2 Scope

The main goal of this project is to identify and alert users to potentially harmful websites created by cybercriminals with the intent of stealing their personal data. This initiative aims to empower users to engage in safe online browsing and protect their sensitive information from malicious individuals seeking to exploit their credentials for illegal purposes.

The primary aims of phishing attacks, as perceived by the attackers, encompass:

Monetary Gain: Phishers often target users to obtain financial benefits by illicitly acquiring their banking credentials.

Anonymity: Rather than utilizing stolen identities themselves, cybercriminals may opt to sell the stolen identities to others who are looking for ways to conceal their own identities and engage in illegal activities.

Reputation and Recognition: Phishers may carry out these attacks for the purpose of gaining recognition and notoriety within their peer groups or communities.

1.4 Literature Survey

[1] Y. Zhang, J. I. Hong, and L. F. Cranor,” Cantina: A Content-based Approach to Detecting Phishing Web Sites,” New York, NY, USA, 2007, pp. 639-648.

In their study, Zhang and colleagues [1] introduced CANTINA, a fresh and innovative method for identifying phishing websites that relies on content analysis. Their approach utilizes the widely recognized TF-IDF algorithm to assess the textual content of web pages independently, without external factors. They also conducted experiments involving straightforward heuristics to effectively decrease the occurrence of false positives.

Remarkably, their system successfully identifies approximately 90% of phishing websites while maintaining an impressively low false positive rate of only 1%.

[2] Sudhanshu Gautam, Kritika Rani and Bansidhar Joshi: Detecting Phishing Websites Using Rule-Based Classification Algorithm: A Comparison: In Springer,2018.

Sudhanshu et al. [2] used association data mining approach. They have proposed rule-based classification technique for phishing website detection. They have concluded that association classification algorithm is better than any other algorithms because of their simple rule transformation. They achieved 92.67% accuracy by extracting 16 features but this is not up to mark so proposed algorithm can be enhanced for efficient detection rate.

[3] Pradeepthi K V and Kannan A," Performance study of classification techniques for phishing URL detection," in 2014 Sixth International Conference on Advanced Computing (ICoAC), 2014, pp. 135-139.

Pradeepthi and Kannan, as detailed in their research denoted by reference [12], conducted an extensive survey focusing on the body of work concerning the classification techniques employed in the detection of phishing URLs. Their research employed a dataset comprising 4500 URLs, and they methodically categorized the features into four distinct groups: lexical features, URL-centric features, network-related attributes, and domain-based characteristics. Subsequently, they conducted experiments that encompassed a variety of machine learning algorithms, including but not limited to Naive Bayes (NB), multi-layer perceptron, J48 decision tree, Logistic Model Tree (LMT), Random Forest (RF), Random Tree, C4.5, ID3, C-RT, and K-nearest neighbor (KNN).

Upon analyzing the outcomes of their experiments, they arrived at the noteworthy conclusion that tree-based classifiers, in particular, prove to be the most fitting and effective approach for the task of phishing URL classification.

Chapter 2

System Analysis

2.1 EXISTING SYSTEM

- In the existing models have low latency. Existing systems do not have a specific user interface. In the current system, different algorithms are not compared.
- The Decision Tree model provided detection accuracy rate at 96%
- The Support vector Machine accuracy rate at 94%

2.2 PROPOSED SYSTEM

2.2.1 Data Set

In research, we amassed a dataset consisting of 11,055 URLs, encompassing both phishing and legitimate websites. Our model is designed with a specific emphasis on detecting phishing attacks, with a primary focus on scrutinizing the features associated with phishing websites. These carefully chosen features encompass various aspects, including the URLs themselves, domain identity, page design and content, the appearance of the web address bar, and even the influence of human social factors. However, for the scope of our paper, our primary focus centers on the analysis of URL and domain name features.

In study concentrates on the examination of these specific attributes to develop an effective method for distinguishing between phishing and legitimate websites. By delving into the characteristics of URLs and domain names, we aim to enhance the accuracy of our phishing detection model and provide valuable insights into the field of online security.

2.2.2 Features extraction

Phishing websites exhibit distinct characteristics and patterns, which can be harnessed as valuable features for identification and analysis. In this section, we comprehensively explore the full spectrum of phishing website features that have been leveraged in prior research endeavors. Additionally, as we delve deeper into the study of phishing attributes and patterns, we unveil novel characteristics that can be incorporated as additional features. In total, our research encompasses 36 distinct phishing features.

To facilitate a structured understanding, we categorize these features into four primary groups, as outlined below, and provide a summary of these features in Table I:

- Features can be derived from the address bar, providing valuable insights into the website's legitimacy.
- Abnormalities within the website's structure and behavior can serve as a source of distinctive features for analysis.
- Extracting features from the HTML and JavaScript code used on the website offers a deeper understanding of its functionality and intent.
- Domain-related features, such as the domain name and registration details, can be pivotal in distinguishing between legitimate and potentially malicious websites.

TABLE 2.1: Feature List

Features Based on	
Address bar based	Using the IP Address
	Long URL to Hide the Suspicious Part
	Using URL Shortening Services “Tiny URL”
	URL’s having “@” Symbol
	Redirecting using “//”
	Adding Prefix or Suffix Separated by (-) to the Domain
	Sub Domain and Multi Sub Domains
	HTTPS (Hyper Text Transfer Protocol with Secure Sockets Layer)
	Domain Registration Length
	Favicon
	Using Non-Standard Port
	The Existence of “HTTPS” Token in the Domain Part of the URL
Abnormal Based Features	Request URL
	URL of Anchor
	Links in <Meta>, <Script> and <Link> tags
	Server Form Handler (SFH)
	Submitting Information to Email
	Abnormal URL
HTML and JavaScript based Features	Website Forwarding
	Status Bar Customization
	Disabling Right Click
	Using Pop-up Window
	I Frame Redirection
Domain based Features	Age of Domain
	DNS Record
	Website Traffic
	PageRank
	Google Index
	Number of Links Pointing to Page
	Statistical-Reports Based Feature

2.3 REQUIREMENT ANALYSIS

2.3.1 FEASIBILITY STUDY

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates.

During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are

- Economic feasibility
- Technical feasibility
- Social feasibility

2.3.2 ECONOMICAL FEASIBILITY

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited.

The expenditures must be justified. Thus, the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

2.3.3 TECHNICAL FEASIBILITY

This study is carried out to check the technical feasibility, that is, the technical requirements of the system.

Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

2.3.4 SOCIAL FEASIBILITY

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it.

2.4 HARDWARE REQUIREMENTS

- Computer/Laptop
- Processer: i5 or more
- 8GB RAM
- Operating System (Windows)
- Internet

2.5 SOFTWARE REQUIREMNTS

- ➢ Jupiter Notebook
- ➢ Python
- ➢ PyCharm
- ➢ Google colab
- ➢ GitHub

Chapter 3

System Design

3.1 MODULE DIVISION

To identify the most potent features and eliminate irrelevant ones among the 36 available features, our approach involves a comprehensive examination of all possible combinations. This meticulous feature selection process is pivotal in refining our classifier's accuracy. Here's a breakdown of the steps involved in building the classifier using Random Forest and Logistic Regression techniques:

1. The initial phase entails partitioning the dataset into training and test subsets. We allocate 80% of the data for training and reserve the remaining 20% for testing.

2. In this critical step, we systematically train and test the classifier using all conceivable combinations of the 32 features. This exhaustive process allows us to identify the strongest features that significantly enhance the detection accuracy.

3. Following the extensive evaluation, we've accumulated a set of features that have demonstrated their effectiveness in boosting accuracy. These selected features move on to the final stage of classifier construction.

4. With the refined subset of features, we proceed to execute the final classifier. This classifier, incorporating the most influential features, is poised to deliver optimal results in the detection of phishing websites.

SYSTEM ARCHITECTURE

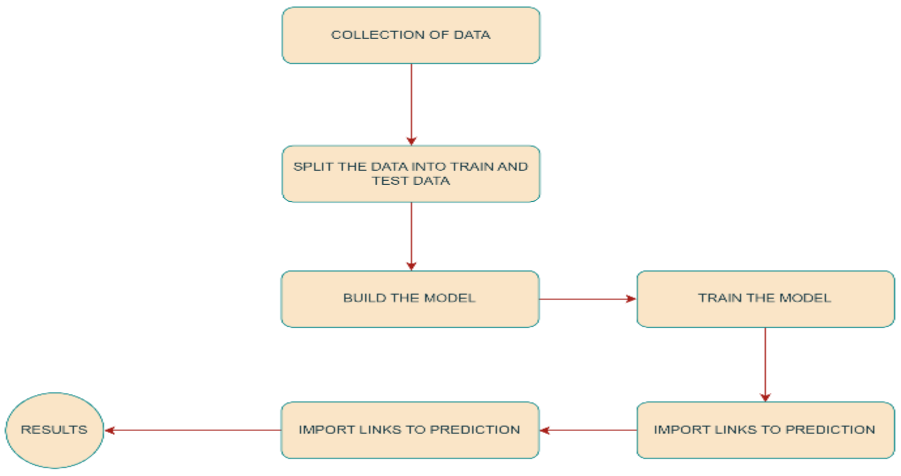

COLLECTION OF DATA
SPLIT THE DATA INTO TRAIN AND TEST DATA
BUILD THE MODEL
TRAIN THE MODEL
IMPORT LINKS TO PREDICTION
IMPORT LINKS TO PREDICTION
RESULTS

Fig 3.1: System Architecture

3.2 UML DIAGRAMS

This section consists of the UML diagrams related to the modules developed such as Activity diagram, Sequence diagram and Use case diagram. UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group. The goal is for UML to become a common language for creating models of object-oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems. The UML is a very important part of developing object-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

3.2.1 USE CASE DIAGRAM

A **use case diagram** is a graphical depiction of a user's possible interactions with a system. A use case diagram shows various use cases and different types of users the system has and will often be accompanied by other types of diagrams as well. The use cases are represented by either circles or ellipses. The actors are often shown as stick figures.

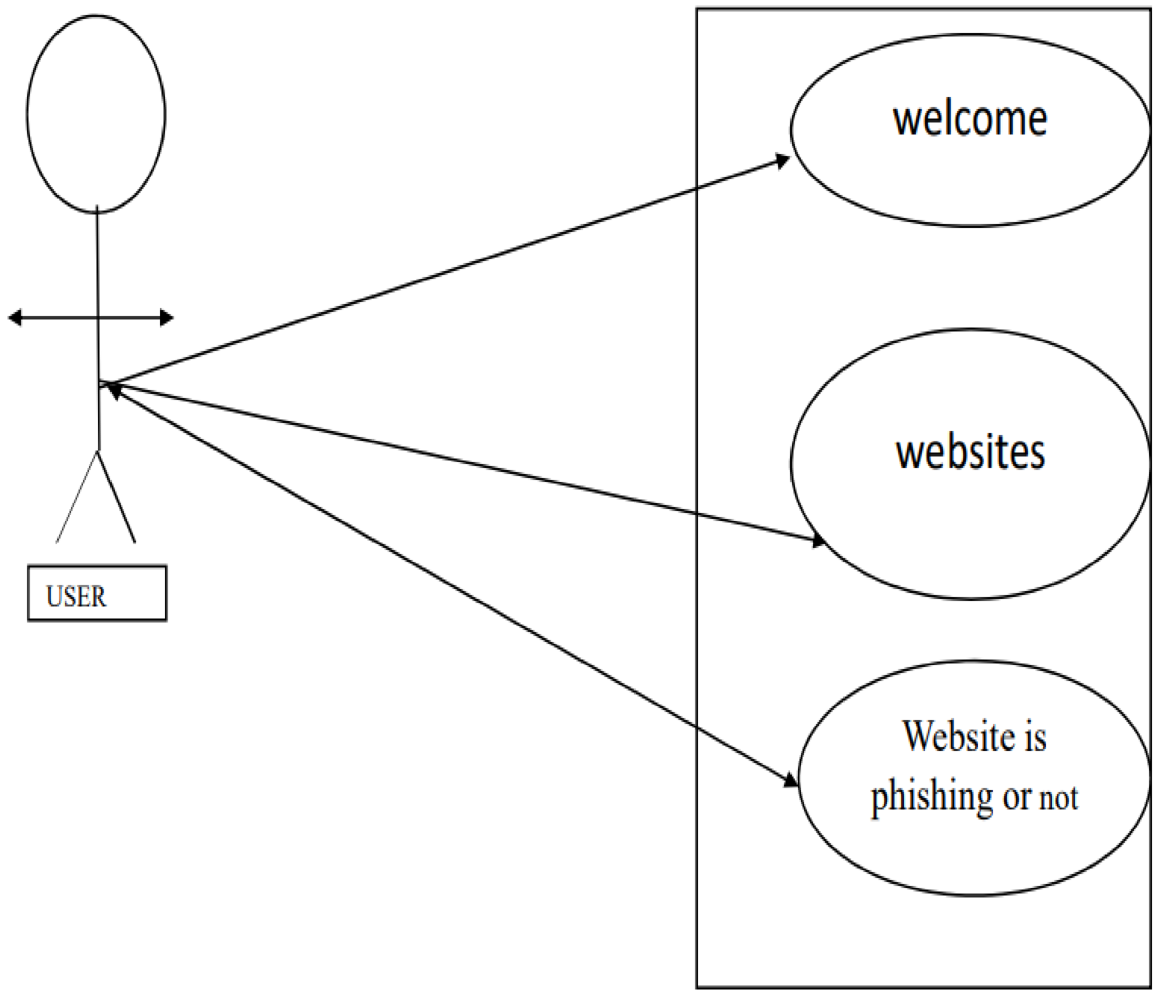


Fig3.2: Use case-website interface

3.2.2 SEQUENCE DIAGRAM

A sequence diagram shows how processes or objects interact with each other over time. It is a dynamic modeling language that focuses on the interactions between objects in the sequential order that those interactions occur.

Sequence diagrams are often used to document the requirements and design of software systems. They can also be used to model business processes and other types of systems**.**

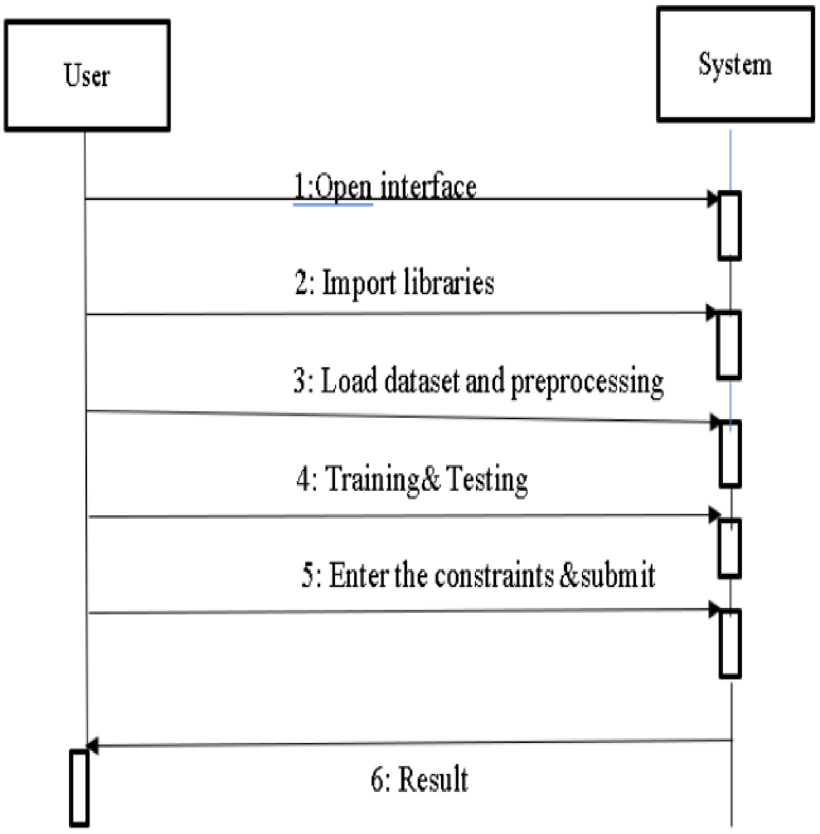
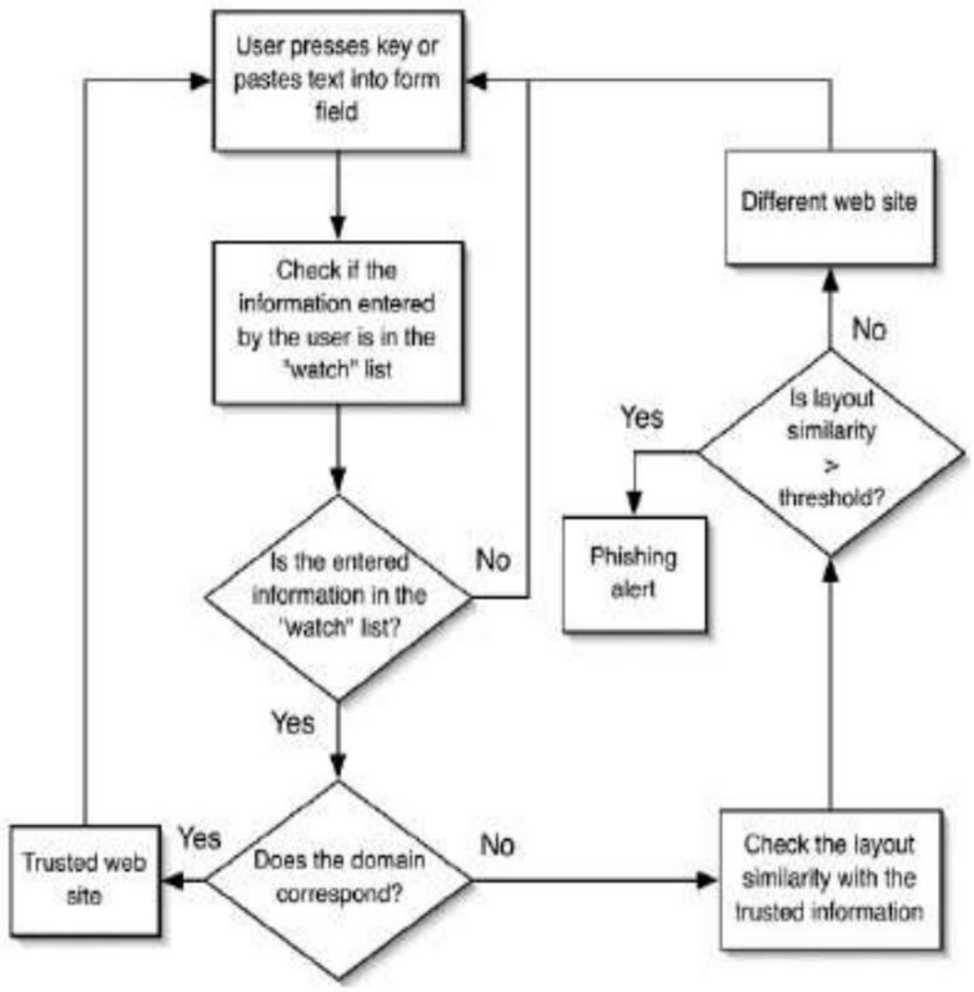


Fig 3.3: Sequence diagram on user and system implementation

3.2.3 DATA FLOW DIAGRAM

A data flow diagram (DFD) is a graphical representation of the flow of data through a system or process. DFDs are used to model and analyze the flow of data through a system, and to identify opportunities for improvement.



User presses key or pastes text into form field
Check if the information entered by the user is in the "watch" list
Is the entered information in the "watch" list?
No
Yes
Does the domain correspond?
Yes
Trusted web site
No
Check the layout similarity with the trusted information
Is layout similarity > threshold?
Yes
Phishing alert
No
Different web site

Fig 3.4: Data Flow Diagram on website

Chapter 4

Implementation and Testing

In order to tackle the issue of phishing, we've devised an innovative solution in the form of a Google Chrome web browser extension. The development of this extension is rooted in the utilization of JavaScript programming language (JS) to effectively identify and thwart phishing attacks.

4.1 CODE:

CODE OF DATASETGEN.PY:

```
import requests
import datetime
import re
import socket
import ssl
import urllib.request

import regex
import whois
from bs4 import BeautifulSoup
from tldextract import extract



def generate_data_set(url):
    data_set = []

    # Converts the given URL into standard format
    if not re.match(r"^https?", url):
        url = "http://" + url
```

```
# Stores the response of the given URL
try:
    response = requests.get(url)
except:

    response = ""

# Extracts domain from the given URL
domain = re.findall(r"://([^/]+)/?", url)[0]

# Requests all the information about the domain
whois_response = requests.get("https://www.whois.com/whois/" + domain)

rank_checker_response = requests.post("https://www.checkpagerank.net/index.php", {
    "name": domain
})

# Extracts global rank of the website
try:
    global_rank = int(re.findall(r"Global Rank: ([0-9]+)", rank_checker_response.text)[0])
except:
    global_rank = -1

def url_having_ip(url):
    # using regular function
    symbol = regex.findall(r'(http((s)?)://)((((\d)+).)*)((\w)+)(/((\w)+))?', url)
    if (len(symbol) != 0):
        having_ip = 1  # phishing
    else:
        having_ip = -1  # legitimate
        return (having_ip)
    return 0

def url_length(url):
    length = len(url)
    if (length < 54):
        return -1
    elif (54 <= length <= 75):
        return 0
    else:
        return 1

def url_short(url):
    if re.findall("goo.gl|bit.ly", url):
        return 1
    else:
        return -1

def having_at_symbol(url):
    symbol = regex.findall(r'@', url)
    if (len(symbol) == 0):
        return -1
    else:
        return 1

def doubleSlash(url):
    if re.findall(r"[^https?:]//", url):
        return -1
    else:
        return 1

def prefix_suffix(url):
    subDomain, domain, suffix = extract(url)
    if (domain.count('-')):
        return 1
    else:
        return -1
```

```
def sub_domain(url):
    subDomain, domain, suffix = extract(url)
    if (subDomain.count('.') == 0):
        return -1
    elif (subDomain.count('.') == 1):
        return 0
    else:
        return 1

def SSLfinal_State(url):
    try:
        # check wheather contains https
        if (regex.search('^https', url)):
            usehttps = 1
        else:
            usehttps = 0
        # getting the certificate issuer to later compare with trusted issuer
        # getting host name
        subDomain, domain, suffix = extract(url)
        host_name = domain + "." + suffix
        context = ssl.create_default_context()
        sct = context.wrap_socket(socket.socket(), server_hostname=host_name)
        sct.connect((host_name, 443))
        certificate = sct.getpeercert()
        issuer = dict(x[0] for x in certificate['issuer'])
        certificate_Auth = str(issuer['commonName'])
        certificate_Auth = certificate_Auth.split()
        if (certificate_Auth[0] == "Network" or certificate_Auth == "Deutsche"):
            certificate_Auth = certificate_Auth[0] + " " + certificate_Auth[1]
        else:
            certificate_Auth = certificate_Auth[0]
        trusted_Auth = ['Comodo', 'Symantec', 'GoDaddy', 'GlobalSign', 'DigiCert',
'StartCom', 'Entrust', 'Verizon',
                        'Trustwave', 'Unizeto', 'Buypass', 'QuoVadis', 'Deutsche Telekom', 'Network
Solutions',
                        'SwissSign', 'IdenTrust', 'Secom', 'TWCA', 'GeoTrust', 'Thawte', 'Doster',
'VeriSign']
        # getting age of certificate
        startingDate = str(certificate['notBefore'])
        endingDate = str(certificate['notAfter'])
        startingYear = int(startingDate.split()[3])
        endingYear = int(endingDate.split()[3])
        Age_of_certificate = endingYear - startingYear

        # checking final conditions
        if ((usehttps == 1) and (certificate_Auth in trusted_Auth) and (Age_of_certificate >=
1)):
            return -1  # legitimate
        elif ((usehttps == 1) and (certificate_Auth not in trusted_Auth)):
            return 0  # suspicious
        else:
            return 1  # phishing

    except Exception as e:

        return 1

def domain_registration(url):
    try:
        w = whois.whois(url)
        updated = w.updated_date
        exp = w.expiration_date
        length = (exp[0] - updated[0]).days
        if (length <= 365):
            return 1
        else:
            return -1
    except:
```

```
        return 0

def favicon(url):
    # ongoing
    return 0

def port(url):
    try:
        port = domain.split(":")[1]
        if port:
            return 1
        else:
            return -1
    except:
        return -1

def https_token(url):
    subDomain, domain, suffix = extract(url)
    host = subDomain + '.' + domain + '.' + suffix
    if (host.count('https')):  # attacker can trick by putting https in domain part
        return 1
    else:
        return -1

def request_url(url):
    try:
        subDomain, domain, suffix = extract(url)
        websiteDomain = domain

        opener = urllib.request.urlopen(url).read()
        soup = BeautifulSoup(opener, 'lxml')
        imgs = soup.findAll('img', src=True)
        total = len(imgs)
        linked_to_same = 0
        avg = 0
        for image in imgs:
            subDomain, domain, suffix = extract(image['src'])
            imageDomain = domain
            if (websiteDomain == imageDomain or imageDomain == ''):
                linked_to_same = linked_to_same + 1
        vids = soup.findAll('video', src=True)
        total = total + len(vids)

        for video in vids:
            subDomain, domain, suffix = extract(video['src'])
            vidDomain = domain
            if (websiteDomain == vidDomain or vidDomain == ''):
                linked_to_same = linked_to_same + 1
        linked_outside = total - linked_to_same
        if (total != 0):
            avg = linked_outside / total

        if (avg < 0.22):
            return -1
        elif (0.22 <= avg <= 0.61):
            return 0
        else:
            return 1
    except:
        return 0

def url_of_anchor(url):
    try:
        subDomain, domain, suffix = extract(url)
        websiteDomain = domain

        opener = urllib.request.urlopen(url).read()
        soup = BeautifulSoup(opener, 'lxml')
        anchors = soup.findAll('a', href=True)
        total = len(anchors)
        linked_to_same = 0
        avg = 0
        for anchor in anchors:
            subDomain, domain, suffix = extract(anchor['href'])
            anchorDomain = domain
            if (websiteDomain == anchorDomain or anchorDomain == ''):
                linked_to_same = linked_to_same + 1
        linked_outside = total - linked_to_same
        if (total != 0):
            avg = linked_outside / total

        if (avg < 0.31):
            return -1
        elif (0.31 <= avg <= 0.67):
            return 0
        else:
            return 1
    except:
        return 0

def Links_in_tags(url):
    try:
        opener = urllib.request.urlopen(url).read()
        soup = BeautifulSoup(opener, 'lxml')

        no_of_meta = 0
        no_of_link = 0
        no_of_script = 0
        anchors = 0
        avg = 0
        for meta in soup.find_all('meta'):
            no_of_meta = no_of_meta + 1
        for link in soup.find_all('link'):
            no_of_link = no_of_link + 1
        for script in soup.find_all('script'):
            no_of_script = no_of_script + 1
        for anchor in soup.find_all('a'):
            anchors = anchors + 1
        total = no_of_meta + no_of_link + no_of_script + anchors
        tags = no_of_meta + no_of_link + no_of_script
        if (total != 0):
            avg = tags / total

        if (avg < 0.25):
            return -1
        elif (0.25 <= avg <= 0.81):
            return 0
        else:
            return 1
    except:
        return 0

def sfh(url):
    # ongoing
    return 0

def email_submit(url):
    try:
        opener = urllib.request.urlopen(url).read()
        soup = BeautifulSoup(opener, 'lxml')
        if (soup.find('mailto:')):
            return 1
        else:
            return -1
    except:
        return 0

def abnormal_url(url):
    return 0;

def redirect(url):
    if len(response.history) <= 1:
        return -1
    elif len(response.history) <= 4:
        return 0
    else:
        return 1

def on_mouseover(url):
    if re.findall("<script>.+onmouseover.+</script>", response.text):
        return 1
    else:
        return -1

def popup(url):
    if re.findall(r"alert\(", response.text):
        return 1
    else:
        return -1

def iframe(url):
    if re.findall(r"[<iframe>|<frameBorder>]", response.text):
        return 1
    else:
        return -1

def age_of_domain(url):
    try:
        w = whois.whois(url)
        start_date = w.creation_date
        current_date = datetime.datetime.now()
        age = (current_date - start_date[0]).days
        if (age >= 180):
            return -1
        else:
            return 1
    except Exception as e:
        print(e)
        return 0

def dns(url):
    # ongoing
    return 0

def web_traffic(url):
    try:
        if global_rank > 0 and global_rank < 100000:
            return -1
        else:
            return 1
    except:
        return 1

def page_rank(url):
    try:
        if global_rank > 0 and global_rank < 100000:
            return -1
        else:
            return 1
    except:
        return 1

def google_index(url):
        try:
            if global_rank > 0 and global_rank < 100000:
                return -1
            else:
                return 1
        except:
            return 1

    def links_pointing(url):
        number_of_links = len(re.findall(r"<a href=", response.text))
        if number_of_links == 0:
            return 1
        elif number_of_links <= 2:
            return 0
        else:
            return -1

    def rightClick(url):
        if re.findall(r"event.button ?== ?2", response.text):
            return 1
        else:
            return -1

    def statistical(url):
        # ongoing
        return -1

    check = [[url_having_ip(url), url_length(url), url_short(url), having_at_symbol(url),
              doubleSlash(url), prefix_suffix(url), sub_domain(url), SSLfinal_State(url),
              domain_registration(url), favicon(url), port(url), https_token(url), request_url(url),
              url_of_anchor(url),       Links_in_tags(url),       sfh(url),       email_submit(url),
abnormal_url(url),
              redirect(url), on_mouseover(url), rightClick(url), popup(url), iframe(url),
              age_of_domain(url), dns(url), web_traffic(url), page_rank(url), google_index(url),
            links_pointing(url), statistical(url)]]
    return check
def main(url):
    return generate_data_set(url)
```

CODE OF RUN.PY:

```
#!/usr/bin/env python
# coding: utf-8

# In[5]:



from sklearn.ensemble import RandomForestClassifier
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import GridSearchCV
import numpy as np
import matplotlib.pyplot as plt

dataset = pd.read_csv("phishcoop.csv")
dataset = dataset.drop('id', 1) #removing unwanted column

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

x_train, x_test, y_train, y_test = train_test_split(x,y,test_size = 0.25, random_state =0 )
parameters = [{'n_estimators': [100, 700],
    'max_features': ['sqrt', 'log2'],
    'criterion' :['gini', 'entropy']}]

grid_search = GridSearchCV(RandomForestClassifier(),  parameters,cv =5, n_jobs= -1)
```

```
grid_search.fit(x_train, y_train)
#printing best parameters
print("Best Accurancy =" +str( grid_search.best_score_))
print("best parameters =" + str(grid_search.best_params_))
classifier = RandomForestClassifier(n_estimators = 100, criterion = "gini", max_features =
'log2',  random_state = 0)
classifier.fit(x_train, y_train)

#predicting the tests set result
y_pred = classifier.predict(x_test)

# In[7]:


import Datasetgen as data


# In[8]:



def run(url):

    try:
        #checking and predicting
        checkprediction = data.main(url)
        #checkprediction=[[0, -1, -1, -1, 1, 1, -1, 1, 0, 0, -1, -1, 1, 1, 0, 0, -1, 0, -1, -1, -1, -1, 1, 0,
0, 1, 1, 1, -1, -1]]
        prediction = classifier.predict(checkprediction)
        if prediction==1:
            return "Warning The website is harmfull for you"
        else:
            return "COOL! You can excess the website" +"\n"+ url
    except:
        return "The website you entered does not exist"
```

CODE OF GUI.PY:

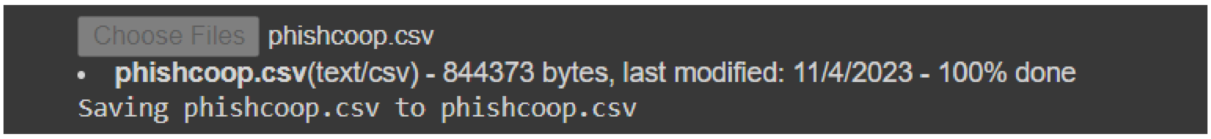
```
from tkinter import *
from PIL import Image, ImageTk
import run

scr = Tk()
scr.geometry('1920x1080+0+0')  # Increased dimensions (width x height) and position (x, y)
scr.resizable(width=True, height=True)

def retrieve_input():
    global us
    input1 = us.get()
    print(input1)
    out = run.main(input1)
    print(out)
    if out == "Warning The website is harmful for you":
        v1 = StringVar()
        v1.set(out)
        l2 = Label(f, text='Phishing / Not Phishing', textvariable=v1, font=('times', 20, 'bold'),
bg='white', fg='red')
        l2.place(x=500, y=300)
    else:
        v1 = StringVar()
        v1.set(out)
        l2 = Label(f, text='Phishing / Not Phishing', textvariable=v1, font=('times', 20, 'bold'),
bg='white', fg='red')
        l2.place(x=500, y=300)

l = Label(scr, text="Phishing Website Detection", font=('times', 20, 'bold'), bg='blue',
fg='white')
l.pack(side=TOP, fill=X)
f = Frame(scr, bg='orange')
f.pack(fill=BOTH, expand=12)
v = StringVar()
l1 = Label(f, text='  Write The Website  ', font=('times', 20, 'bold'), bg='white', fg='black')
l1.place(x=300, y=100)
us = Entry(f, font=('times', 20, 'bold'), bg='white', fg='black', textvariable=v, width=50)
us.place(x=600, y=100)
b1 = Button(f, text='Predict', font=('times', 20, 'bold'), command=retrieve_input)
b1.place(x=700, y=200)

image_label = None  # A global variable to hold the image label
image_displayed = False  # Flag to keep track of image display status

def display_image():
    global image_label, image_displayed

    if not image_displayed:
        image_path         =         "C:/Users/DELL/Downloads/Detecting-Phishing-Website-master
(1)/Detecting-Phishing-Website-master/final/image.png"
        img = Image.open(image_path)
        img = ImageTk.PhotoImage(img)

        # Create a label to display the image
        image_label = Label(f, image=img)
        image_label.image = img
        image_label.place(x=0.5, y=350)
        image_displayed = True
    else:
        # Remove the image label
        image_label.place_forget()
        image_displayed = False
```

```
image_button = Button(f, text="ACCURACY IMAGE", command=display_image, bg="yellow", font=("Arial", 12), fg="black")
image_button.place(x=50, y=100)

result_image_label = None  # A global variable to hold the result image label
result_image_displayed = False  # Flag to keep track of result image display status

def display_result_image():
    global result_image_label, result_image_displayed

    if not result_image_displayed:
        result_image_path = "C:/Users/DELL/Downloads/Detecting-Phishing-Website-master (1)/Detecting-Phishing-Website-master/final/image3.png"  # Replace with the actual path to your result image
        result_img = Image.open(result_image_path)
        result_img = ImageTk.PhotoImage(result_img)

        # Create a label to display the result image
        result_image_label = Label(f, image=result_img)
        result_image_label.image = result_img
        result_image_label.place(relx=0.5, rely=0.7, anchor=CENTER)
        result_image_displayed = True
    else:
        # Remove the result image label
        result_image_label.place_forget()
        result_image_displayed = False

result_button = Button(f, text="Result Image", command=display_result_image, bg="yellow", font=("Arial", 12), fg="black")
result_button.place(relx=0.5, rely=0.5, anchor=CENTER)

scr.mainloop()
```

GOOGLE COLAB

FOR LOGISTIC REGRESSION

```
from google.colab import files
uploaded=files.upload()
```



```
phishcoop.csv
• phishcoop.csv(text/csv) - 844373 bytes, last modified: 11/4/2023 - 100% done
Saving phishcoop.csv to phishcoop.csv
```

```
from sklearn.ensemble import RandomForestClassifier
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import GridSearchCV
import numpy as np
import matplotlib.pyplot as plt

dataset = pd.read_csv("phishcoop.csv")
dataset

dataset = pd.read_csv("phishcoop.csv")
dataset = dataset.drop(labels='id', axis=1)
x = dataset.iloc[: , :-1].values
y = dataset.iloc[: , -1:].values
print(y)

x_train, x_test, y_train, y_test = train_test_split(x,y,test_size = 0.25, random_state =0 )
Classifier = LogisticRegression(random_state=0)
Classifier.fit(x_train, y_train)
y_red = Classifier.predict(x_test)
#predicting the tests set result
y_pred = Classifier.predict(x_test)
```

```
#confusion matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
score = Classifier.score(x_test , y_test)
print(score*100,"%")


from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
logit_roc_auc = roc_auc_score(y_test, Classifier.predict(x_test))
fpr, tpr, thresholds = roc_curve(y_test, Classifier.predict_proba(x_test)[:,1])
plt.figure()
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()

import seaborn as sns
plt.figure(figsize=(9,9))
sns.heatmap(cm, annot=True, fmt=".3f", linewidth=.5, square=True, cmap='Blues_r');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title ='Accuracy Score; {0}'.format(score)
plt.title(all_sample_title, size=15);
plt.savefig("cm.png")

df=pd.read_csv("phishcoop.csv")
df.head
df.info()
pd.set_option('display.max_rows', 100)
df.isna().sum()
pd.reset_option('display.max_rows')
original_dataset = df.copy()
class_map = {'legitimate':0, 'phishing':1}
original_dataset['Result'] = original_dataset['Result'].map(class_map)
corr_matrix = original_dataset.corr()
plt.figure(figsize=(60,60))
color = plt.get_cmap('viridis').copy()   # default color
color.set_bad('lightblue')
sns.heatmap(corr_matrix, annot=True, linewidth=0.4, cmap=color)
plt.savefig('heatmap')
plt.show()
```

FOR RANDOM FOREST

```
from sklearn.ensemble import RandomForestClassifier
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import GridSearchCV
import numpy as np
import matplotlib.pyplot as plt

dataset = pd.read_csv("phishcoop.csv")
dataset

dataset=dataset.drop('id',1)
x = dataset.iloc[ : , :-1].values
y = dataset.iloc[:, -1:].values
print(y)
```

```
from sklearn.model_selection import GridSearchCV
x = dataset.iloc[ : , :-1].values
y = dataset.iloc[:, -1:].values

x_train, x_test, y_train, y_test = train_test_split(x,y,test_size = 0.25, random_state =0 )
parameters = [{'n_estimators': [100, 700],
    'max_features': ['sqrt', 'log2'],
    'criterion' :['gini', 'entropy']}]

grid_search = GridSearchCV(RandomForestClassifier(),  parameters,cv =5, n_jobs= -1)
grid_search.fit(x_train, y_train)
#printing best parameters
print("Best Accurancy =" +str( grid_search.best_score_))
print("best parameters =" + str(grid_search.best_params_))

classifier = RandomForestClassifier(n_estimators = 100, criterion = "gini", max_features =
'log2',  random_state = 0)
classifier.fit(x_train, y_train)
y_pred = classifier.predict(x_test)

#predicting the tests set result
y_pred = classifier.predict(x_test)
#confusion matrix

cm = confusion_matrix(y_test, y_pred)
print(cm)
names = dataset.iloc[:,:-1].columns
importances =classifier.feature_importances_
sorted_importances = sorted(importances, reverse=True)
indices = np.argsort(-importances)
var_imp = pd.DataFrame(sorted_importances, names[indices], columns=['importance'])
#-------------plotting variable importance
plt.title("Variable Importances")
plt.barh(np.arange(len(names)), sorted_importances, height = 0.7)
plt.yticks(np.arange(len(names)), names[indices], fontsize=7)
plt.xlabel('Relative Importance')
plt.show()

score = classifier.score(x_test, y_test)
print(score*100,"%")
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
roc_auc = roc_auc_score(y_test, classifier.predict(x_test))
fpr, tpr, thresholds = roc_curve(y_test, classifier.predict_proba(x_test)[:,1])
plt.figure()
plt.plot(fpr, tpr, label='Random Forest(area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('RANDOM')
plt.show()
```

4.2 TESTING

Testing is a process of executing a program with the aim of finding error. To make our software perform well it should be error free. If testing is done successfully, it will remove all the errors from the software.

TYPES OF TESTING

- WHITE BOX TESTING
- BLACK BOX TESTING

PRINCIPLES OF TESTING: -

a. All the test should meet the customer requirements
b. To make our software testing should be performed by third party
c. Exhaustive testing is not possible. As we need the optimal amount of testing based
d. on the risk assessment of the application.
e. All the test to be conducted should be planned before implementing it
f. It follows pareto rule (80/20 rule) which states that 80% of errors comes from 20%
g. of program components.
h. Start testing with small parts and extend it to large parts.

4.2.1 WHITE BOX TESTING

It is defined as the testing of a software solution's internal structure, design, and coding. In this type of testing, the code is visible to the tester. It focuses primarily on verifying the flow of inputs and outputs through the application, improving design and usability, strengthening security.

4.2.2 BLACK BOX TESTING

Black box testing is a type of software testing in which the functionality of the software is not known. The testing is done without the internal knowledge of the products.

Chapter 5

Results and Discussions

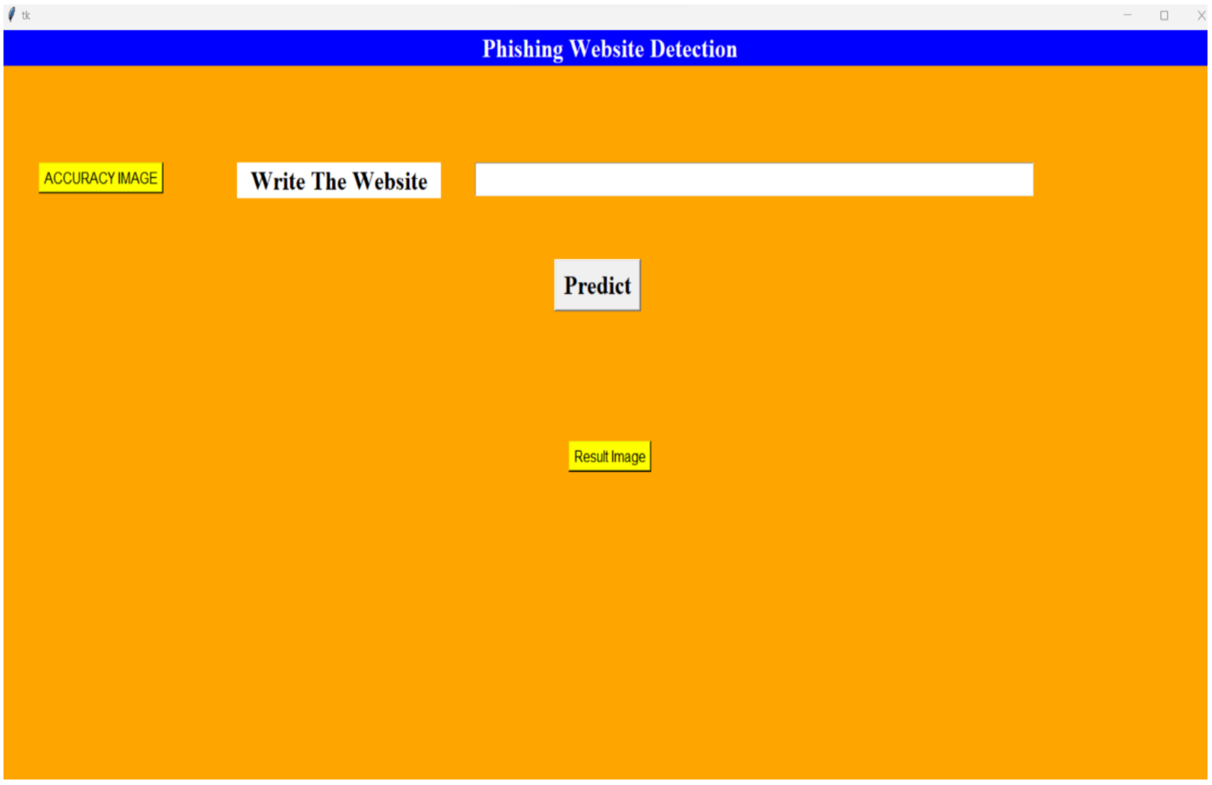
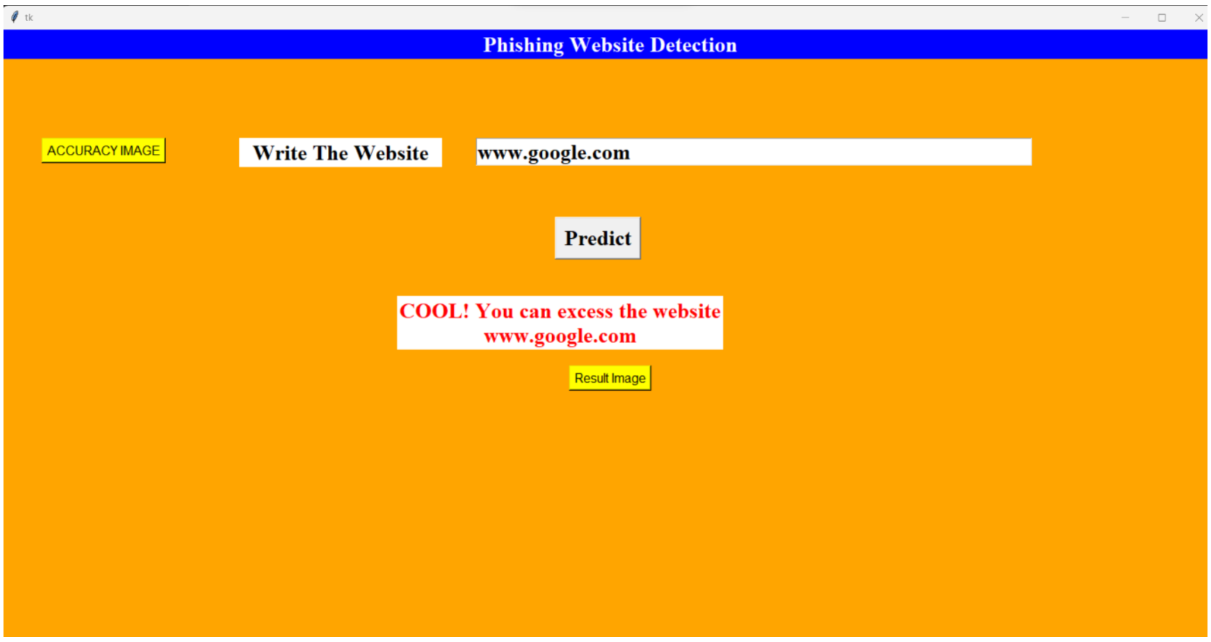
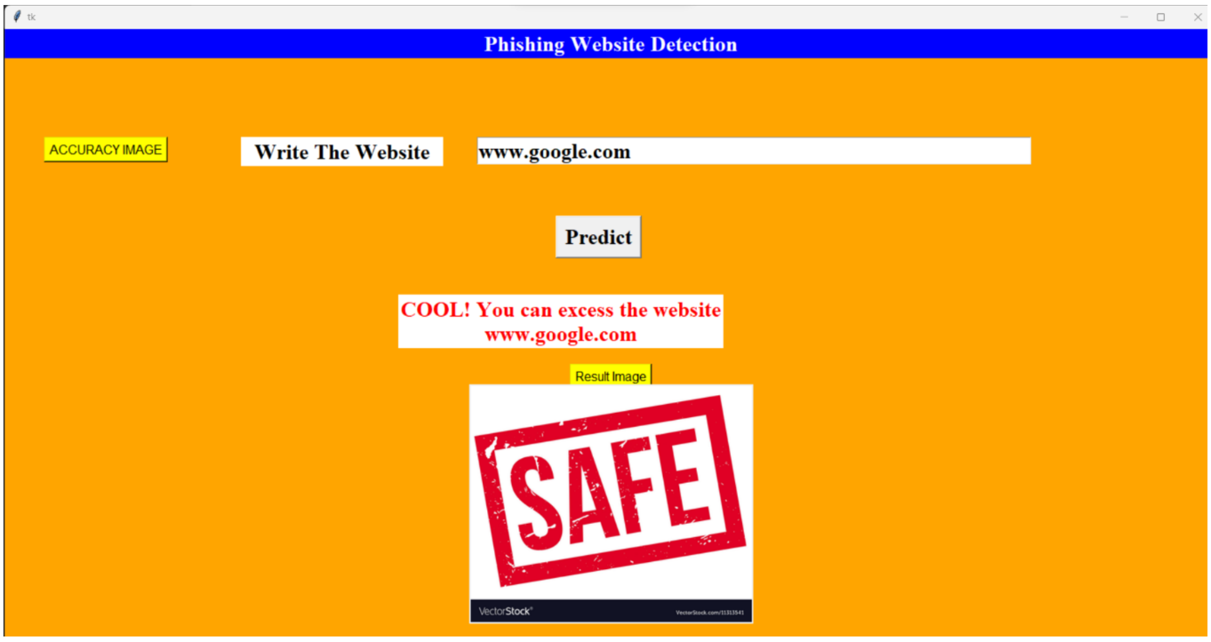


Fig5.1: Display website



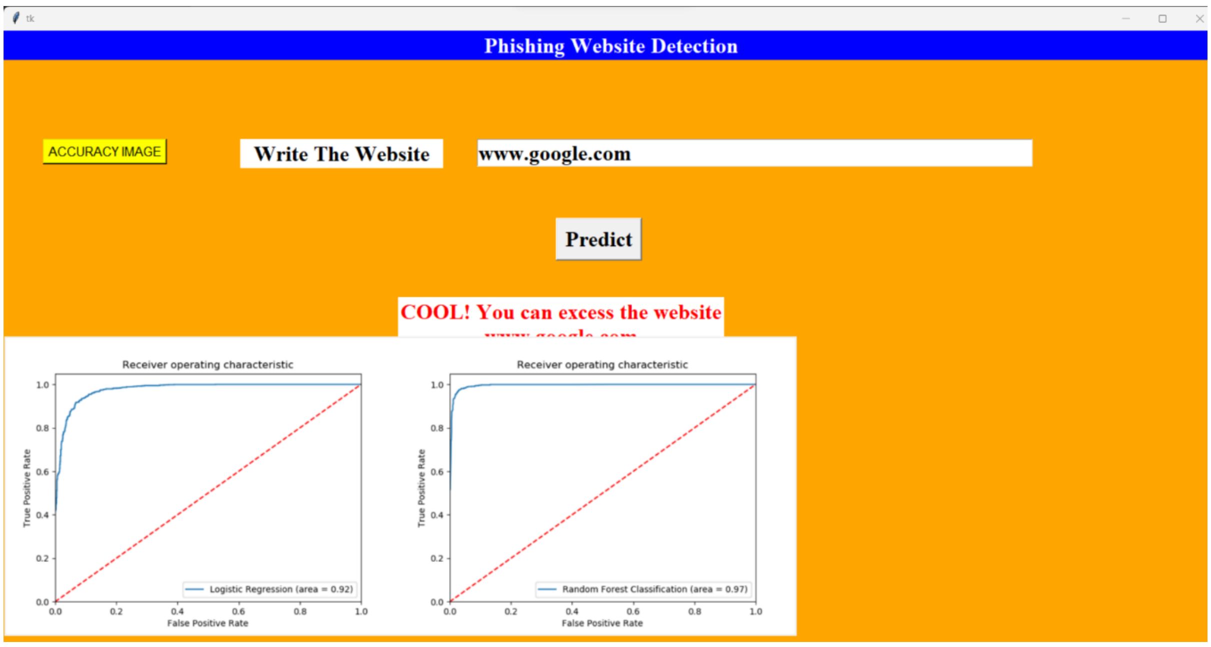
COOL! You can excess the website
www.google.com

Fig 5.2: Predict the phishing website



COOL! You can excess the website
www.google.com

Fig 5.3: Safe URL



Phishing Website Detection

ACCURACY IMAGE

Write The Website

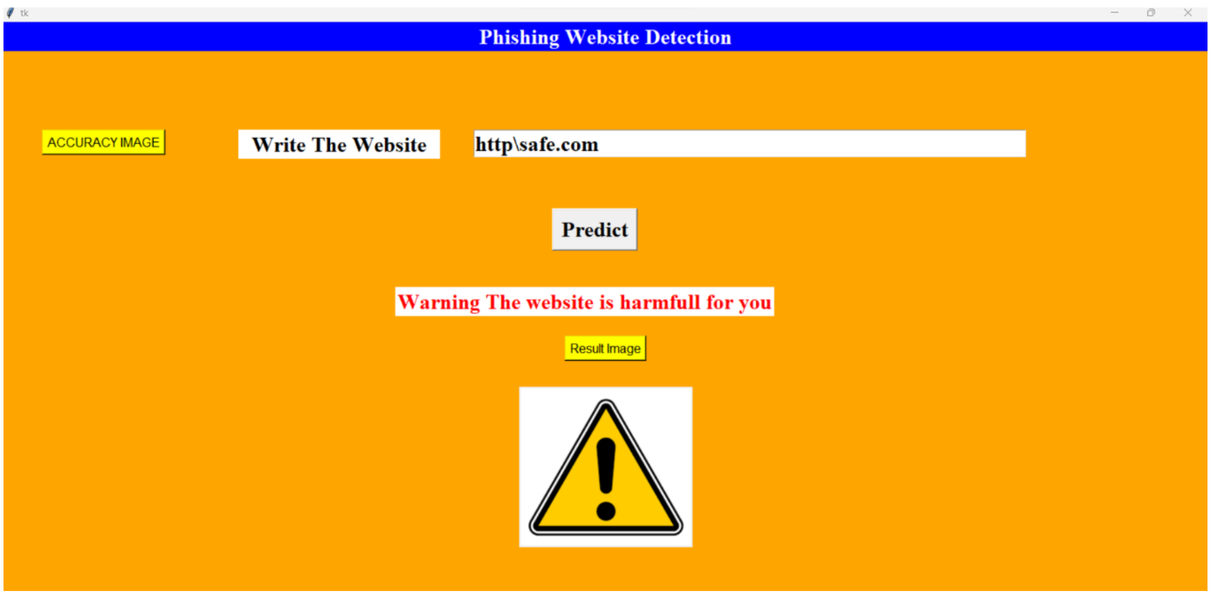
www.google.com

Predict

COOL! You can excess the website

Receiver operating characteristic
True Positive Rate
False Positive Rate
Logistic Regression (area = 0.92)
Receiver operating characteristic
True Positive Rate
False Positive Rate
Random Forest Classification (area = 0.97)

Fig 5.4: Showing accuracy image



Phishing Website Detection

ACCURACY IMAGE

Write The Website

http\safe.com

Predict

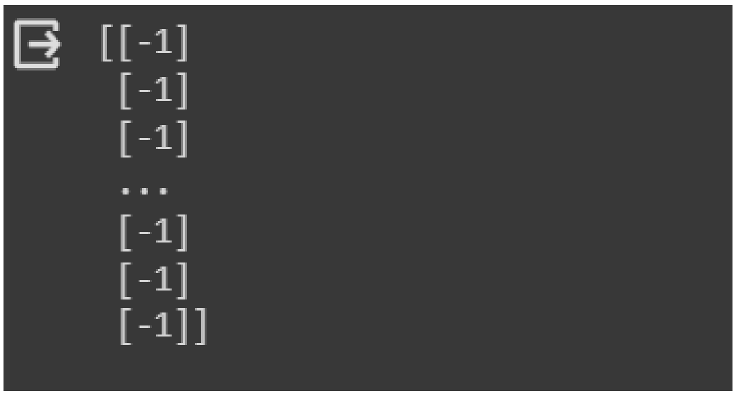
Warning The website is harmfull for you

Result Image

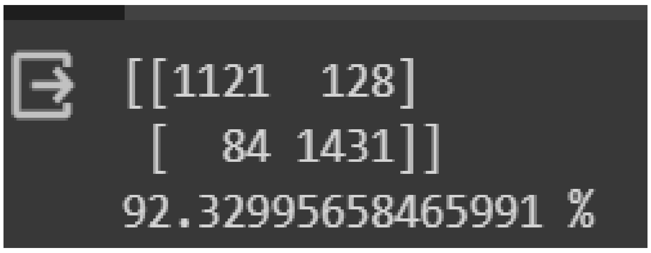
Fig5.5: The website is attacked by phishing

FOR LOGISTIC REGRESSION

The method we've introduced in our approach consistently demonstrates an impressive average accuracy rate of 92%.



```
[[-1]
 [-1]
 [-1]
 ...
 [-1]
 [-1]
 [-1]]
```



```
[[1121  128]
 [  84 1431]]
92.32995658465991 %
```

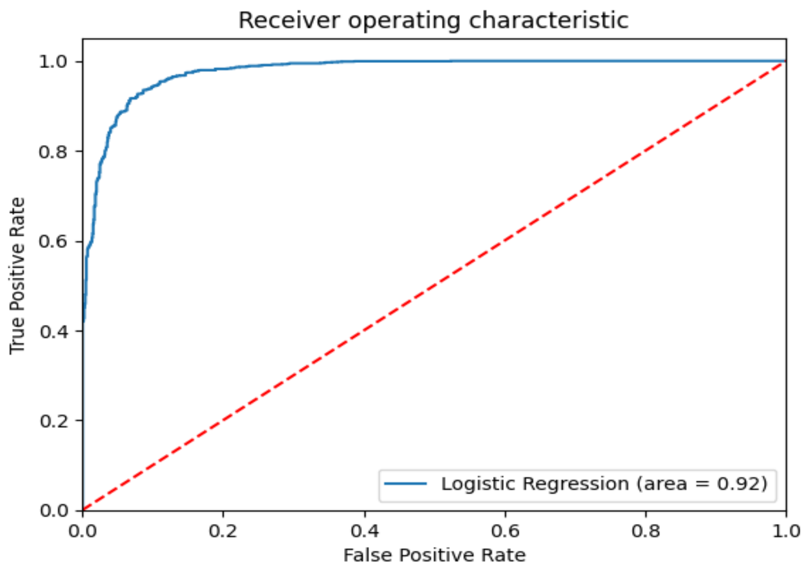


Fig 5.6: Logistic Regression Graph

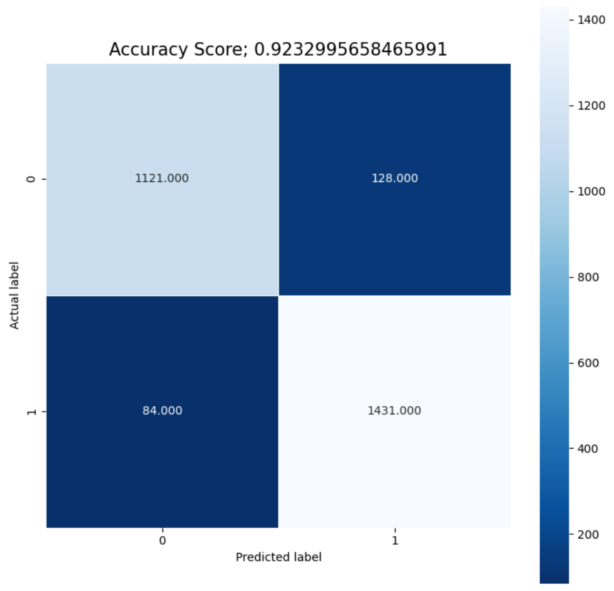
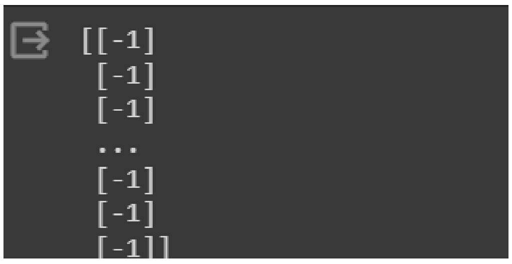


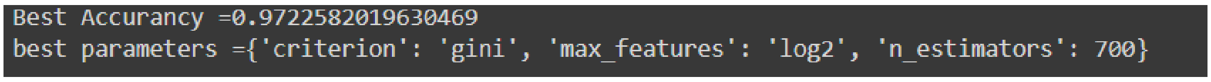
Fig5.7: Logistic Regression Accuracy

FOR RANDOM FOREST

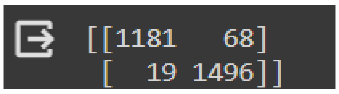
The methodology we've put forward in our approach consistently achieves a noteworthy average accuracy rate of 97%.



```
[[-1]
 [-1]
 [-1]
 ...
 [-1]
 [-1]
 [-1]]
```



```
Best Accurancy =0.9722582019630469
best parameters ={'criterion': 'gini', 'max_features': 'log2', 'n_estimators': 700}
```



```
[[1181   68]
 [  19 1496]]
```

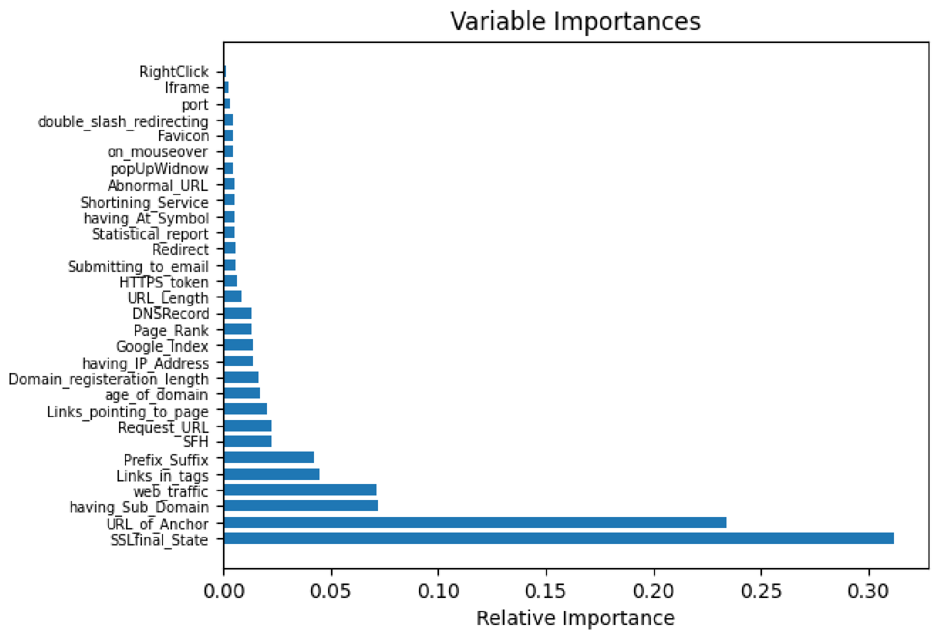


Fig 5.8: variable importance

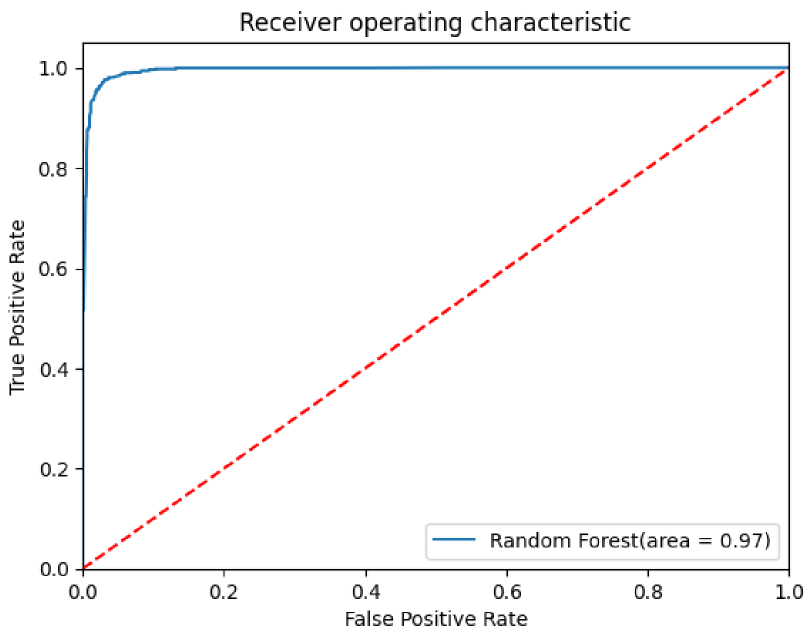


Fig 5.9: Random Forest Accuracy

Chapter 6

Conclusion and Future Work

CONCLUSION

The process of detecting phishing URLs involves two crucial steps. Firstly, we extract pertinent features from the URLs, and subsequently, we employ the developed model, trained on the training dataset, to classify these URLs.

In our study, we exclusively utilized a dataset that provided these pre-extracted features.
One significant challenge, particularly with decision tree classifiers, is the risk of overfitting. Decision trees often excel at classifying the training data but can underperform when applied to a testing dataset.

To elevate the effectiveness of our phishing website detection method, we turned to machine learning technology. Leveraging the Random Forest algorithm, we achieved an impressive 97% accuracy rate while simultaneously minimizing false positives. Our results underscore the advantages of using an algorithm that thrives on larger training datasets, as this approach yielded improved classifier performance. This development represents a significant stride in enhancing the detection of phishing websites through advanced machine learning techniques

FUTURE WORK

It is important to note that the feature scope is a trade-off between accuracy and performance. A wider range of features can lead to better accuracy, but it can also make the model more complex and slower to train and predict. Therefore, it is important to choose a feature scope that is appropriate for the specific problem that you are trying to solve.

In the case of phishing website detection, a wider range of features could potentially lead to better accuracy, but it could also make the model more difficult to maintain and deploy. Therefore, it is important to carefully consider the trade-offs before choosing a feature scope.

Chapter 7

References

[1] Y. Zhang, J. I. Hong, and L. F. Cranor,” Cantina: A Content-based Approach to Detecting Phishing Web Sites,” New York, NY, USA, 2007, pp. 639-648.

[2] Sudhanshu Gautam, Kritika Rani and Bansidhar Joshi: Detecting Phishing Websites Using Rule-Based Classification Algorithm: A Comparison: In Springer,2018.

[3] Pradeepthi K V and Kannan A,” Performance study of classification techniques for phishing URL detection,” in 2014 Sixth International Conference on Advanced Computing (ICoAC), 2014, pp. 135-139.

[4] G. Xiang, J. Hong, C. P. Rose, and L. Cranor,” CANTINA+: A Feature Rich Machine Learning Framework for Detecting Phishing Web Sites,” ACM Trans. Inf. Syst. Secur., vol. 14, no. 2, pp. 21:1-21:28, Sep. 2011.

[5] R. S. Rao and S. T. Ali,” Phish Shield: A Desktop Application to Detect Phishing Webpages through Heuristic Approach,” Procedia Computer Science, vol. 54, no. Supplement C, pp. 147-156, 2015.