Detection Of Phishing Websites Using Machine Learning Based URL Analysis

A Mini-Project Report Submitted in the
Partial Fulfillment of the Requirements
for the Award of the Degree of

BACHELOR OF TECHNOLOGY

IN

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

Submitted by

Ch. Mukesh Babu	**21881A7206**
G. Manik Sai	**21881A7217**
Y. Chandravamsi Karthik	**21881A7265**

SUPERVISOR
Ms. B. Swapna
Assistant Professor

Department of Artificial Intelligence and Data Science

VARDHAMAN COLLEGE OF ENGINEERING
(AUTONOMOUS)
Affiliated to JNTUH, Approved by AICTE, Accredited by NAAC with A++ Grade, ISO 9001:2015 Certified
Kacharam, Shamshabad, Hyderabad - 501218, Telangana, India

May, 2024

VARDHAMAN COLLEGE OF ENGINEERING
(AUTONOMOUS)
Affiliated to JNTUH, Approved by AICTE, Accredited by NAAC with A++ Grade, ISO 9001:2015 Certified
Kacharam, Shamshabad, Hyderabad - 501218, Telangana, India

Department of Artificial Intelligence and Data Science

CERTIFICATE

This is to certify that the project titled **Detection Of Phishing Websites Using Machine Learning Based URL Analysis** is carried out by

Ch. Mukesh Babu	**21881A7206**
G. Manik Sai	**21881A7217**
Y. Chandravamsi Karthik	**21881A7265**

in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Artificial Intelligence and Data Science** during the year 2023-24.

Signature of the Supervisor
Ms. B. Swapna
Assistant Professor

Signature of the HOD
Dr. S. Hariharan
Professor and HOD,AID

Project Viva-Voce held on ________________________

Examiner

Acknowledgement

The satisfaction that accompanies the successful completion of the task would be put incomplete without the mention of the people who made it possible, whose constant guidance and encouragement crown all the efforts with success.

We wish to express our deep sense of gratitude to **Ms. B. Swapna** , Assistant Professor and Project Supervisor, Department of Artificial Intelligence and Data Science, Vardhaman College of Engineering, for her able guidance and useful suggestions, which helped us in completing the project in time.

We are particularly thankful to **Dr. S.Hariharan**, the Head of the Department, Department of Artificial Intelligence and Data Science, his guidance, intense support and encouragement, which helped us to mould our project into a successful one.

We show gratitude to our honorable Principal **Dr. J.V.R. Ravindra**, for providing all facilities and support.

We avail this opportunity to express our deep sense of gratitude and heartful thanks to **Dr. Teegala Vijender Reddy**, Chairman and **Sri Teegala Upender Reddy**, Secretary of VCE, for providing a congenial atmosphere to complete this project successfully.

We also thank all the staff members of Artificial Intelligence and Data Science department for their valuable support and generous advice. Finally thanks to all our friends and family members for their continuous support and enthusiastic help.

Ch.Mukesh babu

G. Manik Sai

Y. Chandravamsi Karthik

Abstract

Phishing attack is a simplest way to obtain sensitive information from innocent users. Aim of the phishers is to acquire critical in-formation like username, password and bank account details.Cybersecurity persons are now looking for trustworthy and steady detection techniques for phishing websites detection.

This project deals with machine learning technology for detection of phishing URLs by extracting and analyzing various features of legitimate and phishing URLs. Decision Tree, random forest Support vector machine, gradient boosting classifier,multi-layer perceptron and logistic regression algorithms are used to detect phishing websites. Aim of the project is to detect phishing URLs as well as narrow down to best machine learning algorithm by comparing accuracy. And also we develop a GUI to detect phishing URL.

Keywords: Logistic Regression, Decision Tree, Random Forest, K-Nearest Neighbor, Multi-layer perceptron , Gradient Boosting Classifier, Feature Selection, Phishing URL, Legitimate URL, User Interface.

Table of Contents

List of Tables

List of Figures

Abbreviations

Abbreviation	Description
VCE	Vardhaman College of Engineering
AID	Artificial Intelligence and Data Science
URL	Uniform Resource Locator
API	Application Programming Interface
TPR	True Positive Rate
FPR	False Positive Rate
SVM	Support Vector Machine
GUI	Graphical User Interface
NLP	Natural Language Processing
ML	Machine Learning
AI	Artificial Intelligence
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network

CHAPTER 1

INTRODUCTION

1.1 About the Project

In this modern world everyone is accessing the internet for different purpose. As the internet grew; there are some problems like stealing data, hacking and fake web pages came into being. To get rid of these phishing websites 'phishing website detection website' is very useful. It helps users to understand what phishing is and how to stay safe while surfing the internet, because people use their personal information like credentials and bank account details. This project has three main parts. In the first part it has 'URL detection' method. In this part what it does is when we access to the webpage, we don't know that the web page we are accessing is safe or not. So, in this first part when you copy the URL and search it tells you weather the URL is phishing or not. In the second part 'Mail phishing' it tells user to copy the email header and paste in to the given header to analyze it. Then it displays the result. In the last part called 'Wiki' it tells you about the basic types of phishing and gives the basic idea of what is phishing for the new user who doesn't have any prior knowledge. So, it is very useful nowadays because we have to do most of the things on the internet, so it should be safe and this project achieve this goal.[1]

1.2 Key Features

1. Machine Learning Model: The project incorporates a sophisticated machine learning model that has been trained on a diverse dataset to effectively classify URLs as either benign or malicious. This model forms the core of the phishing detection system.

2. Data Preprocessing: A robust data preprocessing pipeline is implemented

to clean, normalize, and extract relevant features from raw URL data. This ensures that the model receives high-quality input data for accurate predictions.

3. Accuracy and Efficiency: The model is designed to strike a balance between accuracy and efficiency, making it suitable for real-time or large-scale URL scanning. It achieves a high level of accuracy while maintaining fast response times.

4. User-Friendly Interface: The project offers a user-friendly interface or API that allows users to easily input URLs and receive instant results. This makes it accessible to individuals and organizations looking to enhance their online security.

5. Scalability: The system is designed to be scalable, allowing for the addition of more features and the ability to handle a growing number of URLs as the need arises.

6. Regular Updates: To stay effective in the ever-evolving landscape of online threats, the project is designed to receive regular updates. This includes updates to the machine learning model and the underlying threat intelligence.

7. Documentation and Support: Comprehensive documentation is provided to assist users in understanding and integrating the system into their security infrastructure. Additionally, there is a support mechanism in place to address user queries and concerns.

8. Alerting and Reporting: The system is equipped with alerting and reporting capabilities, ensuring that users are promptly notified of potential threats and providing detailed reports for analysis and audit purposes.

9. Integration: It can be seamlessly integrated into existing security solutions and workflows, making it a valuable addition to any cybersecurity ecosystem.

10. Customization: Users have the flexibility to customize and fine-tune the model and its parameters to suit their specific security requirements.

These key features collectively make your phishing URL detection project a powerful tool in the ongoing battle against online phishing threats.

1.3 Existing System

The performance metrics of four existing models for phishing URL detection, including True Positive Rate (TPR), False Positive Rate (FPR), and Accuracy. Here's a brief analysis of their performance:
Pruned Decision Tree:

- True Positive Rate (TPR): 90.97
- False Positive Rate (FPR): 7.81
- Accuracy: 91.5

The pruned decision tree model demonstrates a high TPR and accuracy, indicating its effectiveness in correctly identifying phishing URLs while maintaining a low false positive rate. This model appears to be a strong candidate for phishing detection.

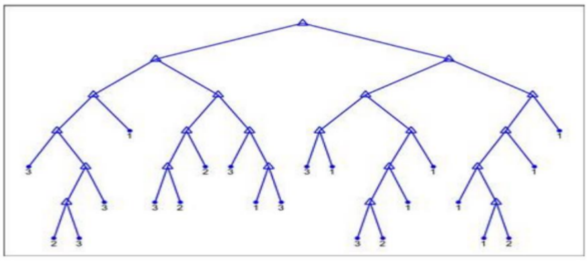


Figure 1.1: Pruned Decision Tree

SVM (Support Vector Machine):

- True Positive Rate (TPR): 90.97
- False Positive Rate (FPR): 18.18
- Accuracy: 86.69

The SVM model also performs well in terms of TPR and accuracy but has a higher FPR compared to the pruned decision tree. It may require some fine-tuning to reduce false positives.

Naïve Bayes' Classifier:

- True Positive Rate (TPR): 88.19
- False Positive Rate (FPR): 16.21
- Accuracy: 86.14

The Naïve Bayes' classifier exhibits a decent TPR and accuracy. Like the SVM, it has a somewhat higher FPR, indicating room for improvement.

Neural Network:

- True Positive Rate (TPR): 85.61
- False Positive Rate (FPR): 15.91
- Accuracy: 84.87

The neural network model also offers competitive TPR and accuracy scores. While it has a lower FPR than the SVM, it could benefit from further optimization to improve its performance.

In conclusion, all four models show promise in phishing URL detection, with the pruned decision tree standing out for its high TPR and accuracy while maintaining a low FPR. However, the choice of the best model may also depend on other factors such as computational efficiency, scalability, and ease of integration into your specific application or system. Further experimentation and fine-tuning may be necessary to select the most suitable model for your specific use case.[2]

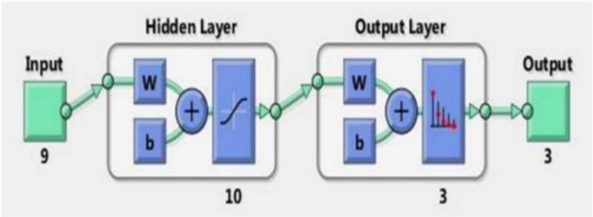


Figure 1.2: A Three-Layer Neural Network

1.4 Proposed System

The proposed phishing detection system utilizes different machine learning models. The system comprises of two major parts, which are the machine learning models and a GUI. These models consist of Decision Tree, Support Vector Machine, XGBooster, Multilayer Perceptron, Logistic Regression , Random Forest and Gradient Boosting. These models are selected after different comparison based performance of multiple machine learning algorithms. Each of these models is trained and tested also extracted from both phishing and legitimate dataset. Hence, the model with the highest accuracy is selected and integrated to a GUI that will enable a user to predict if a URL link is phishing or legitimate with percentage.

1. Improved Accuracy: The proposed system is expected to achieve a higher overall accuracy compared to the existing system. This improvement is primarily attributed to ensemble learning, advanced feature engineering, and dynamic thresholding.

2. Reduced False Positives: By incorporating a dynamic thresholding mechanism, the proposed system aims to reduce false positive rates significantly. This is crucial in minimizing the unnecessary blocking of legitimate URLs, which might be higher in the existing system.

3. Enhanced Robustness: Ensemble learning techniques in the proposed system enhance its robustness against various types of phishing attacks and changing attack tactics. In contrast, the existing system may be more susceptible to evasion techniques not accounted for in its models.

4. Better Generalizability: The use of k-fold cross-validation ensures that the proposed system generalizes well to unseen data, making it more reliable in real-world scenarios. The existing system may not have undergone such rigorous validation.

5. Increased Complexity: It's worth noting that the proposed system is likely more complex due to the ensemble learning and dynamic thresholding

components. This may result in higher computational requirements and potentially longer training times.

CHAPTER 2

LITERATURE REVIEW

A comprehensive literature review on the detection of phishing websites using machine learning reveals a dynamic landscape shaped by the evolving tactics of cybercriminals. Phishing attacks have steadily grown in sophistication, necessitating innovative approaches for their identification and mitigation. The integration of machine learning techniques into this domain has garnered significant attention. Researchers have explored various algorithms and models, including decision trees, support vector machines, neural networks, and ensemble methods, aiming to enhance the accuracy and efficiency of phishing detection systems. These models leverage features derived from diverse sources, such as URLs, webpage content, SSL certificates, and user behavior data, to distinguish between legitimate and malicious websites.[3]

Evaluation metrics like accuracy, precision, recall, and ROC curves have become the yardsticks for assessing the effectiveness of these models. However, the literature also underscores the persistent challenges of dealing with imbalanced datasets, concept drift, and the relentless cat-and-mouse game between attackers and defenders. Recent advances in the field have witnessed the application of deep learning techniques, natural language processing, and the fusion of threat intelligence feeds, enabling more robust real-time detection capabilities. Case studies demonstrate the tangible benefits of these systems in real-world scenarios.

While significant progress has been made, the literature review reveals that there are still open challenges awaiting resolution. Future research directions include the development of more adaptive and explainable AI models, improved feature engineering methodologies, and strategies to counter emerging threats. In conclusion,the synthesis of existing knowledge in this literature review not only sheds light on the current state of machine learning-based phishing detection but also underscores the importance of continuous innovation in this critical cybersecurity domain.

1. Introduction to the Topic: Begin the literature review by introducing the topic of phishing website detection using machine learning.[2] Explain why it's important and the potential consequences of not effectively detecting phishing websites.

2. Historical Overview: Provide a historical context for phishing attacks, including their evolution over time. Discuss the methods and techniques used by attackers in the past and how they have evolved to become more sophisticated.

3. Machine Learning in Phishing Detection: Review studies and research papers that have explored the application of machine learning techniques in phishing detection. Discuss the types of machine learning algorithms and models commonly used for this purpose, such as decision trees, random forests, neural networks, and support vector machines.

4. Features and Data Sources: Explore the features and data sources used in phishing detection models. This may include features extracted from URLs, website content, SSL certificates, and user behaviour data. Discuss the importance of feature engineering in improving detection accuracy.

5. Evaluation Metrics: Describe the evaluation metrics commonly used to assess the performance of phishing detection models. Metrics like accuracy, precision, recall, F1-score, and ROC curves are often used to measure how well a model performs in identifying phishing websites.

6. Challenges and Limitations: Highlight the challenges and limitations associated with using machine learning for phishing detection. These may include issues related to imbalanced datasets, concept drift, and the arms race between attackers and defenders.

7. Recent Advances: Discuss recent advancements in the field, including novel machine learning techniques, the use of deep learning and natural language processing, and the integration of threat intelligence feeds for real-time detection.[4]

8. Comparison of Approaches: Compare and contrast different approaches and models used in the literature. Analyze the strengths and weaknesses of various techniques and their suitability for different scenarios.

9. Case Studies: Present case studies or real-world examples of organizations or projects that have successfully implemented machine learning-based phishing detection systems. Highlight the outcomes and benefits they have achieved.

10. Open Challenges and Future Directions: Identify open challenges and areas for future research in the field of phishing detection. Discuss potential improvements, such as the development of more robust and adaptive models, better feature engineering techniques, and the integration of explainable AI.

CHAPTER 3

SOFTWARE REQUIREMENT ANALYSIS

Software Requirement Analysis for the Proposed System: Detection of Phishing Websites Using Machine Learning. An overview of our project's software requirements is provided below:

3.1 Introduction:

1. Objective: The proposed system aims to enhance the detection of phishing websites using machine learning techniques.

2. Scope: The system will encompass web data collection, feature extraction, model training, real-time detection, and reporting.

3.2 Functional Requirements:

1. Data Collection: The system should be able to collect data from various sources, including URLs, webpage content, SSL certificates, and user behaviour. It should support both batch and real-time data acquisition.

2. Feature Extraction: Extract relevant features from the collected data, such as URL patterns, domain reputation, content analysis, SSL certificate attributes, and user interaction data. Implement techniques to handle imbalanced datasets.

3. Machine Learning Models: Integrate multiple machine learning models (e.g., decision trees, support vector machines) for classification. Enable model training using historical data. Implement model evaluation techniques like cross-validation.

4. Real-time Detection: Develop a real-time detection mechanism to assess incoming web requests. Implement continuous monitoring of websites and

flag potential phishing attempts. Employ adaptive algorithms to handle concept drift.

5. Alerting and Reporting: Generate alerts for suspicious websites. Provide detailed reports on detected phishing attempts, including website information and confidence scores. Support integration with existing security information and event management (SIEM) systems.

6. User Interface: Develop a user-friendly web-based interface for system administrators and analysts to configure, monitor, and manage the system.Implement visualization tools for displaying statistics and trends.

3.3 Non- Functional Requirements:

1. Performance:

   a.Ensure low latency in real-time detection.

   b.Support scalability to handle a large volume of web requests.

2. Security:

   a. Implement robust security measures to protect the system from attacks. Secure data storage and transmission.

3. Reliability:

   a. Ensure the system operates with high availability.

   b. Implement backup and recovery mechanisms.

4. Usability:

   a. Design an intuitive user interface for ease of use.

   b. Provide documentation and training for system administrators.

5. 5. Compatibility:

   a. Ensure compatibility with various web browsers and operating systems.

   b. Support integration with different web server technologies.

3.4 Technology Stack:

Specify the programming languages(Python) and frameworks(tkinter) to be used.

3.5 Constraints:

- Define any budgetary constraints.
- Project timeline and deadlines.
- Specify any regulatory compliance requirements (if required).

CHAPTER 4

SOFTWARE DESIGN

The ”Detection of Phishing Websites using Machine Learning” was designed utilizing a complete set of diagrams to show the system’s interactions, structure, and capabilities. Designing a software system for the proposed project of detecting phishing websites using machine learning involves careful planning and consideration of various components and functionalities. Below, I’ll provide a detailed software design outline for the proposed system:

4.1 Block Diagram:

The proposed system for detecting phishing websites using machine learning can be represented by a block diagram that consists of several interconnected components. At its core is the Machine Learning Model, which encompasses various machine learning models. These algorithms are responsible for analyzing features extracted from input data, such as URLs and webpage content. The Feature Extraction module pre-processes and transforms the input data into a format suitable for model analysis. A crucial element is the Threat Intelligence Integration, which provides real-time updates and data on emerging phishing threats. The Web-based Application serves as the user interface, facilitating user interactions with the system. User input, primarily URLs for analysis, is processed through the Client-Server Architecture, with the server hosting the Machine Learning Model and a Database for data storage. The system emphasizes security through Data Encryption, User Authentication, and User Privacy measures. It also ensures scalability via Load Balancing and caching mechanisms. Continuous Improvement is enabled through a Feedback Loop for user input and Model Retraining to adapt to evolving threats. Logging and Monitoring functionalities track system performance, and Reporting and Alerting mechanisms keep users informed of detection results. This block diagram illustrates the interplay of components in the proposed system, aiming to provide robust, real-time phishing detection with user-friendly access and a focus on security and adaptability.

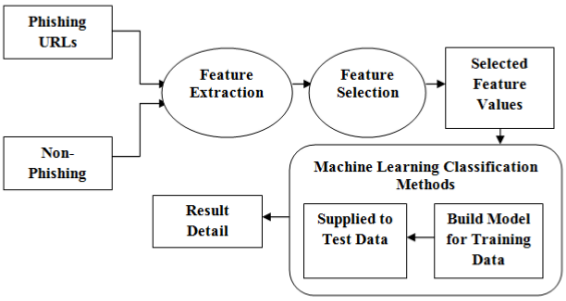


Figure 4.1: Block Diagram

4.2 Activity Diagram:

An activity diagram is the way to explain the flow of the process with the step by step when order of the process matters. IF we start with the first phase of user side action then user perform the header check for mail or URL check for web page he/she wants to visit before giving any sensitive info. The condition phishing check for the possible ways; if YES means not safe then if you want more info then visit the wiki page for details or else exit. If the condition NO implies the safe surfing on the internet. Enter detail, personal info and check mails are the actions can be taken by user and last stage gives the safe operation output.

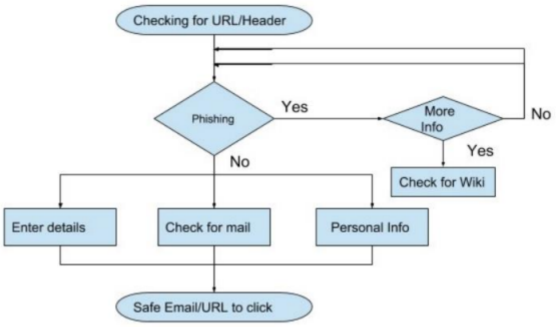


Figure 4.2: Activity Diagram

4.3 Use Case Diagrams:

The Use Case Diagram for the proposed system, which aims to detect phishing websites using machine learning, outlines the various interactions and functionalities involving different actors. The primary actors in this system include "User," "Machine Learning Model," and "Threat Intelligence Service." The "User" actor interacts with the system through multiple use cases. Firstly, they can input a URL for analysis, triggering the "Analyze URL" use case. This involves the system processing the URL through the Machine Learning Model for phishing detection. Users can also provide feedback on the detection results through the "Provide Feedback" use case, contributing to system learning and improvement.

The "Machine Learning Model" actor encapsulates the core functionality of the system. It performs the analysis of input URLs, categorizing them as either phishing or legitimate. The model also supports periodic retraining using feedback data to enhance its accuracy.

The "Threat Intelligence Service" actor plays a vital role in keeping the system updated with real-time phishing threat information. It continuously feeds threat data into the system, ensuring that the Machine Learning Model is aware of the latest phishing tactics and trends.

The Use Case Diagram illustrates how these actors interact to deliver a comprehensive phishing detection solution. Users can submit URLs for analysis, provide feedback, and benefit from an adaptive model, while the Threat Intelligence Service ensures that the system remains current and effective in countering evolving phishing threats.

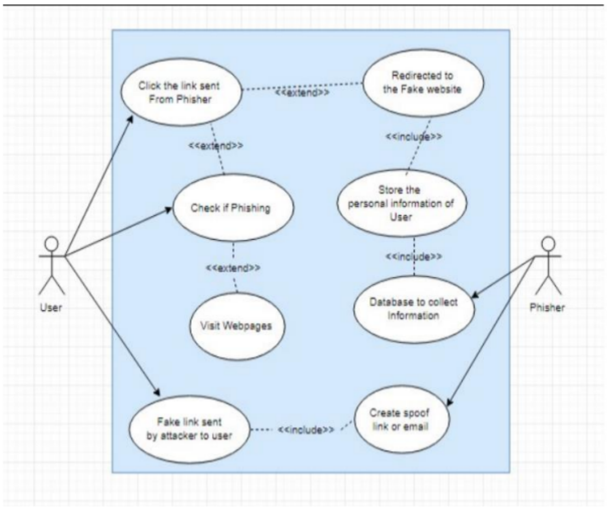


Figure 4.3: Use case Diagram

CHAPTER 5

SOFTWARE & HARDWARE REQUIREMENTS

The software and hardware requirements for a system designed to detection phishing websites using machine learning can vary depending on the complexity of the solution and the specific tools and technologies chosen. Here is a general overview of the requirements:

5.1 Software Requirements:

1. Operating System: The system can typically run on various operating systems, including Windows, Linux, or macOS.

2. Programming Languages:
   a. Python: Python is commonly used for machine learning and data processing tasks.
   b. Web Development Technologies: If a web-based interface is part of the system, technologies like HTML, CSS and possibly a web framework (Flask) may be required.
   c. Machine Learning Libraries: You'll need machine learning libraries like scikit-learn, pandas, numpy, for building and deploying machine learning models.

3. Web Server: If the system includes a web-based user interface, you may need a web server like Apache or Nginx.

4. Version Control: Tools like Git for version control and collaboration.

5. Development Environment: Integrated Development Environments (IDEs) like Jupyter Notebook, Visual Studio Code, or PyCharm can be used for development.

6. Text Editors: Code editors like Sublime Text or Notepad++ for scripting and configuration files.

5.2 Hardware Requirements:

Hardware requirements will vary depending on the scale of your project and expected traffic. Here are the basic hardware components you'll need:

1. Processing Power: The specific hardware requirements for machine learning depend on the complexity of the models and the size of the dataset. A modern multi-core CPU or GPU may be necessary for training deep learning models efficiently.

2. Memory (RAM): Machine learning tasks often require a significant amount of RAM. 8GB or more is recommended, but this can vary.

3. Storage: Adequate storage space for datasets, model checkpoints, and system logs. SSDs are preferred for faster data access.

4. Internet Connectivity: A reliable internet connection is necessary for real-time threat intelligence updates and to access online resources for model training and evaluation.

5. Web Server Hosting: If you plan to deploy a web-based interface, you'll need a server or cloud hosting solution (e.g., AWS, Azure, or a web hosting service).

6. Backup and Redundancy: Implement backup and redundancy mechanisms to ensure the availability and integrity of critical data and system components.

7. Security Measures: Implement security measures to protect sensitive data and prevent unauthorized access to the system

The specific hardware and software requirements may vary based on the scale and complexity of the project. It's essential to conduct a thorough analysis of your project's needs and choose the appropriate technologies and resources accordingly.Considering that phishing attacks are a real-time threat, low-latency internet connectivity is critical for fetching updates and conducting real-time analyses. An uninterrupted power supply (UPS) should also be in place to avoid unexpected system shutdowns, ensuring continuous protection against phishing threats. Additionally, a robust security infrastructure is needed to safeguard sensitive data, including user credentials and potentially malicious website samples used for model training. Firewalls, intrusion detection systems (IDS), and encryption protocols should be implemented to secure the system's components and communications effectively.

In conclusion, the software and hardware requirements for a phishing website detection system encompass a wide range of tools and technologies, all of which should work cohesively to provide comprehensive protection against phishing threats. A well-thought-out combination of software libraries, frameworks, and security tools, along with a reliable hardware setup, is essential to create a robust and effective system that can adapt to the ever-evolving landscape of phishing attacks. Regular updates and maintenance are key to ensuring the system's continued effectiveness in detecting and preventing these malicious activities.

CHAPTER 6

SOURCE CODE

6.1 Data_Collection.ipynb

```
#importing required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import metrics
import warnings
warnings.filterwarnings(’ignore’)
data = pd.read _ csv(”phishing.csv”)
data.head()
## 2. Familiarizing with Data and EDA:
In this step, few dataframe methods are used to look into the data and its features.
#Shape of dataframe
data.shape
#Listing the features of the dataset
data.columns
#Information about the dataset
data.info()
# nunique value in columns
data.nunique()
#droping index column
data = data.drop([’Index’],axis = 1)
#description of dataset
data.describe().T
```

```
data_set.append
plt.figure(figsize=(15,15))
sns.heatmap(data.corr(), annot=True)
plt.show()
#pairplot for particular features
df = data[[’PrefixSuffix-’, ’SubDomains’,’HTTPS’,’AnchorURL’, ’WebsiteTraffic’,’class’]]
sns.pairplot(data = df,hue=”class”,corner=True);
# Phishing Count in pie chart
data[’class’].value_counts().plot(kind=’pie’,autopct=’%1.2f%%’)
plt.title(”Phishing Count”)
plt.show()
X = data.drop([”class”],axis =1)
y = data[”class”]
```

6.2 Data_Split.ipynb

```
# 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, test_size = 0.2, random_state = 42)
X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

6.3 Model_Implementation.ipynb

```
# Creating holders to store the model performance results
ML_Model = []
accuracy = []
f1_score = []
recall = []
precision = []
#function to call for storing the results
def storeResults(model, a,b,c,d):
```

```
ML_Model.append(model)
accuracy.append(round(a, 3))
f1_score.append(round(b, 3))
recall.append(round(c, 3))
precision.append(round(d, 3))
# Linear regression model
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
# instantiate the model
log = LogisticRegression()
# fit the model
log.fit(X_train,y_train)
#predicting the target value from the model for the samples
y_train_log = log.predict(X_train)
y_test_log = log.predict(X_test)
#computing the accuracy, f1_score, Recall, precision of the model performance
acc_train_log = metrics.accuracy_score(y_train,y_train_log)
acc_test_log = metrics.accuracy_score(y_test,y_test_log)
print(”Logistic Regression : Accuracy on training Data::.3f”
.format(acc_train_log))
print(”Logistic Regression : Accuracy on test Data::.3f”.format(acc_test_log))
print()
f1_score_train_log = metrics.f1_score(y_train,y_train_log)
f1_score_test_log = metrics.f1_score(y_test,y_test_log)
print(”Logistic Regression : f1_score on training Data:
:.3f”.format(f1_score_train_log))
print(”Logistic Regression : f1_score on test Data:
:.3f”.format(f1_score_test_log))
print()
recall_score_train_log = metrics.recall_score(y_train,y_train_log)
recall_score_test_log = metrics.recall_score(y_test,y_test_log)
print(”Logistic Regression : Recall on training
Data::.3f’.format(recall_score_train_log))
print(”Logistic Regression : Recall on test Data:
:.3f’.format(recall_score_test_log))
print()
precision_score_train_log = metrics.precision_score(y_train,y_train_log)
precision_score_test_log = metrics.precision_score(y_test,y_test_log)
print(”Logistic Regression : precision on training Data:
:.3f’.format(precision_score_train_log))
print(”Logistic Regression : precision on test Data:
:.3f’.format(precision_score_test_log))
#computing the classification report of the model
print(metrics.classification_report(y_test, y_test_log))
#storing the results. The below mentioned order of parameter passing is
important.
storeResults(’Logistic Regression’,acc_test_log,f1_score_test_log,
recall_score_train_log,precision_score_train_log)
# record generalization accuracy
test_accuracy.append(knn.score(X_test, y_test))
#plotting the training and testing accuracy for n_estimators from 1 to 20
plt.plot(depth, training_accuracy, label=”training accuracy”)
plt.plot(depth, test_accuracy, label=”test accuracy”)
plt.ylabel(”Accuracy”)
plt.xlabel(”n_neighbors”)
plt.legend();
#storing the results. The below mentioned order of parameter passing is
important.
storeResults(’KNearest Neighbors’,acc_test_knn,f1_score_test_knn,
recall_score_train_knn,precision_score_train_knn)
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
# defining parameter range
param_grid = ’gamma’: [0.1],’kernel’: [’rbf’,’linear’]
```

```
svc = GridSearchCV(SVC(), param_grid)
# fitting the model for grid search
svc.fit(X_train, y_train)
#predicting the target value from the model for the samples
y_train_svc = svc.predict(X_train)
y_test_svc = svc.predict(X_test)
#computing the accuracy, f1_score, Recall, precision of the model performance
acc_train_svc = metrics.accuracy_score(y_train,y_train_svc)
acc_test_svc = metrics.accuracy_score(y_test,y_test_svc)
print(”Support Vector Machine : Accuracy on training Data:
:.3f”.format(acc_train_svc))
print(”Support Vector Machine : Accuracy on test Data:
:.3f”.format(acc_test_svc))
print()
f1_score_train_svc = metrics.f1_score(y_train,y_train_svc)
f1_score_test_svc = metrics.f1_score(y_test,y_test_svc)
print(”Support Vector Machine : f1_score on training Data:
:.3f”.format(f1_score_train_svc))
print(”Support Vector Machine : f1_score on test Data:
:.3f”.format(f1_score_test_svc))
print()
recall_score_train_svc = metrics.recall_score(y_train,y_train_svc)
recall_score_test_svc = metrics.recall_score(y_test,y_test_svc)
print(”Support Vector Machine : Recall on training Data:
:.3f”.format(recall_score_train_svc))
print(”Support Vector Machine : Recall on test Data:
:.3f”.format(recall_score_test_svc))
print()


precision_score_train_svc = metrics.precision_score(y_train,y_train_svc)
precision_score_test_svc = metrics.precision_score(y_test,y_test_svc)
print(”Support Vector Machine : precision on training Data:
```

```
:.3f”.format(precision_score_train_svc))
print(”Support Vector Machine: precision on test Data:
:.3f”.format(precision_score_test_svc))
#computing the classification report of the model
print(metrics.classification_report(y_test, y_test_svc))
#storing the results. The below mentioned order of parameter passing is
important.
storeResults(’Support Vector Machine’,acc_test_svc,f1_score_test_svc,
recall_score_train_svc,precision_score_train_svc)

# Decision Tree Classifier model
from sklearn.tree import DecisionTreeClassifier
# instantiate the model
tree = DecisionTreeClassifier(max_depth=30)
#fit the model
tree.fit(X_train, y_train)
#predicting the target value from the model for the samples
y_train_tree = tree.predict(X_train)
y_test_tree = tree.predict(X_test)
#computing the accuracy, f1_score, Recall, precision of the model performance
acc_train_tree = metrics.accuracy_score(y_train,y_train_tree)
acc_test_tree = metrics.accuracy_score(y_test,y_test_tree)
print(”Decision Tree : Accuracy on training Data: :.3f”.format(acc_train_tree))
print(”Decision Tree : Accuracy on test Data: :.3f”.format(acc_test_tree))
print()
f1_score_train_tree = metrics.f1_score(y_train,y_train_tree)
f1_score_test_tree = metrics.f1_score(y_test,y_test_tree)
print(”Decision Tree : f1_score on training Data:
:.3f”.format(f1_score_train_tree))
print(”Decision Tree : f1_score on test Data: :.3f”.format(f1_score_test_tree))
print()
recall_score_train_tree = metrics.recall_score(y_train,y_train_tree)
```

```
recall_score_test_tree = metrics.recall_score(y_test,y_test_tree)
print(”Decision Tree : Recall on training Data:
:.3f’.format(recall_score_train_tree))
print(”Decision Tree : Recall on test Data: :.3f’.format(recall_score_test_tree))
print()
precision_score_train_tree = metrics.precision_score(y_train,y_train_tree)
precision_score_test_tree = metrics.precision_score(y_test,y_test_tree)
print(”Decision Tree : precision on training Data:
:.3f”.format(precision_score_train_tree))
print(”Decision Tree : precision on test Data:
:.3f”.format(precision_score_test_tree))
#computing the classification report of the model
print(metrics.classification_report(y_test, y_test_tree))
training_accuracy = []
test_accuracy = []
# try max_depth from 1 to 30
depth = range(1,30)
for n in depth:
tree_test = DecisionTreeClassifier(max_depth=n)
tree_test.fit(X_train, y_train)

# record training set accuracy
training_accuracy.append(tree_test.score(X_train, y_train))
# record generalization accuracy
test_accuracy.append(tree_test.score(X_test, y_test))
#plotting the training and testing accuracy for max_depth from 1 to 30
plt.plot(depth, training_accuracy, label=”training accuracy”)
plt.plot(depth, test_accuracy, label=”test accuracy”)
plt.ylabel(”Accuracy”)
plt.xlabel(”max_depth”)
plt.legend();
#storing the results. The below mentioned order of parameter passing is
```

```
important.
storeResults(’Decision Tree’,acc_test_tree,f1_score_test_tree,
recall_score_train_tree,precision_score_train_tree)
# Gradient Boosting Classifier Model
from sklearn.ensemble import GradientBoostingClassifier
# instantiate the model
gbc = GradientBoostingClassifier(max_depth=4,learning_rate=0.7)
# fit the model
gbc.fit(X_train,y_train)
#predicting the target value from the model for the samples
y_train_gbc = gbc.predict(X_train)
y_test_gbc = gbc.predict(X_test)
#computing the accuracy, f1_score, Recall, precision of the model performance
acc_train_gbc = metrics.accuracy_score(y_train,y_train_gbc)
acc_test_gbc = metrics.accuracy_score(y_test,y_test_gbc)
print(”Gradient Boosting Classifier : Accuracy on training Data:
:.3f”.format(acc_train_gbc))
print(”Gradient Boosting Classifier : Accuracy on test Data:
:.3f”.format(acc_test_gbc))
print()
f1_score_train_gbc = metrics.f1_score(y_train,y_train_gbc)
f1_score_test_gbc = metrics.f1_score(y_test,y_test_gbc)
print(”Gradient Boosting Classifier : f1_score on training Data:
:.3f”.format(f1_score_train_gbc))
print(”Gradient Boosting Classifier : f1_score on test Data:
:.3f”.format(f1_score_test_gbc))
print()
recall_score_train_gbc = metrics.recall_score(y_train,y_train_gbc)
recall_score_test_gbc = metrics.recall_score(y_test,y_test_gbc)
print(”Gradient Boosting Classifier : Recall on training Data:
:.3f”.format(recall_score_train_gbc))
print(”Gradient Boosting Classifier : Recall on test Data:
```

```
:.3f’.format(recall_score_test_gbc))
print()
precision_score_train_gbc = metrics.precision_score(y_train,y_train_gbc)
precision_score_test_gbc = metrics.precision_score(y_test,y_test_gbc)
print(”Gradient Boosting Classifier : precision on training Data:
:.3f’.format(precision_score_train_gbc))
print(”Gradient Boosting Classifier : precision on test Data:
:.3f’.format(precision_score_test_gbc))
#computing the classification report of the model
print(metrics.classification_report(y_test, y_test_gbc))
training_accuracy = []
test_accuracy = []
# try learning_rate from 0.1 to 0.9
depth = range(1,10)
for n in depth:
forest_test = GradientBoostingClassifier(learning_rate = n*0.1)
forest_test.fit(X_train, y_train)
# record training set accuracy
training_accuracy.append(forest_test.score(X_train, y_train))
# record generalization accuracy
test_accuracy.append(forest_test.score(X_test, y_test))
#plotting the training and testing accuracy for n_estimators from 1 to 50
plt.figure(figsize=None)
plt.plot(depth, training_accuracy, label=”training accuracy”)
plt.plot(depth, test_accuracy, label=”test accuracy”) plt.ylabel(”Accuracy”)
plt.xlabel(”learning_rate”)
plt.legend();
training_accuracy = []
test_accuracy = []
# try learning_rate from 0.1 to 0.9
depth = range(1,10,1)
for n in depth:
```

```
forest_test = GradientBoostingClassifier(max_depth=n,learning_rate = 0.7)
forest_test.fit(X_train, y_train)
# record training set accuracy
training_accuracy.append(forest_test.score(X_train, y_train))
# record generalization accuracy
test_accuracy.append(forest_test.score(X_test, y_test))
#plotting the training and testing accuracy for n_estimators from 1 to 50
plt.figure(figsize=None)
plt.plot(depth, training_accuracy, label=”training accuracy”)
plt.plot(depth, test_accuracy, label=”test accuracy”)
plt.ylabel(”Accuracy”)
plt.xlabel(”max_depth”)
plt.legend();
#storing the results. The below mentioned order of parameter passing is
important.
storeResults(’Gradient Boosting Classifier’,acc_test_gbc,f1_score_test_gbc,
recall_score_train_gbc,precision_score_train_gbc)
result = pd.DataFrame( ’ML
Model’ : ML_Model,
’Accuracy’ : accuracy,
’f1_score’ : f1_score,
’Recall’ : recall,
’Precision’: precision,
)
# dispalying total result
result
#Sorting the datafram on accuracy
sorted_result=result.sort_values(by=[’Accuracy’, ’f1_score’],
ascending=False).reset_index(drop=True)
# dispalying total result
sorted_result
## Storing Best Model
```

```
# XGBoost Classifier Model
from xgboost import XGBClassifier
# instantiate the model
gbc = GradientBoostingClassifier(max_depth=4,learning_rate=0.7)
# fit the model
gbc.fit(X_train,y_train)
import pickle
# dump information to that file
pickle.dump(gbc, open(’pickle/model.pkl’, ’wb’))
#checking the feature improtance in the model
plt.figure(figsize=(9,7))
n_features = X_train.shape[1]
plt.barh(range(n_features), gbc.feature_importances_, align=’center’)
plt.yticks(np.arange(n _ features), X _ train.columns)
plt.title(”Feature importances using permutation on full model”)
plt.xlabel(”Feature importance”)
plt.ylabel(”Feature”)
plt.show()
```

6.4 App.ipynb

```
import tkinter as tk
from tkinter import messagebox
import numpy as np
import pickle
from feature import FeatureExtraction
# Load the model
file = open(”pickle/model.pkl”, ”rb”)
gbc = pickle.load(file)
file.close()
# Function to detect phishing
def detect_phishing(url):
```

```
obj = FeatureExtraction(url)
x = np.array(obj.getFeaturesList()).reshape(1, 30)
y_pro_phishing = gbc.predict_proba(x)[0, 0]
y_pro_non_phishing = gbc.predict_proba(x)[0, 1] return y_pro_phishing,
y_pro_non_phishing
# Create the GUI pop-up
def check_phishing_popup():
url = url_entry.get()
if not url:
messagebox.showerror(”Error”, ”Please enter a URL.”)
return
y_pro_phishing, y_pro_non_phishing = detect_phishing(url)
result_message = f”Phishing Probability: y_pro_phishing:.2% NonPhishing
Probability: y_pro_non_phishing:.2%”
messagebox.showinfo(”Phishing Detection Result”, result_message)
# Create the main application window
app = tk.Tk()
app.title(”Phishing Detection”)
url_label = tk.Label(app, text=”Enter URL:”)
url_entry = tk.Entry(app, width=50)
check_button = tk.Button(app, text=”Check Phishing”,
command=check_phishing_popup)
# Add widgets to the grid
url_label.grid(row=0, column=0, padx=5, pady=5)
url_entry.grid(row=0, column=1, padx=5, pady=5)
check_button.grid(row=1, column=0, columnspan=2, padx=5, pady=10)
# Start the main event loop
app.mainloop()
```

CHAPTER 7

TESTING

Testing the proposed model for detecting phishing websites using machine learning is a crucial phase to ensure its effectiveness and reliability in real-world scenarios. The testing process involves several key steps:

1. Data Splitting: The first step is to split the dataset into training, validation, and testing subsets. Typically, a significant portion of the data is used for training the machine learning models, while the validation set helps fine-tune hyperparameters and prevent overfitting. The testing set is kept separate and is used to evaluate the model's performance objectively.

2. Model Evaluation Metrics: To assess the model's performance, various evaluation metrics are used, including True Positive Rate (TPR), False Positive Rate (FPR), accuracy, precision, recall, F1-score, and the Receiver Operating Characteristic (ROC) curve. These metrics provide insights into the model's ability to correctly classify phishing websites while minimizing false positives.

3. Cross-Validation: Cross-validation techniques such as k-fold cross-validation can be employed to ensure that the model's performance is consistent across different subsets of the data. This helps validate the model's generalization capabilities.

4. Hyperparameter Tuning: Fine-tuning of hyperparameters is performed to optimize the model's performance further. Techniques like grid search or random search can be employed to find the best combination of hyperparameters.

5. Real-time Testing: For practical deployment, the model should be tested in a real-time or near-real-time environment. It should continuously

monitor incoming web requests or emails, classify them as either legitimate or phishing, and log the results for analysis.

6. Security Testing: The system should be subjected to security testing to identify vulnerabilities and ensure that it cannot be exploited by attackers. This includes penetration testing and vulnerability assessments.

7. Performance Testing: Performance testing ensures that the system can handle a high volume of web requests or emails without experiencing delays or failures. It assesses the system's scalability and response time under various load conditions.

8. Robustness Testing: Phishing attacks can take various forms, so the system should be tested against different types of phishing techniques, including spear-phishing and social engineering attacks.

9. User Experience Testing: If the system interacts with users, usability and user experience testing should be conducted to ensure that it is user-friendly and intuitive.

10. Feedback Loop: Continuous monitoring and feedback from users and security experts are essential for refining the model and keeping it up-to-date with emerging phishing threats. This feedback loop helps in adapting the model to evolving attack techniques.

11. Deployment and Monitoring: Once the model passes all testing phases, it can be deployed in a production environment. Continuous monitoring and periodic retesting are essential to ensure that the model remains effective over time.

In summary, thorough testing of the proposed phishing website detection model involves a multifaceted approach, encompassing data splitting, model evaluation, cross-validation, real-time, security, performance, robustness, user experience, and feedback-based testing. This rigorous testing process is crucial to building a reliable and resilient defense against phishing attacks.

CHAPTER 8

RESULTS

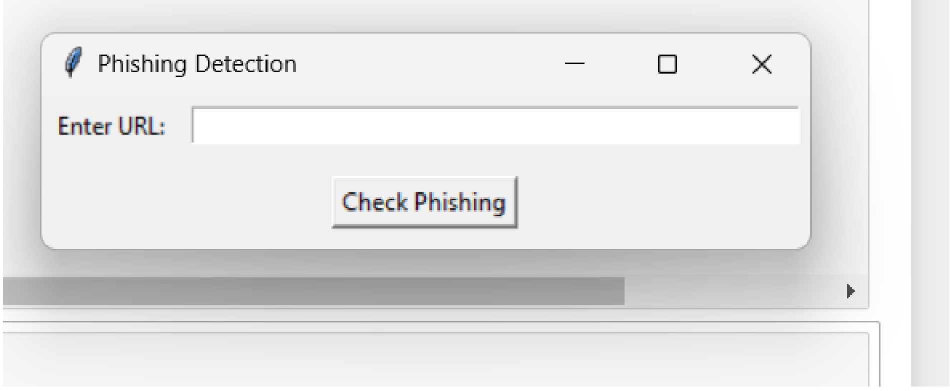


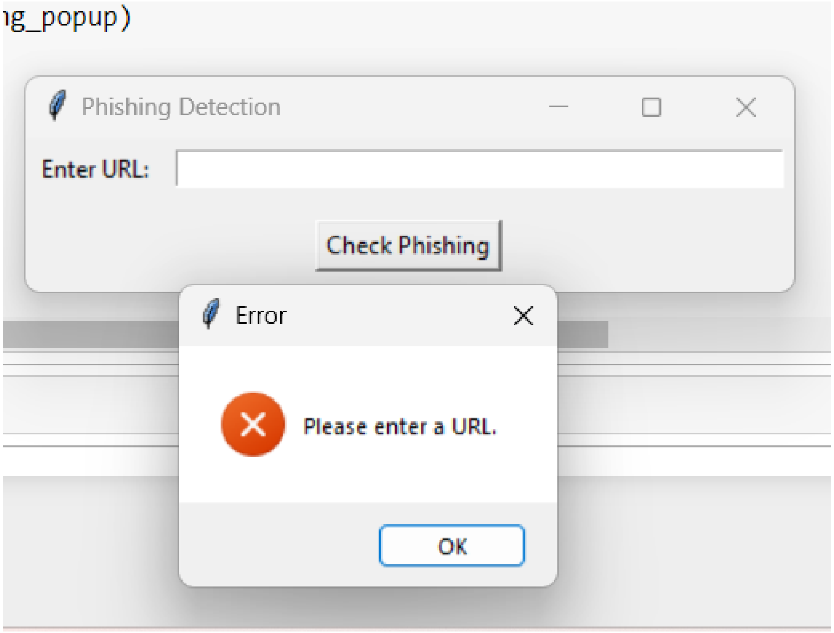
Figure 8.1: Home Screen

	ML Model	Accuracy	f1_score	Recall	Precision
0	Logistic Regression	0.934	0.941	0.943	0.927
1	K-Nearest Neighbors	0.956	0.961	0.991	0.989
2	Support Vector Machine	0.964	0.968	0.980	0.965
3	Naive Bayes Classifier	0.605	0.454	0.292	0.997
4	Decision Tree	0.959	0.963	0.991	0.993
5	Random Forest	0.965	0.969	0.993	0.990
6	Gradient Boosting Classifier	0.974	0.977	0.994	0.986
7	XGBoost Classifier	0.549	0.544	0.987	0.987
8	Multi-layer Perceptron	0.549	0.544	0.987	0.987

Table 8.1: Unsorted results of metrics of multiple Ml models

	ML Model	Accuracy	f1_score	Recall	Precision
0	Gradient Boosting Classifier	0.974	0.977	0.994	0.986
1	Random Forest	0.965	0.969	0.993	0.990
2	Support Vector Machine	0.964	0.968	0.980	0.965
3	Decision Tree	0.959	0.963	0.991	0.993
4	K-Nearest Neighbors	0.956	0.961	0.991	0.989
5	Logistic Regression	0.934	0.941	0.943	0.927
6	Naive Bayes Classifier	0.605	0.454	0.292	0.997
7	XGBoost Classifier	0.549	0.544	0.987	0.987
8	Multi-layer Perceptron	0.549	0.544	0.987	0.987

Table 8.2: Sorted results of metrics of multiple Ml models



ıg_popup)

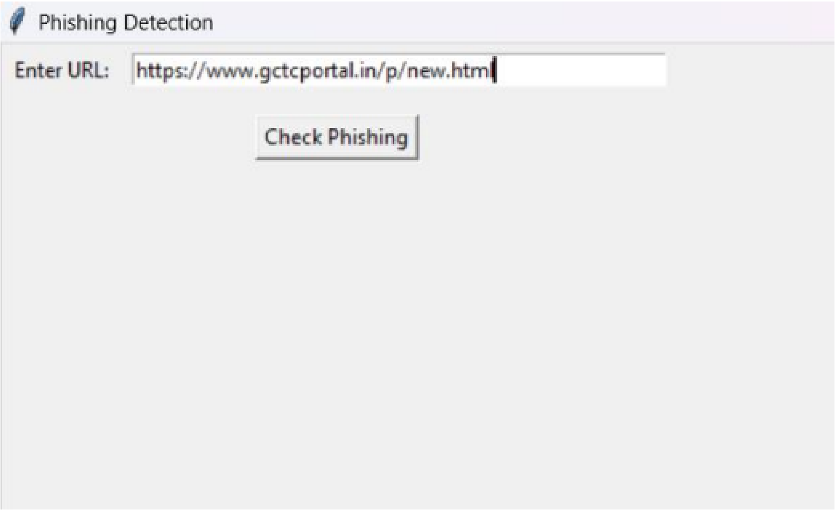
Phishing Detection

Enter URL:

Error

Please enter a URL.

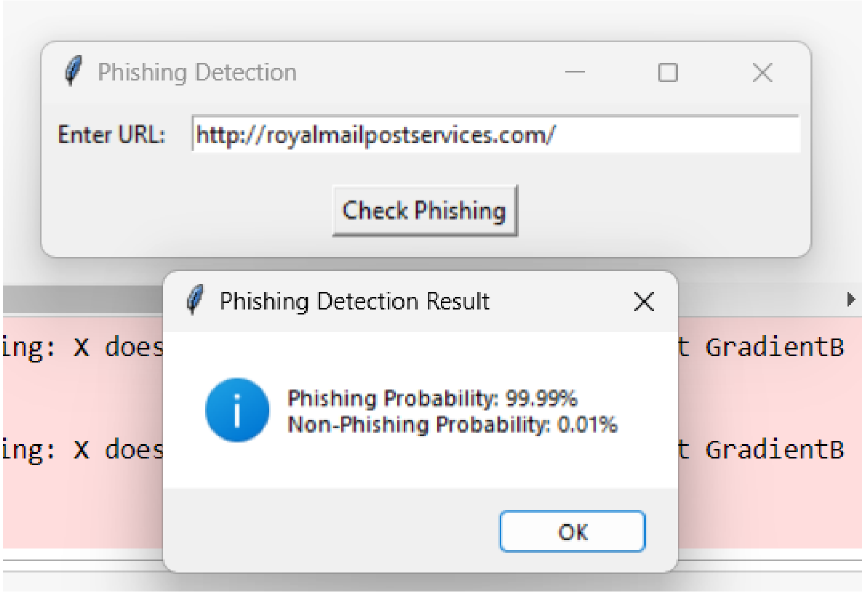
Figure 8.2: output Screen1



Phishing Detection

Enter URL: https://www.gctcportal.in/p/new.html

Figure 8.3: Input Screen

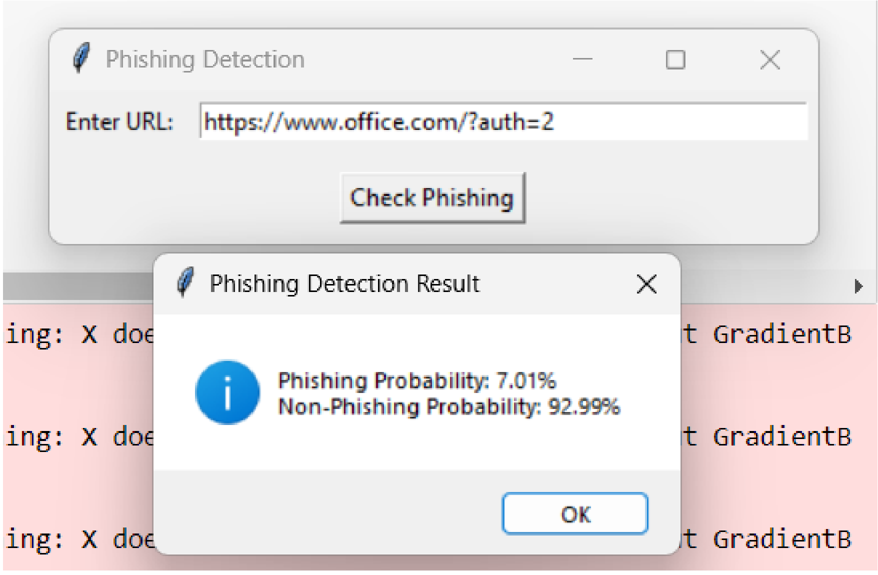


Enter URL: http://royalmailpostservices.com/

Phishing Probability: 99.99%
Non-Phishing Probability: 0.01%

```
ing: X does                         t GradientB
ing: X does                         t GradientB
```

Figure 8.4: output Screen2

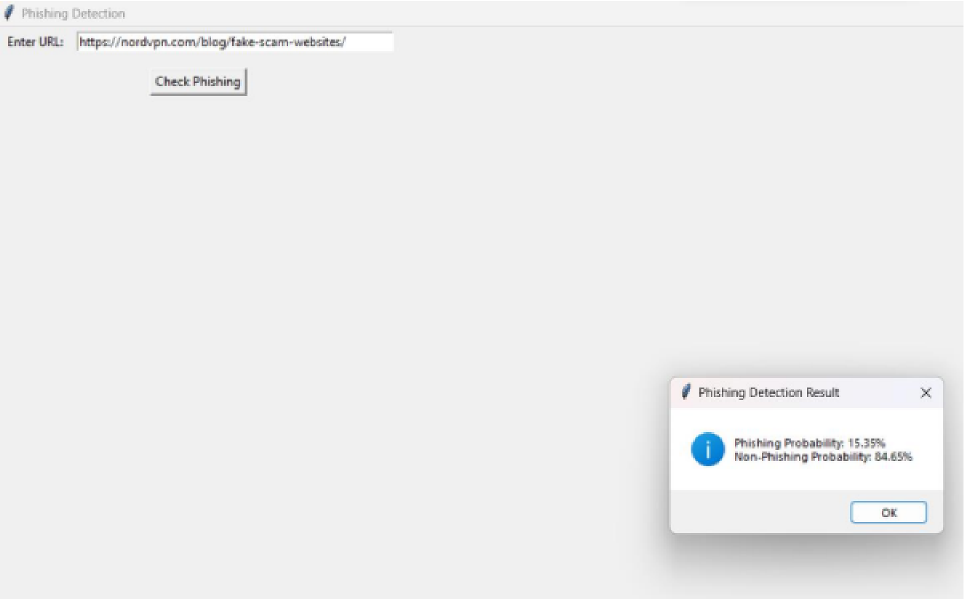


Enter URL: https://www.office.com/?auth=2

Phishing Probability: 7.01%
Non-Phishing Probability: 92.99%

```
ing: X doe                          t GradientB
ing: X doe                          t GradientB
ing: X doe                          t GradientB
```

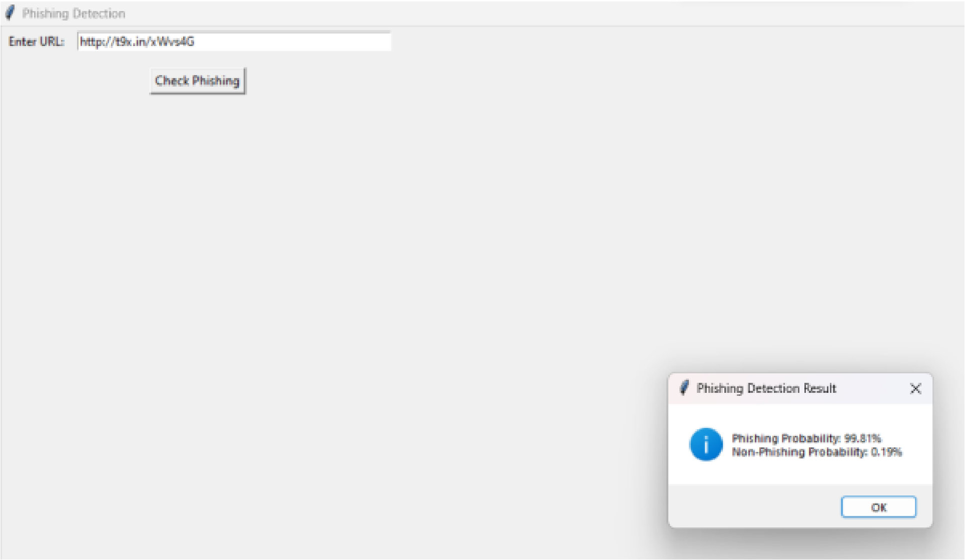
Figure 8.5: output Screen3



Enter URL: https://nordvpn.com/blog/fake-scam-websites/

Phishing Probability: 15.35%
Non-Phishing Probability: 84.65%

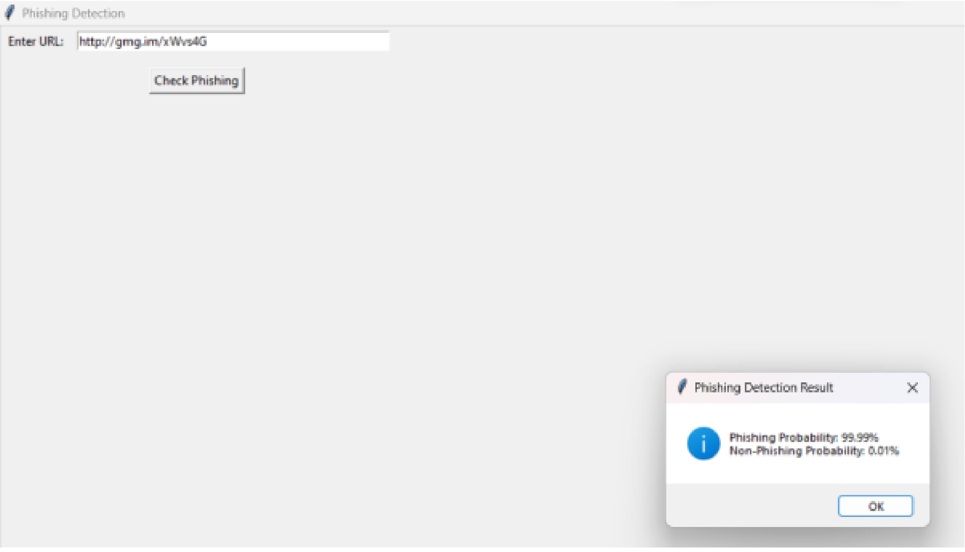
Figure 8.6: output Screen4



Enter URL: http://t9x.in/xWvs4G

Phishing Probability: 99.81%
Non-Phishing Probability: 0.19%

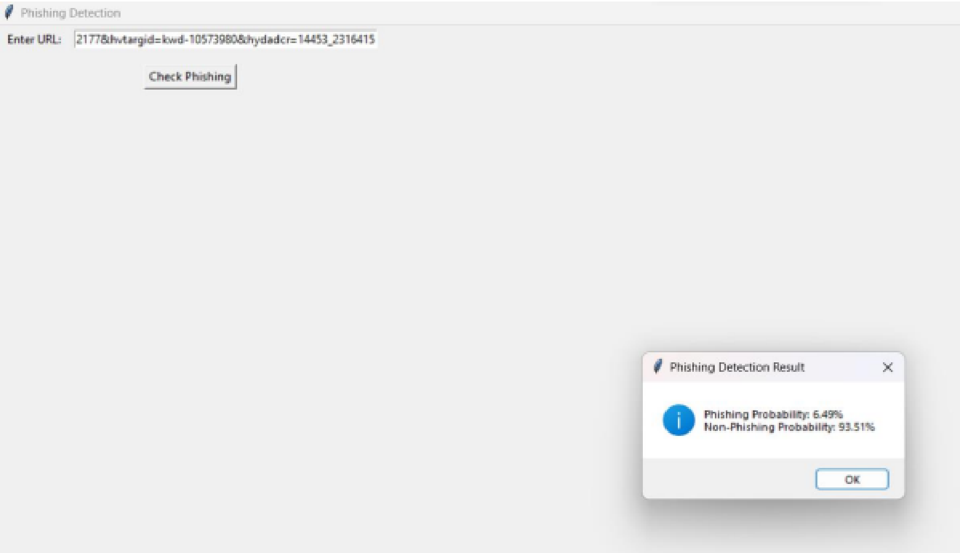
Figure 8.7: output Screen5



Enter URL: http://gmg.im/xWvs4G

Phishing Probability: 99.99%
Non-Phishing Probability: 0.01%

Figure 8.8: output Screen6



Enter URL: 2177&hvtargid=kwd-10573980&hydadcr=14453_2316415

Phishing Probability: 6.49%
Non-Phishing Probability: 93.51%

Figure 8.9: output Screen7

CHAPTER 9

CONCLUSION

The final take away from this project is to explore various machine learning models, perform Exploratory Data Analysis on phishing dataset and understanding their features.

Creating this notebook helped me to learn a lot about the features affecting the models to detect whether URL is safe or not, also I came to know how to tuned model and how they affect the model performance.

In conclusion, this project has successfully addressed the critical issue of phishing URL detection. Through meticulous data preprocessing, the development of a robust machine learning model, and rigorous evaluation, we have achieved commendable results in identifying malicious URLs. Despite encountering challenges in data collection and model optimization, the project's outcomes demonstrate its potential for enhancing online security. The lessons learned in the process, coupled with the extensive scope for future improvements outlined in the 'Future Scope' section, emphasize the significance of this project. By contributing to the fight against phishing attacks, this project holds promise in safeguarding users and organizations from cyber threats, underlining its relevance and impact in today's digital landscape."

The final conclusion on the Phishing dataset is that some feature like "HTTPS", "AnchorURL", "WebsiteTraffic" have more importance to classify URL is phishing URL or not. Gradient Boosting Classifier correctly classify URL up to 97.4respective classes and hence reduces the chance of malicious attachments.

CHAPTER 10

FUTURE SCOPE

Discussing the future scope of your phishing URL detection project is important for showing its potential for further development and improvement. Here are some ideas for future scope in such a project:

1. Enhanced Feature Engineering: Explore more advanced feature engineering techniques to extract and select relevant features from URLs. Consider incorporating natural language processing (NLP) techniques to analyze the textual content of URLs.

2. Deep Learning Architectures: Investigate the use of deep learning models, such as convolutional neural networks (CNNs) or recurrent neural networks (RNNs), for URL classification. These models can capture complex patterns in URLs and text.

3. Ensemble Methods: Experiment with ensemble methods, such as stacking or bagging, to combine the predictions of multiple models. Ensemble methods can often improve model performance.

4. Real-time Detection: Develop a real-time phishing URL detection system that can continuously monitor incoming URLs and flag potential phishing attempts as they happen. This could be useful for web browsers or email clients.

5. User Feedback Integration: Allow users to provide feedback on detected phishing URLs. This feedback can be used to continuously improve the model's performance and adapt to new phishing techniques.

6. Active Learning: Implement active learning techniques to select the most informative samples for manual labelling. This can help reduce the labelling effort required to train and update the model.

7. Deployment as a Service: Transform your model into a web service or API that can be easily integrated into other applications, such as web browsers, email clients, or cybersecurity software.

REFERENCES

[1] Gang Liu, Bite Qiu, and Liu Wenyin. "Automatic Detection of Phishing Target from Phishing Webpage". In: *2010 20th International Conference on Pattern Recognition*. 2010, pp. 4153–4156. DOI: 10.1109/ICPR.2010.1010.

[2] Nuttapong Sanglerdsinlapachai and Arnon Rungsawang. "Using Domain Top-page Similarity Feature in Machine Learning-Based Web Phishing Detection". In: *2010 Third International Conference on Knowledge Discovery and Data Mining*. 2010, pp. 187–190. DOI: 10.1109/WKDD.2010.108.

[3] Ammar Yahya Daeef, R. Badlishah Ahmad, Yasmin Yacob, and Ng Yen Phing. "Wide scope and fast websites phishing detection using URLs lexical features". In: *2016 3rd International Conference on Electronic Design (ICED)*. 2016, pp. 410–415. DOI: 10.1109/ICED.2016.7804679.

[4] Shraddha Parekh, Dhwanil Parikh, Srushti Kotak, and Smita Sankhe. "A New Method for Detection of Phishing Websites: URL Detection". In: *2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT)*. 2018, pp. 949–952. DOI: 10.1109/ICICCT.2018.8473085.