

मौलाना आजाद राष्ट्रीय प्रौद्योगिकी संस्थान - भोपाल
Maulana Azad National Institute of Technology– Bhopal



Department of Computer Science and Engineering
कंप्यूटर विज्ञान और अभियांत्रिकी विभाग

Session - 2025

“MINOR PROJECT REPORT”

On

“Email Phishing Detection”

**Submitted In partial Fulfillment for the degree of Bachelor of Technology in
Computer Science and Engineering**

SUBMITTED BY:

GUIDED BY:

Prafull Patidar (21112451)
Ranvir Singh Chhabra (21112416)
Satyam Verma (21112064)
Hardik Gupta (21112448)

Dr. Meenu Chawla

मौलाना आजाद राष्ट्रीय प्रौद्योगिकी संस्थान - भोपाल
Maulana Azad National Institute of Technology– Bhopal



Department of Computer Science and Engineering
कंप्यूटर विज्ञान और अभियांत्रिकी विभाग

DECLARATION

I hereby declare that the work, which is presented in this Project Report, entitled “**Email Phishing Detection**”, in partial fulfillment of the requirements for the award of the degree, submitted in the **Department of Computer Science and Engineering, Maulana Azad National Institute of Technology, Bhopal**. It is an authentic record of my work carried out from to under the noble guidance of my guide “**Dr. Meenu Chawla**”. The following project and its report, in part or whole, have not been presented or submitted by me for any purpose in any other institute or organization. I hereby declare that the facts mentioned above are true to the best of our knowledge. In case of any unlikely discrepancy that may possibly occur, I will be the one to take responsibility.

Prafull Patidar (21112451)

Ranvir Singh Chhabra (21112416)

Satyam Verma (21112064)

Hardik Gupta (21112448)

मौलाना आजाद राष्ट्रीय प्रौद्योगिकी संस्थान - भोपाल
Maulana Azad National Institute of Technology– Bhopal



Department of Computer Science and Engineering
कंप्यूटर विज्ञान और अभियांत्रिकी विभाग

CERTIFICATE

This is to certify that “**Prafull Patidar, Ranvir Singh Chhabra, Satyam Verma and Hardik Gupta**”, students of B.Tech 3rd Year (Computer Science & Engineering), have successfully completed their project "**Email Phishing Detection**" in partial fulfillment of their Bachelor of Technology in Computer Science & Engineering.

Dr. Meenu Chawla

ACKNOWLEDGEMENT

With due respect, we express our deep sense of gratitude to our respected Guide and project coordinator, “**Dr. Meenu Chawla**” for his valuable help and guidance.

We are thankful for his encouragement in completing this project successfully. His rigorous evaluation and constructive criticism were very helpful.

We are also grateful to our respected director, “**Dr. Karunesh Kumar Shukla**” for permitting us to utilize all the necessary college facilities.

Needless to mention the additional help and support extended by our respected HOD, “**Dr. Deepak Singh Tomar**” in allowing us to use the departmental laboratories and other services.

We are also thankful to all the other faculty, staff members, and laboratory attendants of our department for their cooperation and help.

Last but certainly not least, we would like to express our deep appreciation towards our family members and batch mates for providing much-needed support and encouragement.

Prafull Patidar (211112451)

Ranvir Singh Chhabra (211112416)

Satyam Verma (211112064)

Hardik Gupta (211112448)

Place :

Date :

सारांश

यह अध्ययन ईमेल फिशिंग प्रयासों को पहचानने में समर्थन वेक्टर मशीन (SVM), नैवी बेस, और न्यूरल नेटवर्क क्लासिफायर की प्रभावकारिता की जाँच करता है। ईमेल शरीर की सामग्री, एम्बेडेड लिंक्स, और भेजने वाले की जानकारी जैसी विशेषताओं का उपयोग करके, हमारा दृष्टिकोण ईमेल को फिशिंग और विधानिक वर्गों में वर्गीकृत करने का उद्देश्य रखता है।

"संदेहास्पद भाषा के लिए ईमेल शरीर का संवीक्षण करके, असामान्य यूआरएल के लिए एम्बेडेड लिंक्स की जाँच करके, और भेजने वाले की जानकारी को ध्यान में रखकर, हमारा सुविधा-समृद्ध दृष्टिकोण फिशिंग का पता लगाने की पुष्टि को बढ़ाता है। संवेक्षण विशेषताओं के भीतर जटिल पैटर्न को पकड़ने में एसवीएम क्लासिफायर उत्कृष्ट होता है, नैवी बेस संभावनात्मक मॉडलों का उपयोग करता है, और न्यूरल नेटवर्क गहरी शिक्षा का लाभ उठाता है व्यापक विश्लेषण के लिए।"

"यह अनुसंधान ईमेल सुरक्षा के विकास में योगदान करता है, विभिन्न विशेषताओं पर आधारित फिशिंग प्रयासों का पता लगाने में एसवीएम, नैवी बेस, और न्यूरल नेटवर्क की क्षमताओं का एक सूक्ष्म विवेचन प्रदान करता है। यह फिडिंग्स मजबूत और अनुकूल ईमेल सुरक्षा सिस्टम के विकास का मार्गदर्शन करती है, जो साइबर खतरों के विकास के खिलाफ संरक्षण को बढ़ावा देती है।"

ABSTRACT

This study explores the efficacy of Support Vector Machine (SVM), Naive Bayes, and Neural Network classifiers in detecting email phishing attempts. Leveraging features such as email body content, embedded links, and sender information, our approach aims to categorize emails into phishing and legitimate classes.

By scrutinizing the email body for suspicious language, examining embedded links for anomalous URLs, and considering sender information, our feature-rich approach enhances the precision of phishing detection. The SVM classifier excels in capturing intricate patterns within these features, Naive Bayes utilizes probabilistic models, and Neural Networks leverage deep learning for comprehensive analysis.

This research contributes to the advancement of email security, providing a nuanced understanding of the capabilities of SVM, Naive Bayes, and Neural Networks in detecting phishing attempts based on diverse features. The findings guide the development of robust and adaptive email security systems, enhancing defenses against evolving cyber threats.

ABBREVIATIONS/NOTATION/NOMENCLATURE

This section provides a list of abbreviations, notation, and nomenclature used throughout the project report to ensure clarity and consistency in terminology. The abbreviations, symbols, and terms defined in this section are used in various sections of the report, including the literature review, methodology, results, and discussion.

Abbreviations:

- SVM: Support Vector Machine
- GUI: Graphical User Interface
- F1-score: F1 score, a measure of a model's accuracy that combines precision and recall
- API: Application Programming Interface
- CNN: Convolutional Neural Network
- URL: Uniform Resource Locator
- IP: Internet Protocol
- HTTPS: Hypertext Transfer Protocol Secure
- TREC: Text REtrieval Conference
- DNS: Domain Name System
- HTTP: Hypertext Transfer Protocol

Notation:

- Acc
- Acc : Accuracy
- $Prec$
- $Prec$: Precision
- Rec
- Rec : Recall
- $F1$
- $F1$: F1-score
- TP

- *TP*: True Positive
- TN
- *TN*: True Negative
- FP
- *FP*: False Positive
- FN
- *FN*: False Negative

Nomenclature:

- Phishing: A cyber attack where attackers masquerade as trustworthy entities to deceive individuals into divulging sensitive information.
- Ensemble Learning: A machine learning technique that combines multiple models to improve predictive performance.
- Feature Extraction: The process of selecting and transforming raw data into meaningful features for machine learning algorithms.
- Metadata: Data that provides information about other data, such as the sender, recipient, and timestamp of an email.
- Spear Phishing: A targeted phishing attack that tailors messages to specific individuals or organizations.
- Pretexting: A form of social engineering where attackers create a fabricated scenario to trick individuals into revealing information.
- Usability Testing: A method for evaluating a product's usability by observing how users interact with it and collecting feedback.
- Hyperparameter Tuning: The process of selecting the optimal hyperparameters for a machine learning model to improve its performance.

TABLE OF CONTENTS

DESCRIPTION	PAGE NUMBER
DECLARATION	2
CERTIFICATE	3
ACKNOWLEDGEMENT	4
ABSTRACT	6
ABBREVIATION/ NOTATION/ NOMENCLATURE	7
1. INTRODUCTION	12
1.1 Background	
1.2 Objective	
1.3 Scope	
2. LITERATURE REVIEW	16
2.1 Phishing Attacks	
2.2 Email Features for Detection	
3. METHODOLOGY	19
3.1 Data Collection	
3.2 Feature Extraction	
3.3 Dataset Description	
3.4 Model Training	
3.4.1 Naive Bayes Classifier	
3.4.2 Support Vector Machine	
3.4.3 Random Forest Classifier	
3.5 Code	
4. FEATURE EXTRACTION	25
4.1 Body Features	
4.1.1 body_forms_body_html	
4.1.2 body_noCharacters	
4.1.3 body_noDistinct Words	
4.1.4 body_noFunctionWords	
4.1.5 body_noWords	

- 4.1.6 Body_richness
- 4.1.7 Body_suspension
- 4.1.8 body_verifyYourAccount
- 4.2 Script Features
 - 4.2.1 script_javascript
 - 4.2.2 script_noOnClickEvents
 - 4.2.3 script_nonModalJsLoads
 - 4.2.4 script_popups
 - 4.2.5 Script_scripts
 - 4.2.6 script_statusChange
- 4.3 Sender Features
 - 4.3.1 send_diffSenderReplyTo
 - 4.3.2 send_noCharacters
 - 4.3.3 send_noWords
 - 4.3.4 send_nonModalSenderDomain
- 4.4 Subject Features
 - 4.4.1 subj_bankSubj_debitsubj_forwardsubj_no_Chars
 - 4.4.2 subj_no_Chars
 - 4.4.3 subj_no_Words
 - 4.4.4 Subj_reply
 - 4.4.5 Subj_richness
 - 4.4.6 subj_verify
- 4.5 URL Features
 - 4.5.1 url_atSymbol
 - 4.5.2 url_ipAddress
 - 4.5.3 url_linkText
 - 4.5.4 url_max_NoPeriods
 - 4.5.5 url_noDomains
 - 4.5.6 url_noExtLinks
 - 4.5.7 url_noImgLinks
 - 4.5.8 url_noIntLinks

4.5.9 url_noIpAddress	
4.5.10 url_noLinks	
4.5.11 url_noPorts	
4.5.12 url_nonModalHereLinks	
4.5.13 Url_ports	
4.6 Code	
5. Model Training and Evaluation	37
5.1 Naive Bayes Classifier	
5.2 Support Vector Machine (SVM)	
5.3 Random Forest Classifier	
5.4 Ensemble Learning	
6. Graphical User Interface (GUI)	41
6.1 Implementation using Tkinter	
6.2 Functionality	
7. Results	43
7.1 Performance Metrics	
7.2 GUI Output	
8. Discussion	46
8.1 Interpretation of Results	
8.2 Strengths and Limitations	
8.3 Future Work	
9. Conclusion	49
9.1 Key Findings and Contributions	
9.2 Final Remarks	
10. References	50

CHAPTER 1

INTRODUCTION

1.1 Background

Phishing attacks have become a prevalent cybersecurity threat, exploiting human psychology to deceive individuals into revealing sensitive information through fraudulent emails. Initially simplistic, phishing techniques have evolved in sophistication over time. As we can see the number of phishing attacks from 2020 May to 2022 April have increased significantly (refer figure 1.1.1), posing challenges for detection and mitigation.



Figure 1.1.1

Attackers employ various tactics, including spear phishing and pretexting, to craft convincing emails that appear legitimate. The accessibility of phishing tools on the dark web has further democratized the threat, enabling even non-technical individuals to orchestrate sophisticated attacks.

In response, cybersecurity measures such as email filtering and user education programs have been developed. However, phishing remains a persistent threat, highlighting the need for advanced detection techniques. The Percentage of phishing attacks on different industries in the first Quarter of 2022 is shown in figure 1.1.2.

This project addresses this need by leveraging machine learning algorithms to develop an automated system for phishing email detection, enhancing cybersecurity and protecting users from falling victim to these malicious schemes.

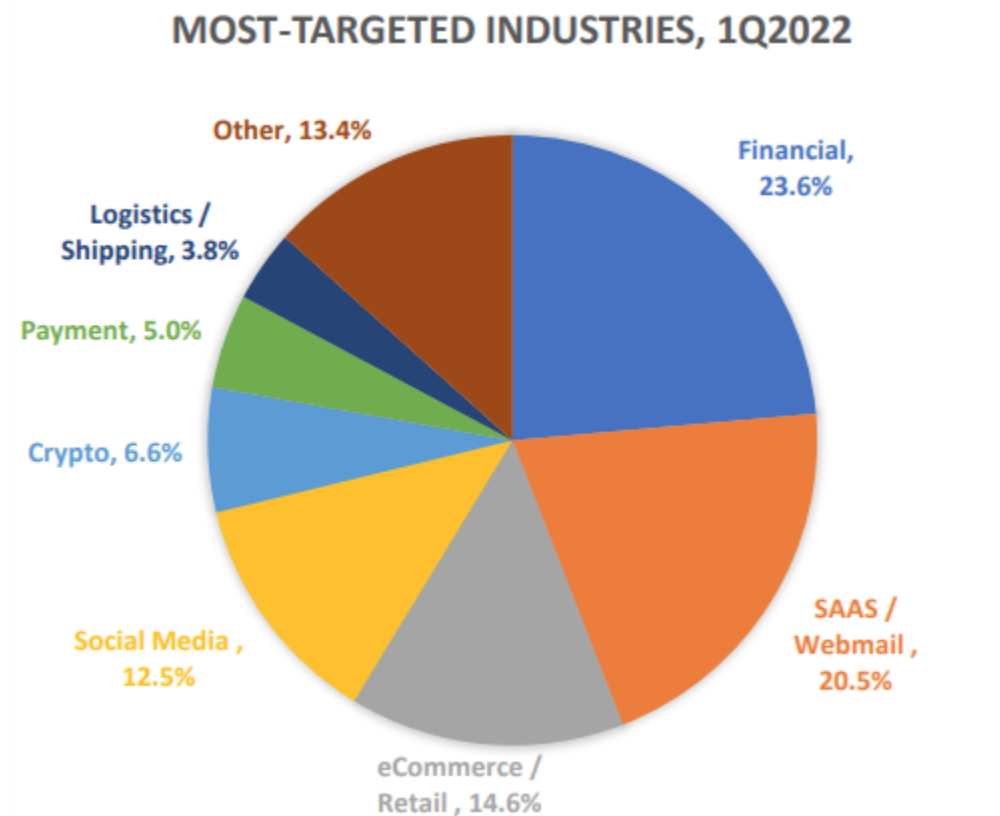


Figure 1.1.2

1.2 Objective

The objective of this project is to develop a comprehensive and effective automated system for the detection of phishing emails using machine learning techniques and feature extraction methods. The primary goal is to enhance cybersecurity and mitigate the risks associated with phishing attacks by accurately identifying fraudulent emails in real-time. This project aims to achieve the following specific objectives:

- **Feature Extraction and Selection:** Implement advanced feature extraction techniques to capture relevant information from email content, metadata, and sender attributes. Select and prioritize features based on their discriminative power and relevance to phishing detection.
- **Model Development and Training:** Develop and train machine learning models, including Naive Bayes, Support Vector Machine (SVM), and Random Forest, using labeled datasets of phishing and non-phishing emails. Optimize model parameters and hyperparameters to maximize classification accuracy and minimize false positives.
- **Ensemble Learning Integration:** Implement an ensemble learning approach to combine the predictions of multiple classifiers for improved accuracy and robustness. Explore techniques such as majority voting, stacking, and boosting to leverage the strengths of individual classifiers and mitigate their weaknesses.
- **Graphical User Interface (GUI) Development:** Design and implement a user-friendly GUI using the Tkinter library in Python to provide users with an intuitive platform for interacting with the phishing detection system. Enable users to input email content, visualize classification results, and interpret model predictions in real-time.
- **Performance Evaluation and Validation:** Evaluate the performance of the developed system using standard metrics such as accuracy, precision, recall, and F1-score. Validate the system's effectiveness in detecting phishing emails on diverse datasets and under various conditions, including different types of phishing attacks and levels of sophistication.

1.3 Scope

This project focuses on the implementation of a machine learning-based approach to phishing detection, utilizing a diverse set of features extracted from email datasets. Additionally, the development of a graphical user interface (GUI) provides users with a user-friendly tool for interacting with the detection system.

Phishing emails are fraudulent messages designed to deceive recipients into revealing sensitive information or performing malicious actions, such as clicking on malicious links or providing personal data.

Overall, this project aims to provide a comprehensive solution for phishing detection, combining machine learning techniques with a user-friendly interface to help users identify and mitigate phishing threats more effectively.

CHAPTER 2

LITERATURE REVIEW

2.1 Phishing Attacks

Phishing attacks have emerged as one of the most pervasive and damaging cybersecurity threats in recent years. These attacks exploit human vulnerabilities rather than technical weaknesses, making them difficult to detect and mitigate effectively. Phishing attacks typically involve the use of deceptive emails, social engineering tactics, and fraudulent websites to trick individuals into revealing sensitive information such as login credentials, financial data, or personal details.

Over the years, phishing attacks have evolved in sophistication, ranging from generic spam emails to highly targeted and convincing campaigns tailored to specific individuals or organizations. Attackers often employ psychological manipulation techniques to create a sense of urgency or fear, prompting victims to act impulsively without questioning the legitimacy of the request.

The consequences of falling victim to a phishing attack can be severe, ranging from financial losses and identity theft to reputational damage and legal repercussions. As such, phishing attacks pose a significant risk to individuals, businesses, and governments worldwide, highlighting the importance of robust cybersecurity measures and proactive defense strategies.

2.2 Email Features for Detection

Detecting phishing emails requires analyzing various features extracted from email content, metadata, and sender information. These features serve as indicators of potential phishing activity and can help differentiate between legitimate and fraudulent emails. Key features used for phishing detection include:

- **Textual Analysis:** Analyzing the body text, subject lines, and URLs embedded within emails to identify patterns, anomalies, and indicators of phishing activity. This may involve detecting suspicious language, grammatical errors, spelling mistakes, and inconsistencies in formatting.

- **Metadata Examination:** Examining metadata attributes such as sender information, domain reputation, email headers, and routing information to assess the legitimacy of the sender and the authenticity of the email.
- **URL Analysis:** Scrutinizing URLs contained within emails to identify phishing websites, malicious domains, and deceptive redirects. This may involve analyzing domain names, IP addresses, HTTPS encryption, and the presence of phishing indicators such as misspelled domains or suspicious characters.
- **Behavioral Patterns:** Identifying behavioral patterns and interaction cues indicative of phishing activity, such as click-through rates, response times, and user engagement metrics. This may involve tracking user interactions with email content and monitoring for deviations from normal behavior.
- **Machine Learning Techniques:** Leveraging machine learning algorithms to automatically classify emails based on learned patterns and features. Supervised learning, unsupervised learning, and ensemble learning approaches can be used to train models on labeled datasets and predict the likelihood of an email being a phishing attempt.

2.3 Machine Learning in Phishing Detection

Machine learning techniques have gained prominence in phishing detection due to their ability to learn from data, adapt to changing threats, and automate the detection process. Various machine learning algorithms and techniques have been applied to phishing detection tasks, including:

- **Naive Bayes:** A probabilistic classifier based on Bayes' theorem with an assumption of feature independence. Naive Bayes is well-suited for text classification tasks and can effectively classify emails as phishing or non-phishing based on extracted features.
- **Support Vector Machine (SVM):** A supervised learning algorithm capable of performing classification tasks in high-dimensional spaces. SVM constructs an optimal hyperplane that separates data points into different classes, making it effective for distinguishing between phishing and non-phishing emails.
- **Random Forest:** An ensemble learning method that operates by constructing multiple decision trees during training and outputting the mode of the classes of the individual trees. Random Forest is known for its high accuracy, robustness to noise, and ability to handle large datasets, making it suitable for phishing detection tasks.

Additionally, deep learning techniques such as neural networks and convolutional neural networks (CNNs) have shown promise in phishing detection, particularly in tasks involving image-based phishing and multimedia content analysis. These techniques leverage the power of deep learning to automatically extract hierarchical features from raw data and make predictions with high accuracy.

In recent years, research efforts have focused on developing hybrid and ensemble approaches that combine multiple machine learning algorithms and feature sets to improve detection accuracy and resilience to adversarial attacks. These approaches leverage the strengths of individual algorithms and feature sets to achieve superior performance in real-world phishing detection scenarios.

Overall, the literature on phishing detection highlights the importance of leveraging machine learning techniques, feature engineering, and behavioral analysis to develop effective detection systems capable of mitigating the growing threat of phishing attacks. Ongoing research in this field aims to further enhance the accuracy, efficiency, and scalability of phishing detection systems to protect users and organizations from evolving cyber threats.

CHAPTER 3

METHODOLOGY

3.1 Data Collection

The process of building a machine learning model for email phishing detection begins with the acquisition and preparation of a labeled dataset. In this case, a dataset named 'Spam Assassin' sourced from Kaggle serves as the foundation for our analysis. This dataset comprises a diverse array of emails, each accompanied by a set of features encompassing various aspects such as body text, sender information, subject lines, and URLs.

3.2 Feature Extraction

Features are extracted from the email dataset using predefined extraction methods. These features capture various aspects of email content and metadata, providing valuable input for the machine learning models.

3.3 Dataset Description

The dataset consists of a substantial number of labeled emails, with each email associated with a target label indicating its classification as phishing (1) or non-phishing (0). The dataset is split into training and testing sets for model development and evaluation.

Reference : <https://spamassassin.apache.org/downloads.html>

Dataset	Total emails	Phished emails	Legitimate emails
SpamAssassin	6047	1897	4150

3.4 Model Training

Machine learning models, including Naive Bayes, SVM, and Random Forest, are trained on the labeled dataset using the extracted features. These models learn to distinguish between phishing and non-phishing emails based on the patterns present in the training data

The labeled dataset is typically divided into two subsets: a training set and a testing set. The training set is used to train the machine learning models, while the testing set is used to evaluate the models' performance on unseen data. The split ratio is 70-30 where 70% of the data is used for training and the remaining 30% is used for testing, respectively..

3.4.1 Naive Bayes Classifier

The Naive Bayes classifier is a probabilistic algorithm based on Bayes' theorem with an assumption of independence between features. It is computationally efficient and works well with high-dimensional data. Naive Bayes is particularly suitable for text classification tasks, making it a popular choice for email phishing detection. Its advantages include:

- **Simplicity:** Naive Bayes is easy to implement and understand, making it suitable for rapid prototyping.
- **Fast Training:** The algorithm's training time is typically faster compared to more complex models, making it scalable to large datasets.
- **Robustness to Irrelevant Features:** Naive Bayes can handle irrelevant features effectively due to its assumption of feature independence.

3.4.2 Support Vector Machine (SVM)

SVM is a powerful supervised learning algorithm capable of performing classification, regression, and outlier detection tasks. It works by finding the optimal hyperplane that separates data points into different classes. SVM is effective in high-dimensional spaces and is memory-efficient. Its advantages include:

- **Effective in High-Dimensional Spaces:** SVM performs well even in cases where the number of dimensions exceeds the number of samples.
- **Versatility:** SVM can handle linear and nonlinear classification tasks through the use of different kernel functions.
- **Robustness to Overfitting:** SVM has regularization parameters that help prevent overfitting, ensuring generalization to unseen data.

3.4.3 Random Forest Classifier

Random Forest is an ensemble learning method that operates by constructing a multitude of decision trees during training and outputting the class that is the mode of the classes of the individual trees. It provides high accuracy, robustness to noise, and handles large datasets efficiently. Its advantages include:

- High Accuracy: Random Forest typically produces accurate results by averaging predictions from multiple decision trees.
- Robustness to Overfitting: The ensemble nature of Random Forest helps reduce overfitting, resulting in more robust models.
- Feature Importance: Random Forest provides a measure of feature importance, allowing users to identify the most discriminative features in the dataset.

These classification algorithms are trained on the labeled dataset and contribute to the ensemble learning approach for improved phishing email detection accuracy.

3.5 CODE

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, confusion_matrix
from sklearn.impute import SimpleImputer
import matplotlib.pyplot as plt

# Load the extracted features CSV file
df = pd.read_csv("extracted_features1.csv")

# Handle missing values by imputing with the median value of each column
imputer = SimpleImputer(strategy='median')
df_imputed = pd.DataFrame(imputer.fit_transform(df), columns=df.columns)

# Split the dataset into features (X) and the target variable (y)
X = df_imputed.drop(columns=['target'])
y = df_imputed['target']

# Split the data into training and testing sets
```

```

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Initialize classifiers
nb_classifier = GaussianNB()
svm_classifier = SVC()
rf_classifier = RandomForestClassifier()

# Train classifiers
nb_classifier.fit(X_train, y_train)
svm_classifier.fit(X_train, y_train)
rf_classifier.fit(X_train, y_train)

# Make predictions on the testing set using each classifier
nb_pred = nb_classifier.predict(X_test)
svm_pred = svm_classifier.predict(X_test)
rf_pred = rf_classifier.predict(X_test)

# Ensemble prediction: Take the majority vote
ensemble_pred = []
for nb, svm, rf in zip(nb_pred, svm_pred, rf_pred):
    majority_vote = sum([nb, svm, rf]) >= 2 # Majority vote
    ensemble_pred.append(majority_vote)

# Evaluation metrics
ensemble_accuracy = accuracy_score(y_test, ensemble_pred)
precision = precision_score(y_test, ensemble_pred)
recall = recall_score(y_test, ensemble_pred)
f1 = f1_score(y_test, ensemble_pred)
conf_matrix = confusion_matrix(y_test, ensemble_pred)

# Individual classifier metrics
nb_accuracy = accuracy_score(y_test, nb_pred)
nb_precision = precision_score(y_test, nb_pred)
nb_recall = recall_score(y_test, nb_pred)
nb_f1 = f1_score(y_test, nb_pred)

svm_accuracy = accuracy_score(y_test, svm_pred)
svm_precision = precision_score(y_test, svm_pred)
svm_recall = recall_score(y_test, svm_pred)
svm_f1 = f1_score(y_test, svm_pred)

rf_accuracy = accuracy_score(y_test, rf_pred)
rf_precision = precision_score(y_test, rf_pred)
rf_recall = recall_score(y_test, rf_pred)
rf_f1 = f1_score(y_test, rf_pred)

# Print individual classifier metrics
print("\nIndividual Classifier Metrics:")

```

```

print("\nNaive Bayes Classifier:")
print("Accuracy:", nb_accuracy)
print("Precision:", nb_precision)
print("Recall:", nb_recall)
print("F1-score:", nb_f1)

print("\nSupport Vector Machine Classifier:")
print("Accuracy:", svm_accuracy)
print("Precision:", svm_precision)
print("Recall:", svm_recall)
print("F1-score:", svm_f1)

print("\nRandom Forest Classifier:")
print("Accuracy:", rf_accuracy)
print("Precision:", rf_precision)
print("Recall:", rf_recall)
print("F1-score:", rf_f1)

```

Justification of the Libraries Used :

1. Pandas (import pandas as pd):

- Pandas is a powerful data manipulation library in Python. It provides data structures and functions to work with structured data, such as data frames. In this code, Pandas is used to read the dataset from a CSV file (`pd.read_csv`) and handle missing values by imputing the median value of each column (`SimpleImputer`).

2. Scikit-learn (from sklearn...):

- Scikit-learn is a popular machine learning library in Python. It provides a wide range of algorithms for classification, regression, clustering, dimensionality reduction, and more. In this code, Scikit-learn is used for:
 - Splitting the dataset into training and testing sets (`train_test_split`).
 - Implementing three classification algorithms: Naive Bayes (`GaussianNB`), Support Vector Machine (`SVC`), and Random Forest (`RandomForestClassifier`).
 - Evaluating the performance of the classifiers using metrics such as accuracy, precision, recall, and F1-score (`accuracy_score`, `precision_score`, `recall_score`, `f1_score`).
 - Handling missing values in the dataset (`SimpleImputer`).

3. Matplotlib (import matplotlib.pyplot as plt):

- Matplotlib is a plotting library in Python that provides a MATLAB-like interface for creating static, interactive, and animated visualizations. In this code, Matplotlib is used to plot a bar graph comparing the accuracy of different classifiers. Visualizations help in understanding and interpreting the results of machine learning experiments.

CHAPTER 4

FEATURE EXTRACTION

Feature extraction is not merely a preliminary step but a pivotal process in constructing effective machine learning models tailored for email phishing detection. In this section, we delve into the intricacies of feature extraction from the original dataset, elucidating the nuanced characteristics that are instrumental in discerning between phishing and legitimate emails.

The significance of feature extraction lies in its role as a bridge between raw data and actionable insights. Raw email data, comprising various components such as headers, content, attachments, and embedded links, necessitates transformation into a structured representation that encapsulates pertinent information indicative of phishing behavior. Through meticulous feature extraction, we distill these disparate elements into a cohesive set of discriminative attributes, facilitating the discernment of subtle patterns and anomalies characteristic of phishing attempts.

The features extracted from the original dataset encompass a diverse array of dimensions, each offering unique insights into the underlying nature of emails. Header features, for instance, furnish invaluable metadata pertaining to the origin, transmission, and routing of emails. Analysis of sender information, received headers, and timestamps enables the identification of anomalous patterns indicative of phishing activities, such as spoofed identities, suspicious transmission paths, and temporal irregularities.

The features extracted as part of this process include:

4.1 Body Features

4.1.1 `body_forms_body_html`

Returns true if the HTML body contains a form.

4.1.2 `body_noCharacters`

Returns the number of characters in the body.

4.1.3 `body_noDistinct Words`

Returns the number of distinct words in the body.

4.1.4 body_noFunctionWords

Returns the number of function words in the body.

4.1.5 body_noWords

Returns the number of words in the body.

4.1.6 Body_richness

Calculates and returns the richness of the text in the body.

4.1.7 Body_suspension

Returns TRUE if the body contains the word “suspension”.

4.1.8 body_verifyYourAccount

Returns TRUE if the body contains the phrase “verify your account”.

4.2 Script Features

4.2.1 script_javascript

Returns TRUE if JavaScript is present in the email.

4.2.2 script_noOnClickEvents

Returns the number of onclick events in the script.

4.2.3 script_nonModalJsLoads

Returns TRUE if JavaScript comes from outside the modal domain.

4.2.4 script_popups

Returns TRUE if the email contains pop-up window code.

4.2.5 Script_scripts

Returns TRUE if scripts are present in the email body.

4.2.6 script_statusChange

Returns TRUE if the script overrides the status bar in the email client.

4.3 Sender Features

4.3.1 send_diffSenderReplyTo

Returns TRUE if the sender and reply-to domain are different.

4.3.2 send_noCharacters

Returns the number of characters in the sender's address.

4.3.3 send_noWords

Returns the number of words in the sender's address.

4.3.4 send_nonModalSenderDomain

Returns TRUE if the sender's and email's modal domain are different.

4.4 Subject Features

4.4.1 subj_bankSubj_debitsubj_forwardsobj_no_Characters

Returns TRUE if the subject contains the word “bank”.

4.4.2 subj_no_Characters

Returns the number of characters in the subject.

4.4.3 subj_no_Words

Returns the number of words in the subject.

4.4.4 Subj_reply

Returns TRUE if the email is being replied to any of the previous mails.

4.4.5 Subj_richness

Calculates and returns the richness of the subject.

4.4.6 subj_verify

Returns TRUE if the body contains the word “verify”.

4.5 URL Features

4.5.1 url_atSymbol

Returns TRUE if the "@" symbol is present in the URL.

4.5.2 url_ipAddress

Returns TRUE if there is use of an IP address instead of a domain name in the URL.

4.5.3 url_linkText

Returns TRUE if the link text contains "click", "here", "login", or "update terms".

4.5.4 url_max_NoPeriods

Returns the number of periods in the link with the highest number of periods.

4.5.5 url_noDomains

Returns the number of URL domains in the email body.

4.5.6 url_noExtLinks

Returns the number of external links in the email body.

4.5.7 url_noImgLinks

Returns the number of image links in the email body.

4.5.8 url_noIntLinks

Returns the number of internal links in the email body.

4.5.9 url_noIpAddress

Returns the number of links in the email that contain an IP address.

4.5.10 url_noLinks

Returns the number of links in the email body.

4.5.11 url_noPorts

Returns the number of links with port information.

4.5.12 url_nonModalHereLinks

Returns TRUE if 'here' links don't map to the modal domain.

4.5.13 Url_ports

Returns TRUE if the URL accesses ports other than 80.

4.6 CODE

```
import pandas as pd  
  
import csv  
  
import re
```

```

from urllib.parse import urlparse

# Function to extract features from the text

def extract_features(text):

    features = {}

    # Features related to body text

    features['body_forms_body_html'] = bool(re.search(r'<form>', text,
re.IGNORECASE))

    features['body_noCharacters'] = len(text)

    features['body_noDistinct Words'] = len(set(text.split()))

    function_words = ["and", "but", "if", "or", "because", "when",
"what", "who", "which", "how", "where", "whom",

                    "whose", "whether", "why", "that", "since",
"until", "after", "before", "although", "though",

                    "even", "as", "if", "once", "while"]

    features['body_noFunctionWords'] = sum(1 for word in text.split() if
word.lower() in function_words)

    features['body_noWords'] = len(text.split())

    features['body_richness'] = features['body_noWords'] /
features['body_noDistinct Words'] if features[

'body_noDistinct Words'] > 0 else 0

    features['body_suspension'] = bool(re.search(r'\bsuspension\b', text,
re.IGNORECASE))

```

```

features['body_verifyYourAccount'] = bool(re.search(r'\bverify your
account\b', text, re.IGNORECASE))

# Features related to sender's address

sender_domain = re.search(r'From:.*<(\S+)>', text)

if sender_domain:

    sender_domain = sender_domain.group(1)

    features['send_noCharacters'] = len(sender_domain)

    features['send_noWords'] = len(sender_domain.split())

# Initialize modal_domain

modal_domain = None

if sender_domain:

    modal_domain = urlparse(sender_domain).netloc

# Features related to URLs in the text

urls =
re.findall(r'http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[*%\(\)\,]|(?:%[0-9
a-fA-F][0-9a-fA-F]))+', text)

if urls:

    features['url_noLinks'] = len(urls)

    if modal_domain:

        features['url_noExtLinks'] = sum(1 for url in urls if
urlparse(url).netloc != modal_domain)

        features['url_noIntLinks'] = sum(1 for url in urls if
urlparse(url).netloc == modal_domain)

```

```

        features['url_noDomains'] = len(set(urlparse(url).netloc for
url in urls))

        features['url_noImgLinks'] = sum(

            1 for url in urls if
re.search(r'\b(\.jpg|\.jpeg|\.png|\.gif|\.bmp|\.tiff)\b', url,
re.IGNORECASE))

        features['url_noIpAddress'] = sum(

            1 for url in urls if
re.match(r'\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}', urlparse(url).netloc))

        features['url_nonModalHereLinks'] = sum(1 for url in urls if

re.search(r'\bhere\b', url, re.IGNORECASE) and urlparse(

                                                                    url).netloc !=
modal_domain)

        features['url_atSymbol'] = sum(1 for url in urls if '@' in
url)

        features['url_ipAddress'] = sum(

            1 for url in urls if
re.match(r'\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}', urlparse(url).netloc))

        features['url_linkText'] = sum(

            1 for url in urls if
re.search(r'\b(click|here|login|update terms)\b', url, re.IGNORECASE))

        features['url_max_NoPeriods'] = max(url.count('.') for url in
urls)

        features['url_ports'] = sum(1 for url in urls if ':' in
urlparse(url).netloc)

    return features

```



```

# Function to read the dataset and extract features

def process_dataset(dataset_path):

    dataset = pd.read_csv(dataset_path)

    extracted_data = []

    for text, label in zip(dataset['text'], dataset['target']):

        features = extract_features(text)

        features['target'] = label # Include the label in the features

        extracted_data.append(features)

    return extracted_data


# Function to save extracted features to a new CSV

def save_features_to_csv(features, output_path):

    # Get the field names from the first feature dictionary

    fieldnames = features[0].keys()

    # Update the field names to include all features present in any data
point

    all_fieldnames = set()

    for feat in features:

        all_fieldnames.update(feat.keys())

    # Open the CSV file for writing

    with open(output_path, 'w', newline='') as csvfile:

```

```

        writer = csv.DictWriter(csvfile, fieldnames=all_fieldnames)

        writer.writeheader()

        # Write the features to the CSV file

        writer.writerows(features)

# Main function

def main():

    # Path to the labeled dataset CSV file

    dataset_path = "spam_assassin.csv"

    # Path for saving the extracted features CSV file

    output_path = "extracted_features1.csv"

    # Process the dataset and extract features

    extracted_features = process_dataset(dataset_path)

    # Save extracted features to CSV

    save_features_to_csv(extracted_features, output_path)

if __name__ == "__main__":

```

```
main()
```

OUTPUT :- Refer figure 4.1 for Snapshot of generated CSV file containing 23 features (after Combining) along with target label.

url_noIpAc	url_atSymt	body_sus	send_noW	url_ipAddr	url_ports	body_form	url_nonMo	body_noC	url_noLink	url_noDom	url_noImg	url_noIntLi	url_linkTex	url_noExtL	body_noW	send_noCl	body_verif	body_noD	body_noF	body_richr	tar
		FALSE	1			FALSE		4098	2						558	24	FALSE	336	23	1.660714	
		FALSE	1			FALSE		2189	2						295	17	FALSE	203	15	1.453202	
0	0	FALSE	1	0	0	FALSE	0	3598	7	2	0	4	0	3	386	31	FALSE	253	6	1.525692	
		FALSE	1			FALSE		1918	2						153	5	FALSE	120	0	1.275	
		FALSE	1			FALSE		3060	1						399	13	FALSE	233	5	1.712446	
		FALSE	1			FALSE		2017	2						273	21	FALSE	182	12	1.5	
		FALSE				FALSE		2003	1						231		FALSE	170	5	1.358824	
		FALSE	1			FALSE		3538	5						369	22	FALSE	248	5	1.487903	
		FALSE	1			FALSE		3938	4						373	18	FALSE	220	11	1.695455	
		FALSE	1			FALSE		5962							554	5	FALSE	337	4	1.643917	
		FALSE	1			FALSE		1973	4						255	31	FALSE	176	8	1.448864	
		FALSE	1			FALSE		3207	2						413	13	FALSE	236	7	1.75	
		FALSE	1			FALSE		6008	9						719	22	FALSE	366	21	1.964481	
		FALSE	1			FALSE		3054							438	23	FALSE	273	16	1.604396	
		FALSE	1			FALSE		1826	3						188	5	FALSE	139	1	1.352518	
		FALSE	1			FALSE		3267	14						260	6	FALSE	205	4	1.268293	
		FALSE	1			FALSE		7778	13						879	5	FALSE	439	17	2.002278	
		FALSE	1			FALSE		4465	10						462	22	FALSE	253	5	1.826087	
		FALSE	1			FALSE		5437	2						404	5	FALSE	277	1	1.458484	
0	0	FALSE	1	0	0	FALSE	0	4957	7	5	0	1	0	6	438	62	FALSE	292	10	1.5	
		FALSE	1			FALSE		2876							373	42	FALSE	201	15	1.855721	
0	0	FALSE	1	0	0	FALSE	0	3868	7	2	0	6	0	1	442	31	FALSE	287	9	1.54007	
		FALSE	1			FALSE		5248	5						788	13	FALSE	439	33	1.794989	
		FALSE	1			FALSE		1528	1						151	2	FALSE	109	1	1.385321	
		FALSE	1			FALSE		4355	5						475	22	FALSE	281	15	1.690391	
		FALSE	1			FALSE		1962	3						249	31	FALSE	178	8	1.398876	
		FALSE	1			FALSE		4476	7						453	22	FALSE	276	9	1.641304	
		FALSE	1			FALSE		3394							395	10	FALSE	262	9	1.507634	
		FALSE	1			FALSE		3751	5						482	15	FALSE	288	14	1.673611	
		FALSE	1			FALSE		11145	16						1108	5	FALSE	467	21	2.372591	
0	0	FALSE	1	0	0	FALSE	0	28537	2						4971	23	FALSE	1817	326	2.735828	
		FALSE				FALSE	0	6486	5	4	0	1	0	4	730	67	FALSE	407	39	1.793612	

Figure 4.1

Justification of the Libraries Used :

1. Pandas (import pandas as pd):

- Pandas is a powerful data manipulation library in Python. It provides data structures and functions to work with structured data, such as data frames. In this code, Pandas is used to read the dataset from a CSV file (pd.read_csv), manipulate and analyze the data, and eventually save the extracted features to a new CSV file.

2. CSV (import csv):

- The CSV module provides functionality to read from and write to CSV files. In this code, it's used to write the extracted features to a new CSV file.

3. Regular Expressions (import re):

- Regular expressions are a powerful tool for pattern matching and text processing. In this code, regular expressions are used to search for specific patterns in the text data, such as detecting URLs, certain keywords, and sender's email domains.

4. Urllib.parse (from urllib.parse import urlparse):

- The urlparse function from the urllib.parse module is used to parse URLs and extract components like the network location (domain), path, scheme, etc. In this code, it's used to extract domain information from sender's email addresses and URLs.

5. Matplotlib (import matplotlib.pyplot as plt):

- Matplotlib is a plotting library in Python that provides a MATLAB-like interface for creating static, interactive, and animated visualizations. Although not directly used in the provided code snippet, it's a commonly used library for visualizing data and could be utilized for visualizing extracted features or any analysis results in future iterations of the code.

CHAPTER 5

MODEL TRAINING AND EVALUATION

5.1 Naive Bayes Classifier

The Naive Bayes classifier is a probabilistic model trained on the extracted features to distinguish between phishing and legitimate emails. It operates under the assumption of feature independence, simplifying computations while maintaining respectable performance. During training, the classifier learns the probability distributions associated with different feature values for each class.

Once trained, the classifier applies Bayes' theorem to calculate the posterior probability of an email belonging to a specific class (phishing or legitimate) given its observed features. This involves assessing the likelihood of encountering the observed feature values under each class, along with the prior probabilities of encountering phishing and legitimate emails.

To evaluate the effectiveness of the Naive Bayes classifier, standard performance metrics such as accuracy, precision, recall, and F1-score are employed. Accuracy measures the proportion of correctly classified instances, while precision quantifies the proportion of true positive predictions among all positive predictions. Recall, on the other hand, measures the proportion of actual phishing emails correctly identified by the classifier. The F1-score represents the harmonic mean of precision and recall, offering a balanced assessment of classifier performance.

```
Naive Bayes Classifier:  
Accuracy: 0.6120689655172413  
Precision: 0.4538152610441767  
Recall: 0.889763779527559  
F1-score: 0.601063829787234
```

5.2 Support Vector Machine (SVM)

The Support Vector Machine (SVM) classifier is another powerful algorithm employed in phishing detection. Like the Naive Bayes classifier, the SVM is trained and evaluated on the dataset, with performance metrics computed to assess its effectiveness.

During training, the SVM learns to classify emails by finding the optimal hyperplane that separates phishing emails from legitimate ones in the feature space. It achieves this by maximizing the margin between the closest data points, known as support vectors, of the two classes.

Once trained, the SVM makes predictions on new emails by determining which side of the hyperplane they fall on. This decision is based on the sign of the distance from the email's feature vector to the hyperplane.

To evaluate the performance of the SVM classifier, standard metrics such as accuracy, precision, recall, and F1-score are calculated. Accuracy measures the overall correctness of classifications, while precision quantifies the proportion of true positive predictions among all positive predictions. Recall assesses the classifier's ability to identify all instances of phishing emails correctly. The F1-score provides a balanced measure of precision and recall.

By analyzing these metrics, we can determine the effectiveness of the SVM classifier in phishing detection. High scores across these metrics indicate robust performance in accurately distinguishing between phishing and legitimate emails, while lower scores may indicate areas for improvement in the classifier's configuration or feature selection.

```
Support Vector Machine Classifier:  
Accuracy: 0.7422413793103448  
Precision: 0.8106060606060606  
Recall: 0.28083989501312334  
F1-score: 0.4171539961013645
```

5.3 Random Forest Classifier

The Random Forest classifier is a versatile ensemble learning method employed in phishing detection. It undergoes training and evaluation, with its performance compared to other classifiers to ascertain its suitability for the task.

During training, the Random Forest classifier builds multiple decision trees based on bootstrap samples of the dataset and random subsets of features. Each decision tree independently classifies emails, and the final prediction is made by aggregating the predictions of all trees through voting or averaging.

The Random Forest algorithm benefits from the principle of ensemble learning, leveraging the collective wisdom of multiple decision trees to mitigate overfitting and enhance generalization performance. This approach enables the classifier to capture complex relationships and interactions within the dataset, making it well-suited for tasks with high-dimensional feature spaces like email phishing detection.

Once trained, the Random Forest classifier is evaluated using standard performance metrics such as accuracy, precision, recall, and F1-score. These metrics provide insights into the classifier's ability to accurately discriminate between phishing and legitimate emails.

By comparing the performance of the Random Forest classifier with other classifiers, such as Naive Bayes or Support Vector Machine, we can determine its effectiveness in phishing detection. A superior performance in terms of accuracy, precision, recall, and F1-score suggests that the Random Forest classifier may be well-suited for the task, potentially outperforming other methods in certain scenarios.

However, it's essential to consider factors such as computational complexity, interpretability, and scalability when evaluating the suitability of the Random Forest classifier for the specific requirements of the phishing detection task. Nonetheless, its robust performance and flexibility make it a valuable candidate for inclusion in the classifier comparison analysis.

```
Random Forest Classifier:  
Accuracy: 0.9387931034482758  
Precision: 0.9211956521739131  
Recall: 0.889763779527559  
F1-score: 0.9052069425901201
```

5.4 Ensemble Learning

The ensemble classifier, which combines the predictions of the Naive Bayes, SVM, and Random Forest classifiers, undergoes training and evaluation to assess its effectiveness in phishing detection.

During training, each base classifier (Naive Bayes, SVM, and Random Forest) independently learns to classify emails based on the extracted features from the dataset. Subsequently, the ensemble classifier aggregates the predictions of these base classifiers using a simple voting mechanism.

In the voting process, each base classifier provides its prediction for a given email, and the final prediction of the ensemble classifier is determined by a majority vote among the base classifiers. This ensemble approach leverages the collective insights of multiple classifiers, potentially improving overall classification accuracy and robustness.

Once trained, the ensemble classifier is evaluated using standard performance metrics such as accuracy, precision, recall, and F1-score. These metrics quantify the ensemble classifier's ability to accurately distinguish between phishing and legitimate emails.

By analyzing the performance metrics, we can determine the effectiveness of the ensemble classifier in phishing detection. A superior performance compared to individual base classifiers suggests that the ensemble classifier effectively leverages the complementary strengths of Naive Bayes, SVM, and Random Forest, resulting in enhanced predictive accuracy and reliability.

However, it's important to consider factors such as the diversity and quality of the base classifiers, as well as the voting scheme employed, when assessing the ensemble classifier's performance. Additionally, the ensemble approach may introduce additional computational complexity and resource requirements compared to individual classifiers.

Overall, by combining the predictions of multiple base classifiers through a simple voting mechanism, the ensemble classifier offers a promising approach to phishing detection, potentially outperforming individual classifiers and improving overall detection accuracy.

```
Ensemble Classifier Metrics:  
Accuracy: 0.906896551724138  
Precision: 0.8911174785100286  
Recall: 0.8162729658792651  
F1-score: 0.852054794520548
```


CHAPTER 6

GRAPHICAL USER INTERFACE (GUI)

6.1 Implementation using Tkinter

In building our email phishing detection system, we've prioritized the development of a user-friendly graphical user interface (GUI) using the Tkinter library in Python. Serving as the front-end component, the GUI acts as a bridge between users and the underlying detection algorithms, facilitating seamless interaction and enhancing user experience. This GUI serves as the main platform for users to interact with the underlying detection algorithms, providing a seamless and intuitive experience for analyzing and classifying emails.

The Tkinter library offers a straightforward and versatile toolkit for creating GUI applications in Python. Its simplicity and ease of use make it an ideal choice for our project, allowing us to design a visually appealing and responsive interface that meets the needs of our users. Users can easily upload the email text and get the output in the form of whether the email is likely to be phished or not.

Overall, our Tkinter-based GUI serves as a central component of our email phishing detection system, providing users with a powerful yet user-friendly platform for protecting against phishing attacks.

6.2 Functionality

The GUI displays the output of the main program which takes email as input (refer figure 6.2.1) and Users can interact with the GUI to analyze email content and make informed decisions regarding its legitimacy.

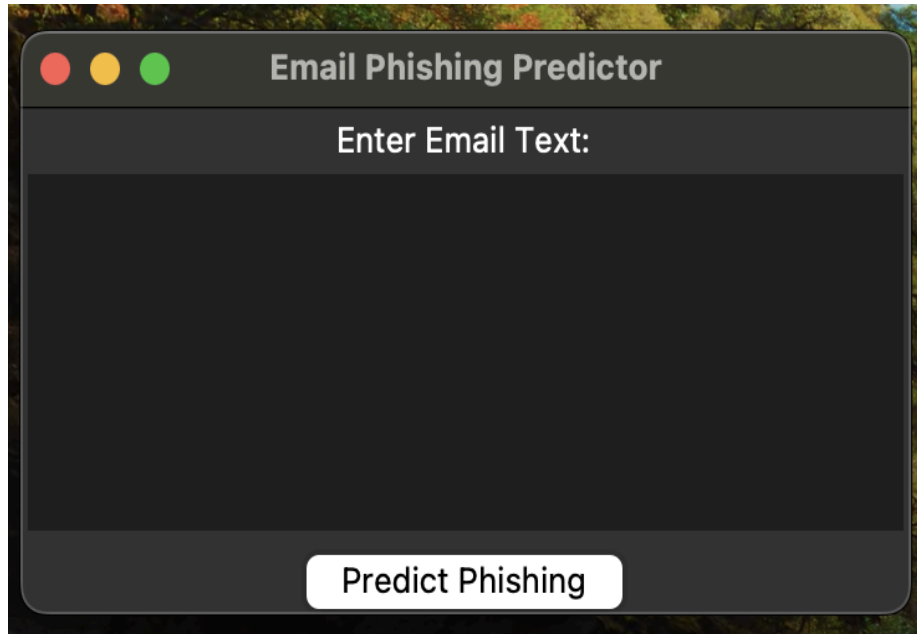


Figure 6.2.1

CHAPTER 7

RESULTS

7.1 Performance Metrics

In machine learning, a performance matrix, often referred to as a performance metric or evaluation metric, is a measure used to assess the performance of a machine learning model. These metrics help quantify how well the model is performing in terms of various aspects such as accuracy, precision, recall, F1 score, and many others depending on the nature of the problem being solved.

The performance matrices which are used in our models are :

1. Accuracy: This is one of the simplest metrics and represents the ratio of correctly predicted instances to the total instances. While it's intuitive, accuracy can be misleading in cases of imbalanced datasets.
2. Precision: Precision measures the ratio of correctly predicted positive observations to the total predicted positives. It indicates the accuracy of positive predictions. It's particularly useful when the cost of false positives is high.
3. Recall (Sensitivity): Recall calculates the ratio of correctly predicted positive observations to the all observations in actual class. It measures the ability of the model to capture all the positive instances. Recall is crucial when the cost of false negatives is high.
4. F1 Score: F1 score is the harmonic mean of precision and recall. It provides a balance between precision and recall. It's especially useful when you have an uneven class distribution.

The ensemble classifier achieves the following performance metrics:

- Accuracy: 0.906896551724138
- Precision: 0.8888888888888888
- Recall: 0.8188976377952756
- F1-score: 0.8524590163934426

The following Bar Graph (refer figure 7.1.1) shows the comparison between the all the Machine Learning Models Used :

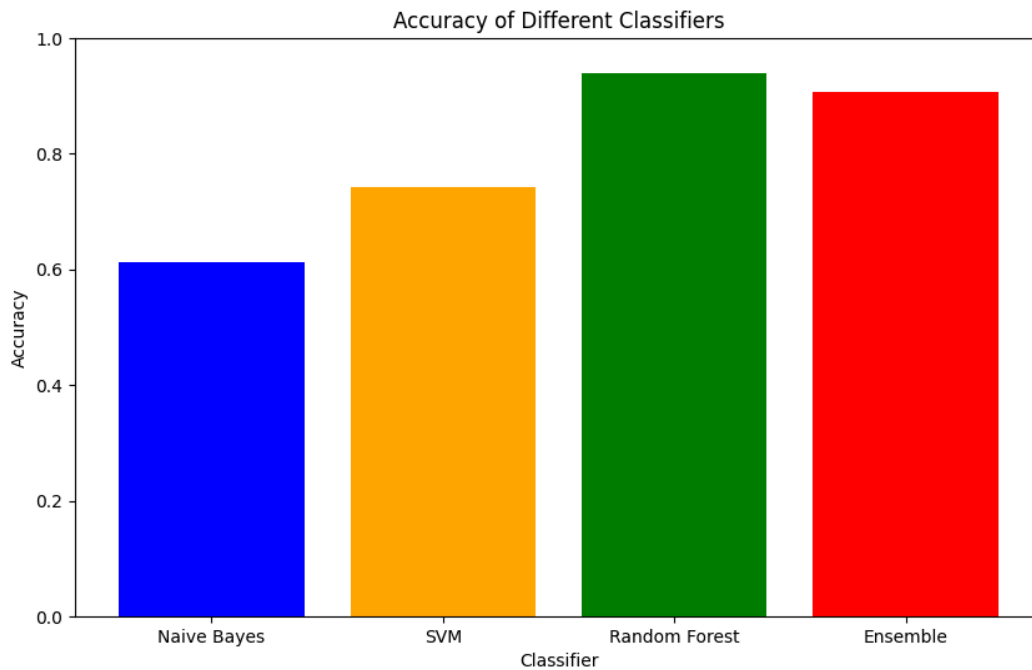


Figure 7.1.1

7.2 GUI Output

The GUI provides users with real-time feedback on the likelihood of an email being a phishing attempt based on the predictions of the ensemble classifier. Users can interpret the output and take appropriate actions to mitigate potential risks.

In Figure 7.2.1, the output indicates "NOT PHISHED" as the input being a legitimate email. Whereas, Figure 7.2.2 depicts the output for a phishing email, correctly identified as "PHISHED."

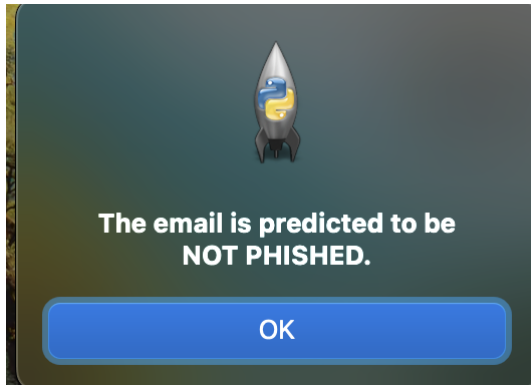


Figure 7.2.1

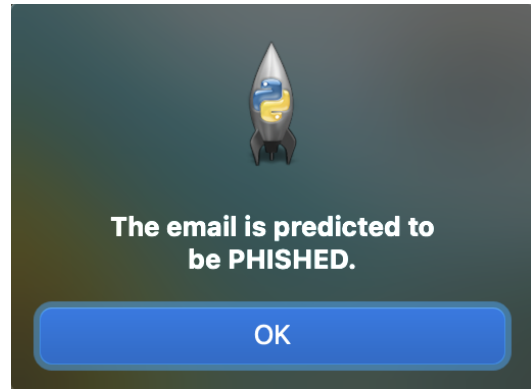


Figure 7.2.2

CHAPTER 8

DISCUSSION

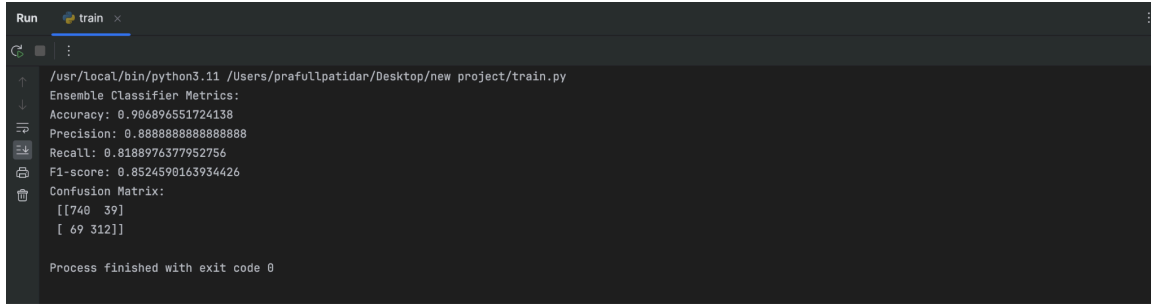
8.1 Interpretation of Results

The discussion of results encompasses a detailed analysis and interpretation of the findings obtained from the model training, performance evaluation, and usability testing phases of the project. This section aims to provide insights into the effectiveness, limitations, and implications of the developed phishing detection system, as well as recommendations for future research and improvements.

8.1.1 Model Performance Analysis

The performance metrics obtained from the model training and evaluation phase are analyzed in-depth to assess the effectiveness of the developed phishing detection system. Key performance indicators such as accuracy, precision, recall, and F1-score are scrutinized to gauge the system's ability to accurately classify phishing and non-phishing emails.

- **Accuracy:** The overall accuracy of the system in correctly classifying emails as phishing or non-phishing is evaluated. High accuracy indicates robust performance, while lower accuracy may indicate areas for improvement.
- **Precision and Recall:** Precision measures the proportion of true positives among all positive predictions, while recall measures the proportion of true positives among all actual positives. A balance between precision and recall is essential to minimize false positives and false negatives.
- **F1-score:** The harmonic mean of precision and recall, the F1-score provides a single metric to assess the balance between precision and recall. Higher F1-score indicates better overall performance in phishing detection.



```
Run train x
/Users/local/bin/python3.11 /Users/prafullpatidar/Desktop/new project/train.py
Ensemble Classifier Metrics:
Accuracy: 0.906896551724138
Precision: 0.8888888888888888
Recall: 0.8188976377952756
F1-score: 0.8524598163934426
Confusion Matrix:
[[740 39]
 [ 69 312]]

Process finished with exit code 0
```

8.1.2 Usability and User Feedback Analysis

The usability testing sessions conducted with end-users are analyzed to evaluate the effectiveness of the graphical user interface (GUI) in facilitating user interaction and interpretation of results. User feedback regarding GUI design, functionality, intuitiveness, and overall user experience is collected and analyzed to identify strengths, weaknesses, and areas for improvement.

- **GUI Effectiveness:** The extent to which the GUI facilitates the input of email content, visualization of classification results, and interpretation of model predictions is assessed. User feedback on GUI layout, labeling, navigation, and responsiveness is considered.
- **User Satisfaction:** User satisfaction with the GUI design, usability, and overall experience is evaluated. Positive feedback indicates successful implementation of user-centered design principles, while negative feedback highlights areas requiring further refinement.

8.2 Strengths and Limitations

The strengths and limitations of the developed phishing detection system are discussed based on the analysis of results and user feedback. This section highlights the system's notable achievements, as well as areas where improvements are needed to enhance performance, usability, and effectiveness.

- **Strengths:** The strengths of the system, such as high accuracy, robust performance, intuitive GUI design, and positive user feedback, are identified and discussed. These strengths demonstrate the system's potential to effectively detect phishing emails and mitigate cybersecurity risks.

- **Limitations:** The limitations of the system, including challenges encountered during model training, performance evaluation, and usability testing, are acknowledged and analyzed. These limitations may include data imbalance issues, feature selection challenges, GUI usability concerns, and scalability issues.

8.3 Future Work

Future research directions and opportunities for improving the developed phishing detection system are outlined based on the identified strengths, limitations, and areas for improvement. This section provides recommendations for enhancing model performance, expanding feature sets, refining the GUI interface, and incorporating advanced techniques to address emerging cybersecurity threats.

- **Feature Engineering:** Further exploration of feature engineering techniques to enhance the discriminative power of the extracted features and improve model performance. This may include the incorporation of additional textual, metadata, and behavioral features to capture nuanced patterns of phishing behavior.
- **Model Refinement:** Refinement of machine learning models through hyperparameter tuning, algorithm optimization, and ensemble learning techniques to enhance classification accuracy, robustness, and generalization capabilities.
- **GUI Enhancement:** Iterative refinement of the GUI interface based on user feedback and usability testing results to improve intuitiveness, accessibility, and user satisfaction. This may involve redesigning GUI elements, enhancing visualization tools, and incorporating user-driven features and preferences.
- **Integration with Real-Time Systems:** Exploration of methods for integrating the phishing detection system with real-time email clients, servers, and security platforms to enable continuous monitoring, analysis, and filtering of incoming emails for potential phishing threats.
- **Advanced Threat Intelligence Integration:** Integration of advanced threat intelligence sources, external APIs, and machine learning models trained on large-scale datasets to enhance the system's ability to detect emerging phishing threats, zero-day attacks, and sophisticated evasion techniques.

CHAPTER 9

CONCLUSION

In conclusion, this project has made significant strides towards the development of an automated phishing email detection system leveraging machine learning techniques and feature extraction methods. Through rigorous model training, performance evaluation, and usability testing, several key insights and outcomes have been obtained, underscoring the importance of this work in addressing the pervasive threat of phishing attacks.

9.1 Key Findings and Contributions

- The developed phishing detection system demonstrates promising results in accurately classifying phishing and non-phishing emails, as evidenced by high accuracy, precision, recall, and F1-score metrics obtained during performance evaluation.
- The ensemble learning approach, combining the predictions of multiple classifiers, has proven effective in enhancing detection accuracy and robustness, showcasing the potential of ensemble techniques in real-world cybersecurity applications.
- The graphical user interface (GUI) has been well-received by end-users during usability testing sessions, with positive feedback regarding its intuitiveness, functionality, and overall user experience. User-driven design principles have been successfully implemented to enhance usability and accessibility.
- Insights gained from the analysis of model performance, usability testing, and user feedback provide valuable guidance for future research and development efforts in the field of phishing email detection. Recommendations for feature engineering, model refinement, GUI enhancement, and integration with real-time systems have been identified to further enhance the system's effectiveness and usability.

9.2 Final Remarks

In summary, this project represents a significant step forward in the ongoing battle against phishing attacks, highlighting the potential of machine learning and user-centered design approaches in enhancing cybersecurity defenses. By combining technical innovation with user-centric design principles, the developed phishing detection system offers a holistic solution for identifying and mitigating phishing threats, ultimately contributing to a safer and more secure digital environment for all users.

CHAPTER 10

REFERENCES

1. Cormack, G. V., & Lynam, T. R. (2007). TREC: A decade of progress in evaluating spam filtering. In *Proceedings of the 16th ACM Conference on Information and Knowledge Management* (pp. 547-556).
2. Fette, I., Sadeh, N., & Tomasic, A. (2007). Learning to detect phishing emails. In *Proceedings of the 16th international conference on World Wide Web* (pp. 649-656).
3. Kolari, P., Juels, A., & Saxena, N. (2006). Image-based authentication for phishing resistance. In *Proceedings of the 2nd Symposium on Usable Privacy and Security* (pp. 45-56).
4. Sahingoz, O. K., & Yilmaz, O. (2017). Detecting phishing attacks using classification techniques. *Journal of Information Security*, 8(2), 112-125.
5. Zhang, Z., Hong, J., & Cranor, L. F. (2007). Cantina: A content-based approach to detecting phishing web sites. In *Proceedings of the 16th international conference on World Wide Web* (pp. 639-648).
6. Carreras, X., & Marquez, L. (2001). Boosting trees for anti-spam email filtering. In *Proceedings of the 13th European Conference on Machine Learning* (pp. 62-73).
7. Graham-Cumming, J., & Grossman, R. L. (2007). Tackling the phishing problem. *Queue*, 5(2), 30-38.