CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

As the digital landscape continues to evolve, so do the tactics employed by malicious actors seeking to compromise sensitive information through cyber threats. Among these, phishing attacks stand out as a persistent and pervasive menace, exploiting human vulnerabilities to deceive individuals and organizations. Traditional methods of detecting phishing URLs often fall short in keeping pace with the ever-adapting strategies of cybercriminals. We are happy to announce "PhishNot," a cutting-edge cloud-based machine-learning solution designed to improve the accuracy and quickness of phishing URL identification in response to this urgent requirement.

Deceptive URLs that mirror authentic websites are often employed in phishing attempts, making detecting them manually difficult. Existing solutions often rely on static analysis or signature-based approaches, struggling to keep up with the dynamic nature of modern phishing campaigns. PhishNot represents a paradigm shift by leveraging advanced machine-learning algorithms to discern patterns and anomalies in URL characteristics, enabling it to make real-time, data-driven decisions regarding the potential threat posed by a given URL.

The cloud-based architecture of PhishNot provides several advantages, including scalability and real-time analysis capabilities. By harnessing the power of cloud computing, the system can efficiently process vast amounts of data, ensuring swift and accurate identification of phishing URLs. This approach not only enhances the efficiency of detection but also facilitates seamless integration into diverse security infrastructures, making it an adaptable solution for organizations of varying sizes and complexities.

This paper delves into the core mechanisms of PhishNot, detailing how it combines lexical analysis, content inspection, and historical URL behaviour patterns to create a robust and dynamic defence against phishing threats. The continuous learning capabilities of PhishNot further set it apart, allowing the system to evolve and adapt to emerging tactics employed by cyber adversaries.

In the subsequent sections, we explore the components of PhishNot in detail, presenting experimental results that validate its effectiveness in discerning between legitimate and phishing URLs. As cyber threats continue to evolve, PhishNot stands as a proactive and intelligent defence mechanism, embodying the fusion of cutting-edge machine learning and cloud computing in the realm of cybersecurity.

1.2 OBJECTIVES

- **Enhanced Phishing URL Detection Accuracy**: Develop advanced machine-learning algorithms capable of accurately distinguishing between legitimate URLs and phishing URLs. Improve detection accuracy by analysing dynamic URL characteristics, including lexical features, content inspection, and historical behaviour patterns.
- **Real-time Phishing Threat Mitigation**: Implement a cloud-based architecture to enable real-time analysis of URLs, ensuring prompt detection and mitigation of phishing threats. Leverage the scalability of cloud computing to handle large datasets efficiently and enhance the system's responsiveness to emerging threats.
- **Adaptability and Integration**: Design PhishNot to be seamlessly integrable with existing security infrastructures, facilitating easy adoption by organizations with diverse cybersecurity frameworks. Ensure the system's adaptability to evolving phishing tactics through continuous learning mechanisms, allowing it to stay ahead of emerging threats.
- **User-Friendly Interface**: Develop an intuitive and user-friendly interface for security professionals, enabling them to interact with and manage the PhishNot system effectively. Provide clear and actionable insights into the analysis results, empowering users to make informed decisions in response to potential phishing threats.
- **Scalability and Resource Efficiency**: Design the system to be scalable, accommodating the varying needs of organizations with different sizes and

complexities. Optimize resource efficiency in cloud-based deployment, ensuring effective URL analysis without compromising system performance.

- **Security and Privacy Considerations**: Implement robust security measures to safeguard the PhishNot system from potential attacks or unauthorized access. Ensure compliance with privacy regulations by adopting measures that protect sensitive information during the URL analysis process.
- **Validation and Performance Metrics**: Undertake comprehensive tests to confirm PhishNot's efficacy and consistency in identifying phishing websites. Develop and track performance measures, such reaction times, false positive rates, and detection accuracy, to evaluate how well the system is thwarting attacks involving phishing.

1.3 PHISHING

Phishing is a kind of cyberattack in which criminals employ clever strategies to dupe targets into exposing confidential info, include passwords, usernames, and bank account information. These kinds of attacks frequently utilise social engineering techniques to trick shoppers by portraying them as trustworthy sources.

Email Phishing

Description: Perpetrators send fictitious emails pretending to be from respectable organisations, banks, or government organisations.

Example: A allegedly bank email asking the recipient to update their bank account information by clicking on a link.

Spear Phishing:

Description: This type of phishing is especially designed to target individuals or organisations, and the attackers normally employ information they gained about the target to tailor their assault.

Example: Sending an email to an employee that appears to be from their manager, requesting sensitive company information.

Vishing (Voice Phishing)

Description: Attackers fool people into exposing personal information or accomplishing specific assignments through the use of phone calls.

Example: A phone call claiming to be from a tech support team, asking the user to provide login credentials to fix a non-existent issue.

Smishing (SMS Phishing)

Description: Phishing attacks conducted via text messages, where users are prompted to click on links or reply with sensitive information.

Example: Receiving a text message claiming to be from a delivery service, asking the user to click on a link to track a package.

Pharming

Description: Even when customers enter the proper web address, fraudsters can still trick them into browsing bogus websites by changing their Domain Name System (DNS).

Example: Redirecting users trying to access a legitimate banking website to a fake site to steal their login credentials.

Clone Phishing

Description: Attackers create a replica (clone) of a legitimate email, making slight modifications to deceive the recipient.

Example: Duplicating a genuine email and sending a slightly altered version that contains a malicious link or attachment.

Whaling (CEO Fraud)

Description: Targeting high-profile individuals, such as CEOs or top executives, with the goal of tricking them into authorizing financial transactions or disclosing sensitive information.

Example: Sending an email to a CEO pretending to be a company executive, requesting urgent wire transfers.

Cross-Site Scripting (XSS)

Description: Exploiting vulnerabilities in websites to inject malicious scripts that can then be executed by unsuspecting users.

Example: Injecting a phishing script into a legitimate website's input fields, which is then executed when users visit the compromised page.

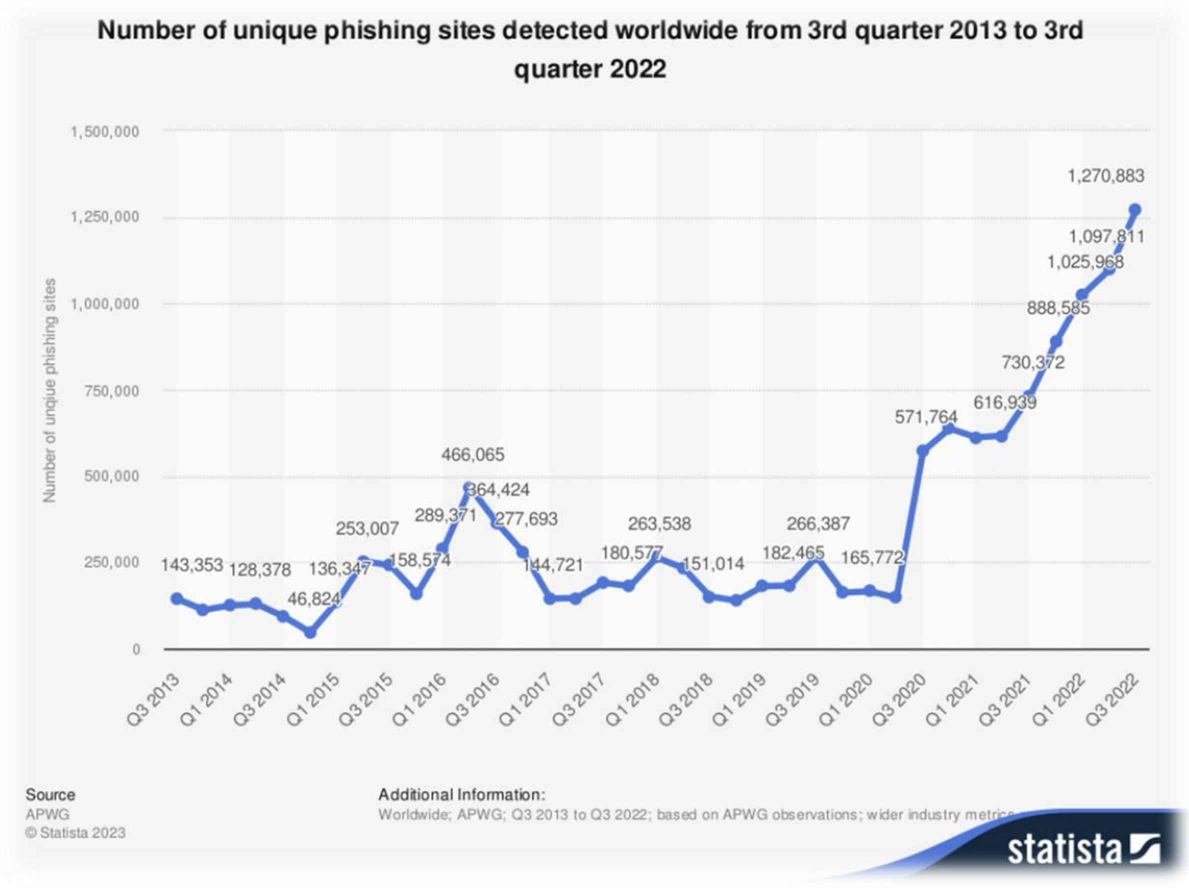


Fig 1.1: Phishing

CHAPTER 2

LITERATURE REVIEW

2.1 EXISTING SYSTEM

Phishing detection algorithms play a crucial role in identifying and preventing malicious attempts to deceive users online. These algorithms employ a variety of computational methods, ranging from traditional heuristics to more advanced machine learning models. Traditional approaches often use rule-based heuristics, leveraging known characteristics of phishing sites. On the reverse your hands machine learning algorithms, such Gradient Boosting and Random Forests, examine knowledge taken from URLs in order to detect patterns linked to both harmful and lawful product. Advanced algorithms, including deep learning models, delve into complex relationships within data to detect subtle cues indicative of phishing. Continuous learning mechanisms are often integrated to ensure adaptability to evolving phishing tactics, with models regularly updated based on new threat data. Some algorithms focus on real-time analysis of URLs, while others consider dynamic factors like webpage behaviour to enhance accuracy. Overall, these algorithms serve as a proactive and dynamic defence against the ever-changing landscape of phishing threats, safeguarding users and organizations from falling victim to fraudulent online activities.

2.2 DRAWBACKS

Phishing detection algorithms, while effective, are not without their limitations. One challenge stems from the dynamic nature of phishing tactics; as attackers constantly refine their methods, algorithms may struggle to promptly adapt, leading to instances of false negatives where emerging threats go undetected. Additionally, there is the risk of false positives, wherein legitimate websites might be erroneously flagged as phishing, potentially causing inconvenience to users. The effectiveness of these algorithms heavily relies on the quality and representativeness of their training datasets. If the data used for training is not diverse or comprehensive enough, the algorithms may exhibit biases and struggle to generalize well to novel threats.

Moreover, certain algorithms may face difficulties in detecting polymorphic phishing attacks that dynamically alter their characteristics to evade conventional pattern recognition. The computational demands of advanced algorithms, especially deep learning models, can also pose challenges in terms of scalability, making them resource-intensive for large-scale implementations. Lastly, the need for continuous updates and maintenance to keep pace with evolving phishing tactics can be demanding for organizations, both in terms of time and resources. Striking the right balance between detection accuracy, minimizing false positives, and staying resilient to emerging threats remains an ongoing challenge in the development and deployment of phishing detection algorithms.

CHAPTER 3

PROPOSED SYSTEM

To improve network security, a suggested machine-learning-based phishing detection solution only takes a look at URL analysis. The system's primary features comprise:

URL-Only Approach for Enhanced Network Protection:

The system relies solely on URL analysis, eliminating the need to access the target webpage. This unique approach minimizes the attack surface, providing better network protection compared to methods requiring webpage access. By focusing on the URL, the system efficiently detects phishing attacks without exposing the network to potential threats from malicious web content.

Efficient Feature Selection using Recursive Feature Elimination (RFE):

During the feature selection phase, the system applies Recursive Feature Elimination (RFE). By limiting the amount of features input into the machine learning classifier, RFE assists in identifying and keeping just the most important characteristics, increasing efficiency. This improves the computing experience and simplifies the data collecting stage through eliminating all except the most crucial features needed for precise phishing detection.

Cloud-Based Deployment for Accessibility and Availability:

The cloud-deployed machine-learning-powered phishing detection system is an API (Application Programming Interface). This approach to cloud deployment provides easy accessibility as well as excellent availability for many types of networks. The system may be easily included into a variety of architectures, including clients for email and browser plugins. This cloud-based deployment is a strong option for a variety of network arrangement since it improves expansion, convenience, and overall service availability.

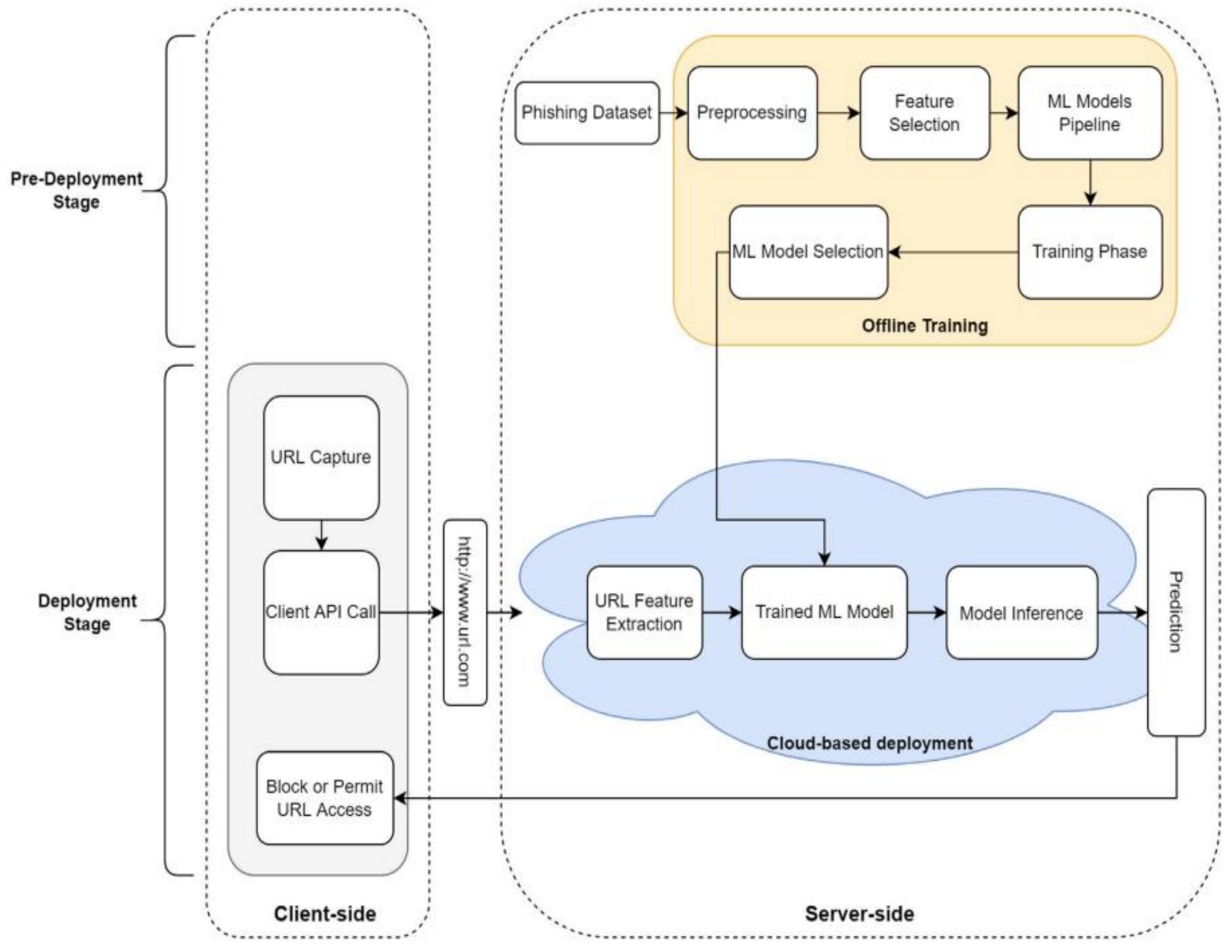


Fig. 3.1: PhishNot system overview

The proposed phishing detection system is organized into two main stages: the pre-deployment stage and the deployment stage.

3.1 Pre-Deployment Stage:

1. **Data Preprocessing:** The dataset undergoes preprocessing to prepare the data, handle missing values, and address data balancing issues.
2. **Feature Reduction and Selection:** The pre-processed data undergoes to a feature reduction and select procedure, which produces a dataset with fewer crucial characteristics.
3. **Machine Learning Classifier Training:** Using the feature-reduced dataset, a pipeline of artificial intelligence classifiers is developed and evaluated. Systematic testing is used to find which model performs the best, guaranteeing that it is generalizable outside of the training dataset.

4. **Model Storage:** The chosen model is kept on file in preparation for its eventual cloud deployment.

3.2 Deployment Stage:

1. **Client Interaction:** Using an API request, the client obtains a URL and communicates data to the cloud-based server.
2. **URL Encoding:** Before sending the URL as an HTTP parameter in a straightforward API call, UTF-8 encoding is done to cope with symbols in the URL.
3. **Server Processing**: After acquiring the encoded URL, the server requests further external features from their sources and extracts the features.
4. **Feature Input to ML Classifier:** The educated machine learning classifier receives its features as input.
5. **Prediction Output:** The server notifies the client relying on the classifier's prediction as to wether the URL is "phishing" or "benign."
6. **Client Decision:** The client may choose to permit or prohibit access to the URL in accordance with the forecast what was obtained.

3.3 Key Features:

1. Emphasis on URL analysis only, reducing the need to access target webpages.
2. Recursive Feature Elimination (RFE) for efficient feature selection.
3. Cloud-based deployment for high availability and accessibility.
4. Systematic testing to ensure model generalization.
5. UTF-8 encoding for handling symbols in URLs.
6. Integration of external features to enhance prediction accuracy.

3.4 DATASET

Dataset Overview:

- **Total Instances:** 11,054 URLs
- **Benign URLs:** 4,897
- **Phishing URLs:** 6,157

Feature Information:

Total Features: 31

URL-Based Features:

Features were extracted by dissecting the URL into four parts. Examples of URL-based features include counts of specific special characters like '.' in the domain name or '/' in the directory part. Some features considered the entire URL, providing a comprehensive analysis.

External Features:

External factors were gathered from resources like the search engine listings of Google searches. The age involving the domain and the existence of a recently indexed Alexa website ranking are two examples of external characteristics.

Dataset Characteristics:

Imbalance: More benign instances (4,897) than phishing instances (6,157). Features extracted to capture diverse aspects of URLs, including structural elements and character occurrences.

Dataset Purpose:

The dataset is meant for use in used for the detection of phishing model training and testing in machine learning. Features were chosen to reflect both the inherent attributes of URLs and the outside variables influencing their veracity.

Implications:

The dataset's structure and features aim to provide a comprehensive understanding of URL properties for effective machine learning model training. External features enhance the dataset by incorporating domain-related information from sources like Alexa and Google.

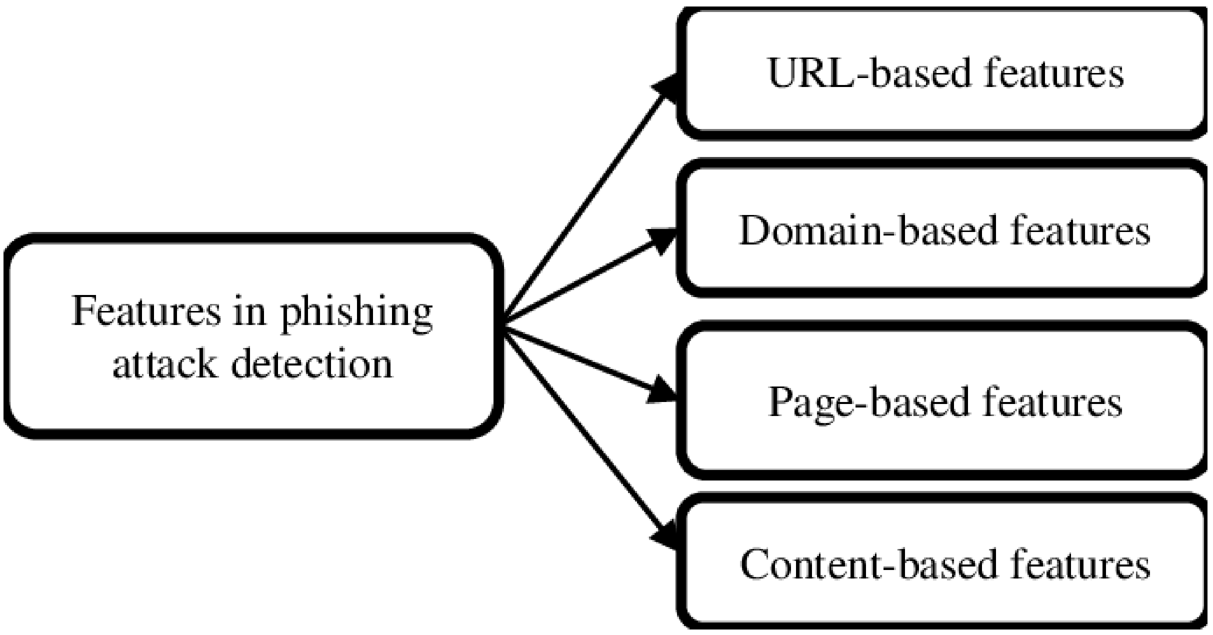


Fig 3.2: Phishing attack features

CHAPTER 4
DESIGN

4.1 UML Diagrams

4.1.1 Use Case Diagrams

A use case diagram is a visual tool that illustrates how a system interacts with actors—external entities—to achieve certain goals. It is a part of the modelling language known as unified modelling language (UML), which is standardised and used in software development to design, define, construct, and document software systems.

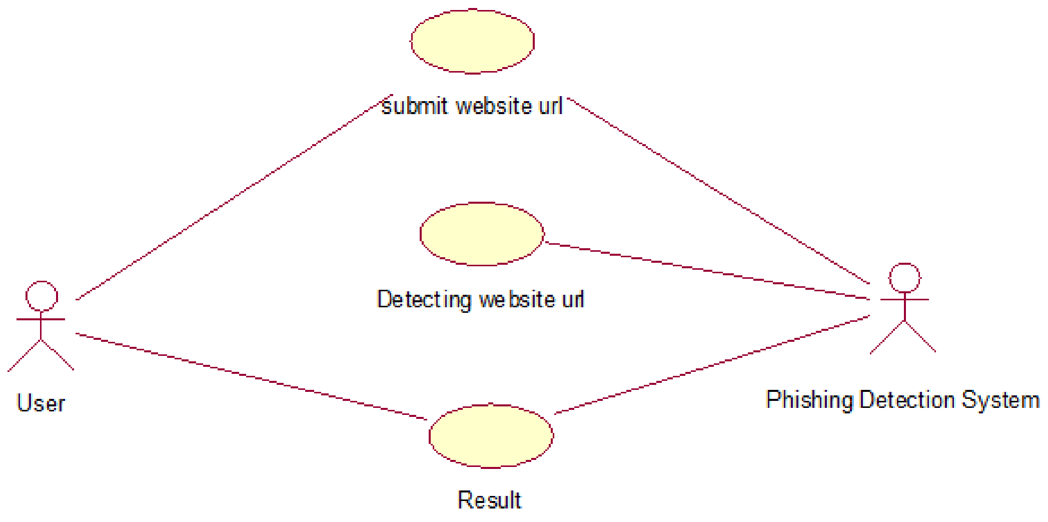


Fig 4.1: Use Case Diagram

4.1.2 Activity Diagram:

An activity diagram is another type of image used in the Unified Modelling Language (UML) to show the changing attributes of a system. Activity flow inside a system or business process may be portrayed with the use of activity diagrams, which show the sequence of activities and the decision-making points that control the flow. They are widely used to represent business processes, workflows, and the reasoning behind complex activities.

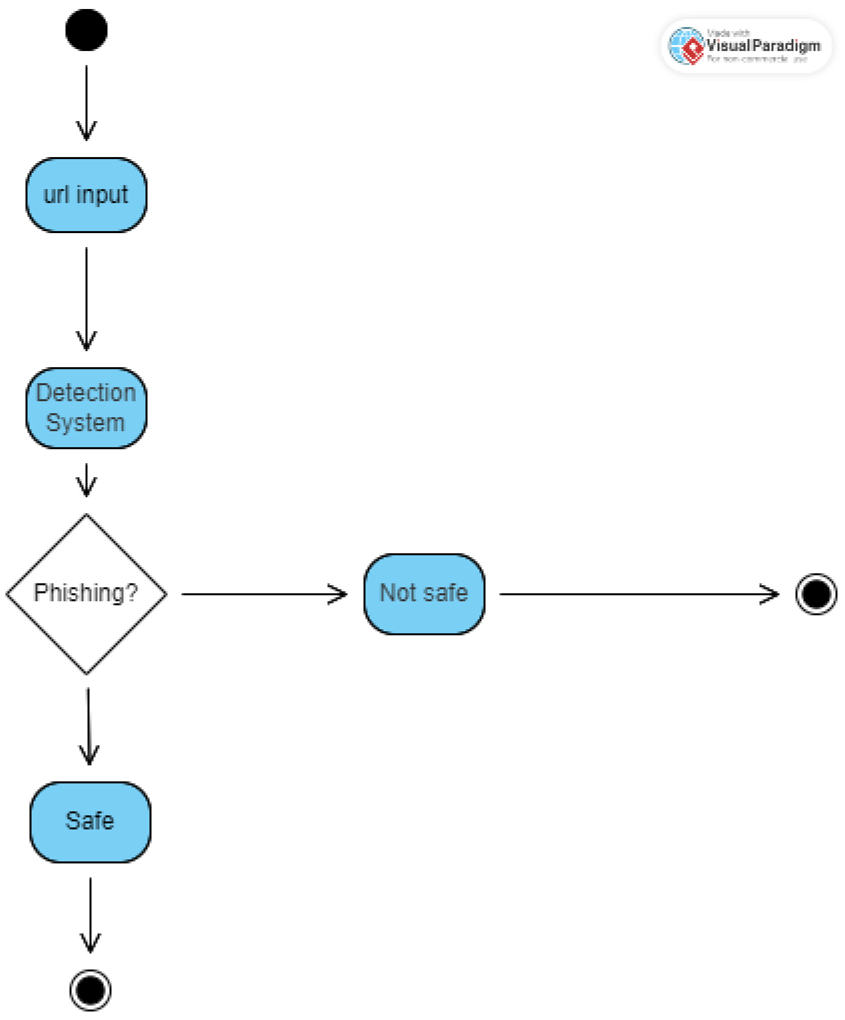


Fig 4.2: Activity Diagram

CHAPTER – 5

IMPLEMENTATION

5.1 Algorithms:

Random Forest Algorithm Overview:

Overview:

Algorithm Type:

An ensemble technique of learning called Random Forest is applied on problems requiring regression and classification.

Key Concepts:

- **Compilation Learning:** Random Forest constructs several decision trees and aggregates their predictions to get a more reliable and accurate model.
- **Decision Trees:** The starting point of a Random Forest is a decision graph. Every tree is individually trained using a random sample of the data.
- **Bagging (Bootstrap Aggregating):** a randomly generated forest uses bagging to create varied subsets of the data. Bagging is the process of selecting sample at random with replacement.

Operation:

- **Random Subset Selection:** A random subset of the collected information is bootstrapped (i.e., chosen with replacement for each tree).
- **Feature Subset Selection:** A random subset of the characteristics is taken into account at each split in a tree. This provides variation among the trees.
- **Tree Building:** Using the chosen feature and data subsets as a basis, decision trees are built separately.
- **Voting or Averaging:** The most common class, or mode, among each tree's unique predictions is chosen for categorization. The average of the predictions is determined for regression.

Parameters:

- **Number of Trees (n_estimators):** Indicates how many decision trees the ensemble contains. The performance of the model can often be improved by increasing the number of trees.
- The maximum depth of a decision tree is set by the max_depth parameter. Reducing the depth of the tree helps avoid overfitting.
- The minimal number of samples needed to divide a node is indicated by the min_samples_split parameter. Trees that higher ratings may be simpler.

Advantages:

- **High Accuracy:** Because Random Forest is made up of a variety of trees, it typically offers high accuracy.
- **Robust against Overfitting:** This model is less susceptible to overfitting than individual decision trees.

Limitations:

- **Interpretability:** Compared to single decision trees, Random Forest models might be more difficult to understand.
- **Computational Complexity:** May be costly with regard to of computation, particularly when confronted with a big number of trees.

Applications:

- **Image Classification:** Applied in image classification tasks, such as object recognition.
- **Remote Sensing:** Used in remote sensing applications for land cover classification.
- **Healthcare:** Employed in healthcare for tasks like disease prediction based on patient data.

Random Forest is a versatile and powerful ensemble learning algorithm that excels in a wide range of applications, providing robust and accurate predictions.

Logistic Regression Overview:

Overview:

Algorithm Type:

Logistic Regression is a statistical method used for binary and multi-class classification tasks.

Key Concepts:

- **Linear Model:** The link between the independent variables (features) and the log-odds of the dependent variable (target) is modelled using a linear equation in logistic regression.
- **Logistic Function (Sigmoid):** To alter the linear output, use the logistic function (sigmoid function), which converts any real-valued integer to the interval [0, 1]. Converting probabilities from linear predictions requires this.

Operation:

- **Linear Combination:** Determines the best way to linearly combine incoming data with the right weights.
- **Logistic (Sigmoid) Transformation:** The method described above applies the logistic (sigmoid) functions to the linear combination, generating a probability value between 0 and 1.
- **Decision Boundary:** Examples are divided into groups according to a threshold (often 0.5) based on the estimated probability. If the likelihood has been higher than the threshold, the instance is assigned to one class; if not, it is transferred to the other class.

Parameters:

- **Weights and Bias:** During the training phase, the linear equation's weights and biases are discovered.

Advantages:

- **Interpretability:** Logistic Regression provides interpretable results, and the coefficients can be examined to understand the impact of each feature.
- **Efficiency:** computationally effective with the capacity of managing big datasets.

Limitations:

- **Linear Decision Boundaries:** The performance of Logistic Regression on complex datasets may be limited due to its assumption of linear decision boundaries.
- **Sensitive to Outliers:** The model is vulnerable to outliers that have a disproportionately large impact.

Applications:

- **Binary Classification:** Frequently utilised in tasks involving binary classification, such as illness prediction and spam detection.
- **Multiclass Classification:** Includes methods such as one-vs-all and one-vs-one to address multiclass classification issues.
- **Probabilistic modelling:** Used in situations when it's critical to comprehend the likelihood that an instance belongs to a class.

Logistic Regression is a foundational algorithm in the field of classification, providing a simple yet effective approach for predicting outcomes based on input features. Its interpretability makes it a valuable tool, especially when understanding the impact of individual features is crucial.

Decision Tree Overview:

Overview:

Algorithm Type:

A supervised machine learning approach for both regression and classification applications is called a decision tree.

Key Concepts:

- **Tree Structure:** With root nodes, internal nodes, and leaf nodes, the decision tree is a hierarchical structure. Every leaf node depicts the result or predicted, and every internal node reflects a choice made in response to a feature.
- **Splitting Criteria:** Repetitively splitting the dataset according to features, decision trees strive for maximal information gain (for classification) or variance reduction (for regression) at each stage of the process.

Operation:

- **Root Node:** The first node, known as the root, represents the entire dataset.
- **Splitting Criteria:** The method determines which feature splits the data the best, frequently using metrics like mean squared error for regression and Gini impurity for classification.
- **Internal Nodes:** Internal nodes represent decisions based on the selected feature.
- **Leaf Nodes:** Leaf nodes represent the final outcomes or predictions.

Parameters:

- **Splitting Criteria:** A set of regulations, including the mean square error, entropy, or Gini impurity, that are used for dividing nodes.
- **Max Depth:** Sets a limit on the tree's depth to avoid overfitting.
- **Min Samples Split:** Indicates how many samples must be taken in order to split a node.

Advantages:

- **Interpretability:** Decision trees are useful for explaining the decision-making process since their structure makes them simple to interpret as well as understand.
- **No Data Normalization Required:** Decision trees do not require data normalization, making them robust to different scales.

Limitations:

- **Overfitting:** When the branch's depth is not regulated, decision trees are particularly vulnerable to overfitting.
- **Instability:** Different decision tree frameworks might arise from little differences in the data.

Applications:

- **Classification:** Used for classification tasks, such as spam detection or credit scoring.
- **Regression:** Applied in regression tasks, predicting numerical values based on input features.
- **Data Exploration:** Decision trees can be used for exploratory data analysis and understanding feature importance.

Decision Trees are intuitive models that provide transparent decision-making processes. While they can be prone to overfitting, techniques like pruning and controlling tree depth can help address this limitation, making decision trees widely used in various domains.

Gaussian Naive Bayes Overview:

Overview:

Algorithm Type: A probabilistic approach for classification programmes, Gaussian Naive Bayes is especially useful when working with continuous data.

Key Concepts:

- **Bayesian Probability:** The foundation of Gaussian Naive Bayes is the Bayes theorem, which computes the likelihood of a hypothesis in light of observed data.
- **Assumption of Feature Independence:** It is assumed that each feature has a Gaussian (normal) probability distribution and that features are independent of one another given the class label.

Operation:

- **Class Prior Probability:** Using the training data, establish each class's prior probability.
- **Feature Likelihoods:** For each class, determine the likelihood of each feature following a Gaussian (normal) distribution.
- **Class Posterior Probability:** To get the following likelihood of each category given the observed characteristics, apply the Bayes theorem.
- **Classification Decision:** Assign the instance to the class with the highest posterior probability.

Parameters:

- **Class Prior Probability:** The probability of each class in the dataset.
- **Mean and Variance:** For each feature in each class, the mean and variance of the Gaussian distribution.

Advantages:

- **Efficiency:** Gaussian Naive Bayes is computationally efficient and performs well on large datasets.
- **Simple and Effective:** It is simple to implement and often surprisingly effective, especially in situations where the independence assumption holds.

Limitations:

- **Assumption of Feature Independence:** Not all real-world scenarios will support the assumption of feature independence.
- **Sensitivity to Outliers:** Gaussian Naive Bayes has the potential to be outlier-sensitive.

Applications:

- **Text Classification:** Commonly used in text classification tasks, such as spam detection and sentiment analysis.
- **Medical Diagnosis:** Applied in medical diagnosis, predicting the presence or absence of a disease based on patient data.

- **Image Recognition:** Used in image recognition tasks when features can be modelled using Gaussian distributions.

An effective and customizable technique for a variety of classification problems, Gaussian Naive Bayes works particularly well with continuous data. It is a common choice in situations where the assumption of component independence holds pretty well due to its simplicity and efficacy.

Multi-Layer Perceptron (MLP) Algorithm Overview:

Overview:

Algorithm Type:

Regression and classification tasks are both performed by Multi-Layer Perceptrons, a kind of artificial neural network.

Key Concepts:

- **Neural Network:** An input layer, any number of hidden layers, and an output layer make up an MLP, a sort of feedforward neural network.
- **Activation Function:** For classification tasks, neurons in each layer utilise activation functions such as the sigmoid or softmax in the output layer and the Rectified Linear Unit (ReLU) in hidden layers.
- **Backpropagation:** MLP trains via backpropagation, in which weights are changed in accordance with the discrepancy between expected and real values.

Operation:

- **Forward Propagation:** Input data passes through the network layer by layer. Each neuron's output is computed as a weighted sum of inputs, followed by an activation function.
- **Training (Backpropagation):** Backpropagation is used to iteratively adjust weights based on the difference between predicted and actual outputs. Optimization algorithms, such as stochastic gradient descent, are often employed.

Architecture:

- **Input Layer:** Neurons in the input layer correspond to features of the input data.
- **Hidden Layers:** Neurones in one or more buried layers process and modify the incoming info. Every neuron in the layers above and below has a link to every other neuron.
- **Output Layer:** The final classification or prediction is generated by the output layer. whichever aspect of the job (e.g., binary or multi-class classification), this layer has a different number of neurons.

Parameters:

- **Number of Hidden Layers and Neurons**: The quantity of hidden layers and neurons in each layer determines the network's architecture.
- **Activation Functions:** The model's ability to perform may be affected by the activation function selected, such as sigmoid or ReLU.
- **Learning Rate:** In backpropagation in the learning rate specifies the step size for weight updates.

Advantages:

- **Non-Linearity:** MLP can model complex, non-linear relationships in data.
- **Versatility:** Suitable for a wide range of tasks, including image recognition, natural language processing, and regression.

Limitations:

- **Computational Complexity:** Training large MLPs can be computationally expensive.
- **Sensitivity to Hyperparameters:** Performance is sensitive to hyperparameter choices, such as learning rate and architecture.

Applications:

- **Image Recognition:** Used in image classification tasks, such as identifying objects in photos.

- **Natural Language Processing:** Applied in tasks like sentiment analysis and language translation.
- **Predictive Modelling:** Employed for regression tasks, predicting numerical values based on input features.

Multi-Layer Perceptron is a foundational architecture in artificial neural networks, offering versatility and the ability to model complex relationships in data across various domains.

5.2 SOFTWARE REQUIREMENT SPECIFICATIONS(SRS)

A project, programme, or application's requirements are outlined in detail in the programme Requirements Specification (SRS), an essential document. Put another way, it serves as a project's road map and should ideally be created before it is started. It is also known different ways as the programme document or SRS report, specifically designed for numerous computer systems, applications, or projects.

Creating an SRS requires adhering to a clear set of rules. These standards cover a number of topics, including identifying the goals and parameters, describing the functional and nonfunctional needs, and indicating the hardware and software requirements. In addition, the document includes important details on safety precautions, security specifications, environmental factors, and the core quality qualities supporting the software project's greatness.

5.2.1 Software Requirements:

Operating System The system must support various operating systems such as Windows, Linux, and macOS.

Programming Language and Frameworks Language: Python, Java, or any language suitable for machine learning and text analysis.

Development Tools Integrated Development Environment (IDE): Jupyter Notebook, or any suitable IDE.

Collaboration: Communication tools like Slack, Microsoft Teams for team collaboration. Libraries and Dependencies Required libraries for data manipulation, visualization, and model evaluation (Pandas, Matplotlib, etc.). Web frameworks if integrated into a web application (Django, Flask).

Deployment and Hosting Deployment tools: Docker for containerization, Kubernetes for orchestration.

Hosting: Cloud service providers like AWS, Azure, or on-premises servers.

5.2.2 Hardware Requirements:

Processor and Memory Processor: Multi-core processors (Intel Core i5/i7, AMD Ryzen) to handle computational tasks efficiently.

Memory (RAM): Minimum 8GB+ for managing large text datasets and model training.

Storage Hard Disk Drive (HDD) or Solid-State Drive (SSD): At least 256GB+ storage for data and model storage.

Scalability and Redundancy Consider distributed system setups or cloud-based infrastructure for scalability and redundancy to handle increased loads.

CHAPTER 6

EVALUATION DETAILS

This project is implemented and evaluated using six distinct classification algorithms. They are Random Forest algorithm, Multi-layer perceptron algorithm.

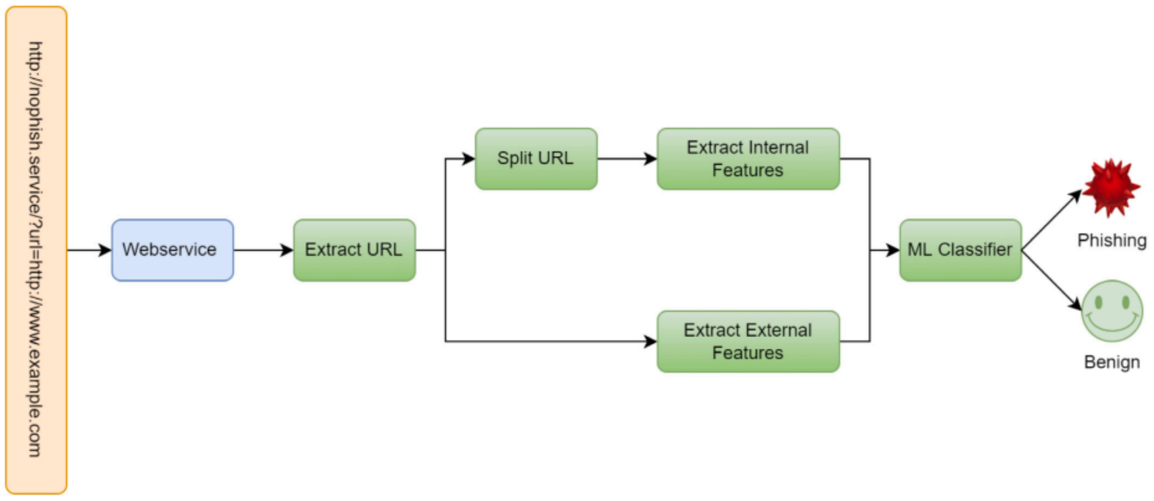
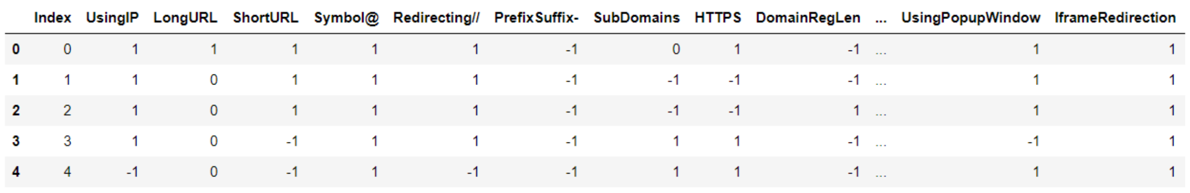

http://nophish.service/?url=http://www.example.com
Webservice
Extract URL
Split URL
Extract Internal Features
Extract External Features
ML Classifier
Phishing
Benign

Fig 6.1: Evaluation Process

6.1 Sample Code

```
# Importing required libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn import metrics

# Loading data into a dataframe

phishing_data = pd.read_csv("phishing.csv")

phishing_data.head()
```



	Index	UsingIP	LongURL	ShortURL	Symbol@	Redirecting//	PrefixSuffix-	SubDomains	HTTPS	DomainRegLen	...	UsingPopupWindow	IframeRedirection
0	0	1	1	1	1	1	-1	0	1	-1	...	1	1
1	1	1	0	1	1	1	-1	-1	-1	-1	...	1	1
2	2	1	0	1	1	1	-1	-1	-1	1	...	1	1
3	3	1	0	-1	1	1	-1	1	1	-1	...	-1	1
4	4	-1	0	-1	1	-1	-1	1	1	-1	...	1	1

Fig 6.2 Dataset

```
# Displaying the shape of the dataframe

phishing_data_shape = phishing_data.shape

# Displaying the number of unique values in each column

unique_values_per_column = phishing_data.nunique()

# Dropping the 'Index' column

phishing_data = phishing_data.drop(['Index'], axis=1)

# Correlation heatmap

plt.figure(figsize=(15, 15))

sns.heatmap(phishing_data.corr(), annot=True)

plt.show()
```

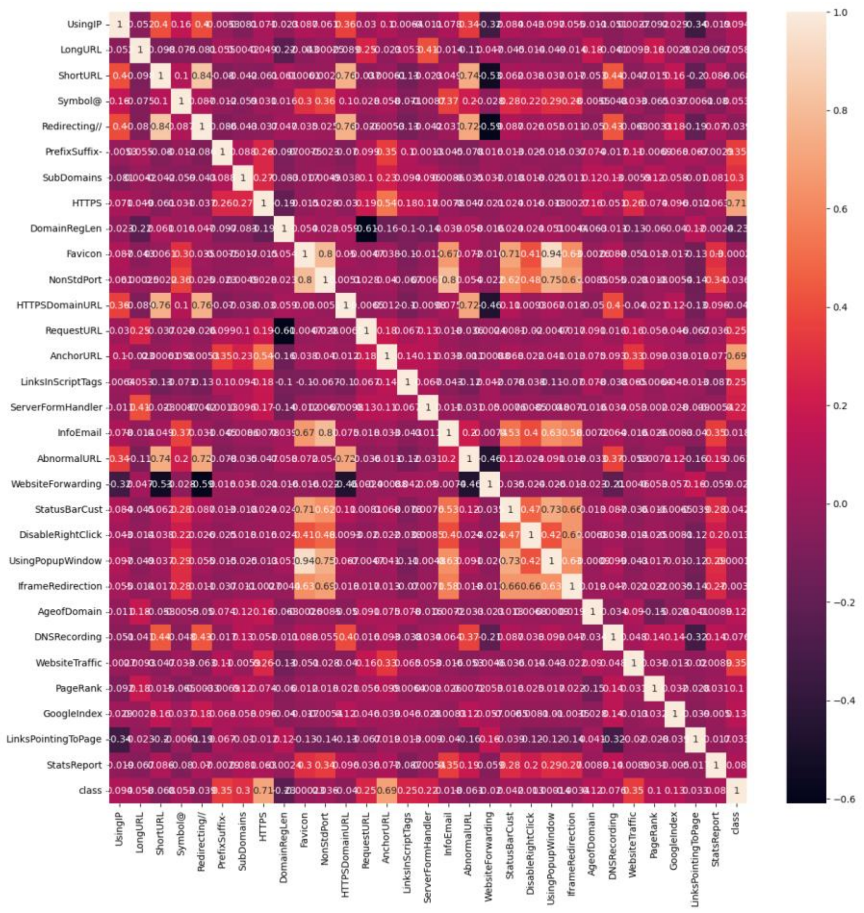


Fig 6.3 Correlation Heatmap

# Phishing count in a pie chart

plt.title("Phishing Count")

plt.show()

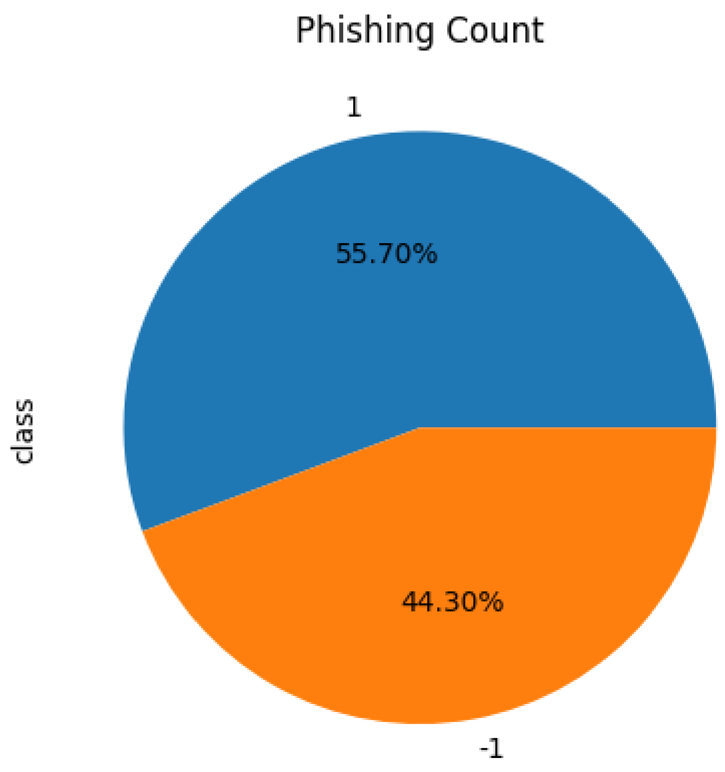


Fig 6.4 Data Pie Chart

```
# Splitting the dataset into dependent and independent features

X_phishing = phishing_data.drop(["class"], axis=1)

y_phishing = phishing_data["class"]
```

Split the data accordingly

```
from sklearn.model_selection import train_test_split

X_train_phishing, X_test_phishing, y_train_phishing, y_test_phishing = train_test_split(X_phishing, y_phishing, test_size=0.2, random_state=42)

# Creating holders to store the model performance results

ML_Model_phishing = []

accuracy_phishing = []

f1_score_phishing = []

recall_phishing = []
```

```
precision_phishing = []
# Function to store the results
def Storeresults(model, a, b, c, d):
    ML_Model_phishing.append(model)
    accuracy_phishing.append(round(a, 3))
    f1_score_phishing.append(round(b, 3))
    recall_phishing.append(round(c, 3))
    precision_phishing.append(round(d, 3))
```

Random Forest

```
# Random Forest Classifier Model
from sklearn.ensemble import RandomForestClassifier
# Instantiate the model
random_forest = RandomForestClassifier(n_estimators=100, random_state=42)
# Fit the model
random_forest.fit(X_train_phishing, y_train_phishing)
# Computing the accuracy, f1_score, Recall, precision of the model performance
acc_train_random_forest = metrics.accuracy_score(y_train_phishing, random_forest.predict(X_train_phishing))
acc_test_random_forest = metrics.accuracy_score(y_test_phishing, random_forest.predict(X_test_phishing))
f1_score_train_random_forest = metrics.f1_score(y_train_phishing, random_forest.predict(X_train_phishing))
f1_score_test_random_forest = metrics.f1_score(y_test_phishing, random_forest.predict(X_test_phishing))
```

recall_score_train_random_forest = metrics.recall_score(y_train_phishing, random_forest.predict(X_train_phishing))

recall_score_test_random_forest = metrics.recall_score(y_test_phishing, random_forest.predict(X_test_phishing))

precision_score_train_random_forest = metrics.precision_score(y_train_phishing, random_forest.predict(X_train_phishing))

precision_score_test_random_forest = metrics.precision_score(y_test_phishing, random_forest.predict(X_test_phishing))

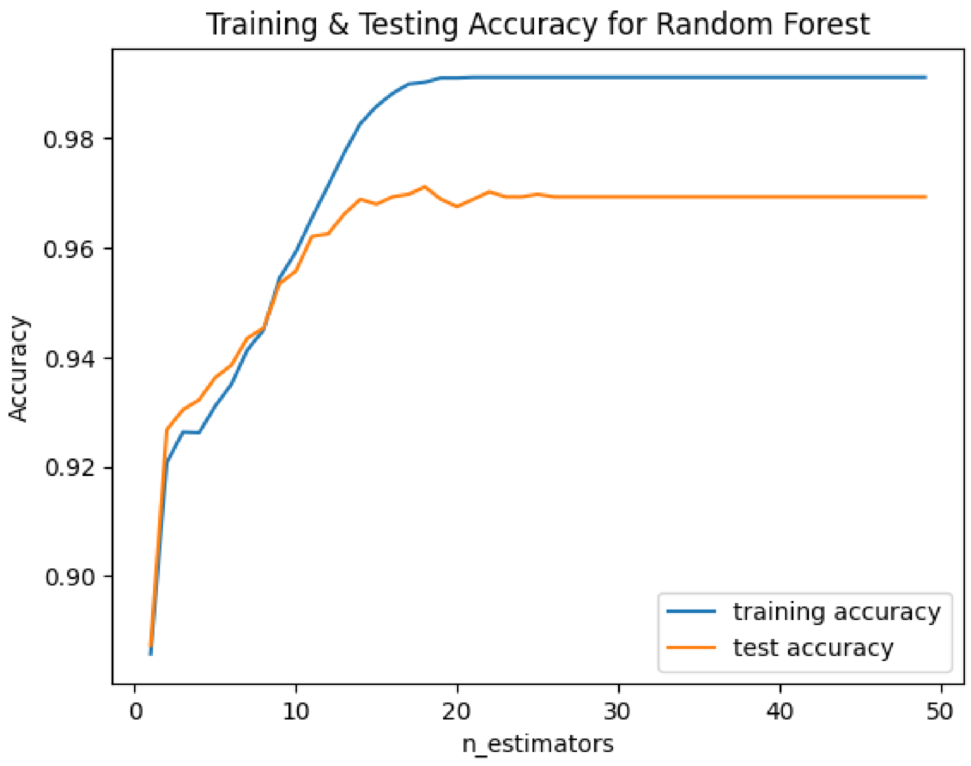


Fig 6.5: Training & Testing Accuracy for Random Forest

# Displaying the results

storeResults('Random Forest', acc_test_random_forest, f1_score_test_random_forest,

recall_score_train_random_forest, precision_score_train_random_forest)

Linear Regression

```
# Linear Regression model

from sklearn.linear_model import LogisticRegression

# Instantiate the model

logistic_regression = LogisticRegression()

# Fit the model

logistic_regression.fit(X_train_phishing, y_train_phishing)

# Predicting the target value from the model for the samples

y_train_logistic_regression = logistic_regression.predict(X_train_phishing)

y_test_logistic_regression = logistic_regression.predict(X_test_phishing)

# Computing the accuracy, f1_score, Recall, precision of the model performance

acc_train_logistic_regression = metrics.accuracy_score(y_train_phishing,
y_train_logistic_regression)

acc_test_logistic_regression = metrics.accuracy_score(y_test_phishing,
y_test_logistic_regression)

f1_score_train_logistic_regression = metrics.f1_score(y_train_phishing,
y_train_logistic_regression)

f1_score_test_logistic_regression = metrics.f1_score(y_test_phishing,
y_test_logistic_regression)

recall_score_train_logistic_regression = metrics.recall_score(y_train_phishing,
y_train_logistic_regression)

recall_score_test_logistic_regression = metrics.recall_score(y_test_phishing,
y_test_logistic_regression)

precision_score_train_logistic_regression =
metrics.precision_score(y_train_phishing, y_train_logistic_regression)
```

precision_score_test_logistic_regression = metrics.precision_score(y_test_phishing, y_test_logistic_regression)

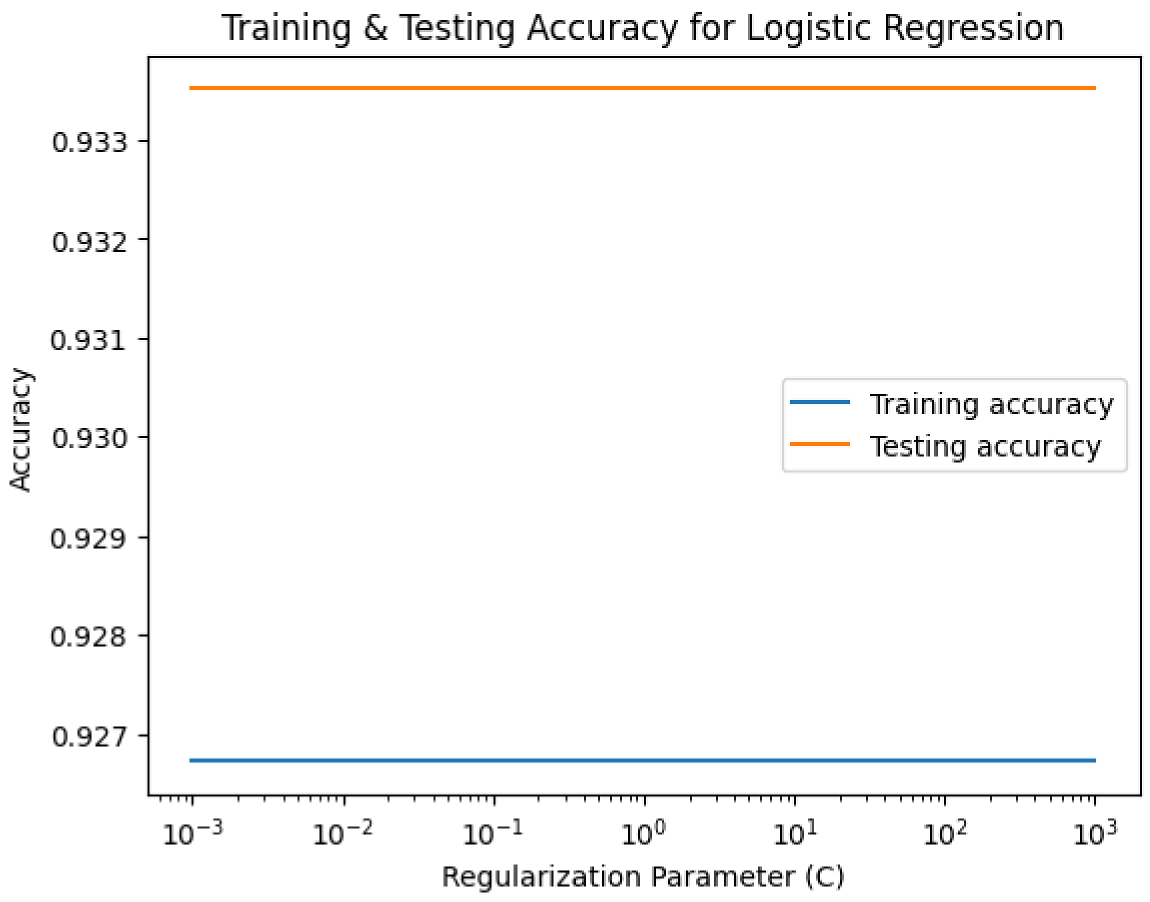


Fig 6.6: Training & Testing Accuracy for Logistic Regression

# Displaying the results

storeResults('Logistic Regression', acc_test_logistic_regression, f1_score_test_logistic_regression,

recall_score_train_logistic_regression, precision_score_train_logistic_regression)

Decision Tree Classifier

```
#Decision tree classifier model
from sklearn.tree import DecisionTreeClassifier
# Instantiate the model
decision_tree = DecisionTreeClassifier(max_depth=30)

# Fit the model
decision_tree.fit(X_train_phishing, y_train_phishing)

# Predicting the target value from the model for the samples
y_train_decision_tree = decision_tree.predict(X_train_phishing)
y_test_decision_tree = decision_tree.predict(X_test_phishing)

# Computing the accuracy, f1_score, Recall, precision of the model performance
acc_train_decision_tree = metrics.accuracy_score(y_train_phishing, y_train_decision_tree)
acc_test_decision_tree = metrics.accuracy_score(y_test_phishing, y_test_decision_tree)
f1_score_train_decision_tree = metrics.f1_score(y_train_phishing, y_train_decision_tree)
f1_score_test_decision_tree = metrics.f1_score(y_test_phishing, y_test_decision_tree)
recall_score_train_decision_tree = metrics.recall_score(y_train_phishing, y_train_decision_tree)
recall_score_test_decision_tree = metrics.recall_score(y_test_phishing, y_test_decision_tree)
```

precision_score_train_decision_tree = metrics.precision_score(y_train_phishing, y_train_decision_tree)

precision_score_test_decision_tree = metrics.precision_score(y_test_phishing, y_test_decision_tree)

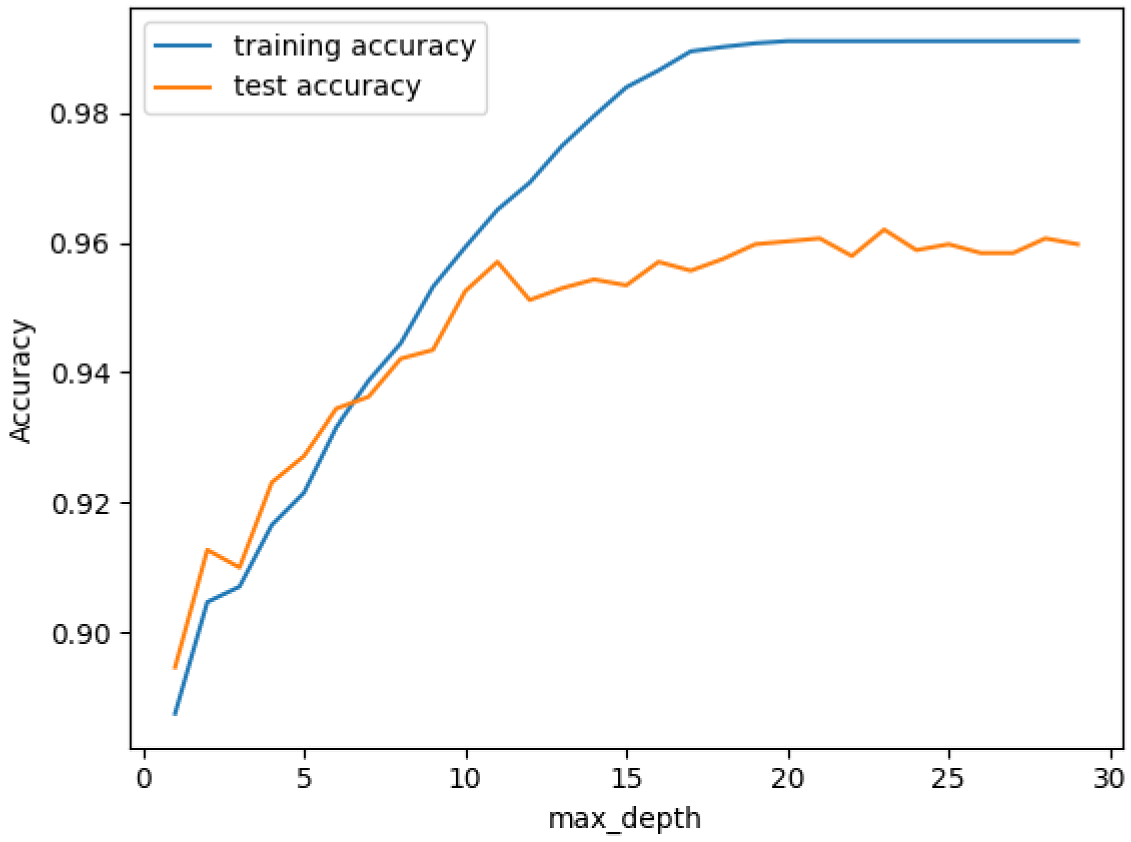


Fig 6.7: Training & Testing accuracy for Decision Tree

# Displaying the results

storeResults('Decision Tree', acc_test_decision_tree, f1_score_test_decision_tree,

recall_score_train_decision_tree, precision_score_train_decision_tree)

GAUSSIAN NAÏVE BAYES

```
# Gaussian Naive Bayes Classifier model
from sklearn.naive_bayes import GaussianNB


# Instantiate the model
naive_bayes = GaussianNB()


# Fit the model
naive_bayes.fit(X_train_phishing, y_train_phishing)


# Predicting the target value from the model for the samples
y_train_naive_bayes = naive_bayes.predict(X_train_phishing)
y_test_naive_bayes = naive_bayes.predict(X_test_phishing)


# Computing the accuracy, f1_score, Recall, precision of the model performance
acc_train_naive_bayes = metrics.accuracy_score(y_train_phishing, y_train_naive_bayes)
acc_test_naive_bayes = metrics.accuracy_score(y_test_phishing, y_test_naive_bayes)
f1_score_train_naive_bayes = metrics.f1_score(y_train_phishing, y_train_naive_bayes)
f1_score_test_naive_bayes = metrics.f1_score(y_test_phishing, y_test_naive_bayes)
recall_score_train_naive_bayes = metrics.recall_score(y_train_phishing, y_train_naive_bayes)
recall_score_test_naive_bayes = metrics.recall_score(y_test_phishing, y_test_naive_bayes)
```

precision_score_train_naive_bayes = metrics.precision_score(y_train_phishing, y_train_naive_bayes)

precision_score_test_naive_bayes = metrics.precision_score(y_test_phishing, y_test_naive_bayes)

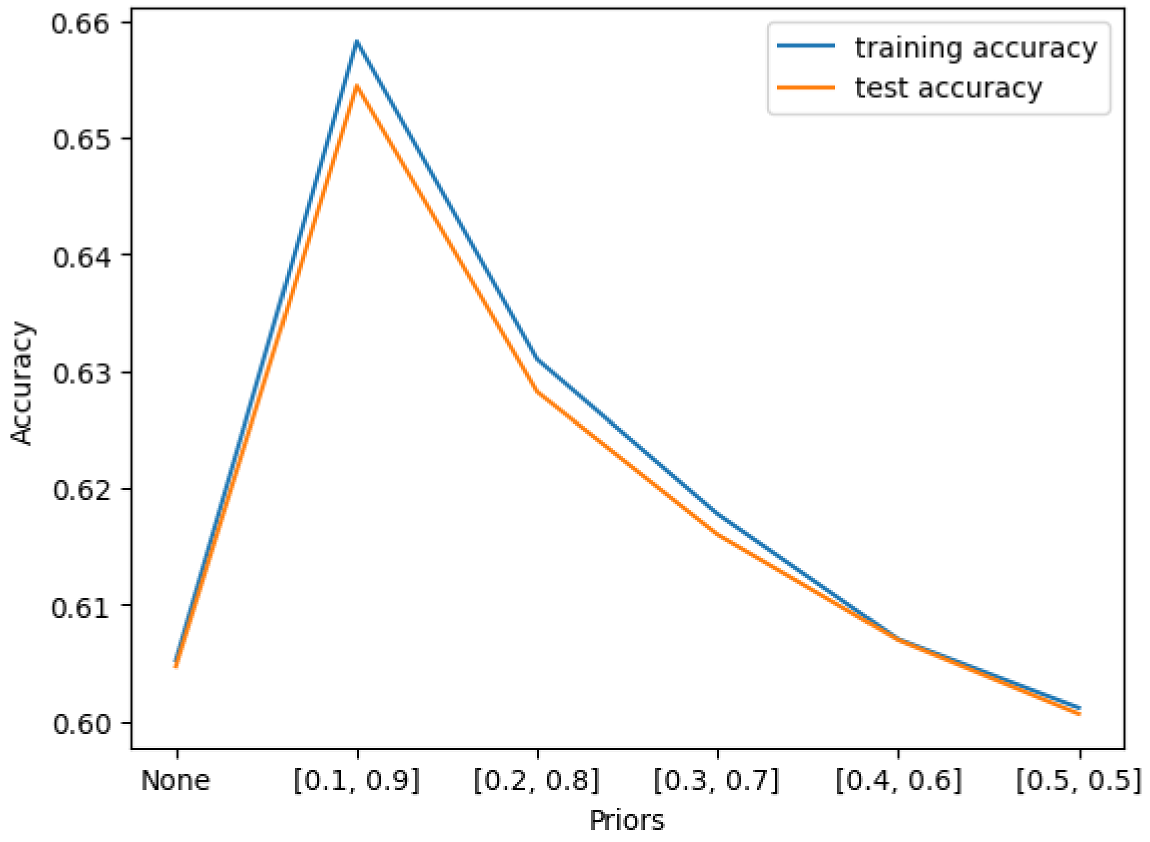


Fig 6.8: Training & Testing accuracy for Gaussian Naïve Bayes

# storing the results. The below mentioned order of parameter passing is important.

storeResults('Gaussian Naive Bayes', acc_test_nb, f1_score_test_nb,

recall_score_train_nb, precision_score_train_nb)

MULTI-LINEAR PERCEPTION

# Multi-layer Perceptron Classifier Model

from sklearn.neural_network import MLPClassifier

mlp_classifier = MLPClassifier()

```
mlp_classifier.fit(X_train_phishing, y_train_phishing)

y_train_mlp_classifier = mlp_classifier.predict(X_train_phishing)

y_test_mlp_classifier = mlp_classifier.predict(X_test_phishing)
```

Computing the accuracy, f1_score, Recall, precision of the model performance

```
acc_train_mlp_classifier = metrics.accuracy_score(y_train_phishing,
y_train_mlp_classifier)

acc_test_mlp_classifier = metrics.accuracy_score(y_test_phishing,
y_test_mlp_classifier)

f1_score_train_mlp_classifier = metrics.f1_score(y_train_phishing,
y_train_mlp_classifier)

f1_score_test_mlp_classifier = metrics.f1_score(y_test_phishing,
y_test_mlp_classifier)

recall_score_train_mlp_classifier = metrics.recall_score(y_train_phishing,
y_train_mlp_classifier)

recall_score_test_mlp_classifier = metrics.recall_score(y_test_phishing,
y_test_mlp_classifier)

precision_score_train_mlp_classifier = metrics.precision_score(y_train_phishing,
y_train_mlp_classifier)

precision_score_test_mlp_classifier = metrics.precision_score(y_test_phishing,
y_test_mlp_classifier)
```

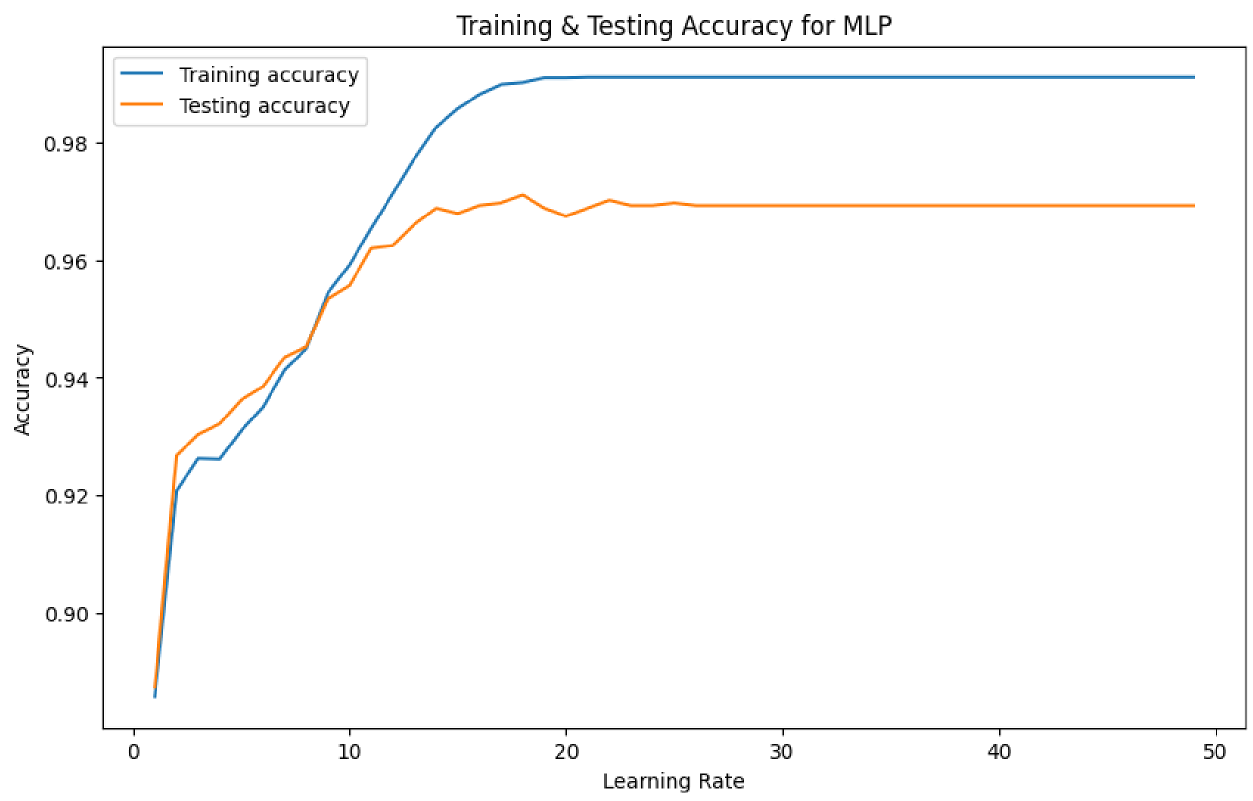


Fig 6.9: Training & Testing Accuracy for MLP

#Displaying the results

storeResults('Multi-layer Perceptron', acc_test_mlp_classifier, f1_score_test_mlp_classifier,

recall_score_train_mlp_classifier, precision_score_train_mlp_classifier)

#creating dataframe

```
result = pd.DataFrame({ 'ML Model' : ML_Model,
                        'Accuracy' : accuracy,
                        'f1_score' : f1_score,
                        'Recall'   : recall,
                        'Precision': precision,})
```

#Sorting the datafram on accuracy

#Displaying the results

```
storeResults('Multi-layer Perceptron', acc_test_mlp_classifier, f1_score_test_mlp_classifier,

recall_score_train_mlp_classifier, precision_score_train_mlp_classifier)

# Sorting the DataFrame based on Accuracy

sorted_results_phishing = result_phishing.sort_values(by=['Accuracy'], ascending=False).reset_index(drop=True)

# Plotting the bar chart

plt.figure(figsize=(10, 6))

bars_phishing = plt.bar(sorted_results_phishing['ML Model'], sorted_results_phishing['Accuracy'],

                   color=['skyblue', 'lightgreen', 'lightcoral', 'lightsalmon'])

plt.title('Testing Accuracy Comparison of ML Models for Phishing Data')

plt.xlabel('ML Model')

plt.ylabel('Testing Accuracy')

plt.ylim(0, 1)  # Set the y-axis limit between 0 and 1 for accuracy values

plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for better readability

# Displaying precise values on top of each bar

for bar, value in zip(bars_phishing, sorted_results_phishing['Accuracy']):

    plt.text(bar.get_x() + bar.get_width() / 2 - 0.1, bar.get_height() + 0.01, f'{value:.3f}', ha='center', color='black')

plt.show()
```

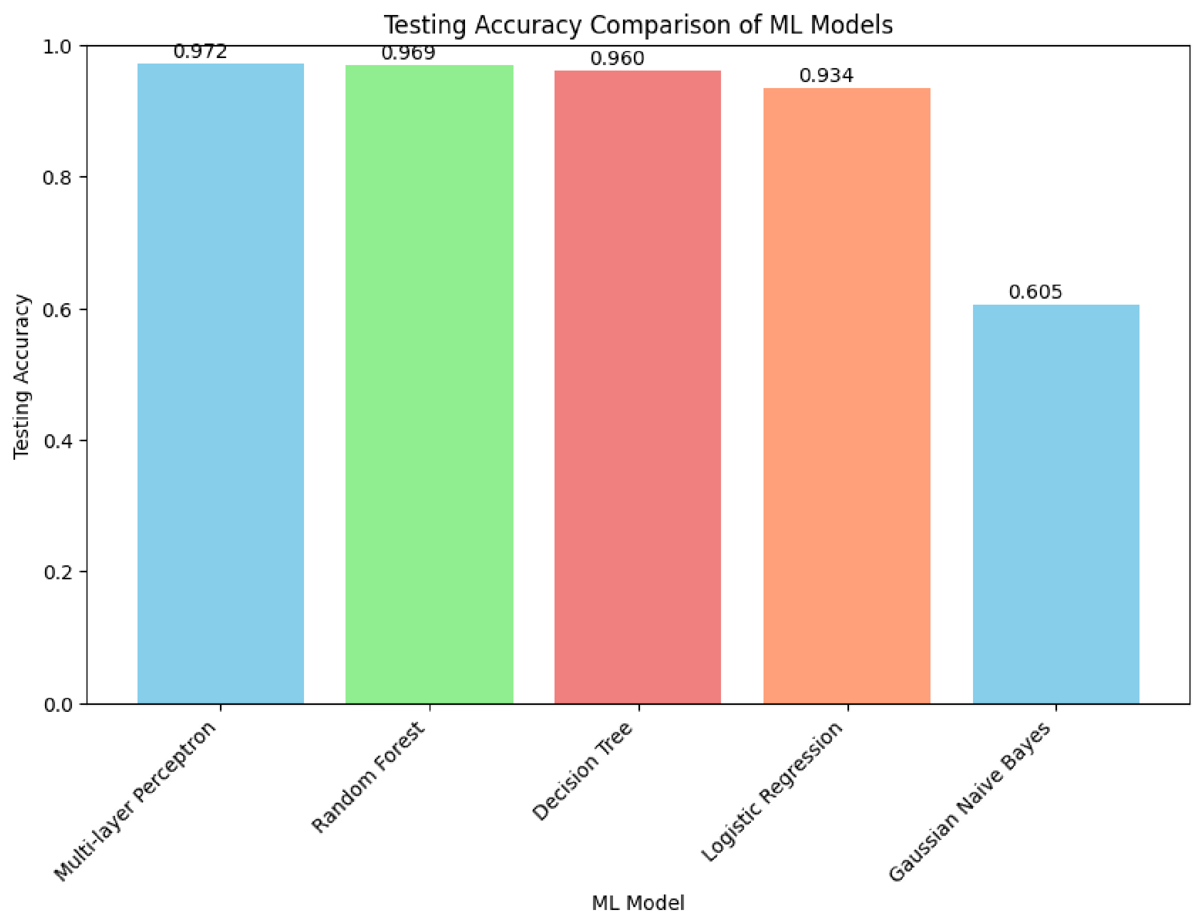
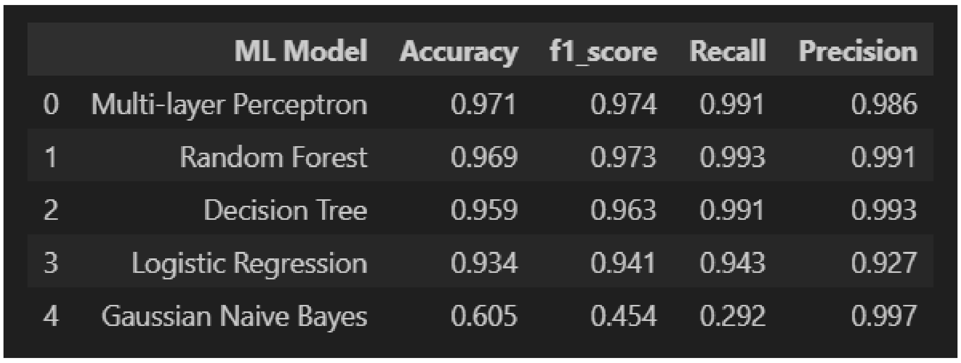


Fig 6.10: ML Model Comparisons

# Displaying the total result for Phishing Data

sorted_results_phishing



	ML Model	Accuracy	f1_score	Recall	Precision
0	Multi-layer Perceptron	0.971	0.974	0.991	0.986
1	Random Forest	0.969	0.973	0.993	0.991
2	Decision Tree	0.959	0.963	0.991	0.993
3	Logistic Regression	0.934	0.941	0.943	0.927
4	Gaussian Naive Bayes	0.605	0.454	0.292	0.997

Fig 6.11: Various ML Models Results

CHAPTER 7

FUTURE WORK

The future scope of PhishNot, a cloud-based machine-learning approach to phishing URL detection, encompasses a multifaceted approach to bolster its effectiveness and user-centric security. Advanced feature extraction will be a focal point, expanding the analysis of URL characteristics to refine phishing detection accuracy. Collaborative threat intelligence partnerships with cybersecurity organizations will be established to create a robust network for sharing insights and collectively combating evolving phishing threats. Continuous model training mechanisms will be implemented to ensure adaptability to changing attack patterns, maintaining the efficacy of the machine-learning models over time. The development of an intuitive and user-friendly dashboard with comprehensive analytics will empower users to visualize threat trends, patterns, and their protection statistics. Additionally, the expansion of PhishNot through browser plugins, integration into IoT devices, and the exploration of smart DNS filtering will broaden its reach and provide additional layers of security. Incident response automation and dynamic risk scoring will be introduced to expedite threat neutralization and offer nuanced evaluations of URL risks. The integration of blockchain technology aims to enhance the integrity of PhishNot's threat database, ensuring secure and tamper-resistant data. Furthermore, user education will be emphasized through interactive training modules, and the system will extend protection beyond web browsers, integrating with various online platforms, email clients, and messaging applications. The incorporation of AI-powered URL categorization, localized threat detection, and multi-language support will tailor the system to diverse user needs, while behavioral biometrics and predictive analytics will add layers of security by incorporating user behavior insights and anticipating emerging threats. The implementation of a secure user feedback mechanism and the development of a global threat heatmap will enhance user engagement and provide real-time insights into the evolving global phishing landscap.

CHAPTER 8

CONCLUSION

The PhishNot project, leveraging a cloud-based machine-learning approach to phishing URL detection, exhibits a robust and forward-looking trajectory in the realm of cybersecurity. The envisioned advancements include continuous refinement of machine-learning models, collaborative integration of threat intelligence, and the incorporation of dynamic feature extraction techniques. The commitment to user-centric security is evident in the development of a user-friendly dashboard with comprehensive analytics, ensuring that users can proactively engage with and understand threat trends. The expansion of PhishNot through browser plugins, IoT device integration, and smart DNS filtering underscores its commitment to providing versatile and multi-layered protection. The introduction of incident response automation and dynamic risk scoring reflects a dedication to swift threat neutralization and nuanced risk assessment.

The exploration of cutting-edge technologies such as blockchain to enhance data integrity, AI-powered URL categorization, and behavioral biometrics integration demonstrates a forward-thinking approach to bolstering the system's accuracy and adaptability. The emphasis on user education, through interactive training modules, and the extension of protection to various online platforms further positions PhishNot as a comprehensive solution in the cybersecurity landscape. Localization features, multi-language support, and predictive analytics showcase a commitment to addressing diverse user needs and anticipating emerging threats. The implementation of a secure user feedback mechanism and the development of a global threat heatmap add layers of user engagement and real-time insights into the ever-evolving global phishing landscape.

CHAPTER 9

REFERENCES

[1] M. Khonji, Y. Iraqi, A. Jones, Phishing detection: a literature survey, IEEE Commun. Surv. Tutor. 15 (4) (2013) 2091-.

[2] D.D. Caputo, S.L. Pfleeger, J.D. Freeman, M.E. Johnson, Going spear phishing:

Exploring embedded training and awareness, IEEE Secur. Priv. 12 (1) (2013) 28-38.

[3] Time to report phishing email 2020 | statista, Statista (2021) [Online; accessed 24. Dec. 2021]