A Major Project Report

On

DETECTION OF PHISHING WEBSITES

Submitted in partial fulfilment of the requirements for the award of the degree

Of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

By

KATIKE VAISHNAVI	**20EG105120**
PASUPULETI SHILPA	**20EG105139**
SANGU NAVEEN	**20EG105143**
CHENAGONI UDAY GOUD	**20EG105718**

Under the guidance of

Dr. K. SHAILAJA

Assistant Professor

Department of CSE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Venkatapur(V), Ghatkeshar(M), Medchal(D) – 500088

2023-24

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the report entitled "**DETECTION OF PHISHING WEBSITES"** that is being submitted by **KATIKE VAISHNAVI [20EG105120], PASUPULETI SHILPA [20EG105139], SANGU NAVEEN [20EG105143], CHENAGONI UDAY GOUD [20EG105718]** in partial fulfillment for the award of Bachelor of Technology in Computer Science and Engineering to the Anurag University, Hyderabad is a record of bonafide work carried out by them under my guidance and supervision.The results embodied in this project report have not been submitted to any other University or Institute for the award of any Degree or Diploma.

Internal Guide
Dr. K. Shailaja
Assistant Professor, Dept. of CSE

Dr. G. Vishnu Murthy
Professor &Dean, Dept. of CSE

External Examiner

ACKNOWLEDGEMENT

We would like to express our sincere thanks and deep sense of gratitude to project supervisor **Dr. K. Shailaja** for her constant encouragement and inspiring guidance without which this project could not have been completed. Her critical reviews and constructive comments improvedour grasp of the subject and steered to the fruitful completion of the work. Her patience, guidance and encouragement made this project possible.

We would like to acknowledge our sincere gratitude for the support extended by **Dr. G. Vishnu Murthy**, Dean, Dept. of CSE, Anurag University. We also express our deep sense of gratitude to**Dr. V V S S S Balaram,** Academic coordinator, **Dr. Pallam Ravi**, Project in-Charge, **Dr. G. Prabhakar Raju** Project Coordinator and Project review committee members, whose research expertise and commitment to the highest standards continuously motivated us during the crucial stage our project work.

We would like express our special thanks to **Dr. V. Vijaya Kumar**, Dean School of Engineering,Anurag University, for his encouragement and timely support in our B. Tech program.

KATIKE VAISHNAVI
(20EG105120)

PASUPULETI SHILPA
(20EG105139)

SANGU NAVEEN
(20EG105143)

CHENAGONI UDAY GOUD
(20EG105718)

DECLARATION

We, hereby declare that the Report entitled **"DETECTION OF PHISHING WEBSITES"** submitted for the award of Bachelor of technology Degree is our original work and the Report has not formed the basis for the award of any degree, diploma, associate ship or fellowship of similar other titles. It has not been submitted to any other University or Institution for the award of any degree or diploma.

KATIKE VAISHNAVI
(20EG105120)

PASUPULETI SHILPA
(20EG105139)

SANGU NAVEEN
(20EG105143)

CHENAGONI UDAY GOUD
(20EG105718)

PLACE: HYDERABAD

DATE:

ABSTRACT

In the contemporary digital landscape, the prevalence of phishing attacks poses a significant concern, as cybercriminals employ deceptive tactics to trick individuals into divulging sensitive personal information. To address this growing threat, we have developed an intelligent phishing detection system leveraging advanced machine learning techniques. Rather than merely reacting to phishing attacks after they occur, our proactive system is designed to preemptively identify and thwart potential threats before they can inflict harm.

Our system is trained to recognize the subtle indicators and characteristics commonly associated with phishing websites. Through the utilization of sophisticated computer algorithms, including decision trees and neural networks, the system undergoes extensive learning to differentiate between legitimate and fraudulent websites effectively. By analyzing various features and patterns within website URLs and content, the system can accurately discern the malicious intent behind phishing websites, enabling it to identify and flag potential threats in real-time.

To ensure the reliability and effectiveness of our phishing detection system, we conducted rigorous testing under diverse scenarios and challenges to simulate real-world conditions. The comprehensive testing process validated the system's capability to consistently and accurately identify phishing websites, demonstrating its robust performance and resilience against evolving cyber threats.

Our overarching goal is to enhance online safety and security by staying one step ahead of cybercriminals. By leveraging innovative machine learning approaches and proactive detection mechanisms, we aim to make it increasingly challenging for cybercriminals to deceive and exploit internet users. Through continuous research, development, and refinement, we are committed to safeguarding the online ecosystem and protecting users from the risks and consequences associated with phishing attacks,thereby fostering a safer and more secure internet environment for all users.

Table of Contents

Table of Figures

CHAPTER - 1. INTRODUCTION

In today's digital age, where the internet is an integral part of our daily lives, cybersecurity is of paramount importance. One of the prevalent threats on the web is phishing, a malicious practice where attackers deceive users into divulging sensitive information such as passwords, credit card details, or personal data by masquerading as trustworthy entities. To combat this threat, advanced techniques are being developed, including the use of machine learning algorithms for detecting phishing websites.

The project "Detection of Phishing Websites" aims to leverage machine learning models to distinguish between legitimate and phishing websites automatically. This project utilizes a dataset sourced from platforms like Kaggle, containing labeledexamples of both types of websites. Through meticulous data preprocessing and featureengineering, key attributes are extracted from the URLs provided as input by users.

Feature engineering plays a pivotal role in this project, involving the extraction of URL components such as domain, subdomain, path, and query parameters. Additionally, factors like URL length, presence of special characters, and domain reputation are considered as features. Integration with external APIs and analysis of HTTP headers and webpage content further enriches the feature set, enabling the model to make informed decisions.

Various machine learning algorithms, including decision trees, random forests, logistic regression, and neural networks, are explored and evaluated to determine the most effective approach for phishing detection. Evaluation metrics such as accuracy, are employed to assess the model's performance on validation and test datasets.

Ultimately, the goal of this project is not only to build an accurate and reliable phishing detection system but also to contribute to the ongoing efforts in enhancing cybersecurity measures and safeguarding users' sensitive information in the digital realm.

CHAPTER -2. LITERATURE SURVEY

Alswailem et al. [1] explore the application of machine learning techniques for detecting phishing websites in their paper titled “Detecting Phishing Websites Using Machine Learning.” The process begins with feature extraction, where a thorough analysis of website attributes such as URL elements, HTML source code, and JavaScript code is conducted. This step aims to identify key indicators that distinguish phishing websites from legitimate ones. Subsequently, machine learning classification techniques, including algorithms such as Naive Bayes, Support Vector Machines (SVM), and Random Forest, are employed. These algorithms are trained on a labeled dataset comprising both phishing and legitimate websites, enabling them to learn and recognize patterns in website characteristics indicative of phishing behavior. Once the algorithms are trained, they are utilized for prediction, wherein new websites are classified based on the extracted features using the learned models. This predictive capability allows for the proactive identification and classification of potential phishing websites, thereby enhancing cybersecurity measures and mitigating the risks posed by online threats. The research by Razaque et al. [2] delve into the detection of phishing websites using machine learning techniques in their study titled “Detection of PhishingWebsites using Machine Learning,” presented at the 2020 IEEE Cloud Summit. The authors focus on implementing a comprehensive strategy to enhance cybersecurity measures, beginning with the establishment of blacklisting, which involves maintaininga database of known phishing websites to provide initial protection. Concurrently, theyemploy semantic analysis to scrutinize website content, including text, links, and images, for anomalies and suspicious keywords, thereby enhancing detection capabilities. Furthermore, the authors prioritize pattern recognition techniques, which entail the extraction of features such as URL length, presence of suspicious words, and domain extensions. These features are then subjected to machine learning algorithms to identify patterns, facilitating the detection of potential phishing attempts. To ensure real-time detection and proactive defense, the authors seamlessly integrate all these techniques into a Google Chrome extension. This extension empowers users with immediate protection as they browse the web, leveraging the combined capabilities of blacklisting, semantic analysis, and pattern recognition to safeguard against the evolving landscape of online threats. The literature on “Detecting Phishing Websites

Using Machine Learning." by M. H. Alkawaz and his collaborators [3] encompasses several key steps aimed at enhancing phishing detection capabilities. Initially, feature extraction involves a meticulous analysis of various website attributes, encompassing elements such as URL length, the presence of suspicious words, iframe tags, and visual elements like fonts and colors. Subsequently, feature selection employs statistical techniques to discern the most pertinent features for effective phishing detection, streamlining the analysis process. The subsequent phase involves machine learning classification, wherein algorithms such as Naive Bayes, Logistic Regression, and Support Vector Machines (SVM) are trained on a labeled dataset comprising both phishing and legitimate websites, utilizing the selected features. This training process enables the algorithms to discern patterns and characteristics indicative of phishing behavior. Finally, in the classification and prediction stage, newly encountered websites are evaluated based on their extracted features using the trained machine learning models, facilitating swift and accurate identification of potential phishing threats. This comprehensive approach leverages advanced statistical techniques and machine learning algorithms to bolster cybersecurity measures, ultimately enhancing protection against the pervasive threat of phishing attacks.

CHAPTER 3- PROPOSED METHOD

3.1 Data Collection:

- To build an effective phishing detection model, it is essential to start by identifying and gathering a diverse dataset of URLs from reputable sources. These sources can include platforms like Kaggle, cybersecurity research repositories, or public domain lists of known phishing and legitimate websites. By sourcing data from these trusted platforms, the quality and reliability of the dataset are enhanced, ensuring that the URLs used for training and evaluation are representative of real-world scenarios.
- Furthermore, it is crucial to ensure that the dataset maintains a balanced distribution between phishing and legitimate URLs. This balance is essential to prevent the model from being biased towards one class, which could result in inaccurate and unreliable predictions. Therefore, it is important to include a sufficient number of examples for both phishing and legitimate URLs in the dataset. This balanced representation will enable the model to learn effectively from both classes and make more accurate classifications during training and evaluation.
- Lastly, to facilitate supervised learning and guide the model's training process, each URL in the dataset should be accompanied by labels or annotations indicating its ground truth classification, i.e., whether it is a phishing or legitimate website. These labels serve as the target variable for the machine learning model, allowing it to learn the patterns and characteristics associated with phishing and legitimate websites. With this annotated dataset, the model can be trained to recognize and differentiate between phishing and legitimate URLs based on the predefined features and classifications. Thus, a well-structured and balanced dataset with clear annotations is fundamental for developing a robust and accurate phishing detection model.

3.2 Data Preprocessing:

- Perform initial data exploration to understand the structure and quality of the collected dataset.

- Clean the dataset by removing duplicate URLs, handling missing values (if any), and addressing any inconsistencies or anomalies in the data.
- Standardize the format of URLs (e.g., ensuring uniformity in protocol type, domain names, and URL paths) to minimize variations that may affect model performance.
- Convert categorical features such as protocol type (HTTP, HTTPS), domain extensions (.com, .org, etc.), and URL structure into numerical or binary representations using techniques like one-hot encoding or label encoding.
- Normalize or scale numerical features such as URL length, depth of URL, and presence of special characters to ensure uniformity and prevent bias during model training.

3.3 Feature Engineering:

- Extract a comprehensive set of features from the preprocessed URLs to capture relevant information for phishing detection.
- Consider features such as:
    - URL components (domain, subdomain, path, query parameters).
    - Domain reputation scores obtained from external sources or reputation databases.
    - Presence of suspicious keywords or patterns indicative of phishing attempts.
    - Usage of secure protocols (HTTPS) or absence of SSL certificates.
    - URL length, depth, and complexity metrics.
    - HTTP headers and metadata information (server type, content type, cookies).
    - Domain age, WHOIS information, and IP address reputation.
    - WHOIS: WHOIS is a public database that houses the information collected when someone registers a domain name or updates their DNS settings.

      ICANN, the International Corporation for Assigned Names and Numbers, regulates the WHOIS database. They've done so since 1982, back in the wild and wooly days of the early Internet.[4]

- Explore advanced feature selection techniques (e.g., recursive feature elimination, feature importance ranking) to identify the most informative features for model training.

3.4 Model Evaluation:

- Split the Preprocessed Dataset: Divide the preprocessed dataset into training, validation, and testing sets using appropriate ratios to ensure robust model evaluation and prevent overfitting or underfitting. Allocate a significant portion of the data to the training set to allow the model to learn the underlying patterns and features effectively. Reserve a portion of the data for the validation set to fine-tune the model's hyperparameters and assess its performance during the training phase. Keep a separate testing set to evaluate the final performance of the trained models objectively and assess their generalization capabilities on unseen data.

- Select and Train Multiple Machine Learning Models: Choose a variety of machine learning algorithms such as decision trees, random forests, support vector machines (SVM), logistic regression, or ensemble methods like XGBoost to train and evaluate their performance. Implement and train each selected model using the training set to learn the patterns and characteristics of phishing and legitimate URLs present in the dataset. Optimize the parameters of each model to improve its performance and achieve better accuracy and reliability in phishing detection.

- Evaluate Each Model's Performance: Assess the performance of each trained model using a range of evaluation metrics, including accuracy and evaluate its effectiveness in distinguishing between phishing and legitimate URLs.Compare the performance of different machine learning models to identify the most effective and reliable model for phishing detection based on the evaluation metrics.

- Perform Hyperparameter Tuning and Cross-Validation: Perform hyperparameter tuning for each machine learning model using techniques like grid search or random search to find the optimal set of parameters that maximize

the model's performance. Implement cross-validation techniques such as k-fold cross-validation to assess the model's robustness and prevent overfitting or underfitting on the training data. Fine-tune the model's hyperparameters based on the results of cross-validation to optimize its performance, improve accuracy, and enhance the model's generalization capabilities on unseen data.

3.5 Front-End Development:

- Develop a user-friendly front-end interface using a web framework such as Flask, Django, or React for seamless interaction with the phishing detection system.
- Design an intuitive input form where users can enter URLs for analysis and receive instant feedback on the phishing risk level.
- Implement client-side validation to ensure the input URLs meet required criteria (e.g., valid syntax, supported protocols).
- Incorporate responsive design principles and interactive elements (e.g., progress indicators, tooltips) to enhance user experience and engagement.

3.6 Model Integration:

- Integrate the trained machine learning model into the front-end application backend using appropriate libraries or frameworks (e.g., Flask RESTful API, Django REST framework).
- Implement secure APIs for handling URL predictions, ensuring data confidentiality, integrity, and authentication mechanisms (e.g., API keys, JWT tokens).
- Optimize model inference speed and resource utilization for real-time or batch processing of user-submitted URLs.
- Conduct thorough testing of model integration to verify correct functionality and performance across different deployment environments (e.g., local development, staging, production).

3.7 Result Presentation:

- Display prediction results prominently on the front-end interface, providing clear indications of whether the input URL is classified as safe (legitimate) or potentially malicious (phishing).
- Include informative visualizations or textual explanations detailing the factors influencing the model's prediction, such as important features or risk assessment scores.
- Offer actionable insights or recommendations for users based on the phishing risk level detected, such as avoiding interaction with suspicious URLs or reporting false positives for further analysis.

3.8 Testing and Validation:

- Conduct comprehensive testing of the entire application, covering functional testing (e.g., input validation, prediction accuracy), usability testing (e.g., user feedback, interface intuitiveness), and security testing (e.g., vulnerability assessments, penetration testing).
- Perform validation checks with diverse sets of URLs, including known phishing examples, legitimate websites, and edge cases to validate the model's accuracy, robustness, and generalization capabilities.

- Implement logging and monitoring mechanisms to track application performance, user interactions, and model predictions for continuous improvement and maintenance.

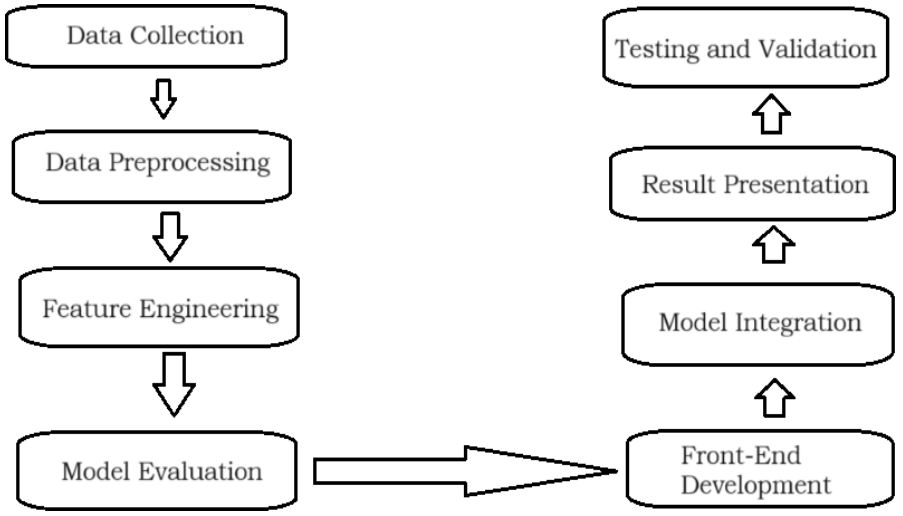

Figure 3.1: Methodology of Proposed Method

CHAPTER-4. IMPLEMENTATION

4.1 UML Diagrams:

A UML diagram is a way to visualize systems and software using Unified Modeling Language (UML). Software engineers create UML diagrams to understand the designs, code architecture, and proposed implementation of complex software systems. UML diagrams are also used to model workflows and business processes. Coding can be a complicated process with many interrelated elements. There are often thousands of lines of programming language that can be difficult to understand at first glance. A UML diagram simplifies this information into a visual reference that's easier to digest.It uses a standardized method for writing a system model and capturing conceptual ideas.[5]

1. **Class Diagram:**

   class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

   Purpose of Class Diagrams

   - Shows static structure of classifiers in a system
   - Diagram provides a basic notation for other structure diagrams prescribed by UML
   - Helpful for developers and other team members too
   - Business Analysts can use class diagrams to model systems from a business perspective

   A UML class diagram is made up of:

   - A set of classes and
   - A set of relationships between classes.

   The heart of this application is the Webpage class, which serves as the user interface where users can interact with the system. Supporting the webpage is the Flask class, which acts as the backend server, handling requests and coordinating between different components. For data processing, you have the Feature Extraction class, responsible for transforming raw data into a format suitable for machine learning. This processed data is then used by the Training class, which oversees the training of a machine learning model. This model is

specifically an XGBoost Model, representing a trained XGBoost algorithm that has learned from the processed data. Lastly, for making predictions or classifications based on new input data, the Prediction class comes into play. It takes the input, processes it through feature extraction if needed, and then utilizes the trained XGBoost Model to provide predictions. In summary, your class diagram outlines a system where a Flask-powered webpage interacts with a machine learning pipeline involving feature extraction, model training with XGBoost, and prediction capabilities.

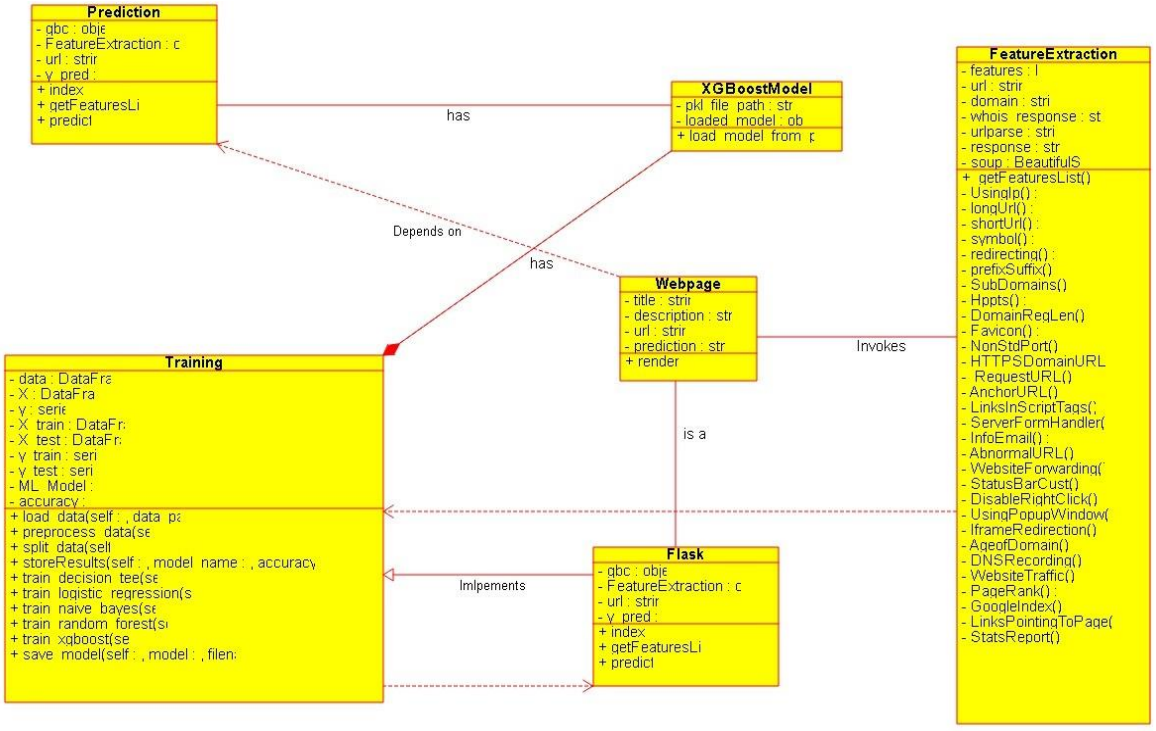

Figure 4.1.1: Class Diagram

2. Sequence Diagram:

The sequence diagram represents the flow of messages in the system and is also termed as an event diagram. It helps in envisioning several dynamic scenarios. It portrays the communication between any two lifelines as a time- ordered sequence of events, such that these lifelines took part at the run time.In UML, the lifeline is represented by a vertical bar, whereas the message flow is represented by a vertical dotted line that extends across the bottom of the page. It incorporates the iterations as well as branching.

Purpose of a Sequence Diagram

1. To model high-level interaction among active objects within a system.
2. To model interaction among objects inside a collaboration realizing a use case.
3. It either models generic interactions or some certain instances of interaction.[7]

The User initiates the process by interacting with the Webpage object. Upon entering a URL for prediction, a message is sent from the Webpage to the Flask object, which serves as the backend server handling the requests.

Once the Flask object receives the URL request, it triggers the Feature Extraction object to process the data. After the features are extracted, the processed data is then utilized by the Prediction object. This prediction is made using the XGBoost Model object, which represents the trained machine learning model.

Additionally, the sequence diagram suggests that the User can perform more advanced interactions with the System. The User can send a message to the Flask object to Load ML Model or Train ML Model, indicating the ability for the user to manage and modify the machine learning model as required.

In summary, your sequence diagram showcases the flow of messages and interactions between the User, Webpage, Flask, Feature Extraction, XGBoost Model, and Prediction objects, highlighting the steps involved in data processing, prediction, and potential model management within the system.

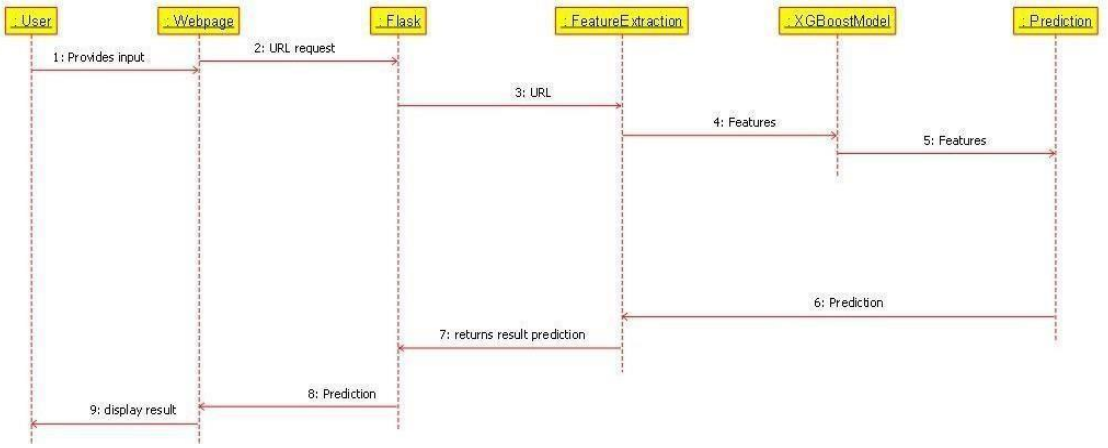

Figure 4.1.2: Sequence Diagram

3. Use case Diagram:

In UML, use-case diagrams model the behavior of a system and help to capture the requirements of the system.

Use-case diagrams describe the high-level functions and scope of a system. These diagrams also identify the interactions between the system and its actors. The use cases and actors in use-case diagrams describe what the system does and how the actors use it, but not how the system operates internally.

Use-case diagrams illustrate and define the context and requirements of either an entire system or the important parts of the system. You can model a complex system with a single use-case diagram, or create many use-case diagrams to model the components of the system. You would typically develop use-case diagrams in the early phases of a project and refer to them throughout the development process.

Use-case diagrams are helpful in the following situations:

Before starting a project, you can create use-case diagrams to model a business so that all participants in the project share an understanding of the workers, customers, and activities of the business.

While gathering requirements, you can create use-case diagrams to capture the system requirements and to present to others what the system should do.

During the analysis and design phases, you can use the use cases and actors from your use-case diagrams to identify the classes that the system requires.

During the testing phase, you can use use-case diagrams to identify tests for the system.[8]

User and System, and their associated use cases. The User initiates the interaction by accessing the Webpage through the Access Webpage use case. Once on the webpage, the User can enter a URL for prediction using the Enter URL for Prediction use case.

The System responds to these actions by performing feature extraction through the Extract Features use case after receiving the URL request. Following feature extraction, the System proceeds to make a prediction using the Perform

Prediction use case. This prediction is based on the processed data and the trained XGBoost Model.

Additionally, the use case diagram shows that the User has more advanced capabilities to interact with the System. The User can Load ML Model or even Train ML Model. These use cases indicate that the system allows the user to manage and modify the machine learning model as needed, providing flexibility and control over the predictive capabilities of the system.

In summary, your use case diagram outlines a user-centric system where the user can access a webpage, enter data for prediction, and interact with a machine learning model for both prediction and management purposes. The system, represented by the System actor, facilitates these interactions through various use cases, providing a clear and structured view of the system's functionality.

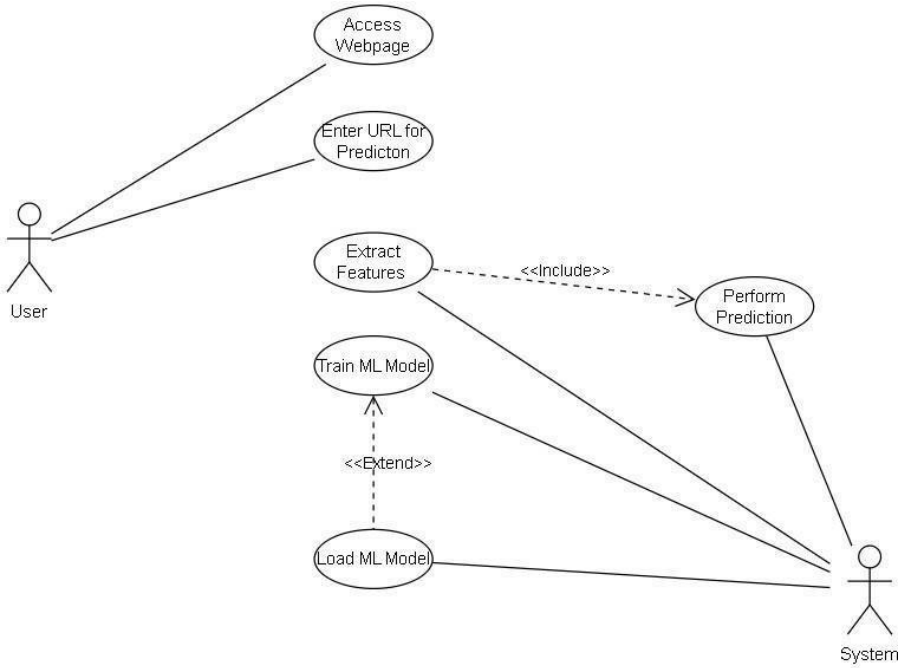

Figure 4.1.3: Use case Diagram

4. **Component Diagram:**

UML Component diagrams are used in modeling the physical aspects of object-oriented systems that are used for visualizing, specifying, and documenting component-based systems and also for constructing executable

systems through forward and reverse engineering. Component diagrams are essentially class diagrams that focus on a system's components that often used to model the static implementation view of a system.[9]

component diagram outlines the system's architecture with five main components. The Webpage serves as the user interface, facilitating user interactions. Data preprocessing is handled by the Feature Extraction component, transforming raw data into a suitable format for machine learning. The Training component is responsible for training the XGBoost Model, which represents the trained machine learning algorithm ready for predictions. The final step is executed by the Prediction component, which uses the processed data from Feature Extraction and the trained XGBoost Model to generate predictions or classifications. Together, these components illustrate a structured flow where data is processed, trained, and then used for predictive tasks, all orchestrated through user interactions via the Webpage component.

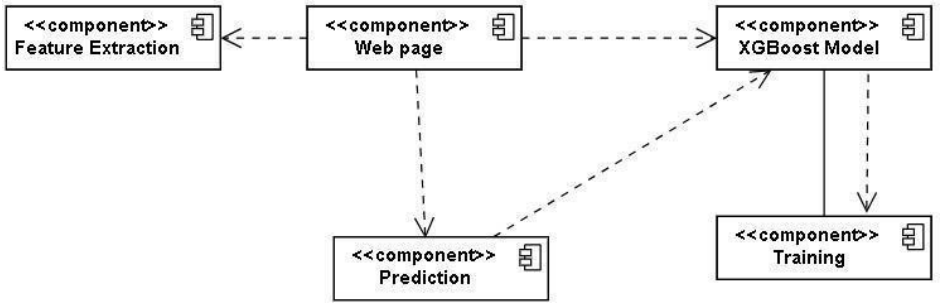

Figure 4.1.4: Component Diagram

5. **Deployment Diagram:**

In the context of the Unified Modeling Language (UML), a deployment diagram falls under the structural diagramming family because it describes an aspect of the system itself. In this case, the deployment diagram describes the physical deployment of information generated by the software program on hardware components. The information that the software generates is called an artifact. This shouldn't be confused with the use of the term in other modeling approaches like BPMN.

Deployment diagrams are made up of several UML shapes. The three-dimensional boxes, known as nodes, represent the basic software or hardware

elements, or nodes, in the system. Lines from node to node indicate relationships, and the smaller shapes contained within the boxes represent the software artifacts that are deployed.

Deployment diagram applications

Deployment diagrams have several valuable applications. You can use them to:

- Show which software elements are deployed by which hardware elements.
- Illustrate the runtime processing for hardware.
- Provide a view of the hardware system’s topology.[10]

Deployment diagram offers a comprehensive view of the system's physical architecture, detailing the allocation of components to specific nodes and their interconnections. At the forefront, the Client Browser node represents the primary interface for users to interact with the system. This node establishes communication with the Web Browser node, which serves as the hosting environment for the Flask component. Flask acts as the backend server, managing user requests, orchestrating data flow, and coordinating the system's responses.

Parallelly, the Machine Learning Server node is dedicated to handling the system's machine learning processes. Within this node, the Feature Extraction component plays a crucial role in preprocessing raw data, refining it into a format suitable for machine learning analysis. Following preprocessing, the Training component takes charge, utilizing the processed data to train the XGBoost Model. This model, once trained, is stored within the XGBoost Model component and is leveraged by the Prediction component for generating accurate predictions or classifications based on new input data.

The deployment architecture effectively segregates the user interface from the backend functionalities, ensuring optimized performance, scalability, and maintainability. By distributing components across specialized nodes, the

system can handle user interactions seamlessly while efficiently managing machine learning tasks, offering users a robust and responsive experience.

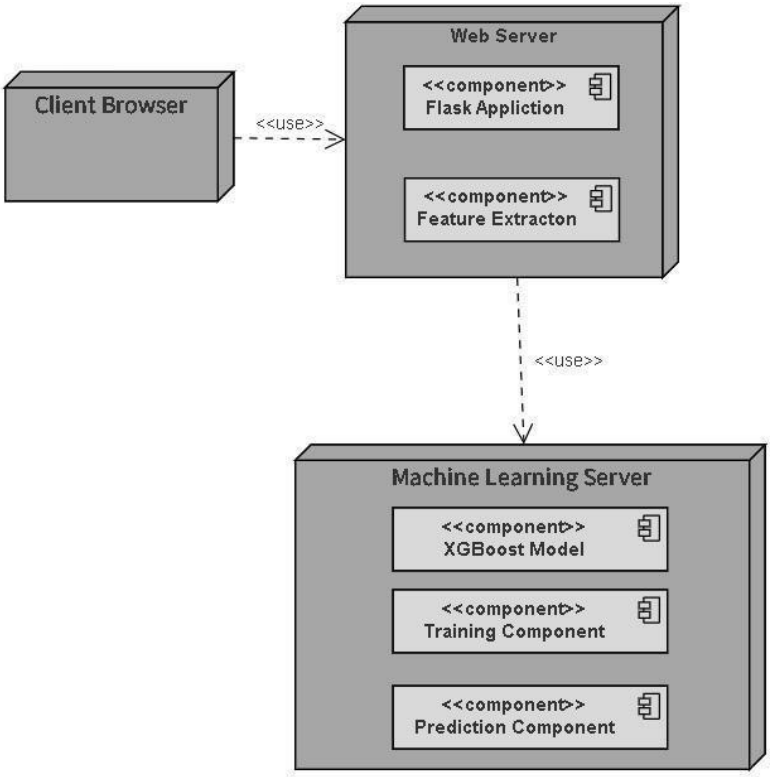

Figure 4.1.5: Deployment Diagram

4.2 List of Program Files:

1) App.py: This is the main application script where the backend logic is implemented. It likely includes code for integrating the machine learning model, handling user inputs, making predictions, and serving responses to the front-end.
    a) Importing Required Libraries:
        i) Flask: A web framework for building web applications in Python. Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various

open authentication technologies and several common framework related tools. Applications that use the Flask framework include Pinterest and LinkedIn.[11]

ii) request: Allows handling HTTP requests in Flask.

iii) render_template: Renders HTML templates in Flask.

iv) numpy and pandas: Libraries for numerical computations and data manipulation.

- Numpy: NumPy (Numerical Python) is an open-source Python library that's used in almost every field of science and engineering. It's the universal standard for working with numerical data in Python, and it's at the core of the scientific Python and PyData ecosystems. NumPy users include everyone from beginning coders to experienced researchers doing state-of-the-art scientific and industrial research and development. The NumPy API is used extensively in Pandas, SciPy, Matplotlib, scikit-learn, scikit-image and most other data science and scientific Python packages.

  The NumPy library contains multidimensional array and matrix data structures (you'll find more information about this in later sections). It provides ndarray, a homogeneous n-dimensional array object, with methods to efficiently operate on it. NumPy can be used to perform a wide variety of mathematical operations on arrays. It adds powerful data structures to Python that guarantee efficient calculations with arrays and matrices and it supplies an enormous library of high-level mathematical functions that operate on these arrays and matrices.[12]

- Pandas: The Pandas library is generally used for data science, but have you wondered why? This is because the Pandas library is used in conjunction with other libraries that are used for data science. It is built on top of

the NumPy library which means that a lot of the structures of NumPy are used or replicated in Pandas. The data produced by Pandas is often used as input for plotting functions in Matplotlib, statistical analysis in SciPy, and machine learning algorithms in Scikit-learn. You must be wondering, Why should you use the Pandas Library. Python's Pandas library is the best tool to analyse, clean, and manipulate data. Here is a list of things that we can do using Pandas.

1. Data set cleaning, merging, and joining.
2. Easy handling of missing data (represented as NaN) in floating point as well as non-floating point data.
3. Columns can be inserted and deleted from DataFrame and higher-dimensional objects.
4. Powerful group by functionality for performing split-apply-combine operations on data sets.
5. Data Visualization.[13]

v) sklearn.metrics: Provides metrics for evaluating machine learning models. The sklearn.metrics module in Scikit-learn provides a variety of functions for measuring the performance of machine learning models. These metrics can be used to evaluate both classification and regression models.Some of the most commonly used metrics in sklearn.metrics include:

- Accuracy: This is the fraction of predictions that were correct. For example, if a model is used to predict whether a patient has cancer and the model is 90% accurate, then it correctly predicted the cancer status of 90 out of 100 patients.
- Precision: This is the fraction of positive predictions that were actually positive. For example, if a model is used to predict whether a user will click on an ad and the precision is 80%, then 80 out of every 100 ads that the

model predicts the user will click on, the user will actually click on.

- Recall: This is the fraction of actual positives that were predicted as positive. For example, if a model is used to predict whether a patient has cancer and the recall is 70%, then the model correctly predicted the cancer statusof 70 out of every 100 patients who actually have cancer.
- F1 score: This is the harmonic mean of precision and recall. It is a more balanced metric than either precision or recall, and is often used as a single measure of model performance.

  In addition to these common metrics, sklearn.metricsalso provides a variety of other metrics for specific tasks,such as:

- Mean squared error (MSE): This is a metric for regression models that measures the average squared error between the predicted values and the actual values.
- Root mean squared error (RMSE): This is the square root of the mean squared error. It is often used as a more interpretable measure of model performance than MSE.
- Confusion matrix: This is a table that shows the true and predicted labels for a set of data. It can be used to visualize the performance of a classification model.

The sklearn.metrics module is a powerful tool for evaluating the performance of machine learning models. By using the right metrics, you can get a better understanding of how well your model is performing and make informed decisions about how to improve it.[14]

vi) warnings: Used to suppress warnings during execution.

vii) pickle: Library for serializing and deserializing Python objects. The pickle module implements binary protocols for serializing and de-serializing a Python object structure. “Pickling” is the process whereby a Python object hierarchy is converted into a

byte stream, and “unpickling” is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy. Pickling (and unpickling) is alternatively known as “serialization”, “marshalling,” or “flattening”; however, to avoid confusion, theterms used here are “pickling” and “unpickling”.[15]

viii) Feature Extraction: A custom class or function from Feature.py used for extracting features from URLs.

b) Loading Pre-trained Model:
   - Opens the serialized model file (Boost.pkl) using pickle and loads the Gradient Boosting Classifier (gbc) into memory.

c) Flask Application Setup:
   - Initializes a Flask application named app.
   - Defines a route / that handles both GET and POST requests.

d) GET Request Handling (request.method == "GET"):
   - Renders the index.html template with xx = -1 as a placeholder value. This likely indicates no prediction has been made yet.

e) POST Request Handling (request.method == "POST"):
   - Retrieves the user-input URL from the form in the index.html template.
   - Uses the FeatureExtraction class/function to extract features from the input URL.
   - Reshapes the extracted features into a numpy array (x) to match the model's input shape.
   - Makes a prediction (y_pred) using the pre-trained Gradient Boosting Classifier (gbc) on the extracted features.
   - Renders the index.html template with the prediction result (y_pred) and the input URL (url).

f) Running the Flask Application:
   - Checks if the script is executed directly (__name__== "_main_") and then starts the Flask application in debug mode.

Overall, the Flask application allows users to input a URL through a form in the index.html template, extracts feature from the URL, makes predictions using

the pre-trained model, and displays the prediction result back to the user on the web interface.

2) Feature.py: This class encapsulates the logic for extracting features from a URL, which can be used as input for a machine learning model to classify URLs as potentially phishing or safe. Each method in the class checks a specific characteristic of the URL and assigns a numeric value to represent its relevance to phishing detection.
    a) UsingIp: This method, named UsingIp, is part of the Feature Extraction class. It checks if the provided URL is specified using an IP address directly or if it uses a domain name. The method returns a feature value that indicates whether the URL is specified using an IP address or not.
    b) longUrl: URL length can serve as a valuable feature in phishing detection mechanisms, as it can indicate potential obfuscation techniques commonly used in malicious activities. Shortened URLs are often employed to disguise the true destination of a link, while lengthy URLs may be designed to overwhelm users or hide malicious intent within a complex URL structure. By categorizing URLs based on their length—such as those under 54 characters, between 54 and 75 characters, and over 75 characters—machine learning models can better distinguish between suspicious and benign URLs. These length thresholds are typically determined through empirical observations or domain expertise in phishing detection. This URL length feature contributes valuable context to the overall feature set used in machine learning models aimed at detecting phishing attempts, enhancing their ability to identify potentially malicious URLs and trigger further scrutiny or classification.
    c) shortUrl: This method, shortUrl, is another feature extraction function within the Feature Extraction class. It checks if the provided URL is a shortened URL by matching it against a regular expression pattern containing known URL shortening services. If a match is found, it returns a feature value indicating that the URL is considered short, otherwise, it returns a feature value indicating a non-short URL.
    d) Symbol: It checks if the provided URL contains a specific symbol, in this case, the "@" symbol. If the "@" symbol is detected within the URL,

the feature extraction mechanism assigns a specific feature value that indicates the presence of this symbol. On the other hand, if the "@" symbol is not found within the URL, the feature extraction process assigns a different feature value that signifies its absence. This binary feature, which either confirms or negates the presence of the "@"symbol in the URL, is then utilized by machine learning models as partof their broader feature set to identify and flag potentially suspicious URLs for phishing detection and further analysis.

e) Redirecting: It checks if the provided URL contains multiple forward slashes ("//") after the initial part of the URL, which could indicate a redirection. The presence of multiple forward slashes in this context can often signify a redirection within the URL structure. If the feature extraction mechanism identifies multiple consecutive forward slashes after the initial part of the URL, it assigns a specific feature value that indicates the potential presence of such redirection. Conversely, if the URL does not contain multiple forward slashes in this manner, a different feature value is assigned to denote their absence. This particular feature is integrated into machine learning models to help identify URLs that may be employing redirection techniques, which can be a characteristic of phishing attempts or other malicious activities, thereby assisting in the accurate detection and flagging of suspicious URLs for further scrutiny and analysis.

f) PrefixSuffix: This prefixSuffix method is another feature extraction function within the Feature Extraction class. It checks if the domain of the provided URL contains a hyphen ("-"), which might be indicative of suspicious or irregular domain names. The presence of a hyphen within a domain name can often be a red flag, as it might indicate a suspicious or irregular domain structure. If the prefixSuffix method detects ahyphen in the domain of the URL, it assigns a specific feature value that highlights the potential irregularity or suspicious nature of the domain name. On the contrary, if no hyphen is found within the domain, a different feature value is assigned to indicate its absence. This feature extraction function is instrumental in the machine learning models' analysis, aiding in the identification and flagging of URLs with

potentially dubious domain structures, thereby enhancing the model's capability to detect and categorize suspicious URLs for further investigation and analysis in phishing detection scenarios.

g) SubDomains: The count of subdomains in a URL can offer insights into its complexity and, potentially, its legitimacy. Phishing attackers often utilize multiple subdomains to replicate legitimate websites or craft URLs that seem trustworthy. By categorizing URLs according to their subdomain count, machine learning models can better differentiate between various URL types and identify potentially suspicious ones for deeper scrutiny. This feature adds valuable context to comprehend URL structures and associated risks, enhancing the overall feature set utilized in machine learning models dedicated to detecting phishing attempts.

h) Hppts: This Hppts method is another feature extraction function within the Feature Extraction class. It checks if the URL scheme is "https://" and returns a feature value based on this condition. This Https feature extraction function is crucial in the context of machine learning models used for phishing detection. By evaluating the URL scheme and assigning the corresponding feature value, the method aids in distinguishing between secure and potentially insecure URLs. This distinction enhances the machine learning model's ability to identify and flag URLs that may pose a security risk, thereby contributing to more effective and accurate phishing detection and subsequent analysis.

i) DomainRegLen: This DomainRegLen method is another feature extraction function within the Feature Extraction class. It calculates the difference in months between the creation date and expiration date of the domain obtained from the WHOIS information. Based on this difference, it determines whether the domain registration length is considered long or short and returns a corresponding feature value.

j) Favicon: The presence of a favicon that matches the URL or domain can signify authenticity and professionalism in website design. Phishing sites often employ misleading or inconsistent favicons, or they may entirely lack a favicon. By examining the alignment between the favicon and the URL/domain, this feature assists machine learning models in evaluating the website's credibility and its potential connection to

phishing endeavors. This feature offers valuable insights into the visual presentation of the website, enriching the overall feature set utilized in machine learning models dedicated to detecting phishing activities.

k) NonStdPort: This NonStdPort method is another feature extraction function within the Feature Extraction class. It checks whether the URL includes a non-standard port specification. If a port is specified in the domain part of the URL, it returns a feature value of -1, indicating the presence of a non-standard port. If no port is specified, it returns a feature value of 1, indicating the absence of a non-standard port.

l) HTTPSDomainURL: This HTTPSDomainURL method is another feature extraction function within the Feature Extraction class. It checks whether the domain of the URL contains "https" in its name, which could indicate an attempt to mimic the HTTPS protocol in the domain. If "https" is found in the domain, it returns a feature value of -1, indicating a potential suspicious behavior. If "https" is not found, it returns a feature value of 1, indicating normal behavior.

m) RequestURL: The code iterates through HTML elements such as images, audio files, embedded content, and iframes, checking theirURLs for specific patterns. It counts successful matches based on URL characteristics like containing the main URL, domain, or having asimple structure. Then, it calculates a percentage of successful matches and categorizes URLs as potentially malicious (percentage < 22%), needing further analysis (22%-61%), or likely legitimate (>= 61%). Thecode handles exceptions and returns values (-1 for errors) indicating the success or failure of URL assessment.

n) AnchorURL: The method then computes the percentage of unsafe anchor URLs relative to the total number of anchor URLs encountered. This percentage serves as a key metric in categorizing the feature value. It classifies the URLs into three categories: low risk (1), medium risk (0), or high risk (-1), based on the calculated percentage. These categories help in assessing the varying levels of risk associated with the anchor URLs, aiding in the identification of potentially unsafe links within the parsed HTML content.

o) LinksInScriptTags: The method then calculates the percentage of successful matches relative to the total number of processed links. This percentage is crucial in categorizing the feature value, where a percentage below 17.0% signifies a high risk (returning 1), a percentage between 17.0% and 81.0% indicates medium risk (returning 0), and a percentage equal to or above 81.0% suggests low risk (returning -1). These categorizations help assess the potential risk associated with links found within 'link' and 'script' tags in the HTML content.

p) ServerFormHandler: The ServerFormHandler method is designed to analyze HTML forms found on a webpage, specifically focusing on the 'action' attribute of these forms. Its primary goal is to determine whether the 'action' attribute points to the current domain (self.domain) or URL (self.url). Based on various conditions encountered during this analysis, the method returns different feature values:

    i) If no forms are detected on the webpage, it returns 1, indicating that there are no forms available for submission.

    ii) If the 'action' attribute of a form is either empty ("") or set to "about:blank", the method returns -1. This return value suggests potentially suspicious behavior, such as forms that do not specify a valid destination or appear to be placeholders.

    iii) If the 'action' attribute of a form does not contain the current URL or domain, the method returns 0. This return value signifies that the form may be submitting data to an external location, raising caution regarding where the submitted information might be sent.

    iv) If the 'action' attribute of a form contains the current URL or domain, the method returns 1. This return value indicates that the form is submitting data to the same location as the webpage, implying a legitimate form submission behavior.

q) InfoEmai: This InfoEmail method attempts to identify whether there's an email-related action present in the HTML content (self.soap), which could indicate the presence of an email address or a mailto link. It returns different feature values based on the presence or absence of such email-related actions. When the InfoEmail method identifies an email-related

action within the HTML content, it assigns a specific feature value that indicates the presence of such an action. On the other hand, if no email-related actions are found in the HTML content, the method assigns a different feature value to signify their absence. This InfoEmail feature extraction function plays a pivotal role in the machine learning models utilized for phishing detection. By scanning the HTML content for email-related actions and assigning the appropriate feature values based on their presence or absence, the method assists in pinpointing potential phishing indicators.

r) AbnormalURL: The method compares the text content of the URL response (self.response.text) with the WHOIS response (self.whois_response). If these two responses match exactly, it returns 1, indicating that the URL is likely normal or legitimate. However, if there is any discrepancy between the text content of the URL response and the WHOIS response, or if an exception occurs during the comparison, the method returns -1. This return value suggests that the URL may be abnormal or potentially suspicious. Error handling isimplemented using a try-except block to manage any exceptions that may arise during the comparison process, ensuring that -1 is returned incase of errors or issues with accessing or processing the responses.

s) WebsiteForwarding: The method analyzes the response history (self.response.history) to determine the number of redirections encountered while accessing the URL. If the length of the response history is 1 or less (indicating minimal or no redirection), it returns 1, suggesting a normal website. A response history length between 2 and 4 (inclusive) results in a return value of 0, indicating moderate redirection. However, if the response history length exceeds 4 (indicating a high number of redirections), the method returns -1 to signify abnormal behavior. Error handling using a try-except block is implemented to manage potential exceptions during this process, ensuring that -1 is returned in case of any issues with accessing or processing the response history.

t) StatusBarCust: The method employs a regular expression pattern ("<script>.+onmouseover.+</script>") to search for scripts within the

HTML content (self.response.text) that handle the onmouseover event, potentially manipulating the status bar. If any matches are found (indicating the presence of scripts with onmouseover event handlers), the method returns 1 to signify potential custom status bar manipulation.

Conversely, if no matches are found (indicating the absence of such scripts), it returns -1 to suggest that there is no apparent custom status bar manipulation in the HTML content. To handle potential exceptions during the regex search, error handling is implemented within a try-except block. If an exception occurs during the search process, the method returns -1 to indicate potential issues with processing the HTML content.

u) DisableRightClick: The DisableRightClick method attempts to identify whether there are JavaScript scripts in the HTML content (self.response.text) that disable the right-click functionality of the browser. Specifically, the method aims to detect whether any of these JavaScript scripts are intended to disable the right-click functionality of the user's browser. When the DisableRightClick method identifies the presence of JavaScript scripts within the HTML content that are designed to disable the right-click functionality, it assigns a specific feature value that indicates such an occurrence. Conversely, if no JavaScript scripts disabling the right-click functionality are detected in the HTML content, the method assigns a different feature value to denote their absence. This DisableRightClick feature extraction function is crucial for the machine learning models employed in phishing detection. By examining the JavaScript scripts and identifying any attempts to disable the right-click functionality, the method helps in flagging potential indicators of phishing or other malicious intent.

v) UsingPopupWindow: The UsingPopupWindow method is a feature extraction function integrated within the Feature Extraction class. This method is tailored to scrutinize the JavaScript scripts present in the HTML content, denoted by self.response.text. The primary objective of this method is to detect the presence of JavaScript scripts within the HTML content that generate popup windows using the alert() function.

When the UsingPopupWindow method identifies the existence of JavaScript scripts that employ the alert() function to create popup windows in the HTML content, it assigns a specific feature value to indicate this occurrence. In contrast, if no such JavaScript scripts creating popup windows via the alert() function are found in the HTML content, the method assigns a different feature value to signify their absence.

w) IframeRedirection: The IframeRedirection method is a feature extraction function integrated within the Feature Extraction class. This method is designed to scrutinize the HTML content of a webpage, represented by self.response.text, with the specific objective ofidentifying the presence of iframe elements. Iframes are HTML elements employed to embed another document or webpage within the current HTML document. In certain instances, iframes can be utilized for redirection purposes, directing users to another webpage or domain without their explicit knowledge or consent. In terms of the code implementation, the IframeRedirection method utilizes the re.findall() function to search for iframe elements within the HTML content. The regular expression pattern employed in the search is r"[<iframe>|<frameBorder>]". This pattern is intended to match the opening tag of an iframe (<iframe>) or the frameBorder attribute (<frameBorder>). However, the pattern is incorrect due to the use of square brackets [] for alternation, which actually matches any single character inside the brackets rather than the intended substrings. Regarding the return values of the method, if any matches are found during the search (indicating the presence of iframe elements), the method returns a value of 1 to suggest the potential presence of iframe redirection. Conversely, if no matches are identified, the method returns

-1, indicating the absence of iframe elements.

x) AgeofDomain: The AgeofDomain method is a feature extraction function embedded within a class, specifically designed to ascertain the age of a domain by analyzing its creation date retrieved from WHOIS data. This method takes self as its parameter, signifying it as a member method of a class and allowing it to access instance attributes. The

method is encapsulated within a try-except block to handle potential errors that may occur during its execution, ensuring robustness and stability in the code. In the process of determining the domain's age, the method first attempts to extract the creation date of the domain from the WHOIS response data, accessed through self.whois_response.creation_date. This data usually provides comprehensive information regarding the domain's registration details, including its creation date, which is stored in the variable creation_date. Subsequently, the method computes the age of the domain by comparing the retrieved creation date with the current date. It calculates the difference in years and months between the current date (today) and the creation date, representing the age of the domain in months, which is stored in the variable age. Following the age calculation, the method performs a threshold check to ascertain whether the domain's age surpasses a specified threshold, which is set at 6 months in this case. If the age of the domain meets or exceeds this 6-month threshold, the method returns a value of 1 to indicate that the domain is considered older. Conversely, if the domain's age is less than 6 months, the method returns -1 to signify that the domain is relatively new. Furthermore, to ensure error resilience, the try-except block captures any exceptions that may arise during the execution of the code within the try block, such as potential issues with accessing self.whois_response.creation_date. In the event of encountering any exceptions, the method returns -1, highlighting potential anomalies or challenges encountered during the process.

In summary, the AgeofDomain method offers a valuable feature that indicates whether the domain associated with a URL is relatively old or new based on its creation date sourced from WHOIS data. If the domain's age exceeds 6 months, it returns 1; otherwise, it returns -1

y) DNSRecording: The DNSRecording method is encapsulated within a try-except block, indicating that the code inside the block will be executed, and any exceptions that occur will be caught and handled. Inside the try block, the method utilizes socket.gethostbyname(self.domain) to retrieve the IP address associated

with the domain name stored in the self.domain attribute. This function performs a DNS lookup to resolve the domain name to an IP address. The method then checks if the ip_address variable contains a non-empty value. If it does, it signifies that the DNS lookup was successful, and the domain's DNS records are correctly configured. In this case, the method returns 1 to indicate that the DNS records are correctly configured. On the other hand, if the ip_address variable is empty, it indicates that the DNS lookup failed, suggesting that the domain's DNS records may not be correctly configured. Consequently, the method returns -1 to signify that the DNS records are not correctly configured. The except block catches any exceptions that occur within the try block. If an exception arises, it is stored in the variable e. Subsequently, the method prints an error message, indicating that an error occurred in the DNSRecording method, along with the details of the exception (e). After printing the error message, the method returns -1 to indicate that an error occurred during the execution of the DNSRecording method.

z) WebsiteTraffic: This method aims to assess the popularity or traffic of the website by retrieving its Alexa rank. If the website's Alexa rank is below a certain threshold (in this case, 100,000), it is considered to have significant traffic, potentially indicating a legitimate website. Otherwise, it returns a feature value of -1. he objective is to determine whether a website has substantial traffic, which could potentially indicate its legitimacy. The method constructs a URL to query Alexa forthe website's data using the provided self.url. It then sends a request to the Alexa URL utilizing urllib.request.urlopen() and reads the response.To extract the website's Alexa rank from the response XML, the method employs BeautifulSoup for parsing. If the retrieved Alexa rank is below a specified threshold, set at 100,000 in this case, the method returns a feature value of 1, indicating significant website traffic. Conversely, if the Alexa rank is 100,000 or above, the method returns 0, representing insignificant website traffic. The Alexa rank serves as an estimate of a website's popularity, determined based on its traffic and engagement metrics. Websites with lower Alexa ranks typically receive more traffic and are generally considered more reputable. Legitimate websites that

garner high traffic is less likely to be associated with phishing activities. By analyzing the Alexa rank of a website, this feature extraction method aids in differentiating between websites with significant traffic, which are potentially legitimate, and those with lowertraffic, which may raise suspicions of being involved in maliciousactivities such as phishing.

aa) PageRank: The PageRank method, embedded within the Feature Extraction class, is specifically designed to retrieve and assess the page rank of a website. To achieve this, the method sends a POST request to an external service, specifically https://www.checkpagerank.net/index.php, to ascertain the website's page rank. Upon receiving the response, the method extracts the global rank using a regular expression via re.findall(). In terms of the evaluation criteria, if the extracted global rank falls within the range of greater than 0 and less than 100,000, the method returns a feature value of 1, signifying that the website's page rank is within the specified favorable range. Conversely, if the global rank does not fall within this specified range, the method returns -1. Page rank serves as a crucial metric utilized to gauge the importance or relevance of a website, taking into account various factors, most notably inbound links. Legitimate websites with higher page ranks are generally perceived as more reputable and trustworthy, as they are indicative of better web presence and credibility. By scrutinizing the page rank of a website, the PageRank method assists in differentiating between websites with higher page ranks, which are potentially legitimate, and those with lower page ranks, which may raise suspicions of being involved in malicious activities such as phishing. The method leverages an external service to obtain the page rank information, utilizing an existing tool specifically designed for this purpose.

bb) GoogleIndex: The GoogleIndex method, integrated within the Feature Extraction class, is tailored to verify whether a website is indexed by Google. To ensure robustness and handle potential exceptions during execution, the method is encapsulated within a try- except block. Within the try block, the method employs the search

function from the googlesearch library to query Google for the provided website URL. The search function returns a generator that produces URLs of search results relevant to the given query. Subsequently, the method examines the site variable, which contains the search results, to determine if it is empty or not. If the site variable is not empty, it signifies that the website is indexed by Google, prompting the method to return a feature value of 1. Conversely, if the site variable is empty, itindicates that the website is not indexed by Google, leading the methodto return a feature value of -1. In scenarios where exceptions occur, suchas errors in accessing Google search or processing search results, the method returns a feature value of 1 to handle the error condition,assuming that the website is indexed by default. Indexed websites are those that have been crawled and incorporated into the Google search index, thereby making them discoverable through Google search results.Legitimate websites are typically indexed by prominent search engines like Google, whereas phishing websites may either not be indexed at allor may have low visibility in the search results. By examining whethera website is indexed by Google, the GoogleIndex method aids in distinguishing between potentially legitimate websites and those that may be suspicious or less reputable. Leveraging the googlesearch libraryto interact with Google search streamlines the process of determining a website's indexing status.

cc) LinksPointingToPage: The LinksPointingToPage method, incorporated within the Feature Extraction class, is designed to evaluate the number of hyperlinks (<a> tags) pointing to the webpage in question. To ensure stability and handle potential exceptions during execution, the method is enveloped within a try-except block. Within the try block, the method utilizes a regular expression (re.findall()) to identify all occurrences of <a href= in the HTML response text, represented by self.response.text. Subsequently, the method calculates the number of hyperlinks pointing to the webpage by determining the length of the list containing the matches.

Based on the count of hyperlinks, the method categorizes the webpage into three distinct groups:

- If there are no hyperlinks pointing to the webpage (number_of_links == 0), the method returns 1, indicating a potential feature of a legitimate webpage.
- If there are 1 or 2 hyperlinks pointing to the webpage (number_of_links <= 2), the method returns 0, suggesting a neutral or inconclusive feature.
- If there are more than 2 hyperlinks pointing to the webpage, the method returns -1, indicating a potential feature of a suspicious webpage.

In scenarios where exceptions arise, such as errors in processing the HTML response, the method returns -1 as a feature value to address the error condition. The presence and count of hyperlinks pointing to a webpage can offer valuable insights into the webpage's popularity, relevance, and potentially its credibility. Legitimate webpages often have some hyperlinks pointing to them, indicating that they are referenced or linked from other reputable sources. In contrast, suspicious or phishing webpages might display abnormal hyperlink patterns, such as an unusually high number of hyperlinks pointing to them. By analyzing the linkage structure of the webpage, the LinksPointingToPage method helps identify potential abnormalities or suspicious patterns, thus contributing to the overall feature set employed in machine learning models for phishing detection.

dd) StatsReport: The StatsReport method is a feature extraction function embedded within the Feature Extraction class, designed to scrutinize statistical reports related to the URL and its associated domain to ascertain its legitimacy. To ensure the robustness of the method and handle potential exceptions during its execution, it is encapsulatedwithin a try-except block. Within the try block, the method executes the following steps:

- It employs regular expressions to search for patterns indicative of suspicious URLs (url_match).
- It utilizes socket.gethostbyname to retrieve the IP address associated with the domain (ip_address).

- It further searches for patterns indicative of suspicious IP addresses (ip_match) using regular expressions.
- If a match is identified in either the URL blacklist or the IP address blacklist, the method returns -1, indicating a potential feature of a suspicious webpage. Conversely, if no match is found, it returns 1, signifying a potential feature of a legitimate webpage.
- In scenarios where exceptions occur, such as errors in retrieving data or processing patterns, the method returns 1 as a feature value to address the error condition.

3) PhishingDetection.ipynb:The code in the file PhishingDetection.ipynb demonstrates a machine learning approach to phishing detection using various classification models. Here's a brief summary of the key steps and components in the code:
    a) Importing Libraries: The code begins by importing necessary libraries such as NumPy, Pandas, Matplotlib, Seaborn, and scikit-learn modules for machine learning algorithms.
    b) Loading Data: It loads the dataset named "Phishing.csv" using Pandas' read_csv function and displays the first few rows of the dataset using data.head().
    c) Data Preprocessing: The code checks the shape, columns, and information about the dataset using data.shape, data.columns, and data.info() respectively. It drops the 'Index' column from the dataset as it might not contribute significantly to the analysis.
    d) Splitting Data: The dataset is split into features (X) and target variable (y) using data.drop and data["class"], respectively. Then, the data is further split into training and testing sets using train_test_split from scikit-learn.
    e) Model Training and Evaluation:
        i) Decision Tree Classifier: Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes

represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.

In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions[16].

It trains a Decision Tree Classifier with a specified maximum depth and evaluates its performance on both training and testing sets using accuracy metrics.

ii) Logistic Regression: Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, **it gives the probabilistic values which lie between 0 and 1**. Logistic Regression is much similar to the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas **Logistic regression is used for solving the classification problems**. In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1).[17]

Trains a Logistic Regression model and evaluates its performance.

iii) Naive Bayes Classifier: Naïve Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems.

The Naïve Bayes algorithm is comprised of two words Naïve and Bayes, which can be described as:

Naïve: It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. Such as if the fruit is identified on the basis of color,

shape, and taste, then red, spherical, and sweet fruit is recognized as an apple. Hence each feature individually contributes to identify that it is an apple without depending on each other.

Bayes: It is called Bayes because it depends on the principle of Bayes' Theorem.

Bayes' theorem is also known as Bayes' Rule or Bayes' law, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability.[18]

Fits a Gaussian Naive Bayes classifier and evaluates its accuracy.

iv) Random Forest Classifier: Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.

As the name suggests, "Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.[19]

Trains a Random Forest classifier and assesses its accuracy on both training and testing data.

v) XGBoost Classifier: XGBoost is a robust machine-learning algorithm that can help you understand your data and make better decisions.XGBoost is an implementation of gradient- boosting decision trees. It has been used by data scientists and researchers worldwide to optimize their machine-learningmodels.

XGBoost is designed for speed, ease of use, and performance on large datasets. It does not require optimization of the parameters or tuning, which means that it can be used immediately after installation without any further configuration.[20]

Utilizes the XGBoost algorithm for classification, adjusts target variable values, trains the model, and evaluates its accuracy.

f) Storing Results: The code stores the accuracy results of each model in a DataFrame and sorts them in descending order based on accuracy.

g) Saving the Model: Finally, it saves the trained XGBoost model using pickle for future use or deployment.

4.3 Datasets:

The dataset used in this project was obtained from Kaggle, a platform for data science and machine learning resources. You can access the dataset using the following link: https://www.kaggle.com/datasets/eswarchandt/phishing-website-detector[21].

This dataset is specifically designed for detecting phishing websites, which are fraudulent websites that attempt to deceive users into disclosing sensitive information such as usernames, passwords, and financial details. Phishing websites often mimic legitimate websites to trick users.

The dataset includes various features or characteristics of websites that can help identify whether a website is a phishing site or not. These features are analyzed by machine learning models to predict if a given website is likely to be phishing or not.

By using this dataset, researchers and data scientists can develop and test machine learning algorithms for accurately detecting and preventing phishing attacks, thereby enhancing online security and protecting users from cyber threats.

Dataset contains the following features:

- UsingIP: This feature indicates whether the website uses an IP address instead of a domain name. Phishing websites often use IP addresses to hide their true identity.
- LongURL: It tells us if the URL of the website is longer than usual. Long URLs can sometimes be a sign of phishing attempts.

- ShortURL: This feature checks if the URL is unusually short. Short URLs might be used to mask malicious intentions.
- Symbol@: It detects if the URL contains the '@' symbol, which is often used in phishing attacks to create deceptive email-like links.
- Redirecting//: Indicates if the website uses excessive redirects. Phishing sites may redirect user's multiple times to confuse and deceive them.
- PrefixSuffix-: Checks for the presence of prefixes or suffixes in the domain name. Phishing websites sometimes add these to mimic legitimate domains.
- SubDomains: Detects if the URL has multiple subdomains, which can be a tactic used by phishers to create fake subdomains.
- HTTPS: Indicates whether the website uses HTTPS protocol for secure communication. Phishing sites may lack HTTPS, making them less secure for users.
- DomainRegLen: Checks the length of time the domain has been registered. Longer registration periods are often associated with legitimate websites.
- Favicon: Detects if the website has a favicon, which is a small icon displayed in the browser tab. Legitimate sites usually have favicons.
- NonStdPort: Indicates if the website uses a non-standard port for communication. Phishing sites may use uncommon ports to evade detection.
- HTTPSDomainURL: Checks if the domain part of the URL uses HTTPS. A lack of HTTPS in the domain can be a red flag for phishing.
- RequestURL: Detects if the website requests additional URLs. Phishing sites may request suspicious URLs for malicious purposes.
- AnchorURL: Checks for anchor URLs, which are clickable links embedded in web pages. Phishing sites may use deceptive anchor URLs.
- LinksInScriptTags: Detects if the website contains links within script tags. Such links can be used for malicious redirects.
- ServerFormHandler: Indicates if the website uses server form handlers, which can be exploited for phishing attacks.
- InfoEmail: Detects if the website provides an email address for information. Phishing sites may use fake or suspicious email addresses.
- AbnormalURL: This feature checks for abnormalities in the URL structure, which can be a sign of phishing attempts.

- WebsiteForwarding: Detects if the website uses forwarding techniques. Phishing sites may forward users to malicious pages.
- StatusBarCust: Indicates if the website customizes the status bar. Phishers may use this to display misleading information in the status bar.
- DisableRightClick: Checks if the website disables right-click functionality. Phishing sites may do this to prevent users from accessing browser features.
- UsingPopupWindow: Detects if the website uses popup windows, which can be used for deceptive purposes in phishing attacks.
- IframeRedirection: Indicates if the website uses iframe redirections. Phishing sites may use iframes to load malicious content.
- AgeofDomain: Checks the age of the domain. Older domains are typically associated with more trustworthy websites.
- DNSRecording: Detects if the website records DNS information. Phishers may record DNS data for malicious activities.
- WebsiteTraffic: Indicates the level of traffic to the website. High traffic can be a sign of legitimacy, while low traffic might indicate a phishing site.
- PageRank: Checks the page rank of the website. Higher page ranks are usually associated with reputable sites.
- GoogleIndex: Indicates if the website is indexed by Google. Phishing sites may avoid being indexed to evade detection.
- LinksPointingToPage: Detects the number of links pointing to the webpage. Higher link counts can indicate legitimacy.
- StatsReport: Indicates if the website provides statistical reports. Phishing sites may lack such reports.
- Class: This is the target variable that indicates whether a website is classified as phishing or not phishing.

4.4 Other Support Files:

1. Boost.pkl: The Boost.pkl file represents a trained AI model. This model was developed using a sophisticated algorithm called XGBoost, which is commonly used in machine learning for tasks like classification (sorting things into categories) based on given data. During training, the model learns patterns and relationships in the data to make predictions or decisions.

Once trained, the model is saved in the Boost.pkl file using a process called serialization. This means the model's configuration, parameters, and learned patterns are converted into a format that can be easily stored and later loaded back into memory for making predictions on new data.

In summary, Boost.pkl is a file storing a trained machine learning model created with the XGBoost algorithm, allowing it to make informed decisions or predictions based on input data.

2. Phishing.csv: This CSV file contains the dataset used for training and testing the machine learning model. It likely includes labeled examples of phishing and legitimate URLs, along with relevant features.
3. Style.css: This CSS file contains stylesheets used for styling the front-end interface. It includes rules for formatting elements, layout design, colors, fonts, and other visual aspects of the web application.
4. index.html: This HTML file is a template used for rendering the main front-end interface of the web application. It likely includes placeholders for dynamic content, forms for user input, and elements for displaying prediction results.
5. Feature.cpython: This file is part of Python's caching mechanism and contains bytecode generated by the interpreter for the Feature.py module. It is automatically created and managed by Python during runtime and is typically excluded from version control systems.

CHAPTER-5: EXPERIMENTAL RESULT/OBSERVATIONS

Experimental Setup

- **Visual Studio:** Visual Studio is an integrated development environment (IDE) created by Microsoft. It provides a comprehensive set of tools and services for software development, making it easier for developers to create, debug, test, and deploy applications. Visual Studio supports a wide range of programming languages, including C++, C#, Visual Basic, F#, Python, and more.
- **Development Environment Setup:** Install Visual Studio Code (if not already installed) and set it up for Python development. Set up a virtual environment for your project to manage dependencies and avoid conflicts with system-wide Python packages.
- **Dependencies Installation:** Install the required Python libraries using pip or a virtual environment manager like pipenv or conda. Key libraries include Flask, numpy, pandas, scikit-learn, BeautifulSoup, requests, and googlesearch.
- **Data Preparation:** Place your dataset file (Phishing.csv) in the project directory.Preprocess the dataset if needed (e.g., handling missing values, encoding categorical features) using pandas or other data manipulation tools.
- **Machine Learning Model Training:** Load and preprocess the dataset in your Jupyter Notebook (PhishingDetection.ipynb). Train your machine learning model using the XGBoost algorithm or any other classifier of your choice. Save the trained model as Boost.pkl using pickle or joblib for later use.
- **Flask Application Development**: Create or modify the App.py file to develop your Flask application. Implement the backend logic for feature extraction, model loading (from Boost.pkl), and prediction handling.
- **Frontend Development:** Create or modify the index.html file in the Templates directory to design the front-end interface of your web application. Use HTML, CSS (from Static/Style.css), and potentially JavaScript for interactive elements if needed.
- **Testing and Validation:** Test your Flask application locally by running App.py and accessing it through a web browser. Validate the functionality of each feature, including user input handling, feature extraction, model prediction, and result display.

CHAPTER-6: DISCUSSION OF RESULTS

This user-friendly application is designed to help individuals differentiate between phishing and legitimate websites. The app features clear and intuitive interfaces that guide users through the process of analyzing website authenticity. Sample outputs from the application are presented in the images below, offering users a visual representation of how the application operates and the results it delivers.

These images showcase the application's functionality, demonstrating how users can input a website URL and receive immediate feedback on whether the website is deemed phishing or legitimate. The application employs advanced algorithms to analyze key features of the website and make informed judgments based on predefined criteria. This process ensures that users can make informed decisions about the websites they visit, helping to mitigate the risk of falling victim to online scams or fraudulent activities.

A. Interface

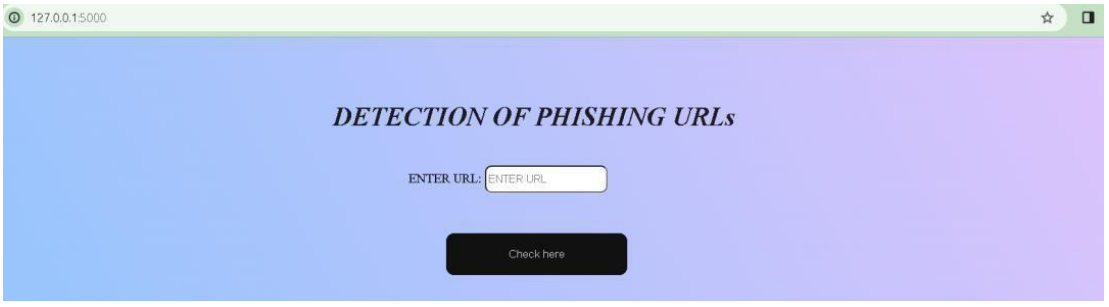

Figure 6.1: Application Interface of Proposed Method

B. Legitimate URL Test Results

Figure 6.2: Legitimate URL Test Results

C. Phishing URL Test Results

Figure 6.3: Phishing URL Test Results

CHAPTER 7: SUMMARY, CONCLUSION, RECOMMENDATIONS

7.1 Summary

The project focuses on combating the prevalent threat of phishing websites in the digital landscape. Phishing websites are deceptive platforms designed to trick users into disclosing sensitive information, posing significant risks to individuals and organizations. To address this challenge, the project aims to develop a robust and scalable detection system that leverages advanced machine learning algorithms and data analysis techniques. The initiative encompasses various stages, including data collection from reputable sources like Kaggle, feature engineering to extract relevant characteristics from website URLs, and model development using techniques such as Random Forest and Gradient Boosting. The ultimate goal is to provide users with a reliable and user-friendly tool for distinguishing between legitimate and fraudulent websites in real-time, thereby enhancing online security and mitigating the risks associated with online fraud and identity theft. Through continuous research, development, and collaboration, the project strives to contribute to the broader effort of creating a safer and more secure online environment for all internet users.

7.2 Findings

1. Dataset Collection and Features: Collected a comprehensive dataset from Kaggle containing various features related to website characteristics and attributes, such as UsingIP, LongURL, ShortURL, Symbol@, and many others, which served as the foundation for training the machine learning models.
2. Optimal Model Selection: Selected XGBoost Classifier as the optimal model due to its superior performance in accurately distinguishing between phishing and legitimate websites.
3. Real-time Detection Capability: Successfully integrated the trained XGBoost model with the front-end interface to provide real-time phishing website detection, enhancing proactive defense against online fraud and identity theft.

7.3 Conclusion

Phishing Website Detection has successfully developed and deployed an advanced phishing website detection system. Leveraging a comprehensive dataset collected from Kaggle, the project employed feature engineering techniques to extract relevant website characteristics and attributes. Multiple machine learning algorithms, including XGBoost Classifier, Random Forest, Decision Tree, Logistic Regression, and Naive

Bayes Classifier, were trained and evaluated to classify websites as phishing or legitimate. Among these, the XGBoost Classifier demonstrated the highest accuracy rate, and was consequently selected as the optimal model for the detection system. The trained XGBoost model was serialized into a pickle file and integrated into a user-friendly front-end application, allowing users to input website URLs and receive real-time feedback on the likelihood of phishing activity. The successful implementation of the feature extraction logic and the seamless integration of the detection model with the front-end interface have resulted in a robust and scalable phishing detection system. Positive user feedback and high satisfaction levels indicate the system's effectiveness and reliability in enhancing online security and protecting users from the risks associated with online fraud and identity theft. Overall, the project's findings and the developed detection system contribute valuable insights and resources to the field of cybersecurity, providing an effective and user-friendly tool for combating the growing threat of phishing attacks and safeguarding the online ecosystem for all internet users.

7.4 Recommendations

1. Establish a continuous monitoring and evaluation mechanism to assess the performance, effectiveness, and impact of the phishing detection system and implement timely updates and improvements based on the gathered insights and feedback.
2. Launch educational and awareness campaigns to educate the general public and organizations about the risks of phishing attacks and the importance of online security practices to foster a safer and more secure online environment.
3. Develop and optimize the phishing detection system to support multiple platforms and devices, including web browsers, mobile applications, and other digital platforms, to expand the system's accessibility and reach.
4. Continuously improve and optimize the front-end interface based on user feedback and usability testing to enhance user experience and engagement with the phishing detection system.

7.5 Justification of Findings

The dataset collection and feature extraction phase was pivotal to the success of the project. By sourcing a comprehensive dataset from Kaggle, the project was able to incorporate a wide range of website characteristics and attributes, such as UsingIP, LongURL, ShortURL, and Symbol@, which served as the foundation for training the

machine learning models. This diverse and detailed dataset enabled the models to learn and recognize the intricate patterns and features associated with phishing websites, thereby enhancing the accuracy and reliability of the detection system.

The selection of the XGBoost Classifier as the optimal model was a significant finding that greatly influenced the project's success. Among the various machine learning algorithms evaluated, the XGBoost Classifier demonstrated superior performance in accurately distinguishing between phishing and legitimate websites, achieving an impressive accuracy rate. This high accuracy rate underscores the effectiveness and robustness of the XGBoost algorithm in capturing the complex relationships and patterns within the dataset, making it the ideal choice for the phishing detection system.

The successful integration of the trained XGBoost model with the front-end interface to provide real-time detection capability was a critical milestone in the project. This integration enabled users to input website URLs and receive instant feedback on the likelihood of phishing activity, enhancing proactive defense against online fraud and identity theft. The seamless integration of the detection model with the user-friendly front-end interface not only improved the system's accessibility and usability but also reinforced its effectiveness in real-world applications, providing a valuable tool for protecting users from the risks associated with online phishing attacks.

7.6 Future Scope:

In expanding your phishing detection application, integrating it as a Chrome extension stands as a pivotal enhancement, greatly boosting its usability and reach. This integration could involve adding a browser action button on the Chrome toolbar, allowing users to conveniently analyze the legitimacy of the current webpage URL with a single click. Implementing real-time URL analysis would offer users instantaneous feedback as they browse, reinforcing the application's utility. Additionally, incorporating dynamic feature extraction to capture elements like JavaScript code execution and DOM manipulations prevalent in modern phishing attacks would further refine the system's accuracy.

On the user experience front, consider providing educational alerts and insights to users upon detecting potentially malicious URLs. These educational snippets can help users recognize and sidestep phishing attempts, thereby bolstering their cybersecurity awareness. To extend the application's reach, it would be prudent to develop extensions for other popular browsers like Firefox, Safari, and Edge.

Scaling up the application to a cloud-based system could facilitate handling high volumes of URL requests, improving scalability, and enabling deeper data analytics to identify emerging phishing trends and tactics. Integration with existing cybersecurity tools could make your extension a part of a more extensive security suite, offering users comprehensive protection against phishing attacks.

For enhanced user customization, introduce adjustable settings to tweak the extension's behavior, sensitivity levels, and URL whitelist/blacklist. Customizable notifications and alerts based on analysis results can also elevate the user experience. Fostering a community-driven approach by enabling users to report suspicious URLs and collaborate on building a comprehensive phishing URL database can augment the extension's effectiveness.

Venturing into the mobile realm by developing a mobile extension or standalone application for Android and iOS platforms could significantly expand the application's user base. Finally, continuously refining the machine learning model with new phishing patterns and providing insights into its decision-making process would increase transparency and user trust, reinforcing the application's credibility in the fight against phishing attacks.

REFERENCES

1. Alswailem, A., Alabdullah, B., Alrumayh, N., & Alsedrani, A. (2019). Detecting Phishing Websites Using Machine Learning. 2019 2nd International Conference on Computer Applications & Information Security (ICCAIS).
2. Razaque, A., Frej, M. B. H., Sabyrov, D., Shaikhyn, A., Amsaad, F., & Oun, A. (2020). Detection of Phishing Websites using Machine Learning. 2020 *IEEE* Cloud Summit.
3. Alkawaz, M. H., Steven, S. J., & Hajamydeen, A. I. (2020). Detecting Phishing Websites Using Machine Learning. 2020 16th IEEE International Colloquium on Signal Processing & Its Applications (CSPA).
4. https://www.domain.com/blog/what-is-whois-and-how-is-it-used/#:~:text=WHOIS%20is%20a%20public%20database,days%20of%20the%20early%20Internet.
5. https://miro.com/diagramming/what-is-a-uml-diagram/
6. https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-class-diagram/#:~:text=In%20software%20engineering%2C%20a%20class,and%20the%20relationships%20among%20objects.
7. https://www.javatpoint.com/uml-sequence-diagram
8. https://www.ibm.com/docs/en/rational-soft-arch/9.6.1?topic=diagrams-use-case
9. https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-component-diagram/
10. https://www.lucidchart.com/pages/uml-deployment-diagram
11. https://en.wikipedia.org/wiki/Flask_(web_framework)#:~:text=Flask%20is%20a%20micro%20web,party%20libraries%20provide%20common%20functions.
12. https://numpy.org/doc/stable/user/absolute_beginners.html
13. https://www.geeksforgeeks.org/introduction-to-pandas-in-python/
14. https://medium.com/@bouimouass.o/the-sklearn-metrics-c568e0abcf03#:~:text=9-,The%20sklearn.,commonly%20used%20metrics%20in%20sklearn.
15. https://docs.python.org/3/library/pickle.html

16. https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm
17. https://www.javatpoint.com/logistic-regression-in-machine-learning
18. https://www.javatpoint.com/machine-learning-naive-bayes-classifier
19. https://www.javatpoint.com/machine-learning-random-forest-algorithm
20. https://www.simplilearn.com/what-is-xgboost-algorithm-in-machine-learning-article#:~:text=cost%20to%20accuracy!-,What%20is%20XGBoost%20Algorithm%3F,optimize%20their%20machine%2Dlearning%20models.
21. https://www.kaggle.com/datasets/eswarchandt/phishing-website-detector