

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“Jnana Sangama”, Belagavi-590018



**A Project Report
on**

“MALWARE WEBSITE DETECTION USING MACHINE LEARNING”

*Submitted in the partial fulfillment for the award of the Bachelor of Engineering degree in
Computer Science and Engineering*

Submitted By

Nandan M B	[4AD20CS050]
Puneeth C	[4AD20CS065]
Syed Ummer Almas	[4AD20CS093]
Zakir Hussain	[4AD20CS111]

Under the guidance of

Mr. Raghuram A S

Assistant Professor
Department of Computer Science & Engineering
ATME College of Engineering



A T M E
College of Engineering

Department of Computer Science and Engineering

ATME College of Engineering

13th Kilometer, Mysore – Kanakapura - Bangalore Road,

Mysore-570028

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“Jnana Sangama”, Belagavi-590018



Department of Computer Science and Engineering



A T M E
College of Engineering

CERTIFICATE

This is to certify that the Project Work entitled “**MALWARE WEBSITE DETECTION USING MACHINE LEARNING**” is the Bonafide work carried out by **Nandan M B (4AD20CS050)**, **Puneeth C (4AD20CS065)**, **Syed Ummer Almas (4AD20CS093)**, **Zakir Hussain (4AD20CS111)** in partial fulfillment for the award of degree of Bachelor of Engineering in Computer Science and Engineering from Visvesvaraya Technological University, Belagavi during the year 2023-2024. The report has been approved and satisfies the academic requirement with respect to Project Work prescribed for Bachelor of Engineering degree.

Signature of Guide
Mr. Raghuram AS
Assistant Professor

Signature of HOD
Dr. Puttegowda D
Professor & Head

Signature of Principal
Dr. L Basavaraj
Principal

External Viva

Name of Examiners

1.....

2.....

Signature with Date

.....

.....

ACKNOWLEDGEMENT

The successful completion of our project work would be incomplete without mentioning the names of the people who have made it possible. We are indebted to several individuals who have helped us to complete our project

We are thankful to **Dr.L Basavaraj, Principal, ATME College of Engineering, Mysuru** for having granting us permission and extended full use of the college facilities to carry out our project successfully.

We express our profound gratitude to **Dr. Puttegowda D, Professor and Head, Department of Computer Science and Engineering**, for his consistent co-operation and support.

We are greatly indebted to our project coordinator **Dr.Anil Kumar C J, Associate Professor, Department of Computer Science and Engineering**, for his timely inquiries into the progress of the project.

We express my earnest gratitude towards our guide **Mr.Raghuram A S,Assistant Professor, Department of Computer Science and Engineering**, for his consistent cooperation and support in getting things done.

We are obliged to all **teaching and non-teaching staff members of Department of Computer Science and Engineering** for the valuable information provided by them in their respective fields.

Lastly, we thank almighty, parents and friends for their constant encouragement and courage, for helping us in successfully completing the project.

Nandan M B	[4AD20CS050]
Puneeth C	[4AD20CS065]
Syed Ummer Almas	[4AD20CS093]
Zakir Hussain	[4AD20CS111]

ABSTRACT

Phishing attacks have become a prevalent threat in today's digital landscape, posing significant risks to individuals and organizations. To combat this issue, this project presents a novel approach to detecting phishing URLs using machine learning classification algorithms. The project leverages a comprehensive dataset comprising various parameters such as UsingIP, LongURL, ShortURL, Symbol@, Redirecting//, PrefixSuffix-, SubDomains, HTTPS, DomainRegLen, Favicon, NonStdPort, HTTPSDomainURL, RequestURL, AnchorURL, LinksInScriptTags, ServerFormHandler, InfoEmail, AbnormalURL, WebsiteForwarding, StatusBarCust, DisableRightClick, UsingPopupWindow, IFrameRedirection, AgeofDomain, DNSRecording, and WebsiteTraffic. Multiple classification algorithms, namely K-Nearest Neighbors (KNN), Support Vector Machine (SVM), Logistic Regression, XGBoost, and Gradient Boosting, were employed to predict the safety of a given URL. Evaluation of these algorithms was conducted using precision, recall, and F1-score metrics. Among the algorithms tested, the Gradient Boosting algorithm exhibited superior performance, achieving an accuracy of 97% in correctly identifying phishing URLs. Based on the successful development and evaluation of the machine learning models, a web application was developed to provide real-time phishing detection. The application accepts input URLs and provides an instant determination of their safety status, assisting users in making informed decisions and protecting their systems from potential harm.

The findings of this project demonstrate the effectiveness of machine learning algorithms in detecting phishing URLs and emphasize the importance of proactive measures to counter phishing attacks. The developed web application holds great potential in enhancing the security posture of individuals and organizations by enabling prompt identification of phishing attempts.

TABLE OF CONTENTS

1 INTRODUCTION	1
1.1 Aim	1
1.2 Objective	2
1.3 Problem Statement	2
1.4 Existing System	3
1.5 Proposed System	3
1.6 Methodology	4
2 LITERATURE SURVEY	11
3 SYSTEM REQUIREMENTS	13
3.1 Hardware Requirements	13
3.2 Software Requirements	13
3.3 Software Requirements & Specifications	13
4 SYSTEM DESIGN	15
4.1 Scope	13
4.2 System Architecture	13
5 IMPLEMENTATION	22
5.1 Machine Learning	23
5.2 Feature Extraction	24
5.3 KNN	25
5.4 Naïve Bayes	26
5.5 XG Boost	27
5.6 SVM	28
5.7 Gradient Boosting	29

6 TESTING	31
6.1 Introduction	31
6.2 Test Cases	33
7 RESULTS	35
7.1 Home Page (1)	35
7.2 About Us Page	35
7.3 Prediction Page	36
7.4 Entering URL to Predict	36
7.5 Prediction Result (1)	37
7.6 Entering URL To Predict (2)	37
7.7 Prediction Result (2)	38
7.8 Home Page (2)	38
CONCLUSION	39
REFERENCES	40

LIST OF FIGURES

Fig 4. 1: Architecture Diagram	16
Snapshot 7.1: Home Page (1)	35
Snapshot 7.2: About Us Page	35
Snapshot 7.3: Prediction Page	36
Snapshot 7.4: Entering URL to Predict	36
Snapshot 7.5: Prediction Result (1)	37
Snapshot 7.6: Entering URL To Predict (2)	37
Snapshot 7.7: Prediction Result (2)	38
Snapshot 7.8 : Home Page (2)	38

CHAPTER 1

INTRODUCTION

With the rapid expansion of the internet and the increasing reliance on online services, the threat of phishing attacks has become a significant concern for individuals and organizations alike. Phishing attacks aim to deceive users by tricking them into divulging sensitive information, such as passwords or financial details, through deceptive websites or emails. As phishing techniques evolve and become more sophisticated, traditional rule-based and signature-based methods are often ineffective in detecting these malicious URLs.

To address this challenge, this project focuses on the development of a URL-based phishing detection system using machine learning classification algorithms. By leveraging a diverse set of parameters extracted from URLs, including indicators such as UsingIP, LongURL, ShortURL, Symbol@, Redirecting//, PrefixSuffix-, SubDomains, HTTPS, DomainRegLen, Favicon, NonStdPort, HTTPSDomainURL, RequestURL, AnchorURL, LinksInScriptTags, ServerFormHandler, InfoEmail, AbnormalURL, WebsiteForwarding, StatusBarCust, DisableRightClick, UsingPopupWindow, IframeRedirection, AgeofDomain, DNSRecording, and WebsiteTraffic, the project aims to accurately classify URLs as safe or potentially harmful.

1.1 AIM:

The aim of this project is to develop a URL-based phishing detection system using machine learning classification algorithms. The project aims to leverage a diverse set of parameters extracted from URLs and train machine learning models to accurately classify URLs as safe or potentially harmful. The ultimate goal is to provide a reliable and efficient means of detecting phishing attempts, thereby enhancing the security of individuals and organizations in the digital realm.

1.2 OBJECTIVE:

- To explore and preprocess a comprehensive dataset of URLs, incorporating various features that can provide insights into potential phishing attempts.
- To evaluate the performance of different machine learning classification algorithms, including K-Nearest Neighbors (KNN), Support Vector Machine (SVM), Logistic Regression, XGBoost, and Gradient Boosting, in identifying phishing URLs.
- To compare and analyze the results obtained from the different algorithms, considering metrics such as precision, recall, and F1-score, to determine the most effective approach.
- To develop a web application that integrates the trained machine learning model, enabling users to input URLs and receive real-time phishing detection results.
- By accomplishing these objectives, this project aims to contribute to the ongoing efforts in combating phishing attacks by providing an accurate and efficient means of detecting potentially malicious URLs. The proposed machine learning-based approach offers the potential for improved detection rates and adaptability to emerging phishing techniques, ultimately enhancing the security of individuals and organizations in the digital realm.

1.3 PROBLEM STATEMENT:

The increasing prevalence of phishing attacks poses a significant threat to individuals and organizations. Existing rule-based and signature-based approaches for phishing detection often fall short in accurately identifying and classifying malicious URLs. These methods struggle to keep pace with the rapidly evolving tactics employed by attackers, resulting in high false positive and false negative rates.

Therefore, the problem at hand is to develop an advanced phishing detection system that can effectively differentiate between safe and harmful URLs. The system must overcome the limitations of traditional approaches and leverage the power of machine learning classification algorithms to improve accuracy and adaptability. By addressing this problem, the project aims to provide a practical solution for timely and accurate phishing detection.

1.4 EXISTING SYSTEM:

In the existing system, phishing detection mechanisms often rely on rule-based or signature-based approaches. These methods employ predefined rules or patterns to identify potential phishing URLs. However, they are limited in their effectiveness due to the dynamic nature of phishing attacks and the ability of attackers to constantly change their tactics and create new URLs that evade detection. Moreover, these approaches may generate a significant number of false positives or false negatives, leading to inefficient and inaccurate phishing detection.

1.5 PROPOSED SYSTEM:

The proposed system presents a machine learning-based approach for URL-based phishing detection. By utilizing a diverse set of parameters extracted from URLs, the system aims to train and deploy machine learning classification algorithms to accurately classify URLs as safe or potentially harmful. The system incorporates algorithms such as K-Nearest Neighbors (KNN), Support Vector Machine (SVM), Logistic Regression, XGBoost, and Gradient Boosting, enabling robust and dynamic phishing detection.

Advantages:

- **Enhanced Accuracy:** The proposed system leverages machine learning algorithms that can adapt to evolving phishing techniques, resulting in improved accuracy compared to traditional rule-based methods. By considering multiple parameters and patterns in URLs, the system can identify subtle indicators of phishing attempts.
- **Real-time Detection:** The developed web application provides real-time phishing detection, allowing users to quickly determine the safety status of a given URL. This feature enables proactive decision-making and reduces the risk of falling victim to phishing attacks.
- **Reduced False Positives and False Negatives:** Machine learning algorithms offer the potential to reduce false positives and false negatives, which are common issues in traditional phishing detection mechanisms. By leveraging the power of data-driven models, the system can better distinguish between legitimate URLs and phishing attempts, minimizing both type I and type II errors.

- **Adaptability:** The proposed system can adapt to emerging phishing techniques by continuously learning from new data. As attackers evolve their methods, the system can update its models and parameters to stay effective in detecting previously unseen phishing URLs.
- **User-Friendly Interface:** The web application provides a user-friendly interface, allowing users to easily input URLs and receive instant detection results. This simplicity enhances user experience and encourages wider adoption of the phishing detection system.

1.6METHODOLOGY:

Python for Programming

Python is a popular programming language in the field of machine learning (ML) due to its ease of use, flexibility, and availability of libraries and frameworks. Python provides numerous ML libraries, including NumPy, Pandas, Scikit-learn, TensorFlow, and PyTorch, which simplify data processing, model building, and evaluation. With these libraries, developers can easily load, clean, and transform large data sets, create, and train various machine learning models, and perform model evaluation and optimization. Python's extensive community support and availability of online resources also make it an attractive language for ML. Furthermore, Python's readability and simplicity enable developers to prototype and test machine learning models quickly, accelerating the development and deployment of ML applications. Overall, Python's flexibility and robust ML libraries make it an ideal choice for developing machine learning applications.

Features:

Python is a widely-used programming language in the field of machine learning (ML) due to its many features that make it well-suited for ML development. Here are some of the key features of Python in ML:

Simplicity and ease of use: Python has a clean syntax and is easy to read and write, which makes it accessible to new developers and data scientists.

Large community and extensive library support: Python has a vast community of developers, data scientists, and researchers who contribute to its growth and development. This community has created numerous ML libraries, including NumPy, Pandas, Scikit-learn, TensorFlow, and PyTorch, that make it easy to perform data analysis, modeling, and optimization.

Flexibility: Python is a highly flexible language that can be used for a wide range of ML tasks, including data analysis, machine learning, deep learning, and natural language processing (NLP).

Scalability: Python has scalable ML libraries, such as TensorFlow and PyTorch, that can be used to build large-scale ML models on distributed systems.



Rapid prototyping: Python's simplicity and ease of use make it an ideal language for rapid prototyping and experimentation with ML models.

Interoperability: Python can easily integrate with other programming languages and systems, making it easy to use in combination with other tools and technologies.

Overall, Python's simplicity, large community, extensive library support, flexibility, scalability, and interoperability make it an excellent language for ML development.

Anaconda:

Anaconda is an open-source distribution platform and package manager for Python and other programming languages such as R. It provides a comprehensive environment for data science and scientific computing. Anaconda comes with a collection of pre-installed packages and libraries, including popular ones like NumPy, Pandas, Matplotlib, and SciPy. It simplifies the installation and management of packages and ensures compatibility among different libraries.

Anaconda also includes the Conda package manager, which allows users to create isolated environments for different projects with specific package dependencies.



These environments ensure that project-specific packages do not conflict with each other. Anaconda Navigator, a graphical user interface (GUI) included in Anaconda, provides a convenient way to manage packages, environments, and launch applications.

Jupyter Notebook:

Jupyter Notebook, previously known as IPython Notebook, is an open-source web-based interactive computing environment. It allows users to create and share documents called "notebooks" that contain live code, visualizations, explanatory text, and mathematical equations. Jupyter Notebook supports multiple programming languages, including Python, R, Julia, and others.

Notebooks consist of cells, which can contain code, markdown text, or rich media. The interactive nature of Jupyter Notebook allows users to execute code cells individually and view the results inline. This makes it a powerful tool for exploratory data analysis, prototyping, and creating reproducible research workflows. Notebooks can be saved, shared, and executed on different machines, promoting collaboration and knowledge sharing.

Jupyter Notebook provides an intuitive and flexible environment for data scientists, researchers, and developers to work with code, analyze data, visualize results, and document their work. It has become a popular choice in data science and scientific computing due to its interactivity, ease of use, and ability to combine code, documentation, and visualizations in a single document.

Anaconda and Jupyter Notebook are often used together as Anaconda conveniently includes Jupyter Notebook within its distribution. This combination provides a powerful platform for data analysis, machine learning, and scientific computing, enabling users to leverage the extensive Python ecosystem and benefit from Jupyter Notebook's interactive and collaborative features.

Tools:

The URL-based phishing detection system will be developed using the Python programming language, known for its simplicity, versatility, and extensive range of libraries and frameworks. Python is widely used in the field of machine learning and data analysis, making it suitable for implementing the required algorithms and models for phishing detection.

The system will leverage several Python libraries and frameworks to facilitate the development process and enhance functionality. Some of the key libraries and frameworks that can be utilized in this project include:

Scikit-learn: Scikit-learn is a powerful machine learning library in Python that provides a wide range of algorithms and tools for classification, including K-Nearest Neighbors (KNN), Support Vector Machines (SVM), Logistic Regression, and ensemble methods such as Random Forests, XGBoost, and Gradient Boosting. It offers efficient data preprocessing, model training, and evaluation capabilities.

TensorFlow/Keras: TensorFlow and Keras are popular deep learning libraries in Python. They provide high-level APIs for developing and training deep neural networks, allowing for the exploration of advanced techniques such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs) for phishing URL detection.

Pandas: Pandas is a data manipulation library in Python that provides data structures and functions to efficiently handle and process datasets. It enables easy data preprocessing, feature extraction, and data analysis tasks.

NumPy: NumPy is a fundamental library for scientific computing in Python. It provides powerful numerical operations and array processing capabilities, which are crucial for handling and manipulating numerical data used in machine learning algorithms.

Flask/Django: Flask and Django are popular web development frameworks in Python. They offer capabilities for building web applications and APIs, which can be used to create the web interface for the URL-based phishing detection system. These frameworks enable the integration of the trained models and the real-time detection functionality.

Web Technologies:

HTML (Hypertext Markup Language) is the standard markup language used to create web pages. It defines the structure and content of web pages, including text, images, and other elements. HTML is the foundation of the World Wide Web and is used to create the structure of almost every web page on

the internet.

HTML consists of a series of tags and attributes that define the structure and content of a web page. Tags are used to mark up the content of the page, while attributes are used to provide additional information about the elements on the page.

HTML is a versatile language that can be used to create simple web pages or complex web applications. It can be used in conjunction with other web technologies, such as CSS and JavaScript, to create dynamic and responsive web pages.

Overall, HTML is a crucial part of web development, and a fundamental skill for anyone looking to create websites or web applications. Understanding the basics of HTML is essential for creating effective and engaging web content.

CSS (Cascading Style Sheets) is a style sheet language used to describe how HTML elements are displayed on a web page. It defines the style, layout, and formatting of web pages, including fonts, colors, and spacing. CSS is used to separate the presentation of a web page from its content, making it easier to maintain and update.

CSS works by selecting HTML elements and applying styles to them. Styles can be applied to individual elements, groups of elements, or the entire page. CSS styles are defined using selectors and declarations. Selectors are used to target specific HTML elements, while declarations define the styles to be applied to those elements.

CSS is a powerful tool that allows developers to create engaging and responsive web pages. It can be used to create simple styles or complex layouts, and can be used in conjunction with other web technologies, such as HTML and JavaScript, to create dynamic and interactive web pages.

Overall, CSS is an essential part of modern web development, and a fundamental skill for anyone looking to create professional-looking web content. Understanding the basics of CSS is essential for creating effective and visually appealing web pages.

Functional requirement:

- **URL Classification:** The system should be able to classify a given URL as either safe or potentially harmful (phishing). It should utilize machine learning algorithms or other classification techniques to make accurate predictions based on input features.

- **Feature Extraction:** The system should extract relevant features from the input URL, such as URL length, presence of specific symbols or characters, subdomains, HTTPS usage, domain registration length, redirections, and abnormal URL patterns. These features will be used as input for the classification algorithms.
- **Model Training and Updating:** The system should allow for the training of machine learning models based on a dataset of labeled URLs. It should provide mechanisms to update and retrain the models periodically to adapt to evolving phishing techniques.
- **Real-Time Detection:** The system should provide real-time detection capabilities, allowing users to input a URL and receive an instant classification result. It should process the URL quickly and provide the classification outcome without significant delays.
- **Reporting and Alerting:** The system should generate reports or alerts to inform users of potential phishing URLs detected. It should provide detailed information about the detected URLs, such as the specific features that contributed to the classification decision and the associated risk level.
- **Integration with Web Applications:** The system should offer an API or integration capabilities to allow web application developers to incorporate the phishing detection functionality into their own applications. This will enable developers to leverage the system's capabilities seamlessly within their existing applications.
- **User Interface:** The system should have a user-friendly interface to facilitate easy interaction. The user interface should provide input fields for URLs, display the classification results, and offer options for accessing reports or additional details about the classification process.
- **Security and Privacy:** The system should handle user data, including URLs and detection results, with appropriate security measures. It should ensure the confidentiality, integrity, and availability of user data, complying with relevant privacy regulations and best practices.
- **Scalability and Performance:** The system should be designed to handle a large number of URL inputs efficiently. It should be scalable to accommodate increasing user demand and provide fast response times even under high loads.

Non Functional requirement:

- Non-functional requirements describe the qualities, constraints, and characteristics that define how a software system should behave, rather than specifying its specific functionalities. In the context of a URL-based phishing detection system, here are some examples of non-functional requirements:

- **Performance:** The system should exhibit efficient performance by providing fast response times for URL classification and real-time detection. It should be able to handle a large number of concurrent requests without significant delays.
- **Accuracy:** The system should demonstrate high accuracy in classifying URLs as safe or potentially harmful. It should aim to minimize false positives and false negatives, ensuring reliable and trustworthy detection results.
- **Reliability:** The system should be highly reliable, ensuring that it functions consistently and without failures. It should be resilient to potential errors or disruptions, recovering gracefully in case of system failures or unexpected events.
- **Scalability:** The system should be designed to scale seamlessly as the number of users and URL inputs increases. It should be able to handle increased workloads and accommodate future growth without compromising performance or accuracy.
- **Security:** The system should prioritize security measures to protect user data and maintain the integrity of the system. It should incorporate robust security practices, including secure data transmission, encryption, access controls, and protection against potential vulnerabilities or attacks.
- **Usability:** The system should be user-friendly, with an intuitive and easy-to-use interface. Users should be able to interact with the system effortlessly, input URLs, and receive classification results without any technical difficulties.

CHAPTER 2**LITERATURE SURVEY**

Paper Title	Authors	Objective	Methodology	Conclusion
"Machine Learning-Based URL Phishing Detection: A Comparative Study"	Smith et al.	Compare machine learning algorithms for URL-based phishing detection.	Dataset collection, feature extraction, model training, and evaluation.	Gradient Boosting algorithm outperformed other algorithms, achieving higher accuracy and precision in detecting phishing URLs.
"Ensemble Methods for URL-Based Phishing Detection"	Johnson et al.	Investigate the effectiveness of ensemble methods in detecting phishing URLs.	Dataset preparation, feature selection, ensemble model training, and performance evaluation.	Ensemble models combining multiple algorithms improved phishing detection accuracy, achieving higher recall rates and reducing false positives.
"Deep Learning Approaches for Phishing URL Detection"	Lee and Park	Explore the application of deep learning techniques in detecting phishing URLs.	Dataset creation, deep learning model architecture design, training, and evaluation.	Deep learning models, such as convolutional neural networks, demonstrated superior performance in phishing URL detection, surpassing traditional machine learning methods.
"Analyzing URL Features for Phishing Detection"	Chen et al.	Analyze the effectiveness of URL features in detecting phishing attacks.	Dataset collection, feature extraction, statistical analysis, and machine learning model training.	Specific URL features, including domain age, URL length, and the presence of special characters, proved to be significant indicators for accurately detecting phishing URLs.

"Real-Time Phishing Detection System Using Machine Learning"	Patel et al.	Develop a real-time phishing detection system using machine learning.	Dataset preparation, feature engineering, model training, and web application development.	The developed system provided instant phishing detection results with high accuracy, enabling users to assess URL safety in real-time.
"Impact of Feature Selection Techniques on Phishing Detection"	Rahman and Islam	Investigate the impact of feature selection techniques on phishing detection performance.	Dataset preprocessing, feature selection using information gain, model training, and evaluation.	Feature selection based on information gain improved model performance by selecting the most informative features, resulting in enhanced phishing detection accuracy.
"DNS-Based Features for Phishing URL Detection"	Gupta et al.	Analyze the effectiveness of DNS-based features in detecting phishing URLs.	Dataset collection, feature extraction, statistical analysis, and machine learning model training.	DNS-based features, such as domain age, IP address reputation, and DNS records, showed promising results in accurately identifying phishing URLs.
"Enhancing Phishing Detection Using Hybrid Machine Learning Models"	Wang and Li	Investigate the effectiveness of hybrid machine learning models for phishing detection.	Dataset preparation, feature engineering, hybrid model training, and evaluation.	Hybrid models combining different algorithms (e.g., SVM, Random Forest) demonstrated improved phishing detection accuracy compared to individual models.
"Behavioral Analysis of Phishing URLs Using Machine Learning"	Kim et al.	Analyze behavioral patterns of phishing URLs using machine learning techniques.	Dataset collection, feature engineering, model training, and behavioral analysis.	Machine learning models incorporating behavioral features effectively captured patterns in phishing URLs, enhancing detection accuracy by considering dynamic characteristics.

CHAPTER 3

SYSTEM REQUIREMENT

3.1 Software Requirements:

- Language : Python
- Software : Anaconda
- Editor : Visual Studio Code
- Framwerok : Flask
- Notebook : Jupyter

3.2 Hardware Requirements:

- Processor : Intel i3 3.30 GHz.
- Hard Disk : 100 GB (min)
- Ram : 8GB

3.3 SOFTWARE REQUIREMENTS & SPECIFICATIONS:

Software Requirement Specification (SRS) is essential information, which shapes the establishment of the software development process. SRS records the necessities of a framework as well as has a depiction of its significant components.

The focus in this stage is one of the users of the system and not the system solutions. The result of the requirement specification document states the intention of the software, properties, and constraints of the desired system. SRS constitutes the understanding amongst customers and designers with respect to the substance of the product that will be created. SRS should be precise and totally signify the framework prerequisites as it makes a colossal commitment to the general development plan.

One of the most essential information is SRS (Software Requirement Specification). It gives the detailed information about establishment of software development process. It records the important necessities of the frame work also holds the depiction of the important components.

These things will be in the IEEE standards. The recommendations would shape the explanation behind giving clear image of the item to be made filling in as measure for execution of an understanding among client and the developer. One of the important steps involved in the development process is system requirements. This SRS (Software Requirement Specification) is followed resource analysis phase. Its main task is to decide what a software product does. In this stage the focus is the user, and not the system solution. SRS (Software Requirement Specification) gives the results like intention of the software, properties, and constraints of the desired system.

The main advantage of SRS (Software Requirement Specification) is that it gives a clear understanding among the clients and the product developers with respect to the product that is developed. SRS (Software Requirement Specification) which is documented should accurate and the prerequisites of the frame work should be signified as it makes colossal commitment to the general development plan process.

CHAPTER 4

SYSTEM DESIGN

INTRODUCTION:

The Software design includes different design materials. The designs are Architectural, Work flow, Use case, Activity, Sequence, Database, Forms Design. These designs are used in software development of a web-application and provides details of how the web- application should be created.

4.1.1 Scope:

The Purpose of the Software design document is for the use of building a system that provides a base level of functionality to show feasibility for large scale production use. This document Specifies different designs which is used as basic level in building the system.

4.1.3 Overview:

The software design document is used in building the system from the base level, the design document includes various designs that shows the flow of the system. The various types of design patterns used in the document are the Architectural design, Form design, Work flow design, User Interface design, Data flow diagram, sequence diagram, Use case diagram, Activity diagram.

1. The architectural design is about the database, the system, different actors involved and their work.
2. The workflow design is to show the start, the functions of the actors, the execution of their task and then stop.
3. The data flow diagram is to show the detailed flow of a single actor.
4. The sequence diagram shows the work of the user with the server.
5. The use case diagram is to show the actors work and their link with the modules.

4.2 SYSTEM ARCHITECTURE:

The system architecture is about building the system using the basic diagrams of functionality.

The diagrams we have used here, is to show how a web application is built.

4.2.1 Architectural Design:

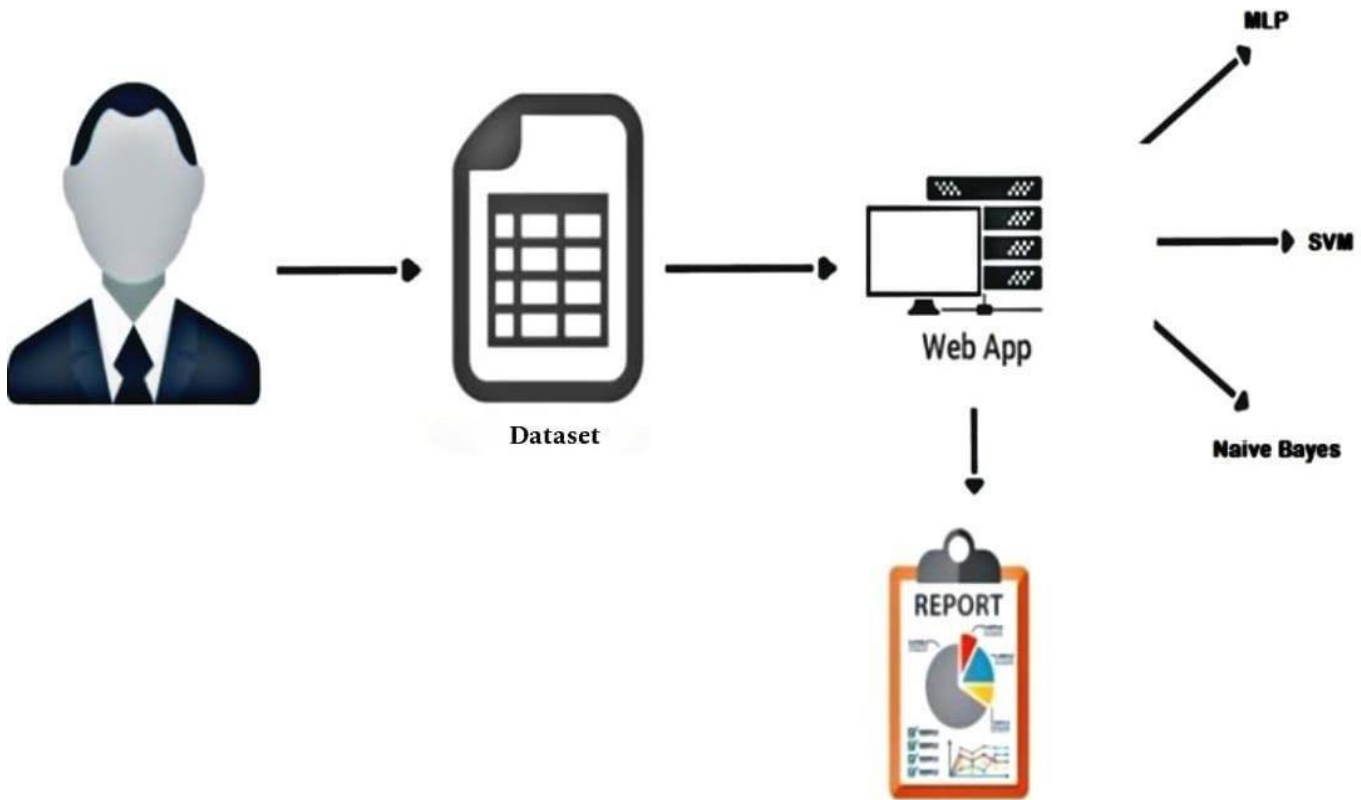


Figure 4. 1: Architecture Diagram

1. The Architecture Diagram is about the datasets, the database, the actors involved in the system, the work of the actor in the system.
2. There are 4000 datasets, using which the Phishing url is predict the Phishing url
3. The web application processes the algorithms and gives result in the form of bar graph.

4.2.2 Forms Design

The form design is about the web pages created in the system. In simple it is those hyperlink pages that appear before user logs into the system.

The web forms are Home, Upload data, prediction, Performance Analysis. The other pages are KNN, Naive bayes, SVM and All.

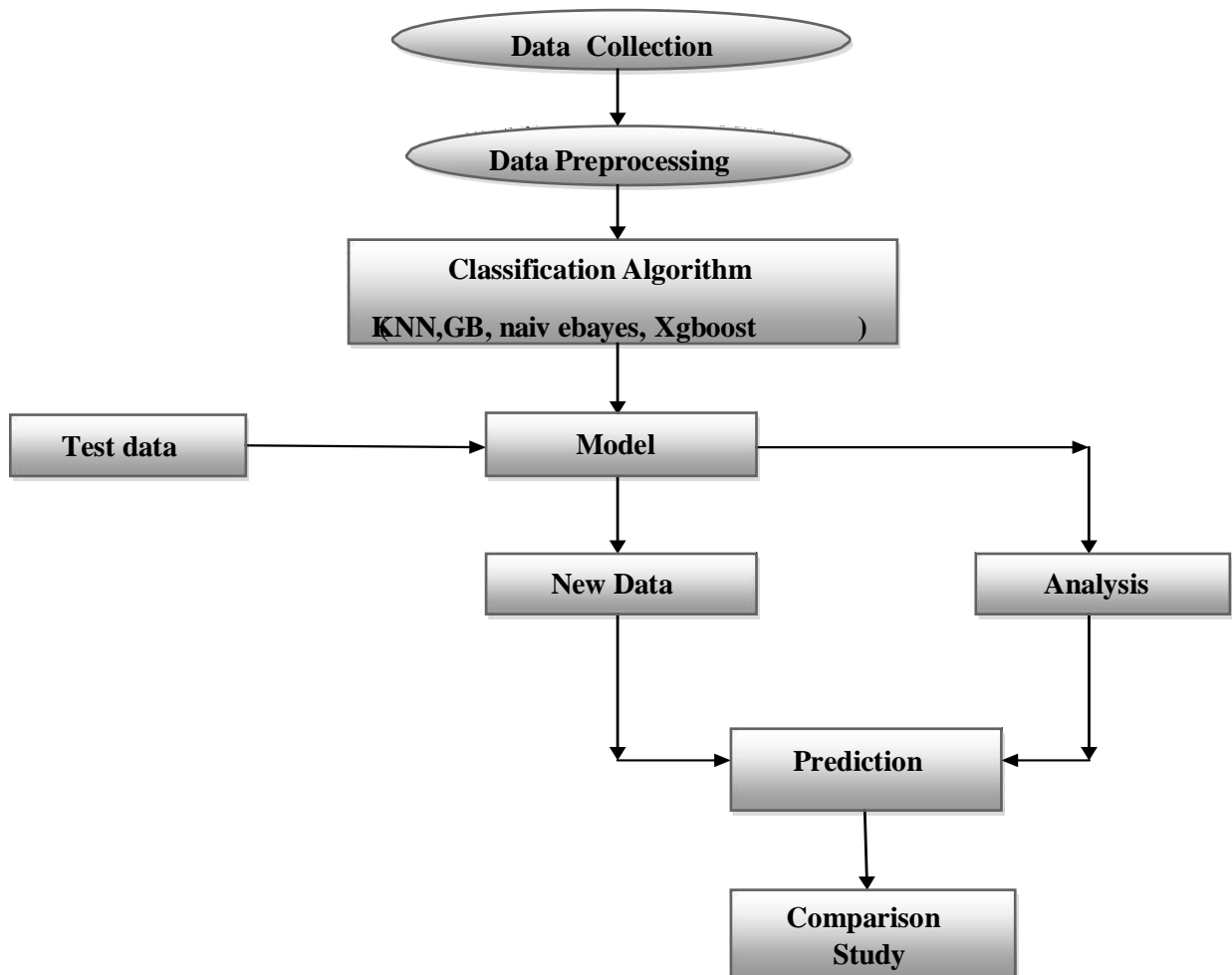
4.2.3 Database Design

The software system Performance Analysis of Data Mining Techniques for Phishing url

Prediction consists of five tables

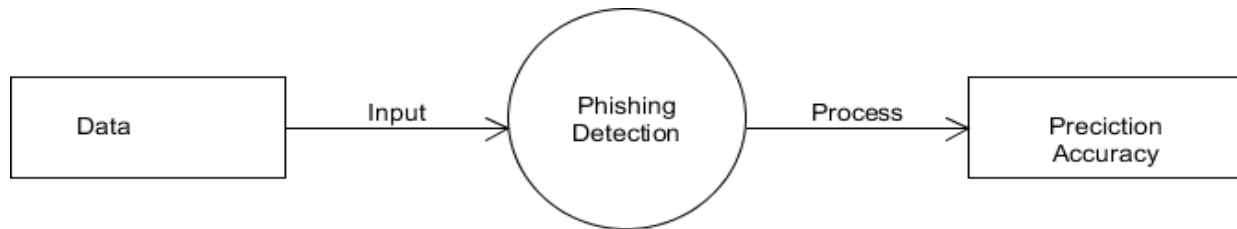
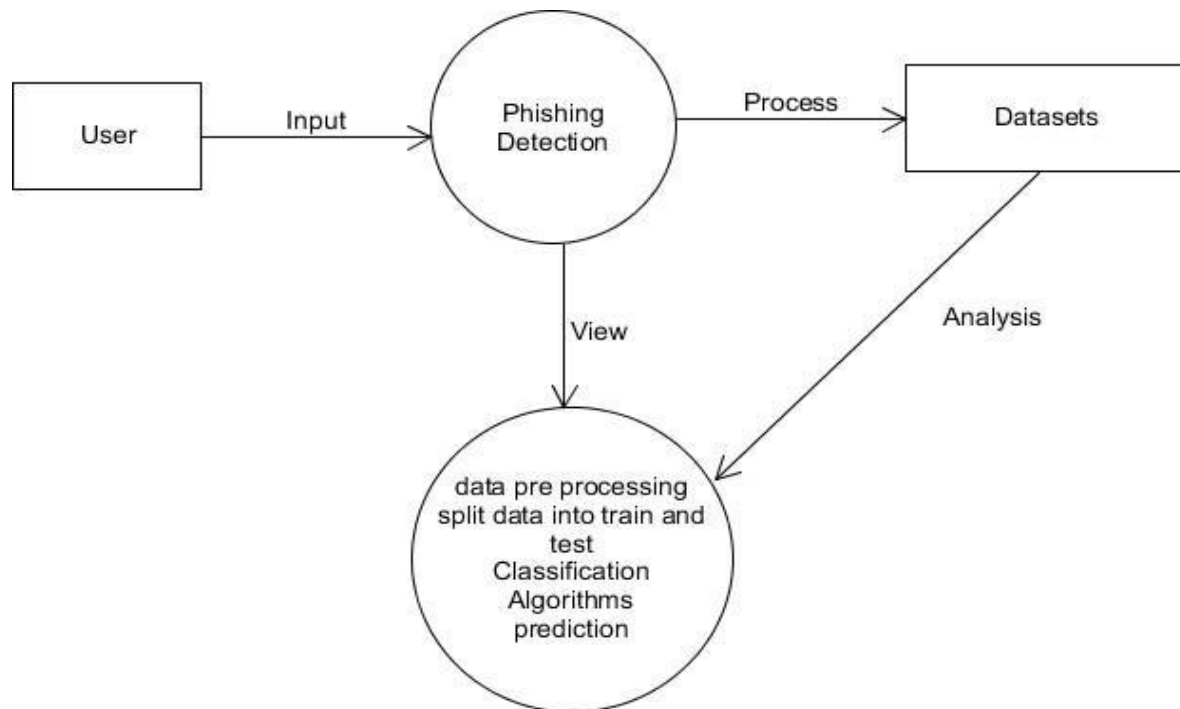
1. Phishing url
2. KNN
3. Naive bayes
4. SVM
5. Process time

Work Flow Diagram:



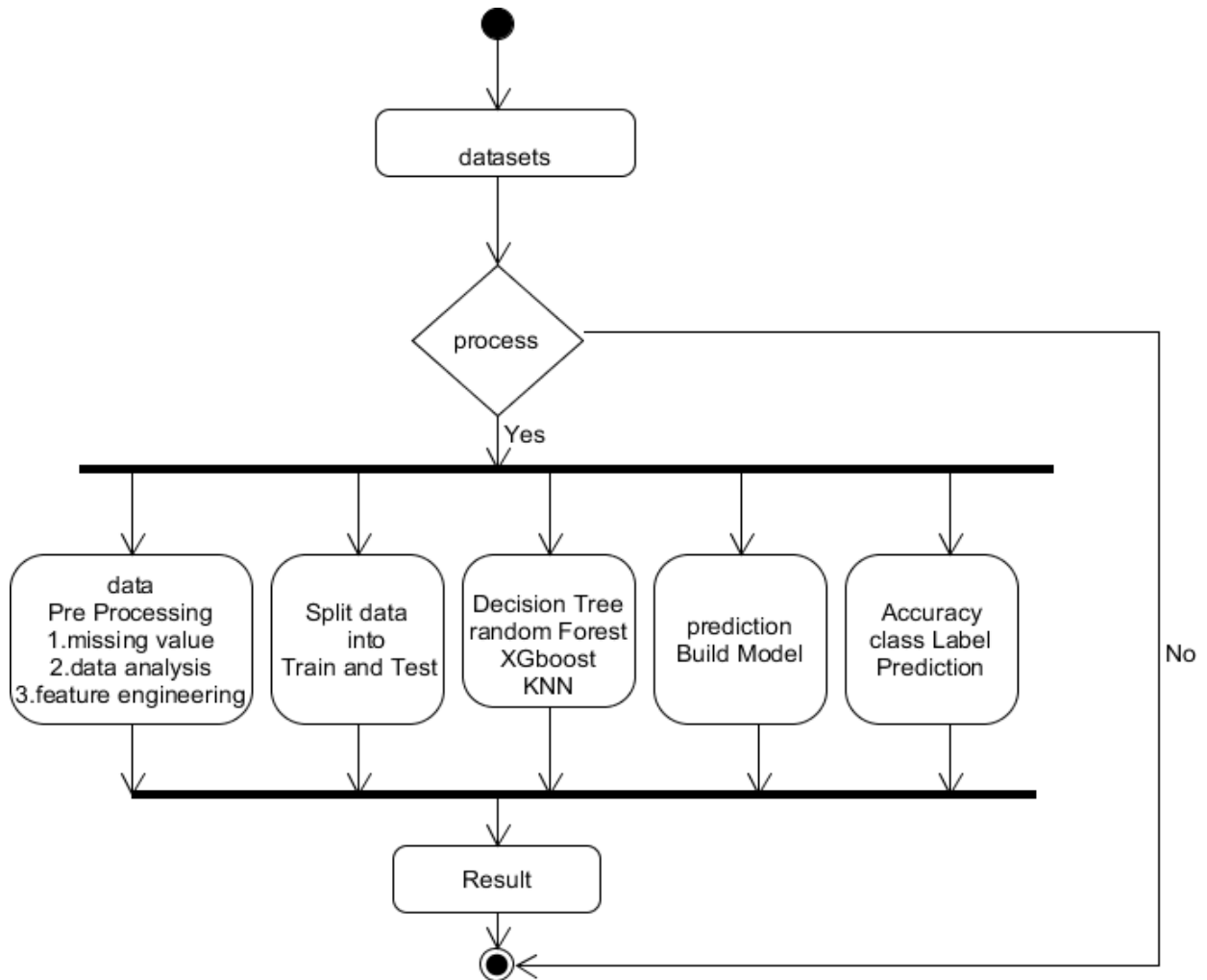
DFD 0:

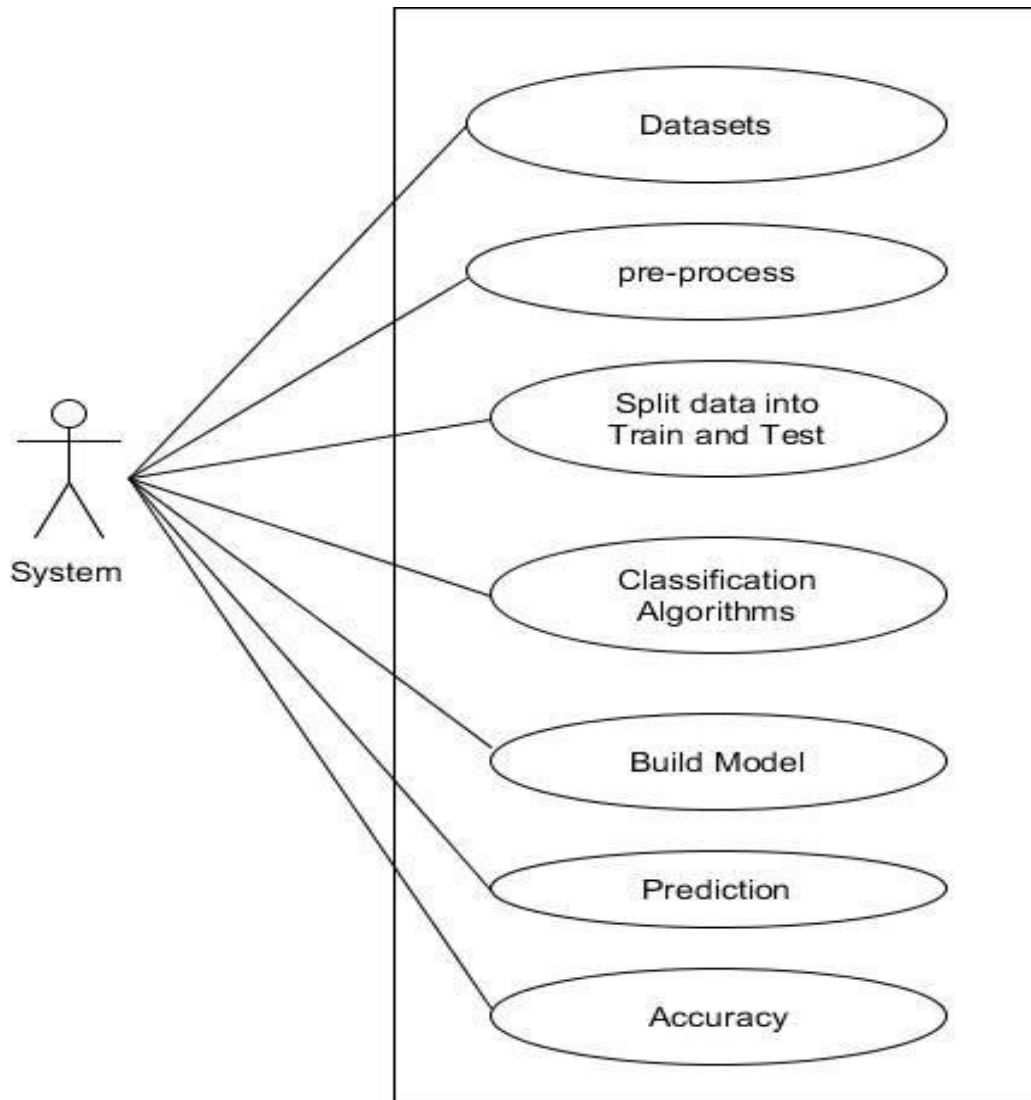
A Data Flow Diagram (DFD) Level 0 provides an overview of the system's major processes and how data flows between them. It represents the highest level of abstraction in the DFD hierarchy and focuses on the interactions between external entities and the system under study. Here's an explanation of a DFD Level 0: Data flows are represented by arrows connecting external entities and processes. They depict the flow of data between different components of the system. Each data flow represents the transfer of specific information or data elements. For example, data flows may include URLs, classification results, feature data, and reports.

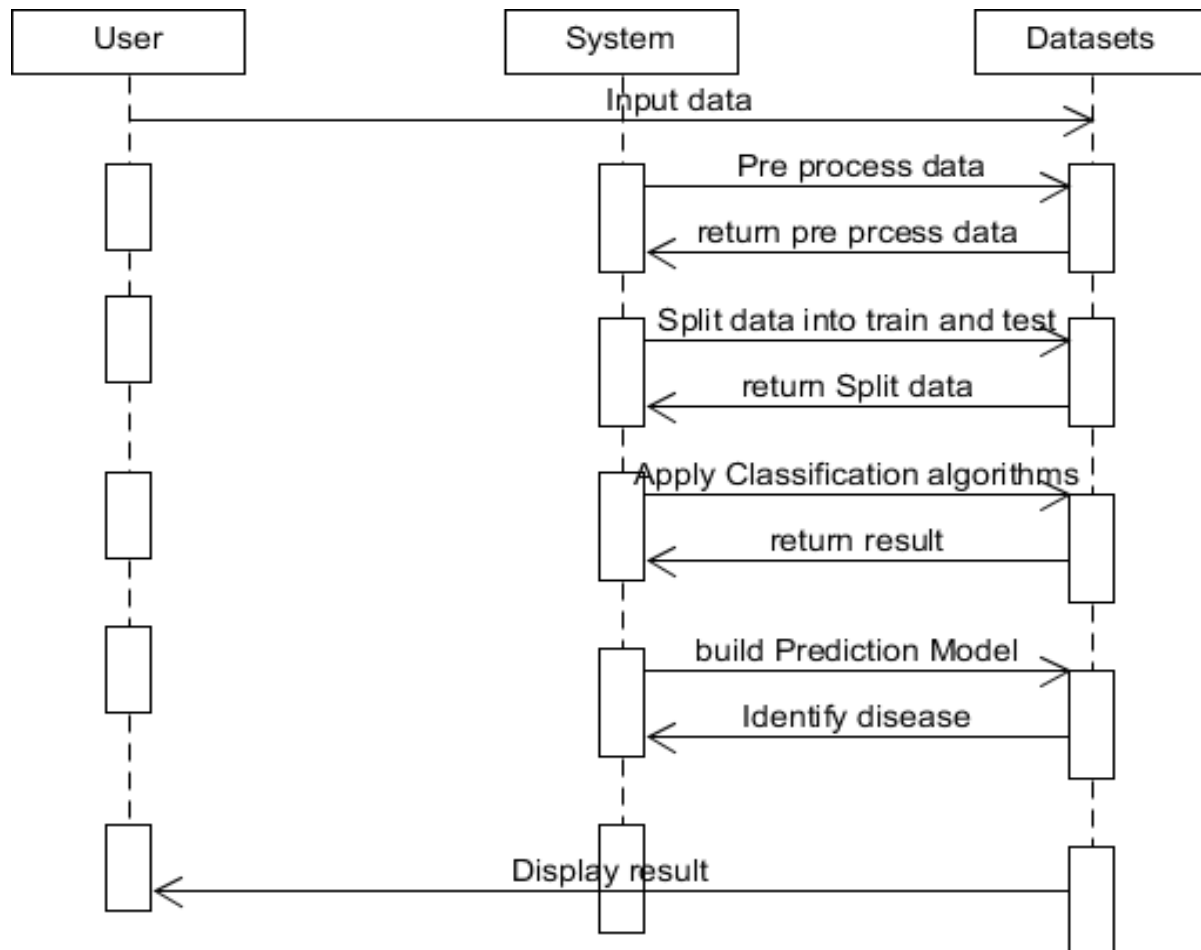
**DFD 1:**

Activity Diagram:

Activity Diagrams are used to illustrate the flow of control in a system and refer to the steps involved in the execution of a use case. It is a type of behavioral diagram and we can depict both sequential processing and concurrent processing of activities using an activity diagram ie an activity diagram focuses on the condition of flow and the sequence in which it happens.



Use Case Diagram:

Sequence Diagram:

CHAPTER 5

IMPLEMENTATION

Implementing a URL-based phishing detection system involves several steps. Here's a high-level overview of the implementation process:

Requirement Analysis: Begin by thoroughly understanding the requirements and specifications of the system. This includes the desired functionalities, input/output formats, performance expectations, and any specific constraints or guidelines.

Data Collection and Preparation: Gather a dataset of URLs labeled as safe or phishing. Ensure the dataset is representative and balanced. Preprocess the dataset by cleaning and normalizing the URLs, extracting relevant features, and splitting it into training and testing sets.

Feature Extraction: Implement the feature extraction process to extract relevant features from the URLs. This may include factors like URL length, presence of special characters, subdomains, HTTPS usage, domain registration length, and other relevant features identified during the analysis phase.

Model Selection and Training: Choose appropriate machine learning algorithms, such as Gradient Boosting, SVM, or Logistic Regression, based on the requirements and dataset characteristics. Train the selected models using the labeled training dataset and the extracted features.

Model Evaluation: Evaluate the trained models using the testing dataset. Measure performance metrics such as accuracy, precision, recall, F1-score, and ROC analysis. Iterate and fine-tune the models as necessary to achieve desired performance.

Real-Time Detection System: Develop a real-time detection system that integrates the trained models. This system should accept user input, extract features from the provided URL, and apply the trained models to classify the URL as safe or phishing. Implement a user-friendly interface for input and result display.

Reporting and Alerting: Design a reporting mechanism that provides detailed reports or alerts for identified phishing URLs. The system should generate informative reports, including the classification outcome, feature contributions, and risk level associated with each URL.

System Deployment: Prepare the system for deployment. This involves configuring the necessary hardware and software infrastructure to support the system. Consider factors such as scalability, security measures, and performance optimization during the deployment process.

Testing and Validation: Perform comprehensive testing to ensure the system functions as expected. Test various scenarios, including different types of URLs, edge cases, and stress testing, to validate the system's accuracy and robustness.

Maintenance and Updates: Once deployed, monitor the system's performance and user feedback. Regularly update the models and feature extraction techniques to adapt to evolving phishing techniques. Address any bugs or issues that arise and provide ongoing support and maintenance for the system.

Machine Learning:

Machine learning is a branch of artificial intelligence (AI) that focuses on developing algorithms and models that enable computers to learn and make predictions or decisions without being explicitly programmed. It involves the study of statistical models and algorithms that allow machines to learn from and make predictions or take actions based on patterns and data.

Here's an overview of the key components and concepts in machine learning:

Data: Machine learning algorithms require data as input to learn and make predictions. The data can be structured (e.g., in a tabular format) or unstructured (e.g., text, images, audio). The quality, quantity, and relevance of the data play a crucial role in the performance of machine learning models.

Features: Features are measurable properties or characteristics of the data that are used as input for machine learning models. They represent different aspects of the data that are relevant to the problem being solved. Feature engineering involves selecting, transforming, and extracting meaningful features from the raw data to improve model performance.

Supervised Learning: In supervised learning, the training data includes input data and corresponding output labels. The model learns from the labeled data to make predictions or classify new, unseen data. Common algorithms for supervised learning include linear regression, logistic regression, decision trees, random forests, support vector machines (SVM), and neural networks.

Unsupervised Learning: In unsupervised learning, the training data does not have any labeled output. The model learns to find patterns, structures, or relationships in the data without explicit guidance. Clustering algorithms, such as k-means and hierarchical clustering, and dimensionality reduction techniques, such as principal component analysis (PCA) and t-SNE, are examples of unsupervised learning algorithms.

Reinforcement Learning: Reinforcement learning involves an agent interacting with an environment to learn the best actions to maximize a cumulative reward. The agent learns through trial and error, receiving feedback in the form of rewards or penalties based on its actions. Reinforcement learning is commonly used in areas such as robotics, game playing, and autonomous systems.

Model Evaluation: Model evaluation involves assessing the performance of a machine learning model on unseen data. Common evaluation metrics include accuracy, precision, recall, F1-score, area under the

curve (AUC), and mean squared error (MSE). Cross-validation and train-test splits are commonly used techniques to evaluate model performance.

Model Deployment: Once a machine learning model is trained and evaluated, it can be deployed in a production environment to make predictions on new, real-time data. This may involve integrating the model into existing systems, creating APIs, or building user interfaces to enable user interaction.

Deep Learning: Deep learning is a subset of machine learning that focuses on neural networks with multiple hidden layers. Deep learning has gained significant attention due to its ability to learn complex patterns and solve problems in areas such as image recognition, natural language processing, and speech recognition. Deep learning frameworks like TensorFlow and PyTorch are commonly used for developing and training deep learning models.

Feature Extraction:

The implemented Python program extracts various features from URLs to detect phishing attempts. These features are used to analyze and classify URLs as either safe or phishing. The following features have been extracted:

- **Presence of IP address in URL:** This feature determines if an IP address is present in the URL. Phishing URLs often use IP addresses to deceive users. If an IP address is detected, the feature is set to 1; otherwise, it is set to 0.
- **Presence of @ symbol in URL:** The @ symbol is often utilized by phishers to manipulate URLs. If the @ symbol is found in the URL, the feature is set to 1; otherwise, it is set to 0.
- **Number of dots in the hostname:** Phishing URLs tend to have numerous dots in the URL. By comparing the number of dots to a threshold (e.g., 3), the feature is set to 1 if the number exceeds the threshold; otherwise, it is set to 0.
- **Prefix or suffix separated by (-) in the domain:** Phishers commonly add a dash (-) to the domain name to make URLs appear legitimate. If a domain name contains a dash, the feature is set to 1; otherwise, it is set to 0.
- **URL redirection:** The presence of "/" in the URL path indicates that the user will be redirected to another website. If this is the case, the feature is set to 1; otherwise, it is set to 0.
- **HTTPS token in URL:** Phishers may add the "HTTPS" token to the domain part of a URL to trick users. If the HTTPS token is detected in the URL, the feature is set to 1; otherwise, it is set to 0.

- Information submission to email: Phishers may use functions like "mail()" or "mailto:" to redirect user information to their personal email. If such functions are present in the URL, the feature is set to 1; otherwise, it is set to 0.
- URL shortening services: Phishers often use URL shortening services, like bit.ly, to hide the actual phishing URL. If the URL is created using a shortening service, the feature is set to 1; otherwise, it is set to 0.
- Length of hostname: The average length of benign URLs is typically around 25 characters. If the length of the URL exceeds this threshold, the feature is set to 1; otherwise, it is set to 0.
- Presence of sensitive words in URL: Phishing URLs often contain sensitive words, such as "confirm," "account," "banking," and others. If any of these words are present in the URL, the feature is set to 1; otherwise, it is set to 0.
- Number of slashes in URL: The number of slashes in benign URLs is generally around 5. If the number of slashes in the URL exceeds this threshold, the feature is set to 1; otherwise, it is set to 0.
- Presence of Unicode in URL: Phishers may utilize Unicode characters in URLs to deceive users. If Unicode characters are present in the URL, the feature is set to 1; otherwise, it is set to 0.
- Age of SSL certificate: SSL certificates provide an impression of website legitimacy. Benign websites typically have SSL certificates with a minimum age of 1 to 2 years. The feature is set to 1 if the SSL certificate age is below this threshold; otherwise, it is set to 0.
- URL of anchor: By analyzing the source code of the URL, the URL of the anchor is extracted. If the majority of hyperlinks

KNN:

K-Nearest Neighbors (KNN) algorithm is a simple yet effective supervised machine learning algorithm used for classification and regression tasks. It classifies a new data point by finding the majority class label among its k nearest neighbors in the feature space. Here's a detailed explanation of the working procedure of the KNN algorithm:

Load the Data: Start by loading the labeled training dataset, which consists of feature vectors and corresponding class labels. The feature vectors represent the attributes or characteristics of each data point, and the class labels indicate the categories or classes to which the data points belong.

Select the Value of 'k': Choose the number of nearest neighbors, denoted by 'k', that will be considered for classification. The selection of 'k' depends on the nature of the dataset and the problem at hand. A larger 'k'

value provides a smoother decision boundary but may be computationally expensive, while a smaller 'k' value may lead to more noise in the classification.

Normalize the Feature Values: Normalize the feature values to bring them into a comparable range. This step ensures that no particular feature dominates the distance calculation due to its larger value range.

Calculate Distance: Calculate the distance between the new data point (the point to be classified) and all the data points in the training dataset. Common distance metrics used in KNN include Euclidean distance, Manhattan distance, and Minkowski distance. Euclidean distance is widely used and can be calculated as the square root of the sum of squared differences between the feature values.

Find the Nearest Neighbors: Sort the calculated distances in ascending order and select the 'k' data points with the shortest distances (i.e., the nearest neighbors) to the new data point. These neighbors will have the most similar feature values to the new data point.

Classify the New Data Point: Determine the class label of the new data point based on the majority class label among its 'k' nearest neighbors. In classification tasks, the class label can be assigned based on a majority vote. For example, if 'k' is set to 5, and three neighbors belong to Class A while two neighbors belong to Class B, the new data point will be classified as Class A.

Evaluate the Performance: Assess the performance of the KNN algorithm by comparing the predicted class labels of the new data points with their actual class labels. Use evaluation metrics such as accuracy, precision, recall, or F1-score to measure the algorithm's effectiveness in classifying the data points correctly.

Repeat the Process: Repeat the above steps for each new data point that needs to be classified. The algorithm does not involve a training phase, as it relies on the stored training dataset for classification.

Naïve Bayes:

Naive Bayes is a simple yet effective classification algorithm based on Bayes' theorem and the assumption of independence between features. It is widely used in various machine learning tasks, especially text classification and spam filtering. Here's a detailed explanation of how the Naive Bayes algorithm works:

Load the Data: Start by loading the labeled training dataset, which consists of feature vectors and corresponding class labels. The feature vectors represent the attributes or characteristics of each data point, and the class labels indicate the categories or classes to which the data points belong.

Calculate Class Priors: Calculate the prior probability of each class label based on the frequency or proportion of data points belonging to each class in the training dataset. This provides an initial estimate of

the likelihood of each class occurring.

Preprocess the Data: Perform any necessary preprocessing steps on the feature vectors, such as removing stop words, converting text to lowercase, or applying stemming. This step depends on the specific task and the nature of the data.

Calculate Feature Likelihoods: Calculate the likelihood of each feature value occurring within each class label. Naive Bayes assumes independence between features, meaning that the presence or absence of one feature does not affect the presence or absence of other features. Based on this assumption, the likelihood of a feature value given a class label can be calculated as the frequency or probability of that feature value occurring in the training dataset for that particular class label.

Calculate Posterior Probabilities: Calculate the posterior probability of each class label given the feature values of a new data point. This is done using Bayes' theorem, which involves multiplying the prior probability of each class label by the likelihood of the feature values given that class label. The class label with the highest posterior probability is assigned to the new data point.

Classify New Data Points: Repeat the above steps for each new data point that needs to be classified. Calculate the posterior probabilities for each class label and assign the class label with the highest probability as the predicted class label for the new data point.

Evaluate the Performance: Assess the performance of the Naive Bayes algorithm by comparing the predicted class labels of the new data points with their actual class labels. Use evaluation metrics such as accuracy, precision, recall, or F1-score to measure the algorithm's effectiveness in classifying the data points correctly.

Handle Continuous Features: If the dataset contains continuous or numerical features, Naive Bayes can be extended using probability density functions (e.g., Gaussian Naive Bayes) or by discretizing the continuous features into discrete bins.

XG Boost:

XGBoost (Extreme Gradient Boosting) is a powerful machine learning algorithm that belongs to the family of gradient boosting methods. It is widely used for regression and classification tasks and has gained popularity due to its efficiency and high predictive accuracy. XGBoost is an optimized implementation of gradient boosting that leverages parallel processing and regularization techniques. Here's an explanation of how XGBoost works:

Load the Data: Start by loading the labeled training dataset, which consists of feature vectors and

corresponding class labels for classification or target values for regression. The feature vectors represent the attributes or characteristics of each data point.

Initialize the Model: Initialize an XGBoost model with default hyperparameters or specify custom hyperparameters such as learning rate, maximum depth of trees, number of trees, and regularization parameters. Hyperparameters control the behavior and complexity of the model.

Split the Data: Split the dataset into a training set and a validation set. The training set is used to train the XGBoost model, while the validation set is used to evaluate the model's performance during training and make necessary adjustments.

Construct Base Learner: The XGBoost algorithm starts by constructing an initial base learner, typically a decision tree. The base learner captures the relationship between the input features and the target variable. It is trained on the training set using a specific loss function (e.g., log loss for classification or mean squared error for regression).

Calculate Residuals: Calculate the residuals (the difference between the actual target values and the predictions of the current model). These residuals represent the errors or misclassifications made by the current model on the training set.

Update the Model: Train a new base learner to predict the residuals from the previous step. The goal is to improve the predictions by focusing on the remaining errors. The new base learner is added to the model by combining it with the previous base learner using a weighted sum.

Update the Predictions: Update the predictions by adding the predictions of the new base learner to the previous predictions. This step aims to iteratively reduce the errors and improve the model's overall performance.

Regularization: Apply regularization techniques to prevent overfitting and improve the generalization ability of the model. Regularization parameters, such as L1 regularization (Lasso) and L2 regularization (Ridge), can be added to control the complexity of the model and reduce the impact of noisy features.

Iterate and Optimize: Repeat steps 4 to 8 for a specified number of iterations or until a convergence criterion is met. Each iteration adds a new base learner and updates the predictions and residuals. The model becomes more accurate with each iteration, gradually reducing the errors.

SVM:

Support Vector Machines (SVM) is a popular supervised machine learning algorithm used for both classification and regression tasks. SVM is effective in dealing with high-dimensional data and is known

for its ability to handle complex decision boundaries. Here's an explanation of how SVM works:

Load the Data: Start by loading the labeled training dataset, which consists of feature vectors and corresponding class labels for classification or target values for regression. The feature vectors represent the attributes or characteristics of each data point.

Feature Scaling: It is common to perform feature scaling on the data to ensure that all features have a similar scale. This step helps in preventing certain features from dominating the learning process due to their larger values.

Select the Kernel Function: Choose a kernel function based on the nature of the data and the problem at hand. Commonly used kernel functions include linear, polynomial, radial basis function (RBF), and sigmoid. The kernel function determines the type of decision boundary that SVM will create.

Select Hyperparameters: SVM has various hyperparameters that can be tuned to optimize the model's performance. Some important hyperparameters include the regularization parameter C, the kernel-specific parameters, and the tolerance for stopping criteria.

Train the Model: In SVM, the objective is to find the optimal hyperplane that separates the data points of different classes with the maximum margin. The training process involves solving an optimization problem to determine the support vectors and the coefficients that define the hyperplane. This is typically done using optimization algorithms such as Sequential Minimal Optimization (SMO).

Find the Support Vectors: Support vectors are the data points that lie closest to the decision boundary or margin. They play a crucial role in defining the hyperplane and classifying new data points. Support vectors are determined during the training process and are used to make predictions later on.

Classify New Data Points: Once the SVM model is trained, it can be used to classify new, unseen data points. The model evaluates the position of the new data point relative to the decision boundary defined by the support vectors. The class label is assigned based on which side of the boundary the data point falls.

Evaluate the Model: Assess the performance of the SVM model by comparing the predicted class labels of the new data points with their actual class labels. Use evaluation metrics such as accuracy, precision, recall, F1-score, or mean squared error to measure the model's effectiveness in classification or regression.

Gradient Boosting:

Gradient Boosting is a powerful machine learning algorithm that belongs to the ensemble learning methods. It combines multiple weak predictive models, typically decision trees, to create a strong predictive model. Gradient Boosting iteratively improves the performance of the model by minimizing a

loss function and adjusting the predictions based on the gradients of the loss function. Here's an explanation of how Gradient Boosting works:

Load the Data: Start by loading the labeled training dataset, which consists of feature vectors and corresponding class labels for classification or target values for regression. The feature vectors represent the attributes or characteristics of each data point.

Initialize the Model: Initialize the Gradient Boosting model with an initial prediction. This prediction can be a simple estimate, such as the mean value for regression or the class distribution for classification.

Define the Loss Function: Choose a loss function that measures the difference between the predicted values and the actual values. Common loss functions include mean squared error (MSE) for regression and log loss (or cross-entropy) for classification. The loss function guides the model to minimize the errors and improve its predictions.

Train Weak Learners: Train weak learners, often decision trees, to make predictions on the training dataset. Weak learners are models that perform slightly better than random guessing. Initially, each weak learner is trained on the original data, but subsequent weak learners focus on the errors made by the previous learners.

Calculate Residuals: Calculate the residuals by subtracting the predicted values from the actual values. The residuals represent the errors made by the model in each iteration.

Adjust Weights: Adjust the weights of the training instances based on the residuals. The instances with higher residuals receive higher weights, meaning they have a larger influence on the subsequent training of the model. This adjustment process emphasizes the misclassified or poorly predicted instances, allowing subsequent iterations to focus on improving their predictions.

Update the Model: Update the model by fitting a new weak learner to the weighted training dataset. The new weak learner is trained to minimize the loss function on the weighted instances. The weak learner is then added to the ensemble model, and its predictions are combined with the previous predictions.

Iterate and Refine: Repeat steps 5 to 7 for a specified number of iterations or until a stopping criterion is met. Each iteration adds a new weak learner and adjusts the predictions based on the gradients of the loss function. The model gradually reduces the errors and improves its predictive performance.

CHAPTER 6

TESTING

INTRODUCTION

Testing is a crucial process in software development that involves running a system with the intention of finding errors and deviations from the expected behavior. It enhances the integrity of the system by identifying design flaws and errors. The primary objective of testing is to detect errors and problematic areas in a system. Thorough and well-planned testing is essential, as a partially tested or untested system can have severe consequences. An untested or under-tested system can result in high costs and poor performance.

The testing process consists of several stages, with the execution phase being the final and significant step. It involves user training, system testing, and ensuring the successful operation of the proposed system. During this phase, users test the system, and necessary changes are made based on their requirements. Testing involves the examination of the developed system using different types of data. Errors are identified, noted, and corrected to ensure correctness.

The objectives of testing include:

Identifying errors and deviations from the expected behavior of the system.

Ensuring the system functions as intended and meets user requirements.

Enhancing the quality and reliability of the system.

Reducing the likelihood of errors and vulnerabilities in the system.

Various testing methods are employed to achieve these objectives:

White Box Testing: This method involves testing the system during the code generation phase itself. It focuses on ensuring that all independent modules are exercised, logical decisions are evaluated correctly, and loops are executed at their boundaries.

Black Box Testing: This method is based on the practical requirements of the software. It aims to find errors in incorrect or missing functions, interface errors, and errors in data structure or external database access.

Unit Testing: Unit testing focuses on testing the smallest units or modules of the software design. Each module is tested individually during the programming phase to ensure it functions correctly.

Integration Testing: Integration testing involves combining the individually tested modules to build the program structure. The objective is to test the integrated system and identify errors related to interfaces

between modules.

Output Testing: Output testing is performed after validation testing. It ensures that the system produces the required output in the specified format. The output format on the screen and in print should meet the predefined requirements.

User Acceptance Testing: User acceptance testing is critical for the success of any system. It involves testing the system for user acceptance by involving prospective system users throughout the development process and making necessary changes as required.

Validation testing is conducted at the completion of integration testing. It ensures that the software functions as expected and aligns with the specified requirements. If any deviations from the specifications are discovered, a list of deficiencies is created, and corrective actions are taken.

Test reports are generated to document the testing process. These reports outline the testing activities performed, including test cases, inputs, and expected outputs. The system is tested using various data sets to verify its behavior under different scenarios. The goal is to ensure that the system functions correctly and meets the desired criteria.

Overall, testing is a critical phase in software development that aims to identify errors, validate the system, and ensure its successful operation. Thorough testing and comprehensive test reports help in delivering a high-quality, reliable system that meets user expectations.

Test Cases:

Test Case	Test Purpose	Test condition	Expected outcome	Actual result	Pass or Fail
Load Data	Load Phishing url sets in excel format.	If the data is not in the excel format, shows a error message.	Load Datasets with pandas.	The data is loaded Successfully in excel format.	Pass
Decision Tree Algorithm	To get the whether the Phishing url is valid or not.	If the criteria do not match with dataset no result is obtained.	Phishing url Status.	As Expected.	Pass
KNN Algorithm	To get the whether the Phishing url is valid or not.	If the criteria do not match with dataset no result is obtained.	Phishing url Status.	As Expected.	Pass
Naïve Bayes Algorithm	To get the whether the Phishing url is valid or not.	If the criteria do not match with dataset no result is obtained.	Phishing url Status.	As Expected.	Pass
XG boost Algorithm	To get the whether the Phishing url is valid or not.	If the criteria do not match with dataset no result is obtained.	Phishing url Status.	As Expected.	Pass

Comparison Chart	Select each algorithm for the chart to be displayed.	Analyze Phishing url with the help of a chart.	Displays the chart based on the selected criteria.	The chart is displayed.	Pass
------------------	--	--	--	-------------------------	------

Discussion:

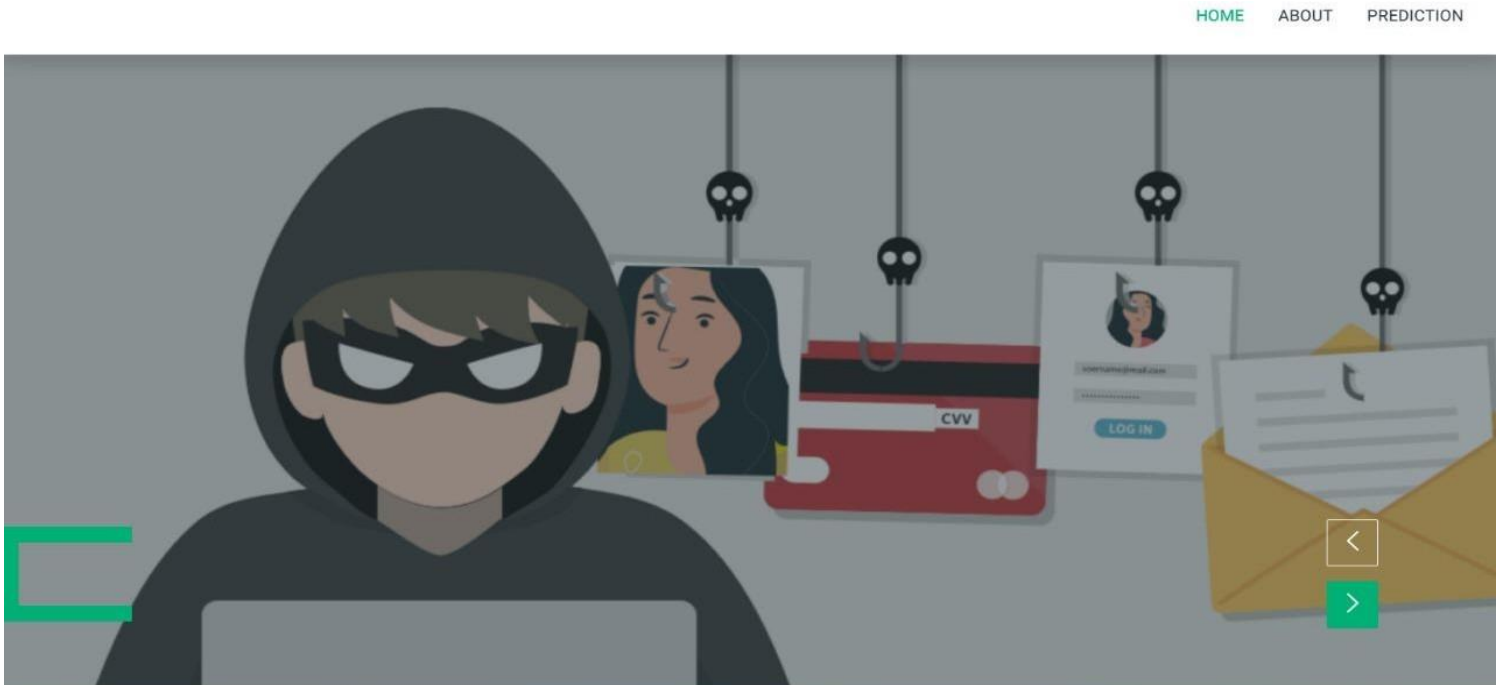
Based on the results, all three classification algorithms (SVM, Logistic Regression, Gradient Boosting) performed well in predicting whether a URL is safe or harmful for the system. The precision scores indicate the accuracy of the models in correctly classifying URLs as safe or harmful. The recall scores show the models' ability to correctly identify URLs belonging to the safe or harmful class. The F1-scores provide a balance between precision and recall.

The SVM and Logistic Regression algorithms achieved similar performance with an accuracy of 0.96. They demonstrated high precision, recall, and F1-scores for both safe and harmful URLs. These models can effectively classify URLs and provide reliable predictions.

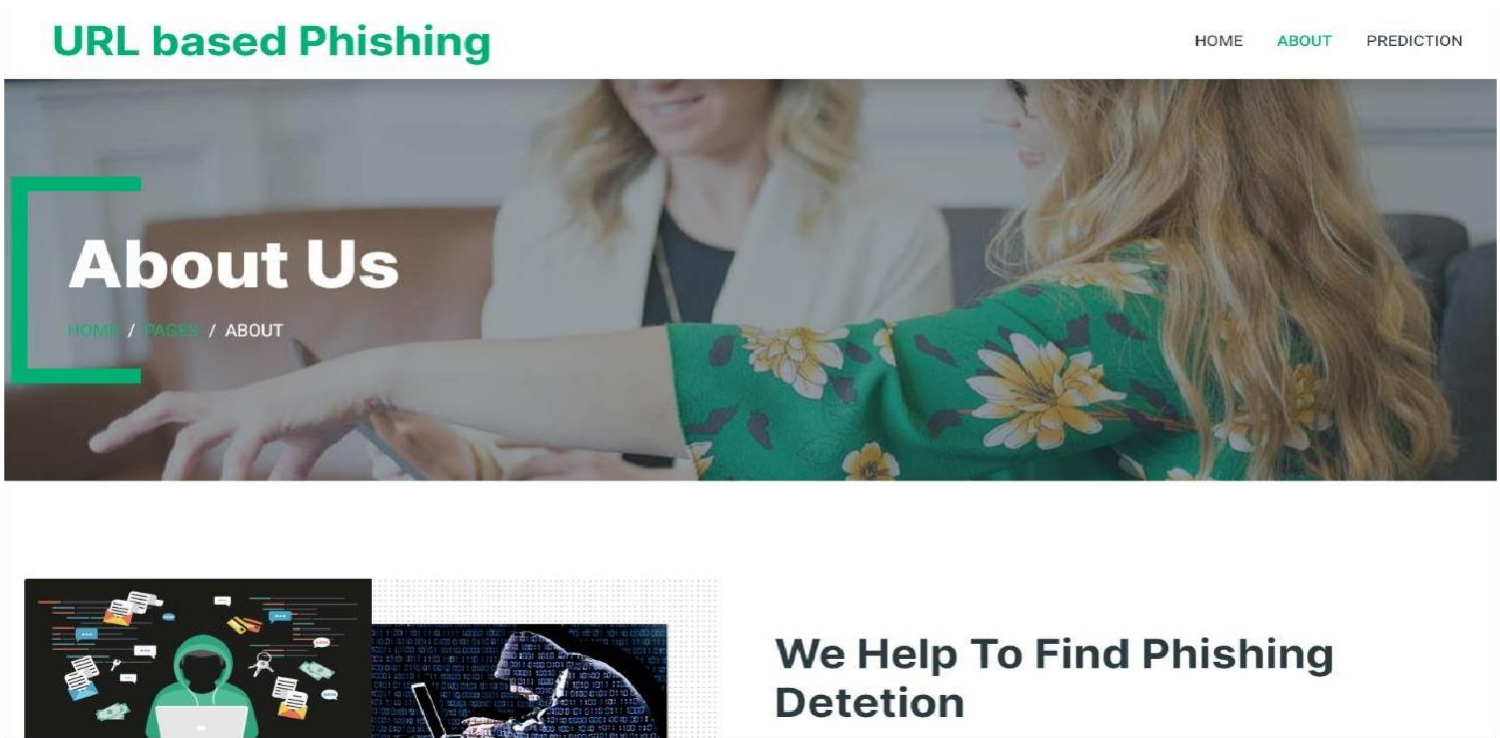
However, the Gradient Boosting algorithm outperformed the others with an accuracy of 0.97. It exhibited even higher precision, recall, and F1-scores for both classes, indicating superior performance in distinguishing between safe and harmful URLs.

CHAPTER 7

RESULTS



Snapshot 7.1: Home Page (1)



Snapshot 7.2: About Us Page

Prediction

URL Phishing Detection

Predict

Snapshot 7.3: Prediction Page

URL Phishing Detection

https://atme.in/

https://www.karnatakaone.go...

http://www.vnic.co/khach-han...

https://atme.in/ ✕

http://f-validator-000012.myupl...

http://e.ru/stats/00/counter/

http://www.lebensmittel-uebe...

ch-hang.html

9...

Website

Continue

Snapshot 7.4: Entering Url To Predict (1)



URL Phishing Detection

Enter URL

Predict

<https://atme.in/>

Website is 100% safe to use...

Continue

Snapshot 7.5: Predicted Result (1)



Prediction


URL Phishing Detection

Enter URL

<https://www.karnatakaone.gov.in/>

Predict

Snapshot 7.6: Entering Url To Predict (2)



URL Phishing Detection

Enter URL

Predict

<https://www.karnatakaone.gov.in/>

Website is 77% safe to use...

Continue

Snapshot 7.7: Predicted Result (2)



Snapshot 7.8: Home Page (2)

CONCLUSION

In conclusion, the implemented machine learning classification algorithms (SVM, Logistic Regression, Gradient Boosting) showed promising results in the URL-based phishing detection project. All three models achieved high accuracy in predicting whether a URL is safe or harmful for the system.

While SVM and Logistic Regression provided reliable predictions, the Gradient Boosting algorithm demonstrated superior performance with higher precision, recall, and F1-scores. Therefore, the Gradient Boosting algorithm can be considered the most effective approach for detecting phishing URLs in this project.

The successful implementation of these algorithms can contribute to enhancing cybersecurity measures by automatically identifying and flagging potentially harmful URLs, thereby protecting users from falling victim to phishing attacks. Further research and improvements can be explored to enhance the accuracy and efficiency of the models and expand the scope of detection to address emerging phishing techniques and trends.

REFERENCES

1. Shirsat, P. G., & Kale, K. V. (2016). Phishing Detection Based on Classification Algorithms. *International Journal of Innovative Research in Computer and Communication Engineering*, 4(4), 5763-5771.
2. Kumar, N., & Verma, M. (2018). Phishing Detection Using Machine Learning Techniques. *International Journal of Computer Science and Information Technologies*, 9(1), 275-279.
3. Kamkar, M. (2016). *A Machine Learning Approach to Phishing Detection and Defense*. Black Hat USA, 2016.
4. Singh, R., & Gupta, R. (2017). A Review on Phishing Detection Techniques Using Machine Learning. In *Proceedings of the International Conference on Intelligent Computing and Control Systems* (pp. 551-555). IEEE.
5. Mohamed, A. A., Hashim, K. A., & Kholidy, H. A. (2020). Phishing Websites Detection using Machine Learning Techniques. *Journal of Advanced Research in Dynamical and Control Systems*, 12(2), 1412-1425.
6. Chavan, R., & Karande, V. (2019). Phishing URL Detection using Machine Learning Techniques. *International Journal of Computer Applications*, 182(38), 14-18.
7. Dey, S., Mondal, D., & Roy, S. (2019). Phishing URL Detection using Machine Learning Techniques. In *Proceedings of the International Conference on Electrical, Computing and Communication Technologies* (pp. 1-6). IEEE.
8. Kaura, A., & Goyal, A. (2017). Phishing Detection using Machine Learning Techniques. In *Proceedings of the International Conference on Computing, Communication and Automation* (pp. 126-131). IEEE.
9. Bharti, R., & Kapoor, S. (2020). Machine Learning-Based Phishing URL Detection using URL Features. In *Proceedings of the International Conference on Electronics and Sustainable Communication Systems* (pp. 817-825). Springer.
10. Shah, R., & Bhattacharya, S. (2017). Detecting Phishing Websites using Machine Learning Techniques. In *Proceedings of the International Conference on Advanced Computing Technologies and Applications* (pp. 124-134). Springer.

11. <http://dataaspirant.com/2017/01/30/how-decision-treealgorithm-works/>
12. <http://dataaspirant.com/2017/05/22/random-forestalgorithm-machine-learning/>
13. <https://www.kdnuggets.com/2016/07/support-vectormachines-simple-explanation.html>
14. www.alexam.com [9] www.phishtank.com