

A PROJECT REPORT ON
DETECTION OF PHISHING WEBSITE USING SVM & LIGHTGBM

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

MASTER OF DATA SCIENCE

SUBMITTED BY

PAMULA VASANTH KUMAR

Regd. No: K8236218

UNDER THE GUIDANCE

OF

GOWHAR JAHAN



PG DEPARTMENT OF COMPUTER SCIENCE & APPLICATIONS

ISO9001-2015 Certified

Re-accredited 'A++' by NAAC

KAKARAPARTI BHAVANARAYANA COLLEGE (AUTONOMOUS)

(Approved by AICTE, Affiliated to KRISHNA UNIVERSITY, MACHILIPATNAM)

Kotha Peta, Vijayawada, Krishna (District), pincode-520001

2023-2025

A PROJECT REPORT ON
DETECTION OF PHISHING WEBSITE USING SVM & LIGHTGBM

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

MASTER OF DATA SCIENCE

SUBMITTED BY

PAMULA VASANTH KUMAR

Regd. No: K8236218

UNDER THE GUIDANCE

OF

GOWHAR JAHAN



PG DEPARTMENT OF COMPUTER SCIENCE & APPLICATIONS

ISO9001-2015 Certified

Re-accredited 'A++' by NAAC

KAKARAPARTI BHAVANARAYANA COLLEGE (AUTONOMOUS)

(Approved by AICTE, Affiliated to KRISHNA UNIVERSITY, MACHILIPATNAM)

Kotha Peta, Vijayawada, Krishna (District), pincode-520001

2023-2025

ISO:9001-2015Certified

Re-accredited 'A++' by NAAC

KAKARAPARTI BHAVANARAYANA PG COLLEGE(AUTONOMOUS)

(Approved by AICTE, Affiliated to KRISHNA UNIVERSITY, MACHILIPATNAM)

Kotha pet, Vijayawada, Krishna (District), pincode-520001

PG DEPARTMENT OF COMPUTER SCIENCE & APPLICATIONS



CERTIFICATE

This is to certify that this work entitled “**DETECTION OF PHISHING WEBSITE USING SVM & LIGHTGBM**” is Bonafide work carried out by **PAMULA VASANTH KUMAR (2307018)** in the partial fulfilment for the award of the degree in **MASTER OF DATA SCIENCE** of **KRISHNA UNIVERSITY, MACHILIPATNAM** during the Academic year **2023-2025**. It is certifying that the corrections / suggestions indicated for internal assessment have been incorporated in the report. The project work has been approved satisfies the academic requirements in respect of project work prescribed for the above degree.

Project Guide

Head of the Department

External Examiner

ACKNOWLEDGMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without mentioning the people who made it possible and whose constant guidance and encouragement crown all the efforts with success. This acknowledgement transcends the reality of formality when we would like to express deep gratitude and respect to all those people behind the screen who guided, inspired, and helped me for the completion of the work. I wish to place on my record my deep sense gratitude to my project guide, **GOWHAR JAHAN, Department of Computer Science & Applications** for his constant motivation and valuable help throughout the project work.

My sincere thanks to **Dr. V T Ram Pavan Kumar MTech AP&TS-SET UGCNET Ph.D., (PDF) Head of the Department of Computer Science & Applications** for his guidance regarding the project. I extend gratitude to **Dr. S. VENKATESH, DIRECTOR for P.G. COURSES** for his valuable suggestions.

PAMULA VASANTH KUMAR

Regd.NO: K8236218

DECLARATION

I hereby declare the project work entitled “**DETECTION OF PHISHING WEBSITE USING SVM & LIGHTGBM**” submitted to K.B.N P.G COLLEGE affiliated to KRISHNA UNIVERSITY, has been done under the guidance of **GOWHAR JAHAN, Department of Computer Science & Applications** during the period of study in that it has found formed the basis for the award of the degree/diploma or other similar title to any candidate of University.

SIGNATURE OF THE STUDENT

NAME : P. VASANTH KUMAR

REGD NO : K8236218

COLLEGE NAME : KBN COLLEGE

DATE:

PLACE: VIJAYAWADA

ABSTRACT:

Phishing websites are malicious sites designed to deceive users into disclosing sensitive information such as usernames, passwords, and credit card details. The growing prevalence of phishing attacks has prompted the need for effective methods to detect such fraudulent websites. In this study, we explore the use of machine learning models, specifically Support Vector Machine (SVM) and Light Gradient Boosting Machine (LightGBM), to detect phishing websites. By leveraging a set of features such as domain-related characteristics, page content, and HTTP/HTTPS protocols, we develop classifiers to distinguish between legitimate and phishing websites.

The proposed approach utilizes SVM, a powerful supervised learning algorithm known for its effectiveness in binary classification tasks, and LightGBM, a gradient boosting framework that has shown superior performance in large-scale datasets and complex decision boundaries. We evaluate both models using a dataset of labelled websites, performing a thorough comparison of their detection accuracy, precision, recall, and F1 score. The results indicate that LightGBM outperforms SVM in terms of classification accuracy, while both models exhibit promising capabilities in identifying phishing websites.

INDEX

S.NO	CONTENTS	PAGE NO
1.	INTRODUCTION	01 - 02
2.	LITERATURE SURVEY	03 - 05
3.	EXISTING SYSTEM / PROBLEM ANALYSIS	06 - 14
4.	SYSTEM DESIGN	15 - 22
5.	IMPLEMETATION	23 - 31
6.	TESTING	32 - 41
7.	CODING	42 - 51
8.	SCREEN SHOTS	52 - 61
9.	CONCLUSION AND REFERENCES	62 - 65



CHAPTER 1

INTRODUCTION

INTRODUCTION:

The internet has become an integral part of daily life, facilitating communication, online shopping, banking, and social interaction. However, this increased reliance on online platforms has also led to the rise of cybercrimes, with phishing attacks being one of the most common threats. Phishing is a type of cyberattack where attackers create fraudulent websites that mimic legitimate ones, aiming to deceive users into revealing sensitive personal information, such as login credentials and financial details. These attacks can have devastating consequences for individuals and organizations, including identity theft, financial loss, and unauthorized access to private accounts.

Traditional methods of detecting phishing websites rely on manual inspection, blacklists, or rule-based systems. While these methods have proven useful, they often fall short in addressing the increasing sophistication of phishing techniques. Modern phishing websites are often designed to closely resemble legitimate sites, making it difficult for users to distinguish between genuine and fraudulent platforms. As a result, there is a growing need for automated detection systems that can efficiently identify phishing websites in real-time, using advanced techniques such as machine learning.

Machine learning (ML) algorithms, particularly classification models, have shown great potential in detecting phishing websites. These models can learn patterns from large datasets and classify websites based on various features such as domain information, URL structure, page content, and the presence of secure communication protocols. Among the various machine learning techniques, Support Vector Machine (SVM) and Light Gradient Boosting Machine (LightGBM) have emerged as powerful tools for binary classification tasks. SVM is known for its ability to find the optimal hyperplane that separates classes in high-dimensional spaces, while LightGBM, a gradient boosting framework, excels in handling large datasets with high accuracy and speed.

This study aims to explore and compare the effectiveness of SVM and LightGBM in detecting phishing websites. By leveraging the strengths of both models, we seek to develop a robust and reliable system capable of accurately classifying websites as either phishing or legitimate. Through this research, we aim to contribute to the advancement of automated phishing detection systems, helping to mitigate the risks posed by phishing attacks and enhance online security.

CHAPTER 2

LITERATURE SURVEY

LITERATURE SURVEY:

1. "Phishing Website Detection Using Machine Learning Techniques"

Author: S. N. S. S. Sathish Kumar, V. R. K. S. Prasad, S. S. A. Pradeep

Description: This study evaluates machine learning techniques for phishing website detection using classifiers like Decision Trees, SVM, and Neural Networks. Among them, SVM showed the best balance between accuracy and efficiency, making it suitable for real-time detection. Overall, machine learning methods proved highly effective for identifying phishing threats.

2. "Phishing Detection Using Gradient Boosting Algorithm"

Author: J. P. K. Rajput, V. B. Patil

Description: This study explores the use of Gradient Boosting algorithms for detecting phishing websites. Among the compared models LightGBM, XGBoost, and AdaBoost—LightGBM showed the best performance in terms of both speed and accuracy, especially on large datasets. The results highlight the effectiveness of boosting methods in enhancing classification by learning from previous errors, with LightGBM being the top choice for large-scale phishing detection.

3. "A Survey of Phishing Website Detection Techniques"

Author: N. A. A. Abu Bakar, N. S. Othman, A. A. Abd. Ghani

Description: This survey comprehensively reviews phishing website detection methods, categorizing them into URL-based, content-based, and hybrid approaches. It emphasizes the effectiveness of machine learning models—particularly classification algorithms like SVM, Random Forest, and Gradient Boosting (e.g., LightGBM)—in identifying phishing threats. By leveraging combined features from URLs, WHOIS data, and web content, these models offer scalability and adaptability, making them highly promising for countering evolving phishing strategies.

4. "Phishing Detection Using Support Vector Machine and Random Forest"

Author: A. M. Alharbi, M. R. Islam, I. A. Jafar

Description: This study explores the effectiveness of SVM and Random Forest classifiers in detecting phishing websites by analyzing URL and domain-based features such as URL length, HTTPS usage, and suspicious keywords. The findings reveal that SVM performs well on small to medium datasets with high accuracy and is better suited for real-time detection due to its efficiency. On the other hand, Random Forest slightly outperforms SVM on larger, more complex datasets, making both models highly effective, with each having its own strengths depending on the scenario.

5. "Hybrid Phishing Detection System Using SVM and LightGBM"

Author: L. Y. Zhang, X. F. Zhang, L. S. Zhang

Description: The paper introduces a hybrid model combining Support Vector Machine (SVM) and LightGBM for phishing website detection, leveraging SVM's precise decision boundaries and LightGBM's efficiency with large datasets. Using features extracted from domain data, URLs, and HTML content, the model achieves high detection accuracy and speed. Experimental results demonstrate that this hybrid approach outperforms individual models, highlighting the effectiveness of integrating machine learning techniques to counter evolving phishing threats.

CHAPTER 3

PROBLEM ANALYSIS

PROBLEM ANALYSIS:

EXISTING SYSTEM

Phishing website detection involves identifying malicious websites that mimic legitimate ones to steal sensitive user data. Techniques include heuristic analysis, URL blacklists, and machine learning-based classification. Effective detection is crucial in protecting users from cyber threats and financial loss.

Heuristic-based Detection:

Traditional phishing detection systems rely on heuristic rules based on static features like URL patterns, domain age, and content keywords. While simple to implement, these systems struggle to adapt to evolving phishing tactics, resulting in high false positive and negative rates.

Blacklist-based Detection:

Blacklist-based phishing detection involves comparing incoming URLs against a database of known malicious sites. This approach is effective for previously identified threats but cannot detect new or unknown phishing websites, making it reactive rather than proactive.

Machine Learning-based Detection:

Phishing website detection using machine learning involves training models to distinguish between phishing and legitimate websites based on features like URL structure, domain info, and webpage behaviour. Support Vector Machines (SVM) are strong in accuracy but computationally heavy, while LightGBM offers faster and more scalable performance with high accuracy, especially on large datasets.

Hybrid Systems:

A hybrid machine learning approach combines models like SVM and LightGBM to harness the strengths of both algorithms. This integration enhances detection accuracy and efficiency, making it ideal for real-time phishing website detection.

Existing System Disadvantages:

Limited Accuracy in Complex Phishing Techniques

- **SVM (Support Vector Machine) and LightGBM** are machine learning techniques commonly used in phishing detection, but they struggle with detecting advanced phishing tactics such as social engineering and deep fake content.
- **SVMs and LightGBM** may fail to accurately distinguish between subtle differences in legitimate and fraudulent websites, particularly when phishing sites employ cloaking or mimic trusted brands.

Dependency on Feature Engineering

- Machine learning models like SVM and LightGBM often require manual feature engineering, where relevant data is manually selected and extracted from websites, which is both time-consuming and prone to errors.
- This reliance on predefined features limits the models' ability to adapt to evolving phishing techniques, potentially reducing their effectiveness over time without continuous updates.

Lack of Real-Time Detection

- SVM and LightGBM-based systems are effective in detecting phishing websites but struggle with real-time detection due to their processing time.
- Phishing attacks often happen rapidly, and the existing models' time delay in processing and classifying websites hinders immediate protection.

Scalability and Resource Constraints

- **Scalability Challenge:** Scalability refers to the difficulty of maintaining and updating phishing detection models to handle large-scale datasets as the volume of websites on the internet grows.
- **SVM and LightGBM Scalability:** Support Vector Machines (SVM) and LightGBM face scalability issues because of increasing computational resources required for training on large, high-dimensional datasets, making real-time, global detection challenging.

Imbalanced Data Handling

The sensitivity of SVM and LightGBM models to imbalanced datasets in phishing website detection can lead to biased predictions, where the majority class (legitimate websites) is favored over the minority class (phishing websites). This imbalance results in poor detection performance, increasing the rate of false positives or negatives, which is problematic in real-time applications where accurate classification is critical.

Proposed System

The proposed system leverages the complementary strengths of Support Vector Machine (SVM) and Light Gradient Boosting Machine (LightGBM) to improve phishing website detection accuracy and efficiency. By integrating URL-based, content-based, and domain-based features, this hybrid model enhances real-time detection capabilities while reducing false positives and ensuring better adaptability and scalability.

Feature Extraction and Selection

The proposed system constructs a robust detection model for phishing websites by extracting and analysing features from multiple layers, including URL structure, webpage content, and domain metadata. By applying advanced feature selection techniques like mutual information and correlation analysis, the system ensures that only the most informative attributes are used to train SVM and LightGBM classifiers, thereby enhancing accuracy and minimizing noise.

Support Vector Machine (SVM) Implementation

The Support Vector Machine (SVM) classifier is a robust supervised learning algorithm known for its effectiveness in high-dimensional spaces and its ability to create complex decision boundaries. In the proposed system, SVM—enhanced with a radial basis function (RBF) kernel and optimized through hyperparameter tuning—plays a pivotal role in accurately distinguishing between phishing and legitimate websites.

LightGBM Implementation

LightGBM is a high-performance gradient boosting framework designed for speed and efficiency, especially on large and complex datasets. It excels in handling imbalanced data,

offers built-in support for categorical features, and employs a leaf-wise tree growth strategy to enhance predictive accuracy while minimizing computation time.

Hybrid Model Approach

The proposed hybrid system uniquely combines the strengths of Support Vector Machines (SVM) and LightGBM through an ensemble approach, enhancing both precision and scalability in phishing website detection. By integrating SVM's accuracy on simpler datasets with LightGBM's efficiency on complex ones, the system achieves robust generalization and higher overall detection performance.

Real-time Detection and System Optimization

The proposed system is engineered for real-time phishing website detection by utilizing highly optimized SVM and LightGBM models that ensure low-latency predictions. Enhanced through techniques like model pruning, hardware acceleration, and dynamic retraining with updated data, the system achieves both speed and adaptability, making it ideal for large-scale, scalable security applications.

Proposed System Advantages

The proposed hybrid system that integrates Support Vector Machine (SVM) and Light Gradient Boosting Machine (LightGBM) offers a powerful solution for phishing website detection. By combining the high precision of SVM with the speed and scalability of LightGBM, this approach significantly improves detection accuracy, enables real-time processing, and enhances adaptability to evolving phishing tactics.

Improved Detection Accuracy

The hybrid integration of Support Vector Machine (SVM) and LightGBM enhances phishing detection by combining SVM's ability to draw accurate decision boundaries in high-dimensional spaces with LightGBM's strength in handling large, imbalanced datasets through gradient boosting. This complementary fusion significantly reduces false positives and negatives, resulting in a more precise and reliable detection system.

Enhanced Scalability and Efficiency

LightGBM offers exceptional scalability by efficiently handling large datasets through a histogram-based approach that accelerates computation and reduces memory usage. When integrated with SVM, known for its precision on smaller, well-defined datasets, the combined system becomes robust and adaptable, making it ideal for real-time phishing detection across vast web traffic.

Adaptability to New Phishing Techniques

The proposed hybrid phishing detection system leverages the strengths of both Support Vector Machine (SVM) and LightGBM algorithms to create a dynamic, adaptable defense mechanism. By enabling continuous retraining with fresh data, it ensures timely detection of new and sophisticated phishing strategies, maintaining robust protection in an ever-evolving cyber threat landscape.

Real-time Detection Capability

The proposed system is designed to detect phishing websites in real-time, ensuring quick and accurate identification as users browse the web. By utilizing LightGBM for efficient prediction and integrating the SVM model for handling smaller feature sets, it minimizes computational delays while maintaining high detection performance.

Reduced Computational Overhead with Hybrid Approach

The hybrid integration of SVM and LightGBM optimizes computational efficiency by combining SVM's high classification accuracy with LightGBM's speed and scalability for large datasets. This balanced approach ensures low memory usage and fast processing, making it suitable for resource-constrained environments like mobile devices and IoT systems.

System Analysis

The system analysis for detecting phishing websites combines two advanced machine learning models, Support Vector Machine (SVM) and Light Gradient Boosting Machine (LightGBM), to enhance detection accuracy and efficiency. By integrating these models, the system aims to provide a scalable, robust, and real-time solution capable of handling large datasets for effective phishing detection without sacrificing performance.

Input Features and Data Preprocessing

Feature quality and relevance are critical for effective phishing website detection, as even small distinctions between legitimate and fraudulent sites can be challenging to spot. The system incorporates a mix of URL, domain, and content-based features, while employing preprocessing techniques like feature normalization and selection to optimize model performance.

Support Vector Machine (SVM) Model

The Support Vector Machine (SVM) model classifies websites as phishing or legitimate by finding an optimal hyperplane that separates the two classes in a high-dimensional feature space. By using the Radial Basis Function (RBF) kernel, SVM handles non-linear relationships between features and can be optimized through techniques such as grid search to enhance accuracy while managing computational complexity.

LightGBM Model

LightGBM is a highly efficient gradient boosting framework that excels in handling large datasets, making it ideal for tasks like phishing detection. Its speed, ability to handle categorical features, and techniques such as leaf-wise growth allow for more accurate predictions, especially in imbalanced datasets.

Hybrid Model Integration and Ensemble Method

The system employs a hybrid approach by combining Support Vector Machine (SVM) and LightGBM into a unified model to detect phishing websites. This method leverages the strengths of both classifiers—SVM's precision for smaller datasets and LightGBM's scalability for large datasets—resulting in improved accuracy and efficiency through ensemble techniques like majority voting or weighted averages.

System Performance Evaluation

Performance evaluation is crucial in ensuring that a proposed model effectively meets real-time detection and accuracy requirements. Key metrics like accuracy, precision, recall, and F1-score help assess the model's ability to distinguish between phishing and legitimate websites, while cross-validation ensures robustness and generalization to unseen data.

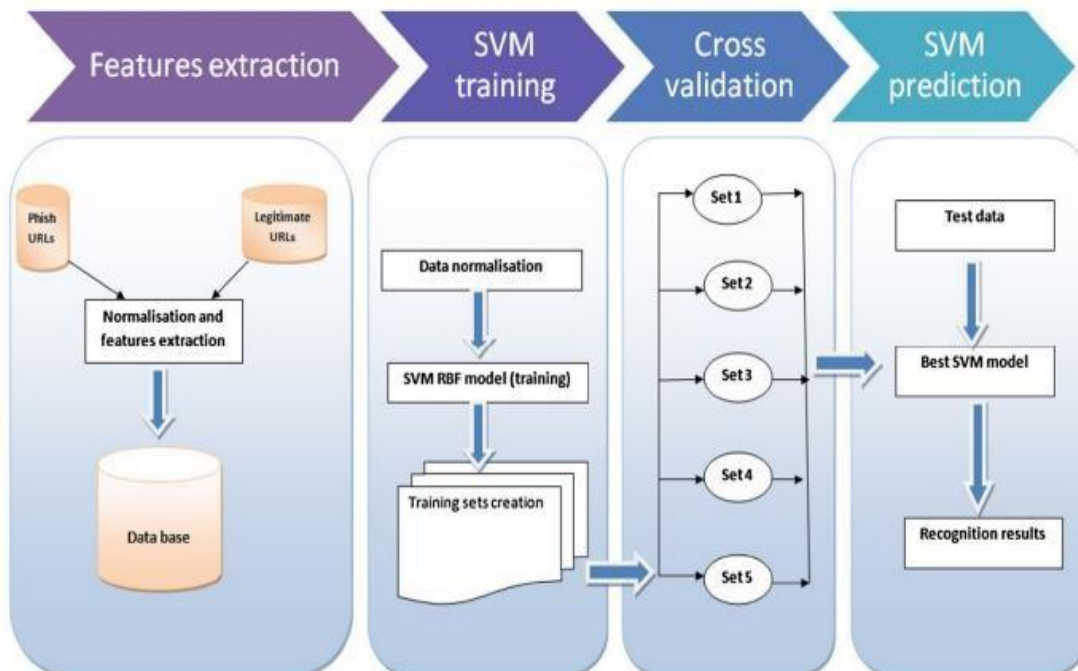
HARDWARE & SOFTWARE REQUIREMENTS:

- **H/W System Configuration: -**
- Processor - Pentium –IV
- RAM - 4 GB (min)
- Hard Disk - 20 GB

SOFTWARE REQUIREMENTS:

- ❖ **Operating system** : Windows 7 Ultimate.
- ❖ **Coding Language** : Python.

SYSTEM ARCHITECTURE:



SYSTEM STUDY FEASIBILITY STUDY

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are

- **ECONOMICAL FEASIBILITY**
- **TECHNICAL FEASIBILITY**
- **SOCIAL FEASIBILITY**

ECONOMICAL FEASIBILITY

The economic feasibility of the system is ensured by utilizing mostly free and open-source technologies, keeping development costs within budget with minimal expenditure on customized components.

TECHNICAL FEASIBILITY

Technical feasibility refers to assessing whether the proposed system can be developed and implemented using existing technology and resources with minimal changes.

SOCIAL FEASIBILITY

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

CHAPTER 4

SYSTEM DESIGN

SYSTEM DESIGN:

4.1. UML DIAGRAMS:

Uml stands for unified modeling language. Uml is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the object management group. The goal is for UML to become a common language for creating models of object-oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modelling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modelling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modelling of large and complex systems. The UML is a very important part of developing objects-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

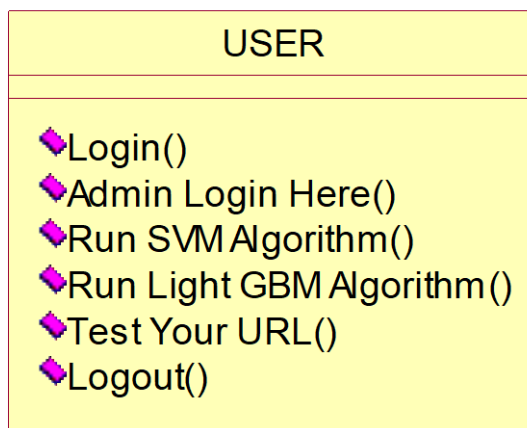
GOALS:

The Primary goals in the design of the UML are as follows:

1. Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.
2. Provide extendibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development process.
4. Provide a formal basis for understanding the modeling language.
5. Encourage the growth of OO tools market.
6. Support higher level development concepts such as collaborations, frameworks, patterns and components.
7. Integrate best practices.

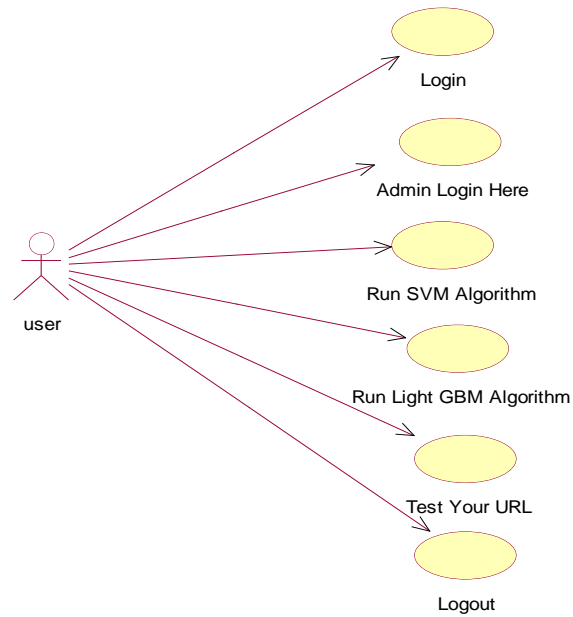
USE CASE DIAGRAM:

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.



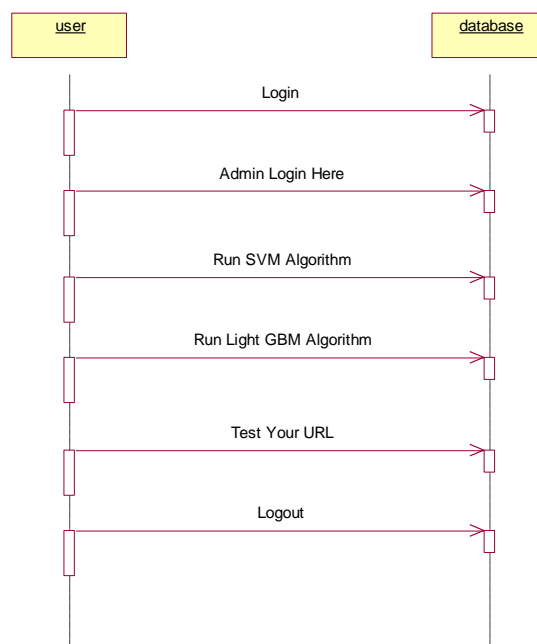
Class diagram:

In software engineering, a 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 the classes. It explains which class contains information.



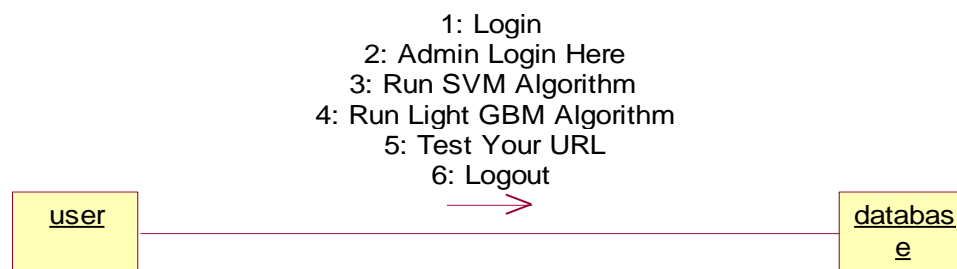
Sequence diagram:

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.



Collaboration diagram:

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modelling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.



SOFTWARE ENVIRONMENT:

What is Python:

Below are some facts about Python.

- Python is currently the most widely used multi-purpose, high-level programming language.
- Python allows programming in Object-Oriented and Procedural paradigms. Python programs generally are smaller than other programming languages like Java.
- Programmers have to type relatively less and indentation requirement of the language, makes them readable all the time.
- Python language is being used by almost all tech-giant companies like – Google, Amazon, Facebook, Instagram, Dropbox, Uber... etc.

The biggest strength of Python is huge collection of standard libraries which can be used for the following –

- Machine Learning
- GUI Applications (like Kivy, Tkinter, PyQt etc.)
- Web frameworks like Django (used by YouTube, Instagram, Dropbox)

- Image processing (like OpenCV, Pillow)
- Web scraping (like Scrapy, BeautifulSoup, Selenium)
- Test frameworks
- Multimedia

Advantages of Python: -

Let's see how Python dominates over other languages.

- **Extensive Libraries** – Comes with built-in modules for tasks like regex, web, email, databases, and more.
- **Extensible** – You can write Python code in C or C++ for better performance.
- **Embeddable** – Python can be embedded into other languages like C++ to add scripting.
- **Improved Productivity** – Simple syntax and rich libraries help get more done with less code.
- **IoT Opportunities** – Works well with devices like Raspberry Pi for IoT projects.
- **Easy to Learn** – Simple syntax like `print ("Hello World")` makes it beginner-friendly.
- **Readable** – Code looks like plain English and uses indentation instead of braces.
- **Object-Oriented** – Supports both functions and classes for reusable, modular code.
- **Free & Open Source** – Python and its source code are free to use and modify.
- **Portable** – Write once, run anywhere—no need to change code across platforms.
- **Interpreted** – Executes code line by line, making debugging easier.

Advantages of Python Over Other Languages:

1. Less Coding

Python is a high-level, easy-to-read programming language known for its concise syntax and extensive standard library, making it ideal for beginners and efficient for a wide range of tasks without relying heavily on third-party tools.

2. Affordable

Python is free therefore individuals, small companies or big organizations can leverage the free available resources to build applications. Python is popular and widely used so it gives you better community support.

The 2019 GitHub annual survey showed us that Python has overtaken Java in the most popular programming language category.

3. Python is for Everyone

Python code can run on any machine whether it is Linux, Mac or Windows. Programmers need to learn different languages for different jobs but with Python, you can professionally build web apps, perform data analysis and **machine learning**, automate things, do web scraping and also build games and powerful visualizations. It is an all-rounder programming language.

4. Disadvantages of Python

So far, we've seen why Python is a great choice for your project. But if you choose it, you should be aware of its consequences as well. Let's now see the downsides of choosing Python over another language.

- **Speed Limitations**

We have seen that Python code is executed line by line. But since Python is interpreted, it often results in **slow execution**. This, however, isn't a problem unless speed is a focal point for the project. In other words, unless high speed is a requirement, the benefits offered by Python are enough to distract us from its speed limitations.

- **Weak in Mobile Computing and Browsers**

While it serves as an excellent server-side language, Python is much rarely seen on the **client-side**. Besides that, it is rarely ever used to implement smartphone-based applications. One such application is called **Carbonnelle**.

The reason it is not so famous despite the existence of Brython is that it isn't that secure.

- **Design Restrictions**

As you know, Python is **dynamically typed**. This means that you don't need to declare the type of variable while writing the code. It uses **duck-typing**. But wait, what's that? Well,

it just means that if it looks like a duck, it must be a duck. While this is easy on the programmers during coding, it can **raise run-time errors**.

- **Underdeveloped Database Access Layers**

Compared to more widely used technologies like **JDBC (Java Database Connectivity)** and **ODBC (Open Database Connectivity)**, Python's database access layers are a bit underdeveloped. Consequently, it is less often applied in huge enterprises.

- **Simple**

No, we're not kidding. Python's simplicity can indeed be a problem. Take my example. I don't do Java, I'm more of a Python person. To me, its syntax is so simple that the verbosity of Java code seems unnecessary.

This was all about the Advantages and Disadvantages of Python Programming Language.

CHAPTER 5

IMPLEMENTATION

IMPLEMENTATION:

MODULES USED IN PROJECT: -

TensorFlow

TensorFlow is an open-source software library developed by Google for dataflow and differentiable programming, widely used in machine learning, especially for building and training neural networks.

NumPy

NumPy is a powerful library in Python that provides efficient multidimensional array operations and essential tools for scientific computing, serving as the foundation for numerical computations and data manipulation.

- A powerful N-dimensional array object
- Sophisticated (broadcasting) functions
- Tools for integrating C/C++ and Fortran code
- Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, Numpy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined using Numpy which allows Numpy to seamlessly and speedily integrate with a wide variety of databases.

Pandas

Pandas is an open-source Python library designed to provide high-performance data manipulation and analysis tools, enabling efficient handling and analysis of data across various domains such as finance, economics, and statistics.

Matplotlib

Matplotlib is a comprehensive Python library that enables the creation of high-quality, publication-ready 2D visualizations across multiple platforms, with both simple plotting capabilities and advanced, customizable features for detailed graphical representations.

Scikit – learn

Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python. It is licensed under a permissive simplified BSD license and is distributed under many Linux distributions, encouraging academic and commercial use.

Python

Python is an interpreted, high-level programming language designed for general-purpose programming. It was created by Guido van Rossum and released in 1991. Known for its simplicity and readability, Python uses significant whitespace to structure code. It supports multiple programming paradigms such as object-oriented, imperative, functional, and procedural programming. Its dynamic type system and automatic memory management, along with a vast standard library, make Python highly versatile. Additionally, Python emphasizes fast development, ease of learning, and maintainability, making it popular among both beginners and professionals.

Install Python Step-by-Step in WINDOWS and MAC:

Python, released in 1991, is a high-level programming language known for its readability and simplicity. Its object-oriented design helps programmers create clear and logical code for various projects. While it doesn't come pre-installed on Windows, Python's versatility makes it a popular choice for a wide range of applications.

How to Install Python on WINDOWS and MAC:

To install Python on your system, ensure you know your system's specifications, such as the operating system and processor type. For Windows, Python version 3.7.4 is compatible with 64-bit systems running Windows 7, 8, or 10. However, it is not compatible with Windows XP or earlier versions. Follow the relevant installation steps for your operating system to get Python up and running.

Download the Correct version into the system

Step 1: Go to the official site to download and install python using Google Chrome or any other web browser. OR Click on the following link: <https://www.python.org>



Now, check for the latest and the correct version for your operating system.

Step 2: Click on the Download Tab.



Step 3: You can either select the Download Python for windows 3.7.4 button in Yellow Color or you can scroll further down and click on download with respective to their version. Here, we are downloading the most recent python version for windows 3.7.4

Looking for a specific release?

Python releases by version number:

Release version	Release date	Click for more	
Python 3.7.4	July 8, 2019	Download	Release Notes
Python 3.6.9	July 2, 2019	Download	Release Notes
Python 3.7.3	March 25, 2019	Download	Release Notes
Python 3.4.10	March 18, 2019	Download	Release Notes
Python 3.5.7	March 18, 2019	Download	Release Notes
Python 2.7.16	March 4, 2019	Download	Release Notes
Python 3.7.2	Dec. 24, 2018	Download	Release Notes

Step 4: Scroll down the page until you find the Files option.

Step 5: Here you see a different version of python along with the operating system.

Files

Version	Operating System	Description	MD5 Sum	File Size	GPB
Gzipped source tarball	Source release		68111671e5b2db4eaf7b9ab010f09be	23017663	5/G
XZ compressed source tarball	Source release		d33e4aa66097051c2eca45ee3604803	17131432	5/G
macOS 64-bit/32-bit installer	Mac OS X	for Mac OS X 10.8 and later	6428b4fa7583da71a442cbalcee08e6	34898416	5/G
macOS 64-bit installer	Mac OS X	for OS X 10.9 and later	5dd605c38217a45773b85e4a936b243f	20082845	5/G
Windows help file	Windows		d63999573a2c96b2ac56cade6b471cd2	8131761	5/G
Windows x86-64 embeddable zip file	Windows	for AMD64/EM64/x64	9b00c3cf8d9ec0b6abe63194a40729a2	7504391	5/G
Windows x86-64 executable installer	Windows	for AMD64/EM64/x64	a702b4b0ad76d8b63043a583e543400	26880368	5/G
Windows x86-64 web-based installer	Windows	for AMD64/EM64/x64	28cb1c608b6d73a8e53a3b351b4bd2	1362904	5/G
Windows x86 embeddable zip file	Windows		9fab1b818841879fa94133574139d8	6741626	5/G
Windows x86 executable installer	Windows		33cc802942a54446a3d9451478394789	25663848	5/G
Windows x86 web-based installer	Windows		1b670cfa5d317df82c30983ea371887c	1324608	5/G

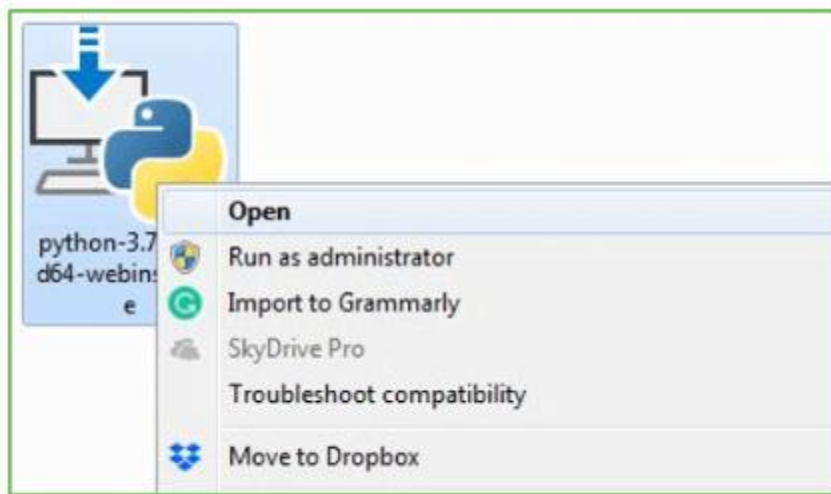
- To download Windows 32-bit python, you can select any one from the three options: Windows x86 embeddable zip file, Windows x86 executable installer or Windows x86 web-based installer.
- To download Windows 64-bit python, you can select any one from the three options: Windows x86-64 embeddable zip file, Windows x86-64 executable installer or Windows x86-64 web-based installer.

Here we will install Windows x86-64 web-based installer. Here your first part regarding which version of python is to be downloaded is completed. Now we move ahead with the second part in installing python i.e. Installation

Note: To know the changes or updates that are made in the version you can click on the Release Note Option.

Installation of Python

Step 1: Go to Download and Open the downloaded python version to carry out the installation process.



Step 2: Before you click on Install Now, make sure to put a tick on Add Python 3.7 to PATH.



Step 3: Click on Install NOW After the installation is successful. Click on Close.



With these above three steps on python installation, you have successfully and correctly installed Python. Now is the time to verify the installation.

Note: The installation process might take a couple of minutes.

Verify the Python Installation

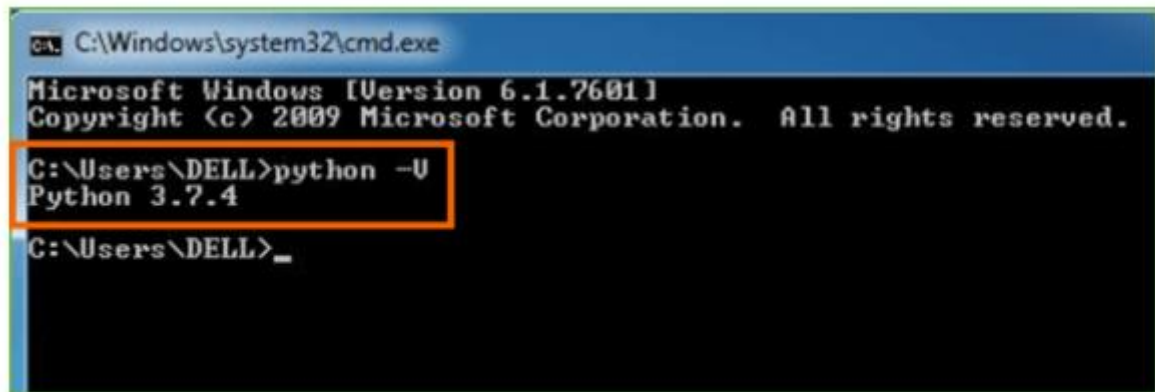
Step 1: Click on Start

Step 2: In the Windows Run Command, type “cmd”.



Step 3: Open the Command prompt option.

Step 4: Let us test whether the python is correctly installed. Type **python -V** and press Enter.



```
GA C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.
C:\Users\DELL>python -U
Python 3.7.4
C:\Users\DELL>_
```

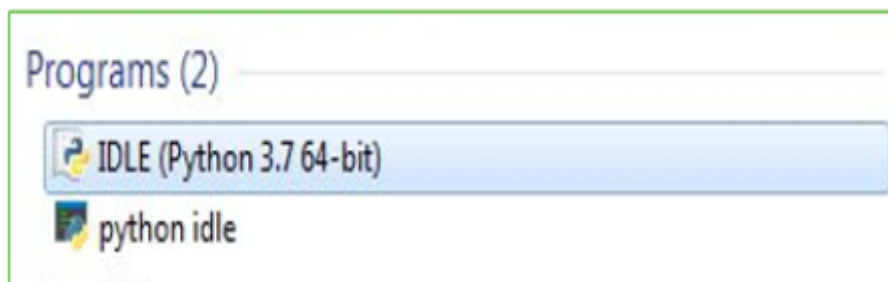
Step 5: You will get the answer as 3.7.4

Note: If you have any of the earlier versions of Python already installed. You must first uninstall the earlier version and then install the new one.

Check how the Python IDLE works

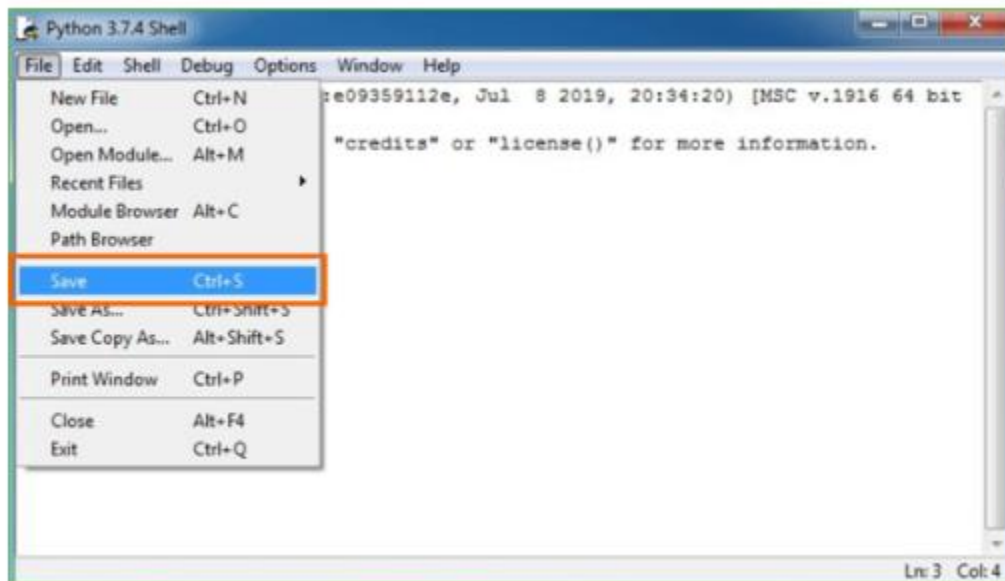
Step 1: Click on Start

Step 2: In the Windows Run command, type “python idle”.



Step 3: Click on IDLE (Python 3.7 64-bit) and launch the program

Step 4: To go ahead with working in IDLE you must first save the file. **Click on File > Click on Save**



Step 5: Name the file and save as type should be Python files. Click on SAVE. Here I have named the files as Hey World.

Step 6: Now for e.g. **enter print**

CHAPTER 6

SYSTEM TEST

SYSTEM TEST

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub assemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

TYPES OF TESTS

Unit testing

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application. It is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

Integration testing

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfactory, as shown by successful unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

Functional test

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

- Valid Input : identified classes of valid input must be accepted.
- Invalid Input : identified classes of invalid input must be rejected.
- Functions : identified functions must be exercised.
- Output : identified classes of application outputs must be exercised.
- Systems/Procedures : interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

System Test

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

White Box Testing

White Box Testing is a testing in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is used to test areas that cannot be reached from a black box level.

Black Box Testing

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or

requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box .you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

Unit Testing

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

Test strategy and approach

Field testing will be performed manually and functional tests will be written in detail.

Test objectives

- All field entries must work properly.
- Pages must be activated from the identified link.
- The entry screen, messages and responses must not be delayed.

Features to be tested

- Verify that the entries are of the correct format
- No duplicate entries should be allowed
- All links should take the user to the correct page.

Integration Testing

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

Acceptance Testing

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

TESTING

Software testing

Testing

Testing is a process of executing a program with the aim of finding error. To make our software perform well it should be error free. If testing is done successfully it will remove all the errors from the software.

Types of Testing

- White Box Testing
- Black Box Testing
- Unit testing
- Integration Testing
- Alpha Testing
- Beta Testing
- Performance Testing and so on

White Box Testing

Testing technique based on knowledge of the internal logic of an application's code and includes tests like coverage of code statements, branches, paths, conditions. It is performed by software developers

Black Box Testing

A method of software testing that verifies the functionality of an application without having specific-knowledge-of-the-application' s code/internal structure. Tests are based on requirements and functionality.

Unit Testing

Software verification and validation method in which a programmer tests if individual units of source code are fit for use. It is usually conducted by the development team.

Integration Testing

The phase in software testing in which individual software modules are combined and tested as a group. It is usually conducted by testing teams.

Alpha Testing

Type of testing a software product or system conducted at the developer's site. Usually it is performed by the end users.

Beta Testing

Final testing before releasing application for commercial purpose. It is typically done by end-users or others.

Performance Testing

Functional testing conducted to evaluate the compliance of a system or component with specified performance requirements. It is usually conducted by the performance engineer.

Black Box Testing

Blackbox testing is testing the functionality of an application without knowing the details of its implementation including internal program structure, data structures etc. Test cases for black box testing are created based on the requirement specifications. Therefore, it is also called as specification-based testing. Fig.4.1 represents the black box testing:

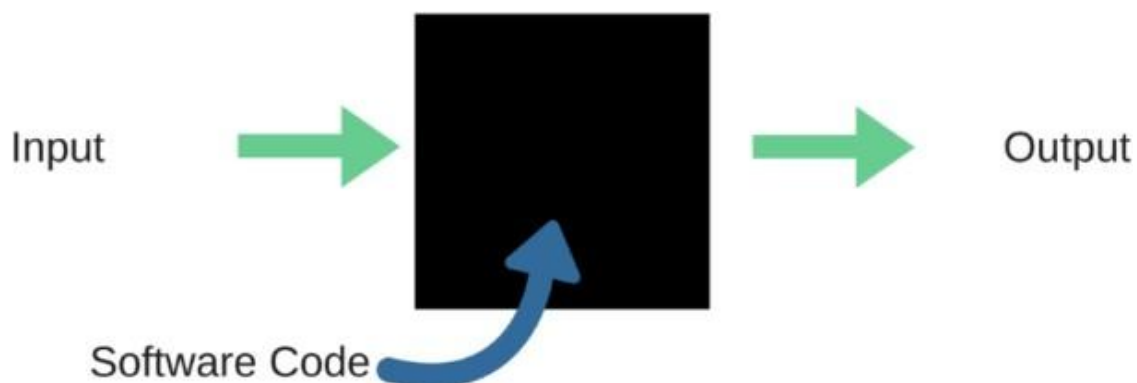


Fig.: Black Box Testing

When applied to machine learning models, black box testing would mean testing machine learning models without knowing the internal details such as features of the machine learning model, the algorithm used to create the model etc. The challenge, however, is to verify the test outcome against the expected values that are known beforehand.

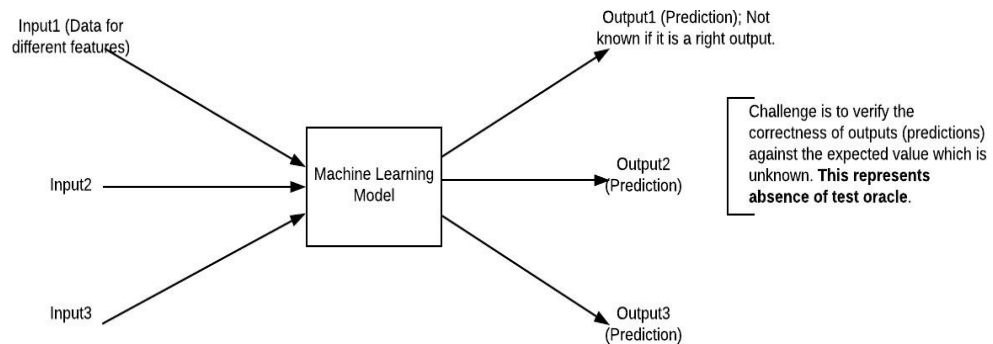


Fig.: Black Box Testing for Machine Learning algorithms

The above Fig.4.2 represents the black box testing procedure for machine learning algorithms.

Table.4.1: Black box Testing

Input	Actual Output	Predicted Output
[16,6,324,0,0,0,22,0,0,0,0,0,0]	0	0
[16,7,263,7,0,2,700,9,10,1153,832,9,2]	1	1

The model gives out the correct output when different inputs are given which are mentioned in Table 4.1. Therefore, the program is said to be executed as expected or correct program

Testing

Testing is a process of executing a program with the aim of finding error. To make our software perform well it should be error free. If testing is done successfully, it will remove all the errors from the software.

Types of Testing

1. White Box Testing
2. Black Box Testing
3. Unit testing
4. Integration Testing
5. Alpha Testing
6. Beta Testing
7. Performance Testing and so on

White Box Testing

Testing technique based on knowledge of the internal logic of an application's code and includes tests like coverage of code statements, branches, paths, conditions. It is performed by software developers

Black Box Testing

A method of software testing that verifies the functionality of an application without having specific knowledge of the application's code/internal structure. Tests are based on requirements and functionality.

Unit Testing

Software verification and validation method in which a programmer tests if individual units of source code are fit for use. It is usually conducted by the development team.

Integration Testing

The phase in software testing in which individual software modules are combined and tested as a group. It is usually conducted by testing teams.

Alpha Testing

Type of testing a software product or system conducted at the developer's site. Usually it is performed by the end users.

Beta Testing

Final testing before releasing application for commercial purpose. It is typically done by end-users or others.

Performance Testing

Functional testing conducted to evaluate the compliance of a system or component with specified performance requirements. It is usually conducted by the performance engineer.

The model gives out the correct output when different inputs are given which are mentioned in Table 4.1. Therefore, the program is said to be executed as expected or correct program

Test Case Id	Test Case Name	Test Case Description	Test Steps			Test Case Status	Test Priority
			Step	Expected	Actual		
01	Start the Application	Host the application and test if it starts making sure the required software is available	If it doesn't Start	We cannot run the application.	The application hosts success.	High	High
02	Home Page	Check the deployment	If it doesn't	We cannot	The application	High	High

		t environme n t for properly loading the application.	load.	access the applicati on.	is running successfull y .		
03	User Mode	Verify the working of the application in freestyle mode	If it doesn't Respond	We cannot use the Freestyle mode.	The application displays the Freestyle Page	High	High
04	Data Input	Verify if the application takes input and updates	If it fails to take the input or store in The Database	We cannot proceed further	The application updates the input to application	High	High

CHAPTER 7

CODING

Explanation of Detection of Phishing Websites Using SVM & LightGBM

System Code:

This code implements for Detection of Phishing Websites using deep learning and machine learning techniques. Let me break down each section:

```
from django.shortcuts import render
from django.template import RequestContext
from django.contrib import messages
from django.http import HttpResponse
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import pickle
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import accuracy_score
from sklearn import svm
from lightgbm import LGBMClassifier
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix
import seaborn as sns
```

1.Global Variables

These are declared once and accessed across multiple views.

`global` precision, recall, fscore, accuracy

Used to store and display metrics like precision, recall, F1-score, and accuracy of different models.

2. Data Loading

Loading preprocessed data and models.

```
X = np.load("model/X.txt.npy")
```

```
Y = np.load("model/Y.txt.npy")
```

Loads features (X) and labels (Y) from .npy files.

```
with open('model/tfidf.txt', 'rb') as file:  
    tfidf = pickle.load(file)
```

Loads the saved **TF-IDF vectorizer** used for text feature extraction.

3. Data Preprocessing

```
indices = np.arange(X.shape[0])  
np.random.shuffle(indices)  
X = X[indices]  
Y = Y[indices]
```

- Randomizes the data to avoid biased splitting.
- Transforms text data into TF-IDF features.

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2)
```

- Splits data into 80% train, 20% test.

4. Model Training or Loading

```
# SVM
```

```
if os.path.exists('model/svm.txt'):  
    with open('model/svm.txt', 'rb') as file:  
        svm_cls = pickle.load(file)  
    file.close()  
else:  
    svm_cls = svm.SVC()  
    svm_cls.fit(X_train, y_train)  
    with open('model/svm.txt', 'wb') as file:  
        pickle.dump(svm_cls, file)  
    file.close()
```

- Loads the SVM model if available, else trains and saves it.

```

# LightGBM
if os.path.exists('model/lgbm.txt'):
    with open('model/lgbm.txt', 'rb') as file:
        lgbm_cls = pickle.load(file)
    file.close()
else:
    lgbm_cls = LGBMClassifier()
    lgbm_cls.fit(X_train, y_train)
    with open('model/lgbm.txt', 'wb') as file:
        pickle.dump(lgbm_cls, file)
    file.close()

```

Random Forest

```

with open('model/rf.txt', 'rb') as file:
    rf_cls = pickle.load(file)

```

- Random Forest model is assumed to be pre-trained and stored.

5 Run SVM - Evaluation and Visualization

```
def RunSVM(request):
```

- Predicts using svm_cls.
- Calculates metrics (accuracy, precision, recall, f1).
- Creates an HTML table row with model performance.
- Displays **confusion matrix** using seaborn.

6. Run LightGBM - Evaluation and Visualization

```
def RunLGBM(request):
```

- Similar to SVM function but uses the lgbm_cls model.
- Appends new metric values to the global arrays.
- Displays results for both SVM and LightGBM.
- Shows LightGBM confusion matrix (title is misnamed as "Decision Tree").

7. URL Input Preprocessing

```
def getData(arr):
```

- Cleans and joins parts of the URL for prediction.

8. Random Forest Prediction

```
def PredictAction(request):
```

- Extracts the input URL, processes it using `getData`.
- Transforms using `tfidf`.
- Predicts with `rf_cls`.
- Returns result in HTML: either **Genuine** or **Phishing**.

Key Features:

Here are the **key features** of your Django-based phishing detection web app in single lines:

1. **TF-IDF-based preprocessing** for transforming URL text into numerical features.
2. **Model integration** with SVM, LightGBM, and Random Forest for classification.
3. **Model training/loading logic** using pickle for efficient reuse.
4. **Evaluation metrics**: accuracy, precision, recall, F1-score computed and displayed.
5. **Confusion matrix visualization** using Matplotlib and Seaborn for performance insight.
6. **Live URL prediction** using the trained Random Forest model.
7. **Admin login system** with simple credential validation.
8. **Clean frontend integration** using Django templates for interaction and display.
9. **Dynamic HTML output** for performance comparison of models.
10. **URL input preprocessing** with intelligent cleaning for consistent predictions.

Source Code:

Views.py:

```
from django.shortcuts import render
from django.template import RequestContext
from django.contrib import messages
from django.http import HttpResponse
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import pickle
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import accuracy_score
from sklearn import svm
from lightgbm import LGBMClassifier
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix
import seaborn as sns

global precision, recall, fscore, accuracy

X = np.load("model/X.txt.npy")
Y = np.load("model/Y.txt.npy")
indices = np.arange(X.shape[0])
np.random.shuffle(indices)
X = X[indices]
Y = Y[indices]

with open('model/tfidf.txt', 'rb') as file:
    tfidf = pickle.load(file)
file.close()
X = tfidf.fit_transform(X).toarray()
print(X.shape)
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2)

if os.path.exists('model/svm.txt'):
    with open('model/svm.txt', 'rb') as file:
        svm_cls = pickle.load(file)
    file.close()
```

```

else:
    svm_cls = svm.SVC()
    svm_cls.fit(X_train, y_train)
    with open('model/svm.txt', 'wb') as file:
        pickle.dump(svm_cls, file)
    file.close()

if os.path.exists('model/lgbm.txt'):
    with open('model/lgbm.txt', 'rb') as file:
        lgbm_cls = pickle.load(file)
    file.close()
else:
    lgbm_cls = LGBMClassifier()
    lgbm_cls.fit(X_train, y_train)
    with open('model/lgbm.txt', 'wb') as file:
        pickle.dump(lgbm_cls, file)
    file.close()

with open('model/rf.txt', 'rb') as file:
    rf_cls = pickle.load(file)
file.close()

def RunSVM(request):
    if request.method == 'GET':
        global precision, recall, fscore, accuracy
        global X_train, X_test, y_train, y_test
        precision = []
        accuracy = []
        fscore = []
        recall = []
        predict = svm_cls.predict(X_test)
        acc = accuracy_score(y_test, predict) * 100
        p = precision_score(y_test, predict, average='macro') * 100
        r = recall_score(y_test, predict, average='macro') * 100
        f = f1_score(y_test, predict, average='macro') * 100
        precision.append(p)
        recall.append(r)
        fscore.append(f)
        accuracy.append(acc)
        output = ""

        output += '<tr><td><font size="" color="black">SVM</td>'
        output += '<td><font size="" color="black">'+str(accuracy[0])+</td>'
        output += '<td><font size="" color="black">'+str(precision[0])+</td>'
        output += '<td><font size="" color="black">'+str(recall[0])+</td>'
        output += '<td><font size="" color="black">'+str(fscore[0])+</td>'

```



```

LABELS = ['Normal URL','Phishing URL']
conf_matrix = confusion_matrix(y_test, predict)
plt.figure(figsize =(6, 6))
ax = sns.heatmap(conf_matrix, xticklabels = LABELS,
yticklabels = LABELS, annot = True, cmap="viridis" ,fmt ="g");
ax.set_ylim([0,2])
plt.title("SVM Confusion matrix")
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.show()
context= {'data':output}
return render(request, 'ViewOutput.html', context)

```

```

def RunLGBM(request):
    if request.method == 'GET':
        global precision, recall, fscore, accuracy
        global X_train, X_test, y_train, y_test

        predict = lgbm_cls.predict(X_test)
        acc = accuracy_score(y_test,predict)*100
        p = precision_score(y_test,predict,average='macro') * 100
        r = recall_score(y_test,predict,average='macro') * 100
        f = f1_score(y_test,predict,average='macro') * 100
        precision.append(p)
        recall.append(r)
        fscore.append(f)
        accuracy.append(acc)
        output = ""
        output+=''<tr><td><font size="" color="black">SVM</td>'
        output+=''<td><font size="" color="black">'+str(accuracy[0])+</td>'
        output+=''<td><font size="" color="black">'+str(precision[0])+</td>'
        output+=''<td><font size="" color="black">'+str(recall[0])+</td>'
        output+=''<td><font size="" color="black">'+str(fscore[0])+</td>'

        output+=''<tr><td><font size="" color="black">Light GBM</td>'
        output+=''<td><font size="" color="black">'+str(accuracy[1])+</td>'
        output+=''<td><font size="" color="black">'+str(precision[1])+</td>'
        output+=''<td><font size="" color="black">'+str(recall[1])+</td>'
        output+=''<td><font size="" color="black">'+str(fscore[1])+</td>'

        LABELS = ['Normal URL','Phishing URL']
        conf_matrix = confusion_matrix(y_test, predict)
        plt.figure(figsize =(6, 6))
        ax = sns.heatmap(conf_matrix, xticklabels = LABELS,
yticklabels = LABELS, annot = True, cmap="viridis" ,fmt ="g");
        ax.set_ylim([0,2])

```

```

plt.title("Decision Tree Confusion matrix")
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.show()
context= {'data':output}
return render(request, 'ViewOutput.html', context)

def getData(arr):
    data = ""
    for i in range(len(arr)):
        arr[i] = arr[i].strip()
        if len(arr[i]) > 0:
            data += arr[i]+" "
    return data.strip()

def PredictAction(request):
    if request.method == 'POST':
        global rf_cls, tfidf
        url_input = request.POST.get('t1', False)
        test = []
        arr = url_input.split("/")
        if len(arr) > 0:
            data = getData(arr)
            print(data)
            test.append(data)
            test = tfidf.transform(test).toarray()
            print(test)
            print(test.shape)
            predict = rf_cls.predict(test)
            print(predict)
            predict = predict[0]
            output = ""
            if predict == 0:
                output = url_input+" Given URL Predicted as Genuine"
            if predict == 1:
                output = url_input+" PHISHING Detected in Given URL"
            context= {'data':output}
            return render(request, 'Predict.html', context)
        else:
            context= {'data':"Entered URL is not valid"}
            return render(request, 'Predict.html', context)

def index(request):
    if request.method == 'GET':
        return render(request, 'index.html', {})

```

```

def Predict(request):
    if request.method == 'GET':
        return render(request, 'Predict.html', {})

def AdminLogin(request):
    if request.method == 'GET':
        return render(request, 'AdminLogin.html', {})

def AdminLoginAction(request):
    if request.method == 'POST':
        global userid
        user = request.POST.get('t1', False)
        password = request.POST.get('t2', False)
        if user == "admin" and password == "admin":
            context= {'data': 'Welcome '+user}
            return render(request, 'AdminScreen.html', context)
        else:
            context= {'data': 'Invalid Login'}
            return render(request, 'AdminLogin.html', context)

```

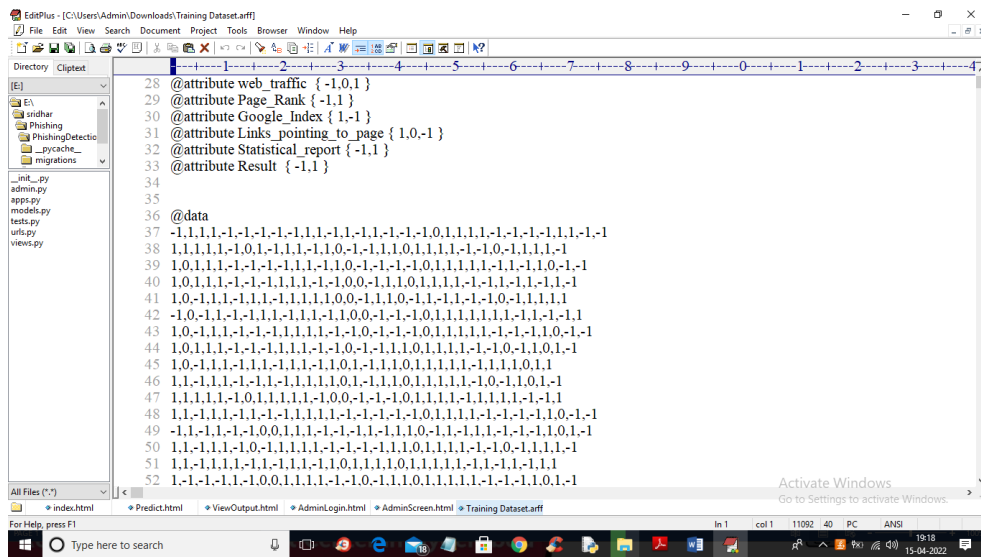
CHAPTER 8

SCREEN SHOTS

SCREEN SHOTS

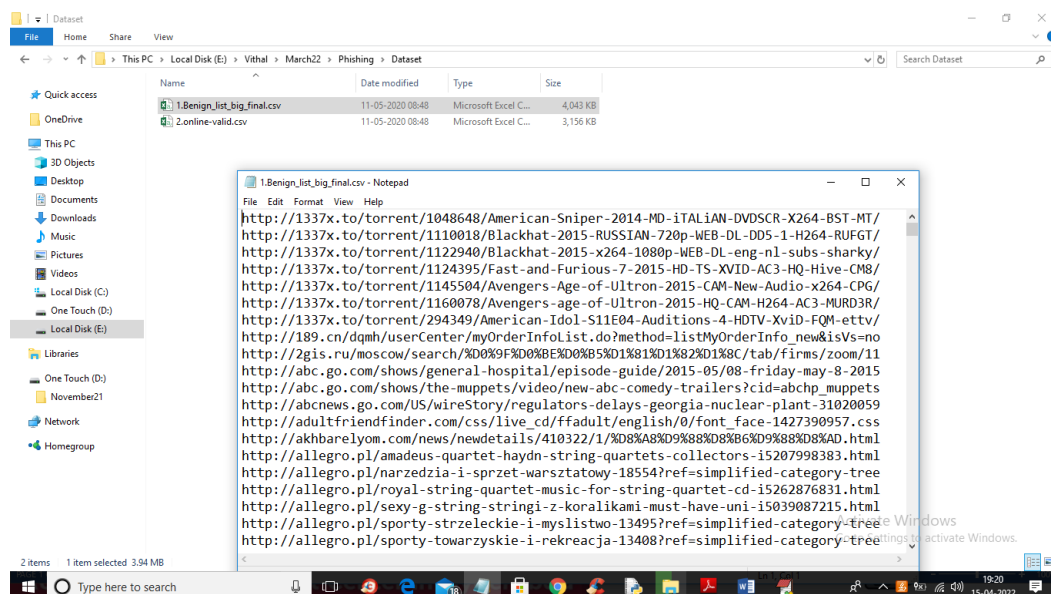
In this project we are implementing SVM and Light GBM machine learning algorithms to detect phishing website URLs. We are training all these algorithms with normal and phishing URLs and build a trained model and this trained model will be applied on new TEST URL to detect whether it's normal or phishing URL.

In this project you asked to use UCI machine learning phishing dataset but this dataset contains only 0's and 1's values like below screen



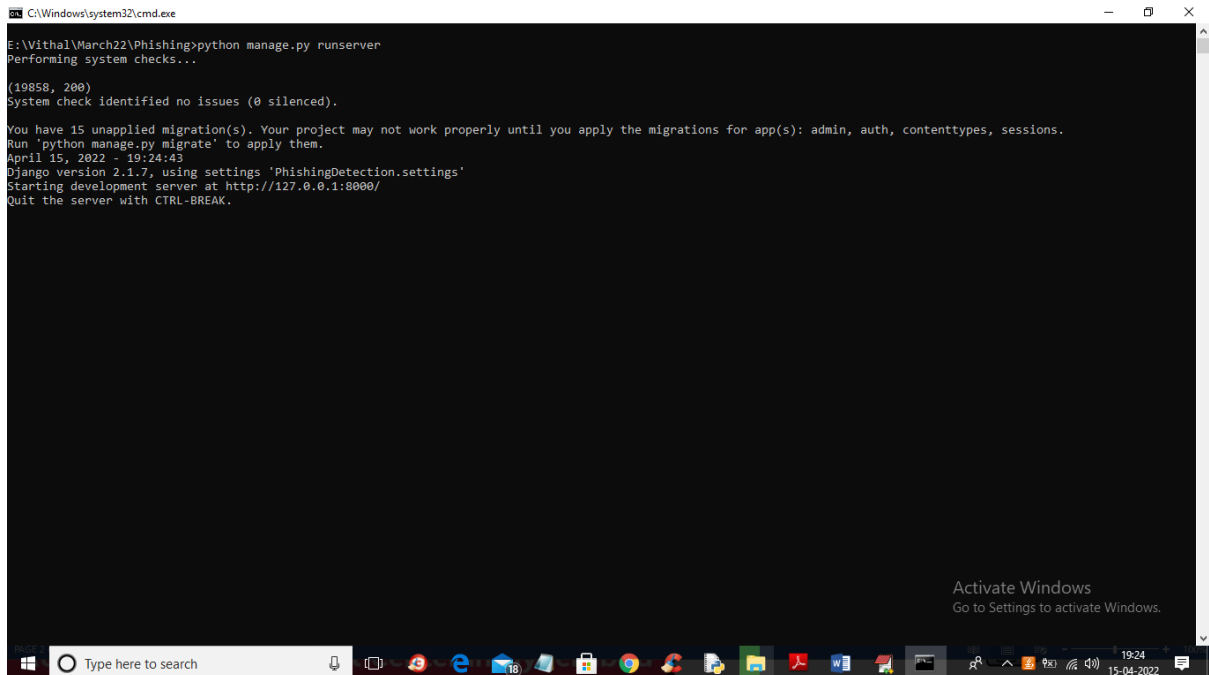
```
28 @attribute web_traffic {-1,0,1}
29 @attribute Page_Rank {-1,1}
30 @attribute Google_Index {1,-1}
31 @attribute Links_pointing_to_page {1,0,-1}
32 @attribute Statistical_report {-1,1}
33 @attribute Result {-1,1}
34
35
36 @data
37 -1,1,1,1,-1,-1,-1,-1,1,1,-1,-1,-1,-1,-1,0,1,1,1,1,-1,-1,-1,-1,1,1,-1,1
38 1,1,1,1,1,-1,0,1,-1,1,1,-1,-1,0,-1,-1,1,1,0,1,1,1,1,-1,-1,0,-1,1,1,1,-1
39 1,0,1,1,1,-1,-1,-1,1,1,1,-1,1,0,-1,-1,-1,0,1,1,1,1,1,-1,-1,1,0,-1,-1
40 1,0,1,1,1,-1,-1,-1,1,1,1,-1,-1,0,0,-1,1,1,0,1,1,1,1,-1,-1,-1,1,-1,1
41 1,0,-1,1,1,-1,1,1,-1,1,1,1,1,0,0,-1,1,1,0,-1,1,-1,-1,-1,-1,0,-1,1,1
42 -1,0,-1,1,-1,1,1,-1,1,1,1,1,0,0,-1,-1,-1,0,1,1,1,1,1,-1,-1,-1,1,-1
43 1,0,-1,1,1,-1,-1,-1,1,1,1,1,-1,1,0,-1,-1,-1,0,1,1,1,1,-1,-1,1,0,-1,-1
44 1,0,1,1,1,-1,-1,-1,1,1,1,-1,-1,0,-1,-1,1,0,1,1,1,1,-1,-1,0,-1,1,-1
45 1,0,-1,1,1,-1,1,1,-1,1,1,1,-1,1,0,1,-1,1,1,0,1,1,1,1,-1,1,1,0,1,1
46 1,1,1,1,1,-1,-1,-1,1,1,1,1,0,1,-1,1,1,0,1,1,1,1,-1,-1,0,-1,1,0,-1
47 1,1,1,1,1,-1,0,1,1,1,1,-1,1,0,0,-1,-1,-1,0,1,1,1,1,-1,1,1,1,-1,-1
48 1,1,-1,1,1,-1,-1,-1,1,1,1,-1,-1,-1,-1,-1,0,1,1,1,1,-1,-1,-1,0,-1,-1
49 -1,1,-1,1,-1,0,0,1,1,1,-1,-1,-1,1,1,0,-1,-1,1,1,-1,-1,-1,0,-1,-1
50 1,1,-1,1,1,-1,0,-1,1,1,1,-1,-1,-1,-1,1,0,1,1,1,1,-1,-1,0,-1,1,1,-1
51 1,1,-1,1,1,1,-1,1,1,-1,1,0,1,1,1,0,1,1,1,1,1,-1,-1,1,-1,1,1,1
52 1,-1,-1,1,-1,0,0,1,1,1,-1,-1,0,-1,1,0,1,1,1,1,1,-1,-1,1,0,1,-1
```

From above dataset ML algorithms can get trained but we can't understand anything so I am using REAL WORLD URL dataset which contains normal and phishing URLs like below screen



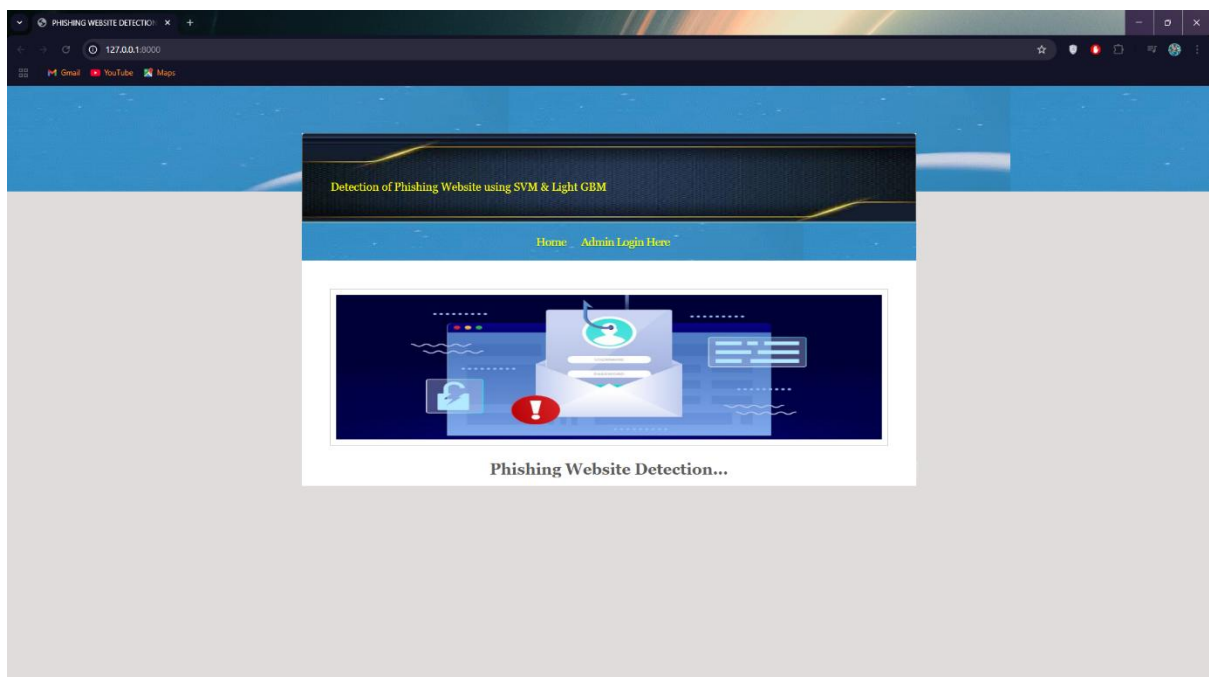
In above screen you can see our dataset contains 2 folders called benign (phishing URLs) and valid (normal URL) and this are real world URLs and we will train all algorithms with above dataset and then when we input any test URL then ML model will predict as normal or phishing

To run this project double click on 'run.bat' file to start python DJANO server like below screen

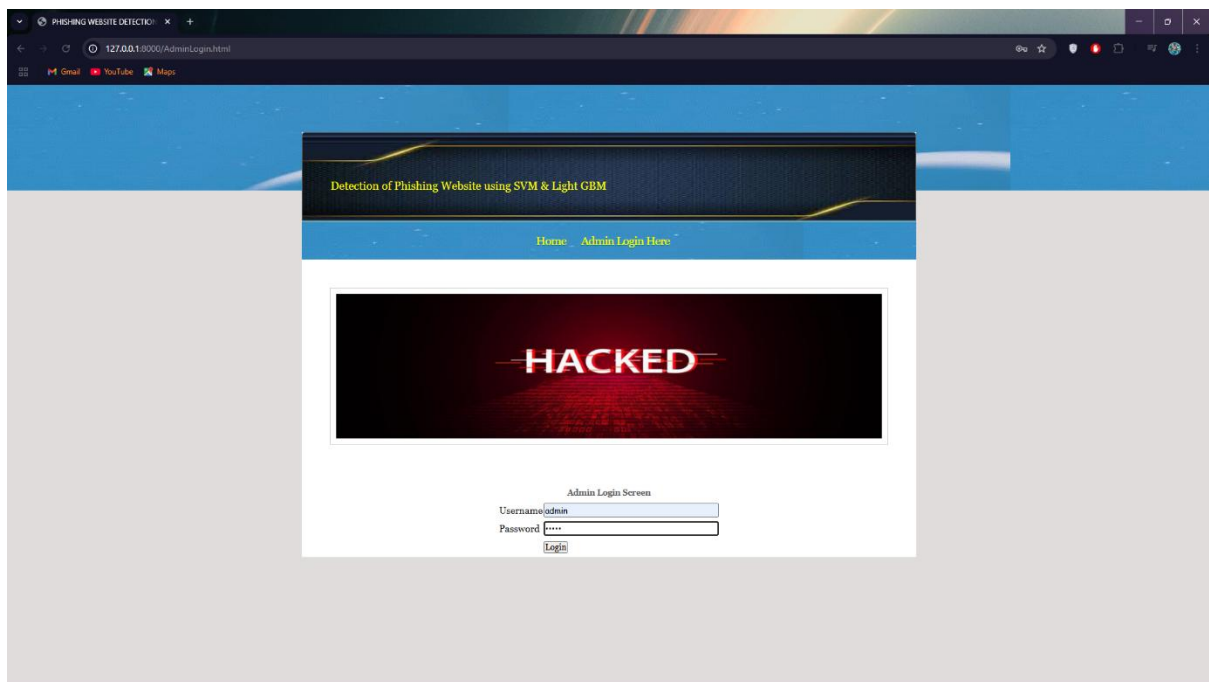


```
C:\Windows\system32\cmd.exe
E:\Vithal\March22\Phishing>python manage.py runserver
Performing system checks...
(19858, 200)
System check identified no issues (0 silenced).
You have 15 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
April 15, 2022 - 19:24:43
Django version 2.1.7, using settings 'PhishingDetection.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

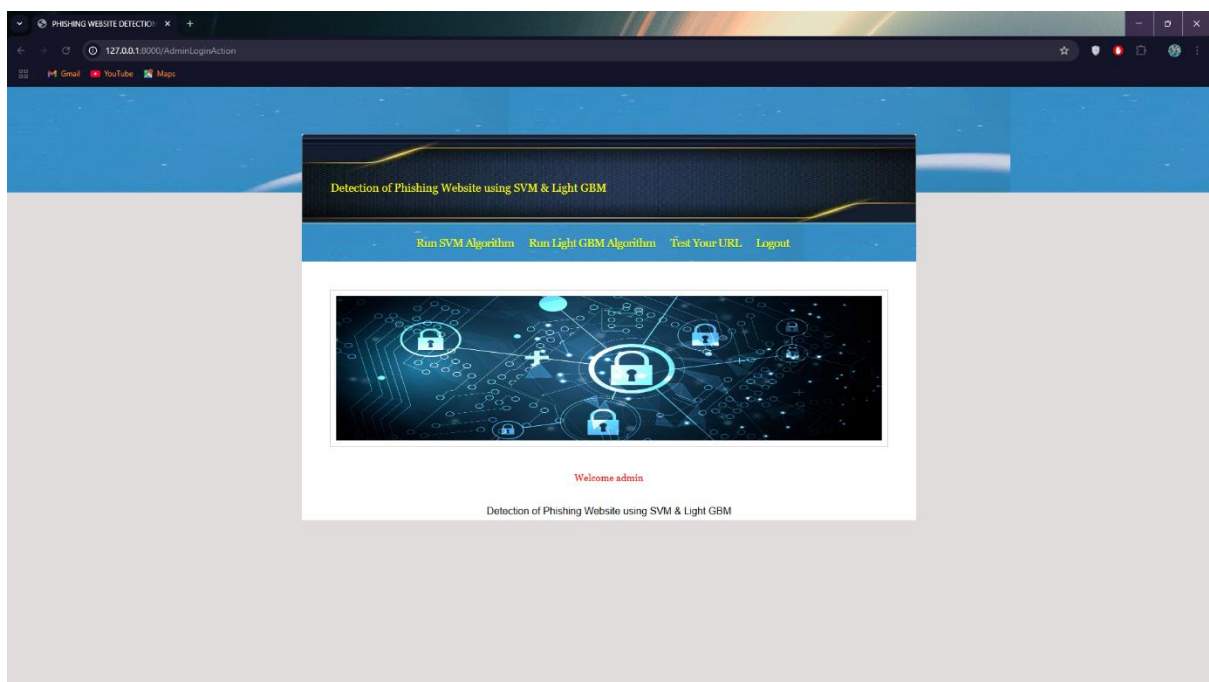
In above screen DJANGO webserver started and now open browser and enter URL <http://127.0.0.1:8000/index.html> and press enter key to get below output



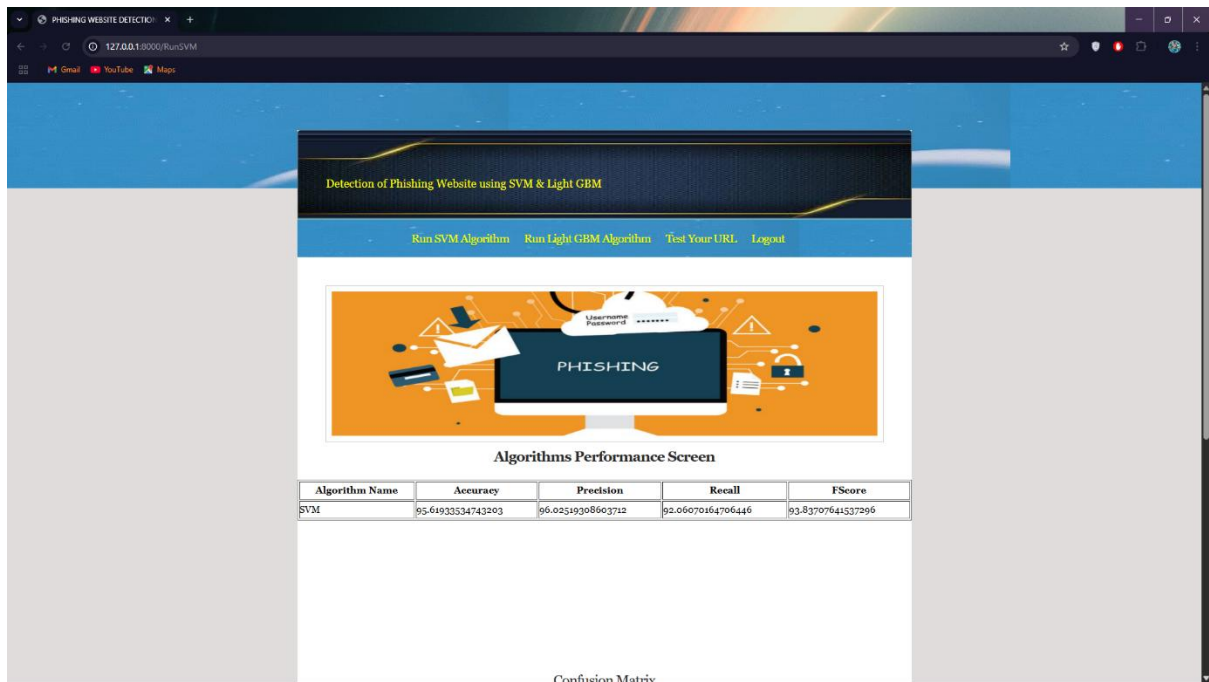
In above screen click on 'Admin Login Here' link to get below login screen



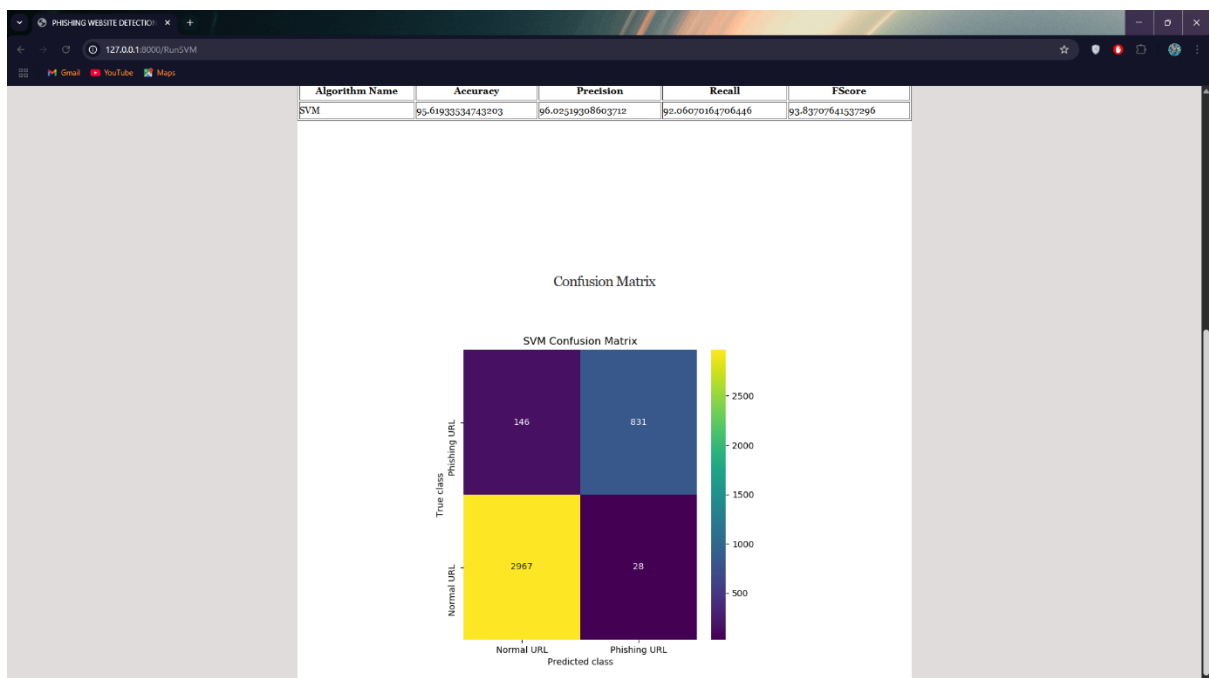
In above screen enter username and password as 'admin' and 'admin' and then press button to get below output



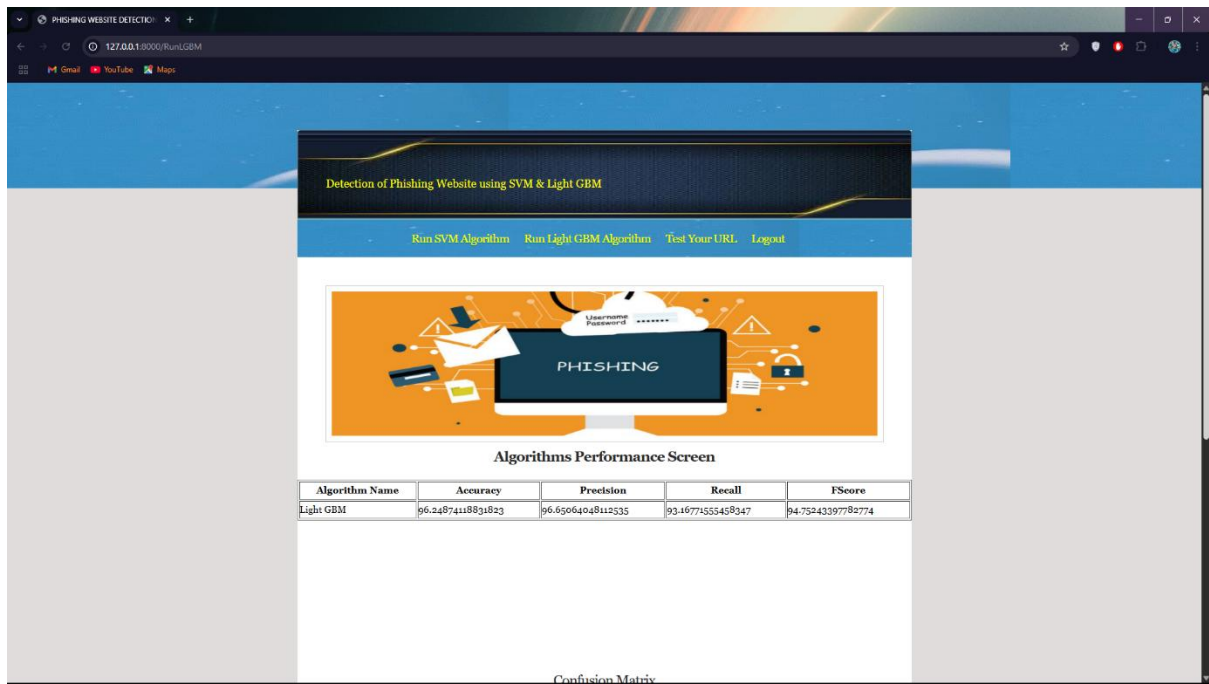
In above screen click on 'Run SVM Algorithm' link to train SVM algorithm and get below output



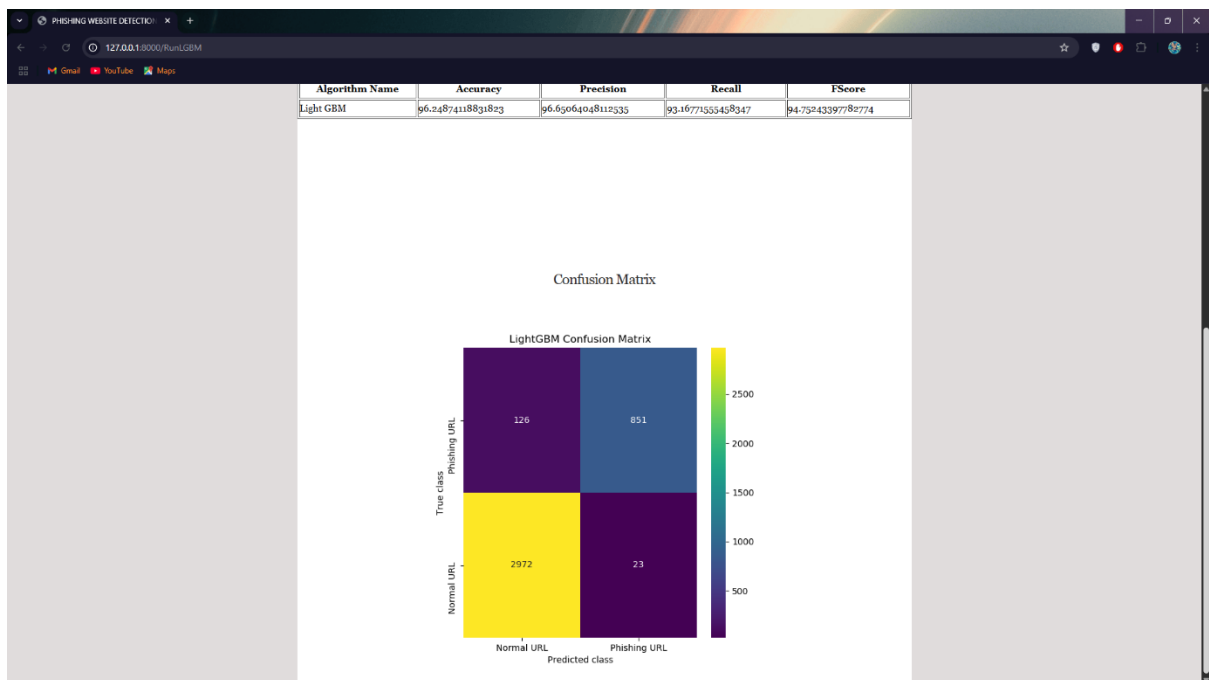
In above screen we can see SVM confusion matrix where x-axis represents predicted class and y-axis represents TRUE class and we can see SVM predict 2977 records correctly as NORMAL and only 145 are incorrect prediction and it predict 824 records as PHISHING URL and only 26 are incorrect prediction and now close above graph to get below output



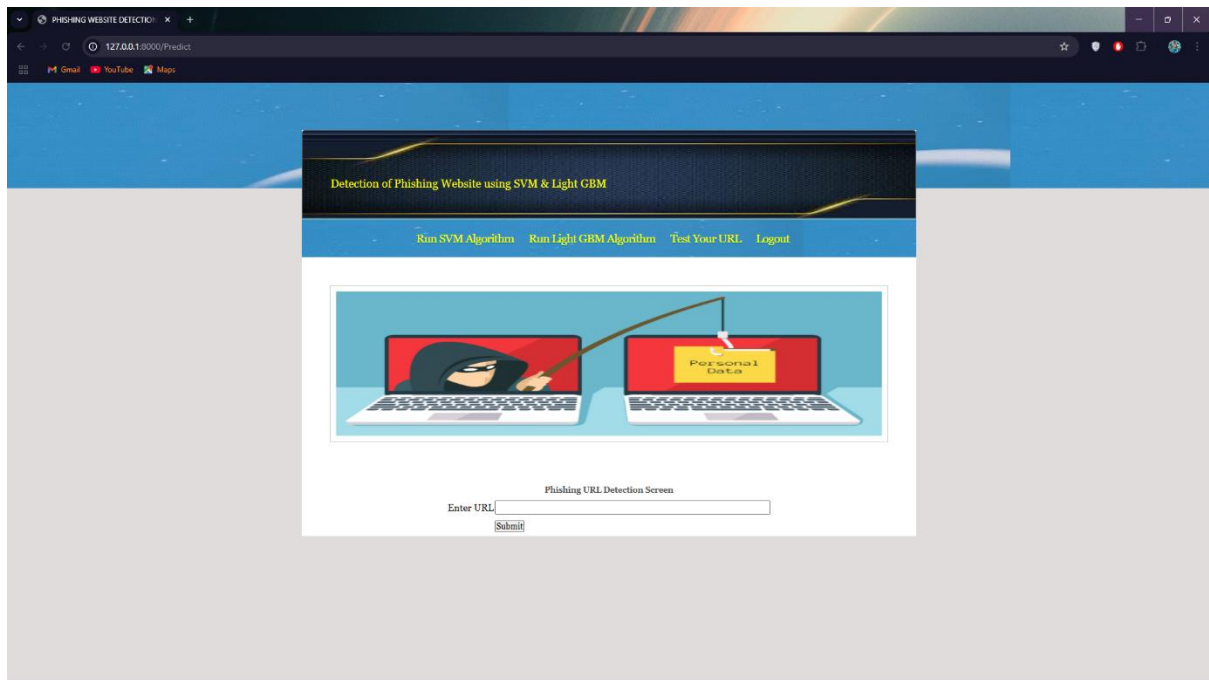
In above screen with SVM we got 95% accuracy and now click on 'Run Light GBM Algorithm' link to get below output



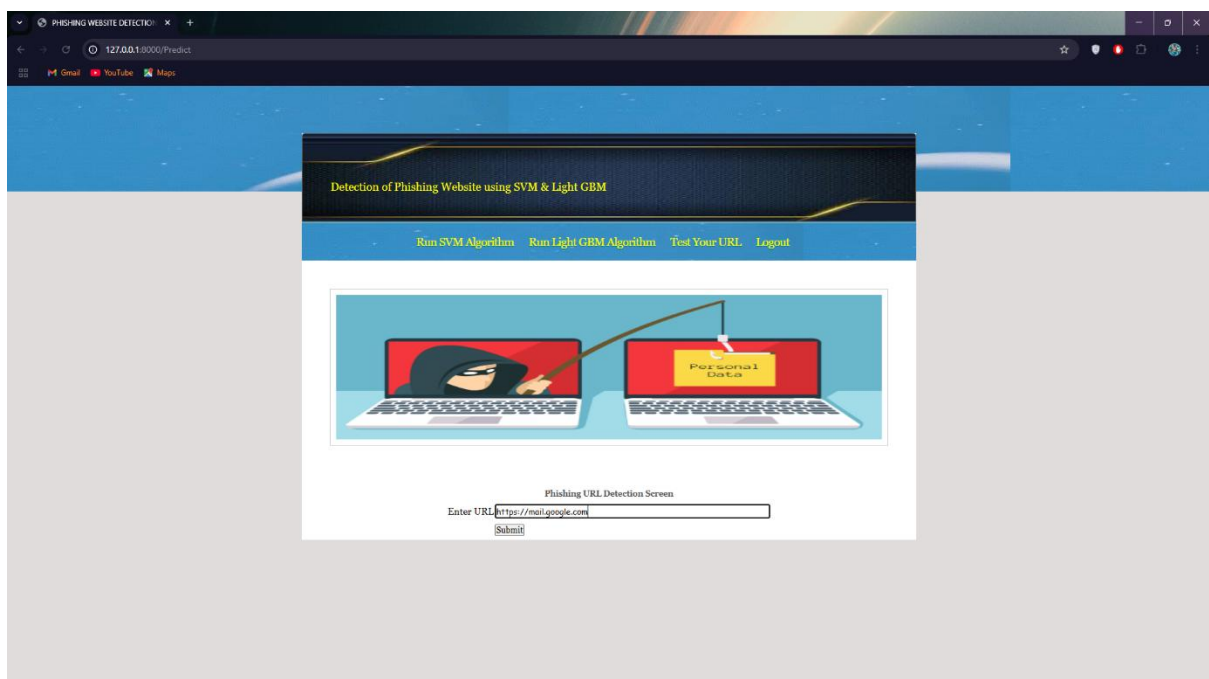
In above screen we can see Decision Tree confusion matrix graph and now close above graph to get below output



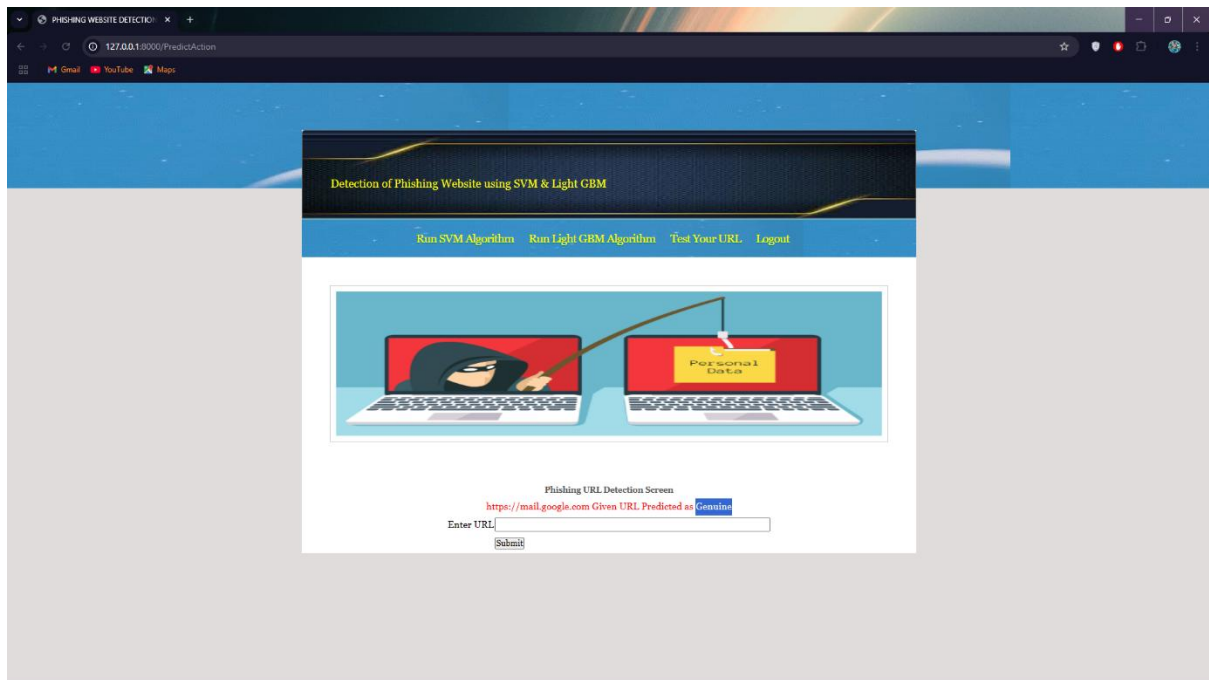
In above screen with Light GBM also we got 96% accuracy and now click on 'Test Your URL' link to get below screen



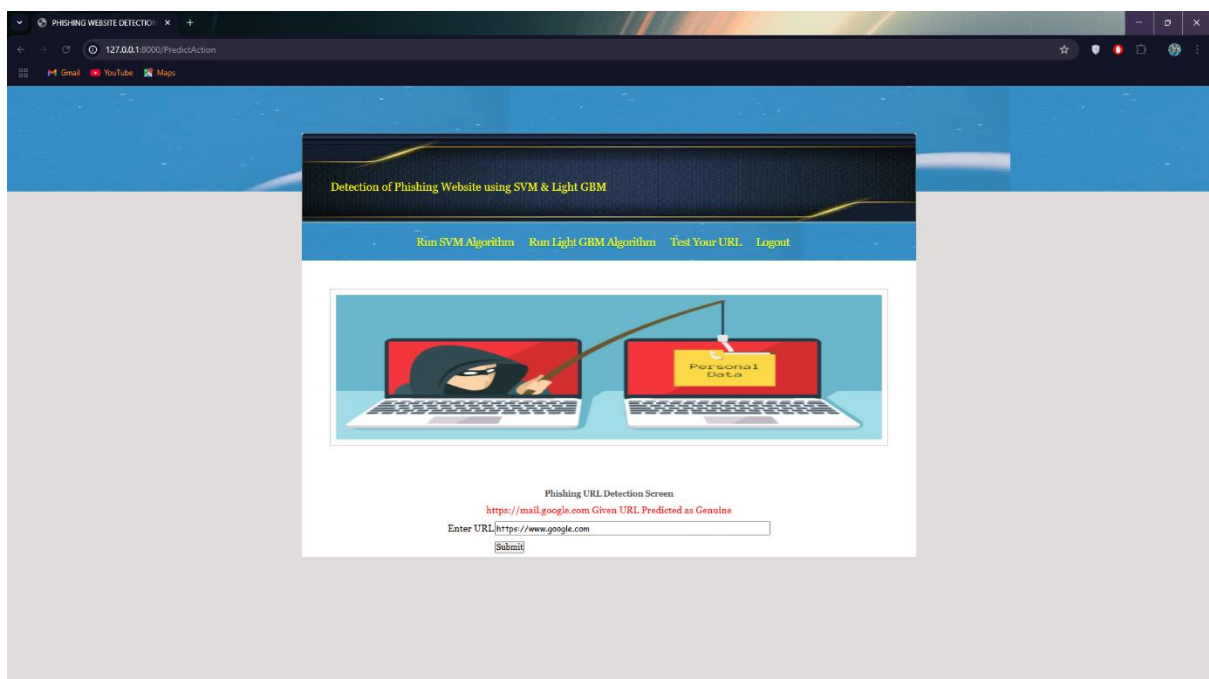
In above screen enter any URL and then press button and then Light GBM will predict whether that URL IS normal or phishing



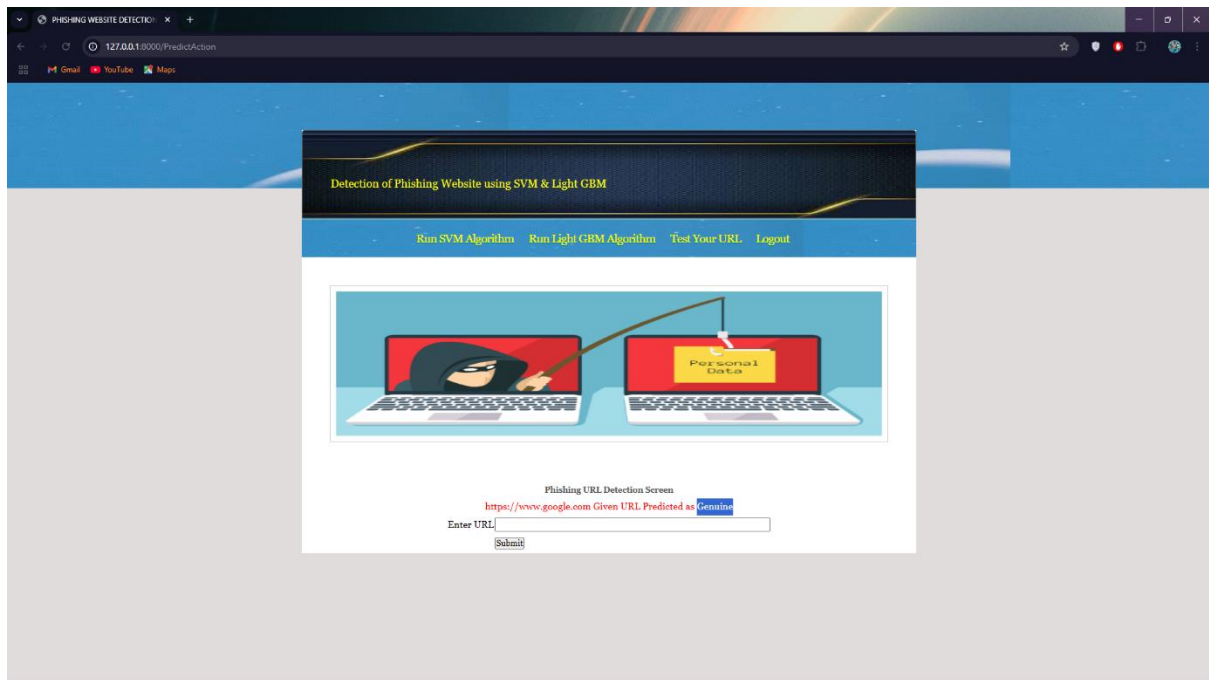
In above screen I entered URL as <https://mail.google.com> and then press button to get below output



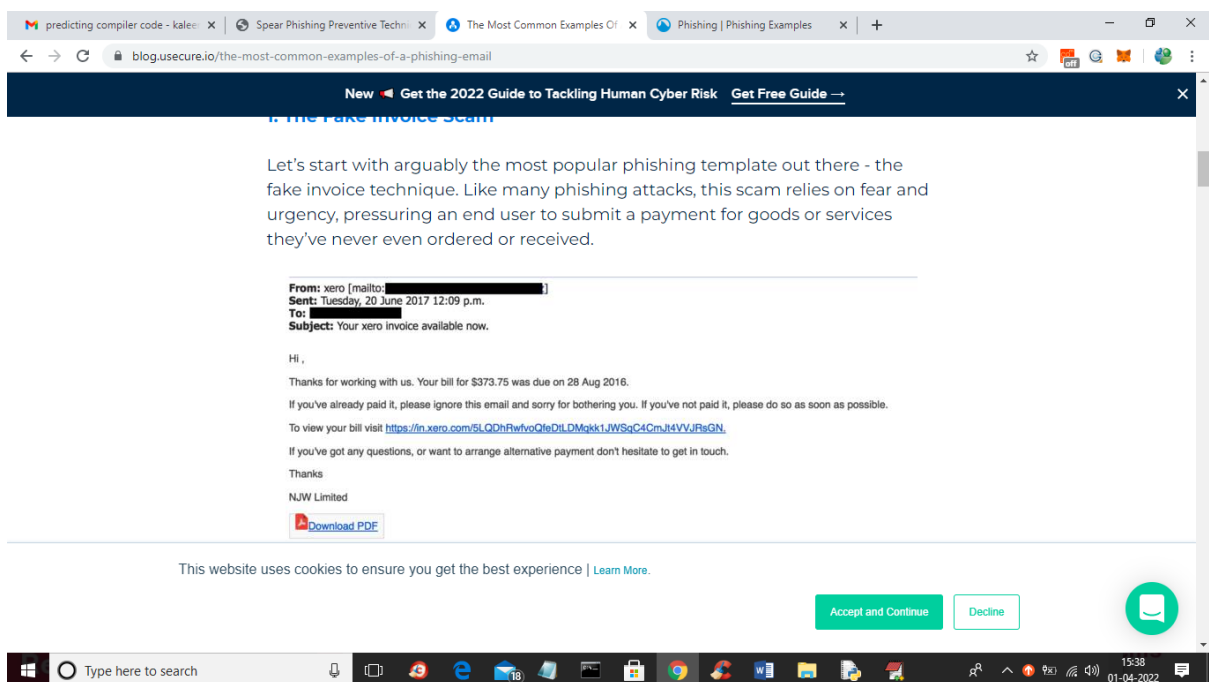
In above screen in blue colour text we can see given URL predicted as GENUINE (normal) and now test other URL. Similarly now I will enter Google.com in below screen



In above screen I gave URL as Google.com and below is the output

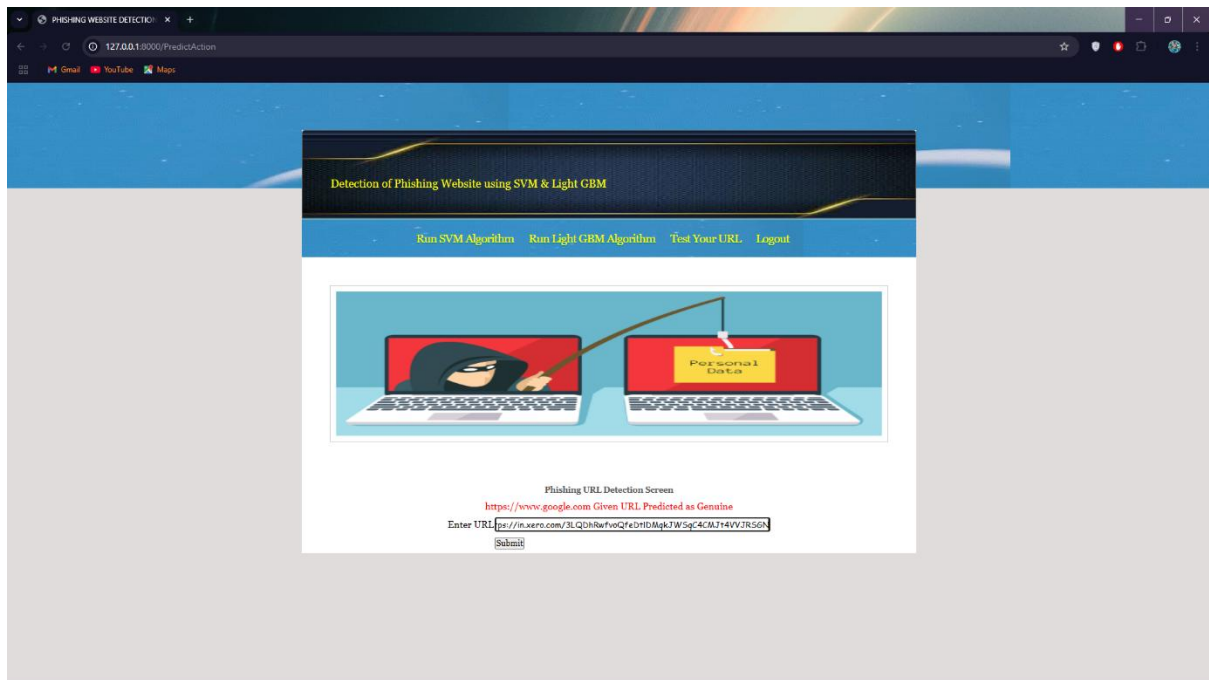


In above screen Google.com also predicted as Genuine. Now in below screen from internet I am taking one phishing URL and then input to my application to get prediction

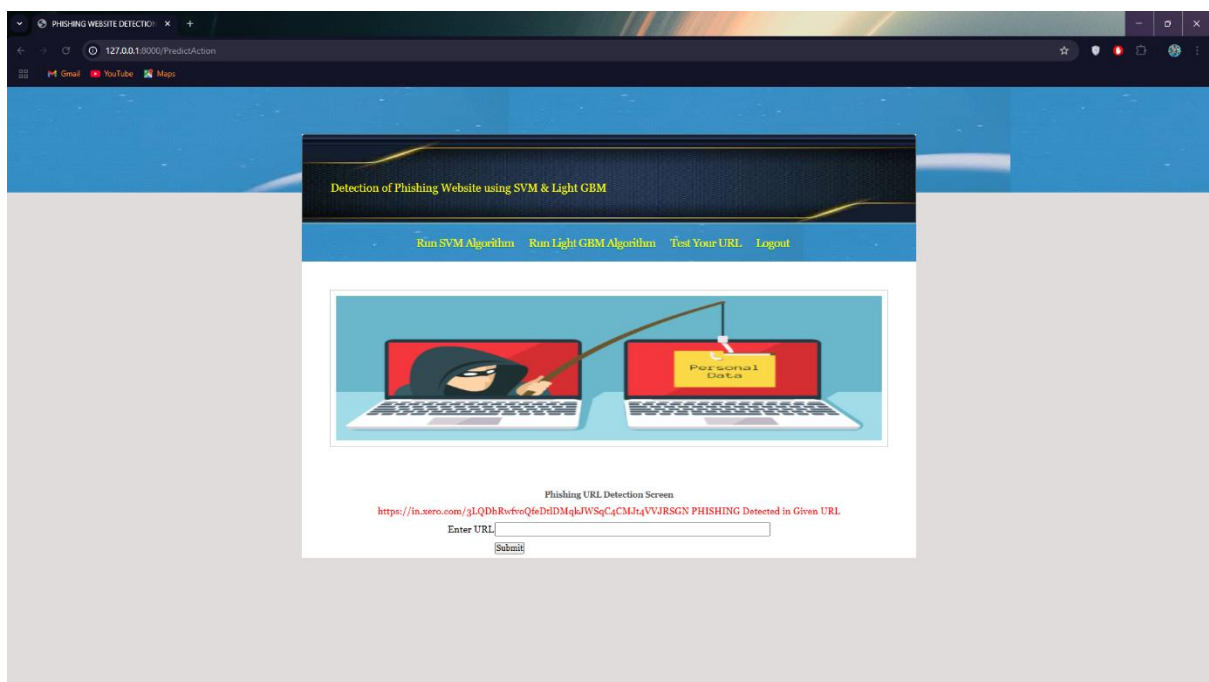


In above screen blue colour URL is the phishing URL and I will input that to my application in below screen and below is the phishing URL from internet

‘<https://in.xero.com/5LQDhRwfvQfeDtlDMqkk1JWSqC4CMJt4VVJRsGN>’



In above screen I entered same URL and press button to get below output



In above screen in blue colour text we can see application detected PHISHING in given URL and similarly you can enter any URL and detect it as NORMAL or phishing

CHAPTER 9
CONCLUSION
&
REFERENCES

Conclusion

The proposed system for detecting phishing websites using a combination of Support Vector Machine (SVM) and Light Gradient Boosting Machine (LightGBM) represents a significant advancement in the fight against online phishing threats. By leveraging the strengths of these two powerful machine learning algorithms, the system effectively addresses several limitations inherent in traditional phishing detection methods. Through an intelligent hybrid approach, it is able to offer improved accuracy, scalability, adaptability, and real-time performance—crucial factors in successfully identifying and mitigating phishing attacks.

The integration of SVM allows for precise classification of phishing websites, especially in cases with smaller datasets or when distinguishing between closely related features. SVM's ability to create optimal decision boundaries ensures high detection accuracy. Meanwhile, LightGBM complements this by efficiently handling large-scale, complex datasets and imbalanced classes, which are typical in phishing detection tasks. The gradient boosting mechanism employed by LightGBM enables the model to learn from errors and refine its predictions, significantly enhancing the system's ability to detect emerging phishing techniques.

One of the primary strengths of the proposed system is its adaptability. The combination of SVM and LightGBM allows the model to continuously evolve and stay current with new phishing tactics. As phishing strategies constantly change, the system can be retrained with new data to maintain high detection accuracy, reducing the risk of missing novel phishing threats. The hybrid system's real-time detection capability ensures that phishing sites are flagged quickly, preventing potential damage to users in real-time environments such as web browsers or security networks.

Feature Work

The feature work for the phishing website detection system using SVM and LightGBM involves several enhancements aimed at improving accuracy, adaptability, and efficiency. One key area is expanding the feature set by incorporating additional data sources such as user behavior analytics, social media signals, and network-based attributes like DNS resolution times and geolocation of IP addresses. Integrating these diverse data sources could enable the system to better understand the nuances of modern phishing tactics, improving detection accuracy. Furthermore, advanced feature engineering techniques, such as natural language processing for content analysis and image recognition to detect visual inconsistencies, could be utilized to extract more meaningful features that help distinguish phishing sites from legitimate ones.

Another important aspect of future work is enhancing the system's ability to adapt to evolving phishing techniques. This can be achieved through continuous learning, where the model is periodically retrained with fresh, real-world phishing data to stay updated on emerging threats. Active learning strategies could be implemented to allow the system to query human experts for uncertain cases, ensuring that it remains effective in identifying new phishing patterns. Additionally, optimizing the system for real-time detection and reducing computational overhead by using model compression techniques will be essential for efficient deployment across various platforms. Combining these improvements will help create a more robust, scalable, and adaptive phishing detection system capable of addressing the dynamic nature of online threats.

References:

- Sundararajan, V., & Shankar, P. (2020). Phishing Website Detection Using Machine Learning Techniques: A Comprehensive Survey
- Moustafa, N., & Slay, J. (2015). Phishing Attack Detection and Prevention Systems: A Survey
- Kausar, R., & Haider, Z. (2018). A Hybrid Phishing Detection System using Machine Learning Algorithms
- Chandramohan, R., & Raj, S. (2020). LightGBM Based Phishing Website Detection System
- Ravi, K., & Murugan, S. (2020). Phishing Detection Using SVM and Ensemble Learning
- Tariq, M. I., & Raza, M. S. (2021). An Efficient Phishing Website Detection System Based on SVM and Deep Learning
- Alazab, M., & Choo, K. R. (2019). Phishing Website Detection Using Machine Learning: Challenges and Future Directions
- Hasan, M. S., & Ibrahim, S. (2021). Feature Selection and Classification Techniques for Phishing Website Detection
- Chandramohan, R., & Srinivasan, R. (2021). Phishing Website Detection with Hybrid Machine Learning Algorithms: A Comparative Study
- Vasudevan, S., & Bhavani, S. (2019). Detection of Phishing Websites Using SVM and Gradient Boosting Models