ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

Proactive Phishing Website URL Scanner

A Major Project report on

Major Project submitted to Anurag University in Partial fulfillment of the requirements for the award of the Degree of Bachelor of Technology in Artificial Intelligence/ Artificial Intelligence and Machine Learning

Submitted by
D R Dinesh Kumar 21EG507101
S. Sujeeth Reddy 20EG107145
B. Aditya 20EG107104

Under the Guidance of

T. Neetha
Assistant Professor

Department of Artificial Intelligence
School of Engineering
ANURAG UNIVERSITY
2020-2024

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

DEPARTMENT OF ARTIFICIAL INTELLIGENCE

CERTIFICATE

This is to certify that the project report titled Proactive Phishing Website URL Scanner is being submitted by D R Dinesh Kumar, S. Sujeeth Reddy, B. Aditya, bearing 21EG507101, 20EG107145, 20EG107104**,** in IV B.Tech I/II semester *Artificial Intelligence/ Artificial Intelligence and Machine Learning* is a record bonafide work carried out by them. The results embodied in this report have not been submitted to any other University for the award of any degree.

Student's Name	Signature's Signature
1. D R Dinesh Kumar	1.
2. S. Sujeeth Reddy	2.
3. B. Aditya	3.

Ms. T. Neetha
External Examiner

Dr. A. Mallikarjuna Reddy

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

ACKNOWLEDGEMENT

We owe our gratitude to Prof.S. Ramchandram , Vice-Chancellor, Anurag University, for extending the University facilities to the successful pursuit of our project so far and his kind patronage.

We acknowledge our deep sense of gratitude to Prof. Balaji Utla , Registrar, Anurag University, for being a constant source of inspiration and motivation.

We wish to record our profound gratitude Dr. V. Vijay Kumar, Dean – School of Engineering, for his motivation and encouragement.

We sincerely thank Dr. A. Mallikarjuna Reddy,Associate Professor and the Head of the Department of Artificial Intelligence, Anurag University, for all the facilities provided to us in the pursuit of this project.

We owe a great deal to our project coordinator Ms. T. Neetha, Assistant Professor, Department of Artificial Intelligence, Anurag University for supporting us throughout the project work.

We are indebted to our project guide Ms. T. Neetha, Assistant Professor, Department of Artificial Intelligence, Anurag University. We feel it's a pleasure to be indebted to our guide for his valuable support, advice, and encouragement and we thank him for his superb and constant guidance towards this project.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

CONTENTS

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

ABSTRACT

The proliferation of phishing attacks represents a critical challenge to cybersecurity, necessitating the development of advanced detection systems. Our project, "Phishing Website Detection Using Machine Learning," aims to address this challenge by leveraging sophisticated machine learning algorithms to meticulously analyze and distinguish phishing websites from legitimate ones. By systematically examining various features and patterns within web content, such as URL anomalies, use of secure protocols, and other distinctive markers, the project seeks to accurately identify and classify phishing attempts. The approach encompasses comprehensive data collection, meticulous preprocessing to enhance data quality, and the employment of diverse machine learning models tailored for optimal performance in real-time detection scenarios. This endeavor not only focuses on enhancing online security measures but also on ensuring user-friendly interaction to facilitate widespread adoption. Through the integration of advanced machine learning techniques and a keen focus on the dynamic nature of cyber threats, this project endeavors to contribute significantly to the proactive defense against phishing attacks, thereby bolstering the integrity and trustworthiness of online spaces.

A pivotal aspect of our methodology is the adoption of gradient boosting algorithms, a powerful ensemble learning technique renowned for its effectiveness in handling complex and nonlinear data. By integrating gradient boosting into our analysis, we significantly improve the model's ability to learn from and adapt to the intricacies of phishing website characteristics, ensuring a robust detection mechanism. This advanced algorithm iteratively corrects errors from previous models and combines weak predictors to form a strong predictive model, offering unparalleled accuracy in real-time phishing detection. The choice of gradient boosting reflects our commitment to employing cutting-edge technology to tackle the dynamic and evolving nature of cyber threats, balancing detection sensitivity with minimal false positives to ensure a seamless web experience for users.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

LIST OF FIGURES

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

LIST OF TABLES

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

SYMBOLS AND ABBREVIATIONS

Symbols and Abbreviations	Description
API	Application Program Interface
GUI	Graphical User Interface
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
JSON	JavaScript Object Notation

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

CHAPTER ONE
INTRODUCTION

In the digital age, phishing attacks pose a severe threat to online security, necessitating advanced detection mechanisms. This project focuses on developing a machine learning-based solution for accurately identifying and classifying phishing websites. By leveraging sophisticated algorithms to analyze web content and adapt to evolving threats in real-time, the aim is to enhance cybersecurity measures and provide users with proactive defense against the ever-growing menace of phishing attacks.

A. Understanding URLs: Key Takeaways

In the vast and interconnected expanse of the internet, Uniform Resource Locators (URLs) serve as the essential navigational coordinates that guide users to their desired destinations. At its core, a URL is a specific type of Uniform Resource Identifier (URI) that provides the means to access information by specifying its location on the internet. This seemingly simple line of text is a cornerstone of web navigation, enabling the seamless connection between users and a myriad of digital resources, from websites and email addresses to files and services.

A URL is meticulously structured, comprising various components that offer clues about its destination's nature and security. The anatomy of a URL includes the protocol (such as HTTP or HTTPS), indicating how data between the browser and the web server should be transmitted the domain name, which identifies the site and is a key factor in assessing its legitimacy and the path, leading to specific resources on the server like web pages or images. Additional elements like query strings and fragments may further specify the type of content or section of the page the URL directs to.

Understanding URLs is crucial not only for navigating the internet with ease but also for safeguarding against cybersecurity threats. Phishing attacks, for example, often exploit subtle modifications in URLs to deceive users into visiting malicious sites. By familiarizing oneself with the structure and characteristics of URLs, users can develop a keen eye for spotting anomalies and protecting their digital identities from potential threats. This foundational knowledge empowers internet users to explore the digital world more safely and confidently, making informed decisions about the trustworthiness and authenticity of the web pages they visit.

B. Navigating the Deceptive Waters of Phishing Attacks: A Comprehensive Overview

Phishing attacks stand as a formidable vector for cyber threats, masterfully engineered to exploit human psychology and trust. These attacks cleverly masquerade as legitimate communications, often appearing to originate from well-known entities such as banks, social platforms, or even government bodies. The essence of phishing lies in its deceitful attempt to glean sensitive information from unsuspecting individuals, including passwords, financial data, or personal identification details. The method is simple yet alarmingly effective: attackers craft emails, text messages, or social media communications that bear a striking resemblance to authentic correspondence. These communications are designed to alarm or entice the recipient, presenting scenarios that demand immediate action, such as verifying account details or clicking on a link to address a supposedly urgent issue.

The sophistication of phishing attempts has grown significantly, with attackers continually refining their strategies to bypass awareness and security measures. These emails or messages often include malicious links or attachments that, once engaged with, can lead to the compromise of personal data or the installation of malware on the victim's device. The success of a phishing attack hinges on its presentation by convincingly mimicking the tone, style, and visual elements of legitimate entities. Attackers can create a veneer of credibility that persuades victims to act against their better judgment.

Understanding the nature of phishing attacks is crucial in the digital age, where information is as valuable as currency. The repercussions of falling prey to such attacks extend beyond individual losses to encompass broader implications for organizational security and privacy. As such, recognizing the signs of phishing and fostering a culture of caution and verification stands as a critical defense mechanism. By educating users on the subtleties of phishing schemes and encouraging a healthy skepticism of unsolicited requests for sensitive information, both individuals and organizations can bolster their defenses against this pervasive cyber threat. In navigating the complexities of the digital landscape, awareness and vigilance are key allies in thwarting the efforts of phishing attackers and protecting the sanctity of digital information.

C. The Mechanics of Phishing Attacks: A Closer Look

Phishing attacks, a prevalent form of cyber deception, exploit human factors to breach personal and organizational security barriers. These attacks are meticulously designed to trick individuals into divulging confidential information, leveraging the thin veneer of legitimacy to ensnare unsuspecting victims. Understanding how phishing attacks are orchestrated is crucial for developing effective countermeasures and fostering a secure digital environment.

At their core, phishing attacks follow a deceptive yet highly strategic process. Initially, attackers identify their target audience, which could range from individual users to employees within specific organizations. The goal is to gather enough information about the targets to craft convincing messages that resonate on a personal or professional level. This preparatory phase often involves

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

collecting email addresses, names, and any relevant data that can be used to personalize the attack, making the fraudulent communication seem all the more legitimate.

The next step involves creating the phishing content itself. Attackers design emails, websites, or social media messages that mimic the look and feel of legitimate sources. These messages often include logos, language, and formatting that are remarkably similar to those used by trusted entities, such as financial institutions, tech companies, or government agencies. The content typically induces a sense of urgency or fear, prompting the recipient to act swiftly—be it clicking on a link, downloading an attachment, or directly providing sensitive information like passwords or credit card numbers.

The delivery of phishing attacks hinges on the use of electronic communication, most commonly email. Advanced techniques, such as spoofing email addresses, make these messages appear to come from legitimate sources, further blurring the lines between authenticity and fraud. Upon engaging with the phishing content, victims might be directed to counterfeit websites where their information is harvested, or they might inadvertently download malware, providing attackers with unauthorized access to their devices and networks.

Then the culmination of a phishing attack sees the attacker leveraging the stolen information for malicious purposes, ranging from financial theft and identity fraud to launching further cyberattacks against larger networks. The sophistication and apparent legitimacy of these attacks make them notoriously difficult to detect and prevent, emphasizing the need for continuous education, vigilance, and advanced cybersecurity measures.

In essence, the mechanics of phishing attacks reveal a calculated exploitation of trust and the human tendency to respond to urgent requests. By dissecting the stages of these attacks, individuals and organizations can better prepare themselves to identify and counteract these cyber threats, safeguarding their information in an increasingly interconnected world.

D. Decoding Gradient Boosting: A Deep Dive into Enhanced Machine Learning

Gradient boosting is a powerful machine learning technique that has gained significant attention for its ability to produce highly accurate models. Part of the ensemble learning family, gradient boosting improves prediction accuracy by combining the outcomes of multiple weaker models to form a strong predictive model. This method works by sequentially adding predictors to an ensemble, each correcting its predecessor, thus incrementally improving the model's accuracy.

The process begins with a basic model that makes predictions on the dataset. The algorithm then assesses where this initial model went wrong, focusing on improving the predictions in areas of greatest error. Subsequent models are added, each tasked with correcting the errors identified by the ensemble thus far. This iterative process continues until a specified number of models are built or improvements become negligible, resulting in a final model that robustly predicts outcomes.

A key element of gradient boosting is its use of the gradient descent algorithm, an optimization technique used to minimize the loss, or error, by adjusting the model parameters. The loss function quantifies how far the predictions deviate from the actual results, guiding the algorithm in adjusting the model weights to minimize this discrepancy.

Gradient boosting is versatile, applicable to both regression and classification problems, and capable of handling various types of input data. Despite its strengths, the technique requires careful tuning of parameters, such as the number of trees in the model and the depth of each tree, to prevent overfitting. Overfitting occurs when the model is too complex, capturing noise in the training data that hinders its performance on new data.

This machine learning approach is computationally intensive, particularly as the number of models in the ensemble grows. However, the trade-off in computational demand is often justified by the significant improvement in prediction accuracy. Gradient boosting ability to navigate complex datasets and uncover intricate patterns makes it a valuable tool in the data scientist's toolkit, contributing to advancements in various fields from financial forecasting to medical diagnosis.

E. Unveiling the Tactics: Pioneering Phishing Attacks Through URLs

Phishing attacks have long been a staple in the cybercriminal's arsenal, exploiting human trust and curiosity to gain unauthorized access to personal and sensitive information. Among the myriad of techniques employed, the manipulation of URLs stands out as a particularly insidious method. This strategy involves crafting URLs that, at first glance, appear benign or even identical to those of legitimate websites. By pioneering the use of deceptive URLs, attackers have elevated phishing to a highly effective form of digital deception, capitalizing on the subtle nuances of web addresses to dupe unsuspecting victims.

The process typically begins with the selection of a target—often a well-known financial institution, social media platform, or email service provider. Attackers meticulously create URLs that mimic those of the targeted entity, employing tactics such as typosquatting, where the URL contains slight misspellings that are easily overlooked, or domain spoofing, which involves using a domain name that closely resembles the legitimate one, but with minor alterations such as substituting letters or adding additional characters.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

These deceptive URLs are then embedded in emails, text messages, or social media posts that purport to be from the legitimate entity. The messages are designed to alarm or entice the recipient, suggesting that immediate action is required—be it updating account information, verifying login details, or claiming a reward. Clicking on the fraudulent URL directs the victim to a phishing site, a convincing replica of the genuine website, where any data entered, such as usernames, passwords, or credit card numbers, is captured by the attackers.

The sophistication of URL-based phishing attacks lies in their ability to bypass initial scrutiny. Modern web users are often advised to check URLs for legitimacy before clicking, but the deliberate subtlety of these malicious URLs challenges even the most vigilant. The use of secure HTTPS connections and padlock icons in phishing sites further complicates this issue, lending an unwarranted layer of credibility to these fraudulent web addresses.

Understanding the mechanics behind URL-based phishing attacks is crucial for both individuals and organizations striving to safeguard their digital domains. Awareness and education on the common traits of malicious URLs, coupled with the implementation of advanced security measures such as multi-factor authentication and regular security training, can significantly mitigate the risks associated with these deceptive tactics. As cybercriminals continue to refine their methods, staying informed about the evolving landscape of phishing attacks becomes a paramount defense strategy, ensuring that users can navigate the internet's vast resources with confidence and security.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

CHAPTER TWO
LITERATURE SURVEY

A. *Existing System*

The realm of phishing URL detection is marked by a diverse array of methodologies, each designed to combat the multifaceted threat posed by phishing attacks. Research in this area spans from employing traditional machine learning algorithms, such as Support Vector Machines (SVM) and Naïve Bayes, to leveraging more sophisticated techniques like Artificial Neural Networks (ANN) and the Bag of Words approach. These methodologies have been instrumental in advancing the field, offering varied perspectives and tools for identifying malicious URLs. However, despite their innovations, these approaches encounter several limitations that may compromise their effectiveness against the continuously evolving strategies of cyber adversaries.

➢ *Traditional Machine Learning Algorithms*

Machine learning algorithms like SVM and Naïve Bayes have been foundational in phishing URL detection, providing a statistical basis for distinguishing between benign and malicious URLs. These algorithms analyze URL features and web content to learn patterns associated with phishing. While they have proven effective in various contexts, their reliance on pre-defined feature sets may limit their adaptability to new phishing techniques that deviate from established patterns.

➢ *Advanced Techniques: ANN and Bag of Words*

Artificial Neural Networks (ANN) and the Bag of Words technique represent more complex approaches to phishing detection. ANNs mimic human brain functionality, offering the potential for deep learning models to identify subtle and complex patterns in data. The Bag of Words approach, on the other hand, analyzes the textual content of websites, transforming it into a format that machine learning models can process. These advanced techniques are promising for their ability to learn from vast amounts of data and detect phishing attempts with high accuracy.

➢ *Overview of Existing Methodologies*

Current research in phishing URL detection showcases a variety of methodologies, each bringing unique insights into identifying and neutralizing phishing threats. These methods range from traditional machine learning algorithms, like Support Vector Machines (SVM) and Naïve Bayes, to more complex Artificial Neural Networks (ANN) and Bag of Words techniques. Despite their contributions, these approaches often grapple with limitations, such as reliance on outdated models, which may not effectively counter new phishing strategies.

B. *Limitations of Existing System*

While the existing Proactive Phishing Website URL Scanner system provides robust protection against phishing attacks, it also has several limitations that should be acknowledged and addressed:

- False Positives And Negatives: Despite utilizing sophisticated detection techniques, the system may occasionally misclassify URLs, leading to false positives (legitimate URLs flagged as phishing) or false negatives (phishing URLs classified as safe). These inaccuracies can erode user trust and lead to unnecessary interruptions or security breaches.
- Dependency On Known Patterns: The system heavily relies on known phishing indicators and patterns for detection. This approach may be less effective against emerging or zero-day phishing attacks that do not conform to established patterns, potentially leaving users vulnerable to novel threats.
- Limited Detection Coverage: The effectiveness of the system is contingent upon the comprehensiveness and timeliness of its phishing databases and blacklists. If these resources are incomplete or outdated, the system may fail to detect newly established phishing websites or sophisticated phishing campaigns, exposing users to risks.
- Resource Intensive: Real-time URL analysis and phishing detection processes can be computationally intensive, requiring significant system resources and potentially impacting performance. This could result in delays in URL processing, user experience degradation, or increased operational costs, particularly during periods of high traffic or malicious activity.
- Over-Reliance On Technical Indicators: While the system analyzes technical aspects of URLs such as syntax, domain reputation, and SSL certificate validity, it may overlook contextual or behavioral cues that could indicate phishing attempts. Sophisticated attackers may exploit this limitation by crafting URLs that evade technical scrutiny but still deceive users through social engineering tactics.
- Lack Of User Education And Awareness: Despite providing warnings and notifications to users, the system may not adequately educate them about the risks associated with phishing or provide guidance on safe online behavior. Without proper awareness and training, users may inadvertently ignore or bypass warnings, increasing their susceptibility to phishing attacks.
- Scalability Challenges: As the volume of web traffic and the diversity of phishing threats continue to grow, the system may face scalability challenges in terms of processing capacity, storage requirements, and infrastructure scalability. Scaling the system to accommodate increasing demand while maintaining performance and reliability may necessitate significant investments in resources and technologies.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

C. Gaps Identified

Identifying gaps in the existing Proactive Phishing Website URL Scanner system is crucial for enhancing its effectiveness and resilience against evolving cyber threats. Here are some key gaps identified:

- Detection Accuracy Discrepancies: The system may misclassify URLs, leading to false positives (legitimate URLs flagged as phishing) or false negatives (phishing URLs classified as safe). Addressing this gap requires refining detection algorithms to minimize inaccuracies and improve overall detection accuracy.
- Limited Coverage and Timeliness: The system's effectiveness relies on the comprehensiveness and timeliness of its phishing databases and blacklists. However, gaps in coverage or delays in updating these databases can result in the system failing to detect newly established phishing websites or sophisticated phishing campaigns.
- Technical Indicators vs. Contextual Analysis: While the system analyzes technical aspects of URLs, it may overlook contextual or behavioral cues that could indicate phishing attempts. Enhancing the system with contextual analysis capabilities can help bridge this gap and improve detection accuracy against socially engineered phishing attacks.
- User Awareness and Education: Despite providing warnings and notifications, the system may not adequately educate users about phishing risks or promote safe online behavior. Bridging this gap requires integrating user awareness and education initiatives into the system to empower users to identify and respond to phishing threats effectively.
- Resource Efficiency and Scalability: Real-time URL analysis and phishing detection processes may strain system resources, impacting performance and scalability. Addressing this gap involves optimizing resource utilization and scalability to ensure efficient operation even during periods of high traffic or malicious activity.
- Adaptability to Emerging Threats: The system's reliance on known phishing indicators and patterns may make it less effective against emerging or zero-day phishing attacks. Enhancing the system's adaptability to rapidly evolving threats requires integrating advanced threat intelligence capabilities and proactive monitoring mechanisms.
- Integration with Security Ecosystem: The system may operate in isolation from other security tools and systems, limiting its ability to leverage synergies and share threat intelligence. Bridging this gap involves integrating the system with broader security ecosystems to enhance collaboration, threat response, and overall security posture.

D. Problem Statement

The increasing sophistication of phishing attacks poses a significant threat to online users, compromising sensitive information such as login credentials, personal details, and financial data. Traditional methods of detecting phishing websites often fall short of keeping pace with evolving phishing techniques. The challenge is to develop an intelligent and proactive solution that leverages machine learning algorithms to accurately identify and classify phishing websites, thereby enhancing cybersecurity measures for end-users.

Traditional approaches to phishing detection, which typically rely on blacklists and heuristic rules, are increasingly inadequate against the backdrop of these evolving threats. These methods struggle to adapt to the rapid pace at which phishing techniques change, often resulting in a reactive rather than proactive stance against attacks. The limitations of such approaches underscore the urgent need for more sophisticated and dynamic solutions.

E. Objectives

To address the identified gaps and enhance the effectiveness of the Proactive Phishing Website URL Scanner system, the following objectives are proposed:

- Enhance Detection Accuracy: Develop and implement advanced detection algorithms to minimize false positives and negatives, thereby improving the overall accuracy of phishing URL detection.
- Expand Coverage and Timeliness: Enhance the system's phishing databases and blacklists to ensure comprehensive coverage of known phishing websites and timely updates to detect emerging threats effectively.
- Integrate Contextual Analysis: Incorporate contextual and behavioral analysis capabilities into the system to complement technical indicators, enabling more robust detection of socially engineered phishing attacks.
- Promote User Awareness and Education: Integrate user awareness and education initiatives within the system to empower users with the knowledge and skills to recognize and respond to phishing threats effectively.
- Optimize Resource Efficiency and Scalability: Implement optimizations to enhance resource efficiency and scalability, ensuring that the system can operate efficiently even under high traffic or malicious activity conditions.
- Adapt to Emerging Threats: Enhance the system's adaptability to rapidly evolving phishing threats by integrating advanced threat intelligence capabilities and proactive monitoring mechanisms.
- Integrate with Security Ecosystem: Foster integration with broader security ecosystems to facilitate collaboration, threat intelligence sharing, and coordinated response efforts against phishing threats.
- Ensure Compliance and Privacy: Implement measures to ensure compliance with relevant regulations and standards governing cybersecurity and user privacy, thereby enhancing trust and accountability in the system's operation.
- Continuous Improvement and Evaluation: Establish processes for ongoing evaluation, feedback collection, and iterative improvement of the system's performance, features, and effectiveness in mitigating phishing threats.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

- User-Centric Design and Experience: Prioritize user-centric design principles to enhance the usability, accessibility, and overall user experience of the system, fostering user adoption and engagement.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

CHAPTER THREE
PROPOSED SYSTEMS

The proposed system for the Proactive Phishing Website URL Scanner is designed to enhance the detection and prevention of phishing attacks using advanced machine learning algorithms and comprehensive data analysis. The system aims to address the limitations of existing solutions by offering improved accuracy, broader coverage, and real-time detection capabilities.

A. Architecture

➢ *Project Architecture*

- **Data Collection Module:** Gathers URLs for analysis from various sources, including user inputs, web crawlers, and threat intelligence feeds.
- **Preprocessing Module:** Normalizes and cleans the collected URL data, preparing it for analysis. This includes extracting features such as domain name characteristics, URL length, use of secure protocols, and the presence of suspicious tokens.
- **Feature Engineering:** Utilize techniques to transform raw data into meaningful features, focusing on predictive attributes for phishing detection.
- **Machine Learning Module:** Utilizes a combination of machine learning algorithms to analyze the preprocessed URLs and classify them as phishing or legitimate. Algorithms like Random Forest, Support Vector Machines (SVM), and Deep Learning models (e.g., Convolutional Neural Networks) can be employed.
- **Real-Time Analysis Engine:** Offers on-the-fly analysis of URLs accessed by users, leveraging the trained machine learning models to provide instant classifications.
- **Alerts and Reporting System:** Generates notifications for users and administrators about detected threats, and compiles reports on system performance, detection accuracy, and threat trends.
- **Feedback Loop:** Collects user feedback and detection outcomes to continuously refine and improve the machine learning models and detection algorithms.

➢ *Model Architecture:*

- **Foundation Model:** The process starts with a base model, often a simple decision tree. This model makes initial predictions for all instances in the dataset.
- **Calculate Errors:** After the base model makes predictions, the algorithm calculates the errors or differences between the predicted and actual outcomes.
- **Build a New Model:** A new decision tree model is then built to predict the errors identified by the previous model. This step aims to correct the mistakes of the base model.
- **Combine Models:** The predictions from the new model are combined with the predictions from the previous model(s) to update the predictions closer to the actual values. This step may involve weighting the new model's contributions based on its performance.
- **Learning Rate:** The algorithm applies a learning rate to the contribution of each new model. A smaller learning rate requires more models to achieve high accuracy but can lead to better generalization.
- **Iterate:** Steps repeated for a predefined number of iterations or until further improvements become minimal. Each iteration aims to reduce the residual errors from the previous models.
- **Ensemble of Trees:** The final model is an ensemble of all the individual decision trees built during the iterations. Each tree corrects the errors of the ensemble of trees that came before it.
- **Prediction:** To make a prediction, the input is passed through all the trees, and their outcomes are combined. The ensemble's final output is the cumulative result of adding up all these contributions, which aims to predict the target variable accurately.
- **Handling Overfitting:** Techniques such as subsampling the data for each tree and applying regularization can help manage overfitting, ensuring the model generalizes well to unseen data.
- **Output:** The final output is a powerful model that combines the strengths of numerous simple models into a more accurate and robust classifier.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

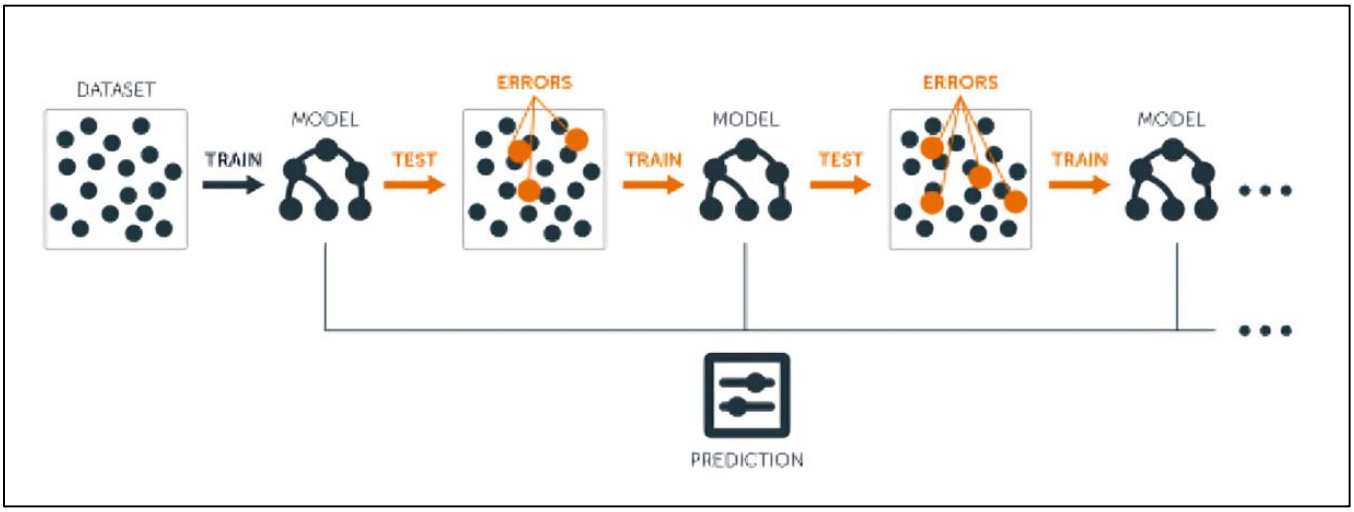

Fig 1: Project Architecture

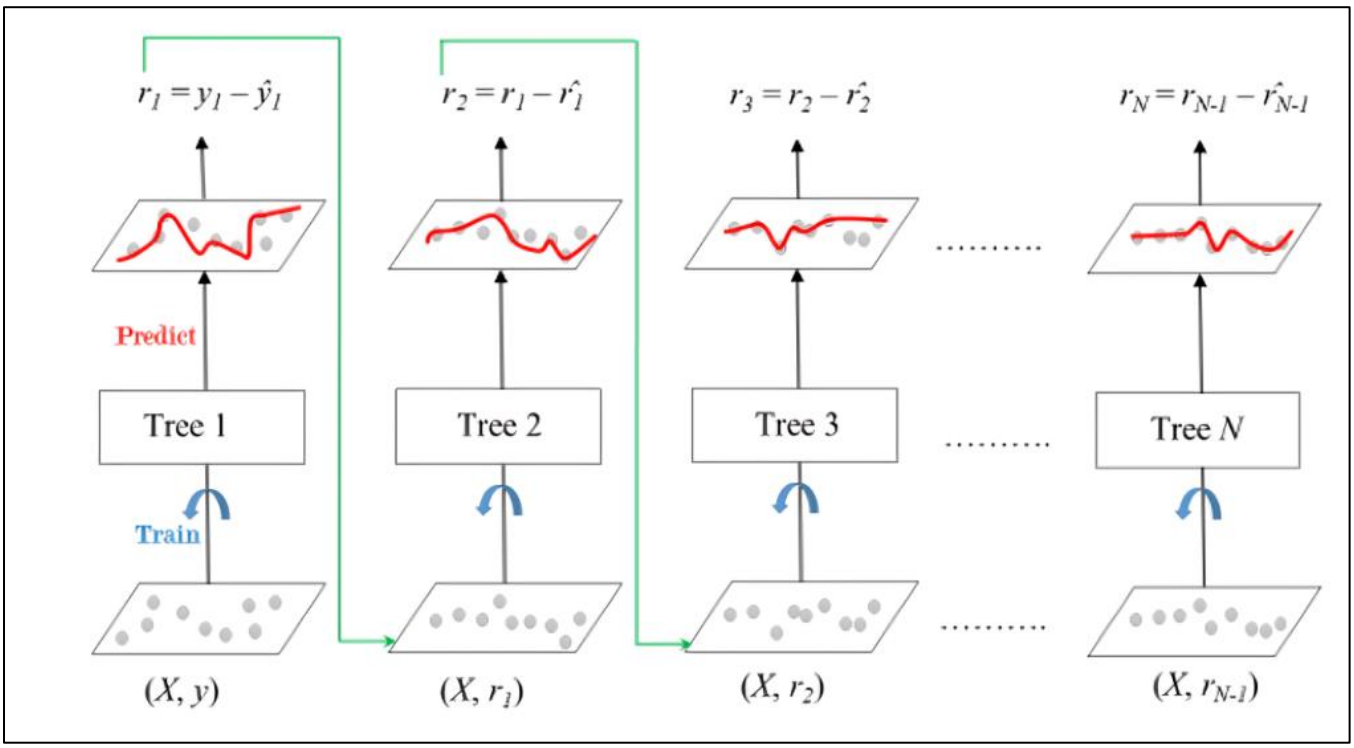

Fig 2: Gradient Boosting Algorithm Architectures

➢ *Training Process*

- **Initialize with a Base Model:** The training starts with the creation of a simple model, often a decision tree. This initial model is trained on the dataset to make the first set of predictions for the target variable.
- **Calculate Initial Errors:** After the base model has made predictions, the algorithm calculates the errors, which are the differences between the predicted values and the actual target values in the training data.
- **Sequential Model Training:** For each iteration after the first, the algorithm focuses on the errors made by the ensemble of all previous models. Then a new decision tree model is trained, but instead of predicting the actual target values, it predicts the errors (residuals) from the previous ensemble predictions.
- **Combine Models Predictions:** After training a new model on the residuals, its predictions are scaled by a factor known as the learning rate and then added to the previous models' predictions. This step updates the ensemble's predictions to be closer to the actual target values.
- **Apply Learning Rate:** The learning rate (a value between 0 and 1) scales the contribution of each new tree. A smaller learning rate means that each tree's predictions have less impact, requiring more trees to achieve high accuracy but often leading to a model that generalizes better.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

- **Iterate to Improve:** The process of training new models on the residuals and updating the ensemble's predictions is repeated for a predefined number of iterations or until improvements in prediction accuracy become negligible. Each iteration is designed to further reduce the residual errors.
- **Ensemble Creation:** The final Gradient Boosting Classifier model is an ensemble of all the decision trees trained during the iterations. The ensemble represents the cumulative corrections to the predictions, aiming to closely match the actual target values.
- **Prevent Overfitting:** Various strategies, such as limiting the depth of the decision trees, introducing regularization techniques, or using a subset of data for training each tree (stochastic gradient boosting), are employed throughout the training process to prevent the model from overfitting to the training data.
- **Model Completion:** At the end of the training process, the Gradient Boosting Classifier is a robust ensemble model capable of making accurate predictions on new, unseen data. The ensemble's final prediction for a given input is the sum of the base model's prediction and all the corrections made by the subsequent models.

➢ *Algorithms:*

- **Feature Selection:** Techniques like Principal Component Analysis (PCA) to identify the most relevant features for phishing detection.
- **Classification Algorithms:** Machine learning algorithms such as Random Forest, SVM, and Neural Networks for the classification of URLs.
- **Anomaly Detection:** Algorithms like Isolation Forest for identifying outliers or anomalies that might indicate novel phishing threats.

➢ *Methods:*

- **Continuous Learning:** Implement a continuous learning system where the model is periodically retrained with new data to adapt to evolving phishing techniques.
- **User Feedback Incorporation:** Integrate a mechanism for users to report false positives/negatives, contributing to the dataset and refining the model's accuracy.

B. Requirements & Specifications

➢ *Client Requirements:*

- **Accessibility:** The scanner must be easily accessible to users, possibly through browser extensions, mobile applications, or web services.
- **Ease of Use:** Interface should be user-friendly, with clear options for scanning URLs and understanding alerts.
- **Real-Time Alerts:** Users should receive immediate notifications about the risk associated with the URLs they attempt to access.

➢ *Software Requirements:*

Software Requirements specify the logical characteristics of each interface and software components of the system. The following are some software requirements.

Software	Description
Python	Programming language for system development
Scikit-Learn	It is the most useful and robust library for machine learning in Python
Flask	It is a web framework, it's a Python module that lets you develop web applications easily.
Visual Studio Code	Visual Studio Code, also commonly referred to as VS Code, is a source-code editor developed by Microsoft for Windows, Linux and macOS.
Git	Git is a distributed version control system that tracks changes in any set of computer files, usually used for coordinating work among programmers who are collaboratively developing source code during software development.

➢ *Hardware Requirements:*

Hardware interfaces specify the logical characteristics of each interface between the software product and the hardware components of the system. The following are some hardware requirements.

Hardware	Description
CPU/GPU	Computational resources for model training Minimum 8GB
Memory	RAM for processing large datasets and model training 16GB And Above
Storage	Solid State drive (SSD) for storing historical data, trained model weights, and application files

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

CHAPTER FOUR
DESIGN

A. DFD (Data Flow Diagram):

Illustrates the flow of information and the processes involved in the system, such as URL collection, preprocessing, analysis, and alert generation.

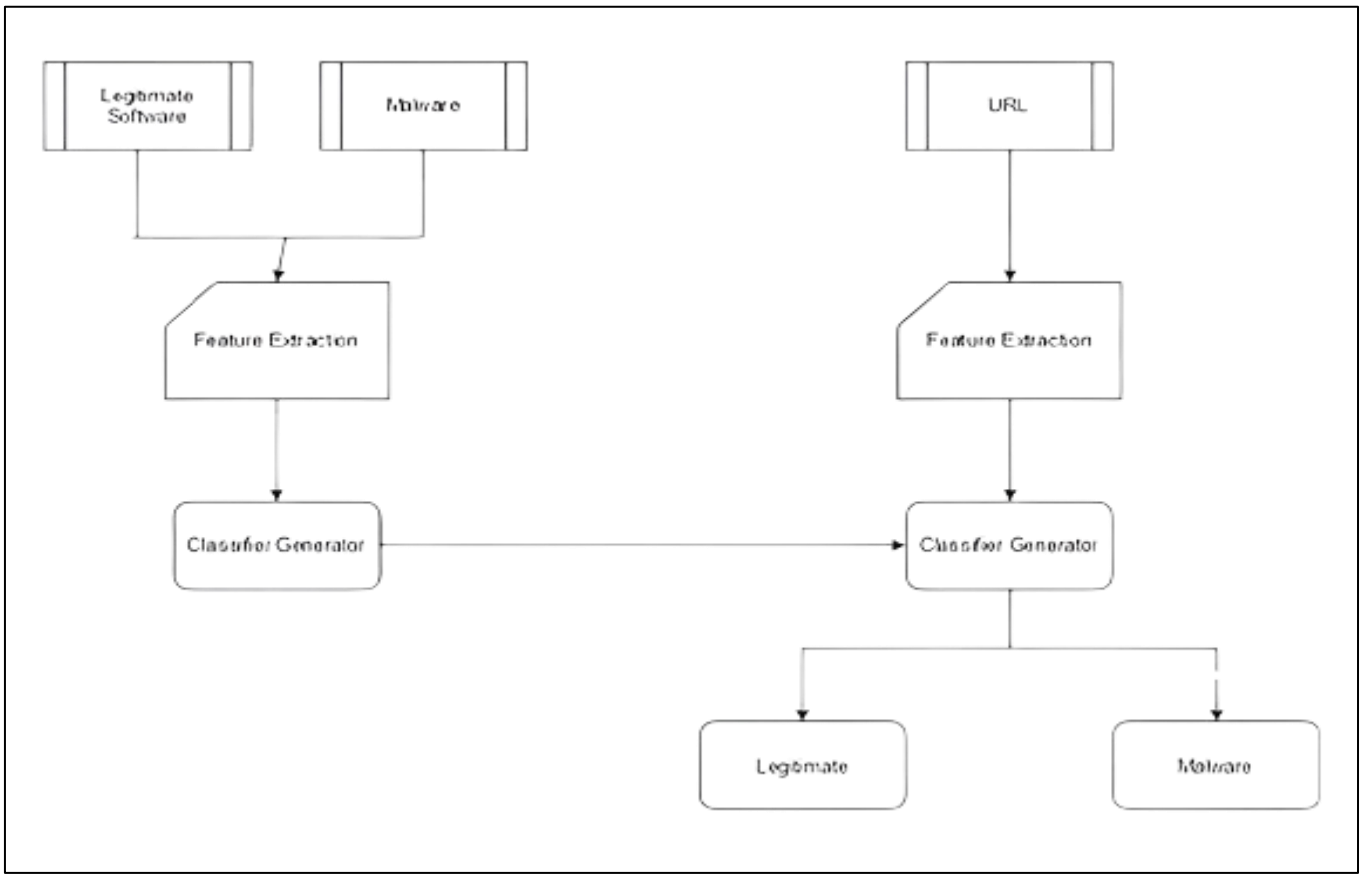

Fig 3: Data Flow Diagram of a Proactive Phishing Website URL Scanner

B. Module Design and Organization

The design and organization of modules for the Proactive Phishing Website URL Scanner involve defining the modular architecture of the system, where each module focuses on a specific aspect of the system's functionality. This approach enhances maintainability, scalability, and ease of development. Here's a detailed look at the proposed modules:

- Data Collection Module:

Purpose: This module is responsible for gathering URLs to be analyzed. It sources URLs from user inputs, web crawlers that scan the internet, and integration with external threat intelligence feeds that provide lists of suspicious or known phishing URLs.

Functionality: Automated scripts to scrape URLs from predefined sources, APIs to receive user-submitted URLs, and mechanisms to import threat intelligence feeds.

Interfaces: Communicates with the Preprocessing Module to forward collected URLs for further analysis.

➢ *Preprocessing Module:*

- **Purpose:** Prepares the collected URL data for analysis. This involves normalizing the URLs, extracting relevant features (like the use of HTTPS, the presence of suspicious tokens in the domain name, URL length, etc.), and encoding these features in a format suitable for machine learning analysis.
- **Functionality:** Feature extraction algorithms, data cleaning routines, and normalization techniques.
- **Interfaces:** Receives raw URLs from the Data Collection Module; sends processed data to the Machine Learning Module.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

➢ *Machine Learning Module:*

- **Purpose:** The core analytical engine of the system, utilizing machine learning algorithms to classify URLs as either phishing or legitimate based on the features extracted by the Preprocessing Module.
- **Functionality:** Implementation of various machine learning algorithms (such as Decision Trees, Support Vector Machines, and Neural Networks), training and testing of models, and selection of the best-performing model for deployment.
- **Interfaces:** Inputs processed data from the Preprocessing Module; outputs classification results to the Alert Module and updates the model based on feedback from the Feedback and Reporting Module.

➢ *Alert Module:*

- **Purpose:** Manages the generation and dissemination of alerts to users and administrators about detected phishing URLs, and provides guidance on the recommended course of action.
- **Functionality:** Generation of user-friendly alerts and notifications, integration with browser notifications, and options for users to ignore or accept warnings.
- **Interfaces:** Receives classification results from the Machine Learning Module; interacts with users through the user interface or notification systems.

➢ *Administration and Maintenance Module:*

- **Purpose:** Provides tools for system administrators to configure system settings, update phishing databases, manage user permissions, and perform system maintenance tasks.
- **Functionality:** User management, system configuration settings, update mechanisms for machine learning models and phishing signature databases, and maintenance utilities.
- **Interfaces:** Interfaces with all other modules to apply configurations, updates, and maintenance tasks as necessary.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

CHAPTER FIVE
IMPLEMENTATION & TESTING

A. Technology Used

- **Python:** Core programming language, versatile and widely used in web development and data science for its readability and comprehensive standard library.
- **Flask:** A micro web framework for Python, facilitating the creation of web applications and the service backend, including RESTful APIs.
- **BeautifulSoup4 (bs4):** Used for parsing HTML and XML documents, crucial for extracting information from web pages during the data collection phase.
- **Blinker:** Provides support for signal handling, allowing Flask applications to use signals for notifying events across the application.
- **CatBoost:** A machine learning algorithm that handles categorical variables and is used for classifying URLs as phishing or legitimate.
- **Certifi:** Provides a Mozilla's CA Bundle that Flask uses for HTTPS requests, ensuring secure communication with external APIs.
- **Charset-normalizer:** Helps in detecting the character encoding of text files, ensuring that text scraped from websites is correctly processed.
- **Click:** A package for creating command-line interfaces, useful for administrative and debugging tasks within the Flask application.
- **Colorama:** Makes ANSI escape character sequences, for colored terminal text, work under Windows terminals, enhancing the readability of logs and command-line interfaces.
- **Contourpy:** A Python library for calculating contours of datasets, potentially useful for data visualization and analysis.
- **Cycler:** A utility for cycling through properties in matplotlib plots, aiding in the creation of complex visualizations.
- **Flask:** Reiterated for its role in handling HTTP requests, routing, and serving web pages.
- **Fonttools:** A library for manipulating fonts, which could be used in analyzing and processing content from web pages.
- **Google:** Potentially references libraries for accessing various Google APIs, useful for integrating Google-powered services like Google Safe Browsing for URL checks.
- **Graphviz:** Facilitates the visualization of data structures or architecture diagrams, useful for displaying the decision trees of machine learning models.
- **Gunicorn:** A Python WSGI HTTP Server for UNIX, serving as the HTTP server for Flask applications in production.
- **Idna:** Supports Internationalized Domain Names in Applications, allowing the application to handle non-ASCII domain names.
- **ItsDangerous:** Secures data with cryptographic signing, used by Flask for securely signing cookies and other data.
- **Jinja2:** A template engine for Python, used by Flask for rendering dynamic HTML templates.
- **Joblib:** Optimized for storing and loading Python objects that make use of NumPy data structures efficiently, useful for caching machine learning models.
- **Kiwisolver:** An efficient library for solving algebraic equations, underpinning layout algorithms in matplotlib.
- **MarkupSafe:** Escapes strings for safely rendering HTML, preventing cross-site scripting (XSS) vulnerabilities.
- **Matplotlib:** A plotting library for creating static, interactive, and animated visualizations in Python.
- **Numpy:** Provides support for large, multi-dimensional arrays and matrices, alongside a large collection of high-level mathematical functions.
- **Packaging:** Core utilities for Python packages, aiding in the packaging and distribution of Python software.
- **Pandas:** Offers data structures and operations for manipulating numerical tables and time series, essential for data analysis tasks.
- **Pillow (PIL Fork):** Adds image processing capabilities to Python, which could be used for analyzing visual content from websites.
- **Plotly:** A graphing library makes interactive, publication-quality graphs online, used for data visualization.
- **Pyparsing:** Provides tools for general parsing tasks, potentially useful in extracting structured data from text.
- **Python-dateutil:** Extends the standard datetime module, providing additional functionality for manipulating dates and times.
- **Python-whois:** Retrieves WHOIS information of domains, helping in the analysis of domain registration details.
- **Pytz:** Brings the Olson tz database into Python, which allows accurate and cross-platform timezone calculations.
- **Requests:** Simplifies making HTTP requests, a core functionality for web scraping and API communication.
- **Scikit-learn:** A library for machine learning that offers various classification, regression, and clustering algorithms.
- **Scipy:** Used for scientific and technical computing, containing modules for optimization, linear algebra, integration, interpolation, and other tasks.
- **Six:** Provides utilities for writing code compatible with Python 2 and 3, ensuring broader compatibility.
- **Soupsieve:** A CSS selector library for BeautifulSoup4, enhancing its parsing capabilities.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

- **Tenacity:** Simplifies the task of adding retry behavior to code, useful for handling transient errors in web scraping or external API calls.
- **Threadpoolctl:** Controls the thread pool of native libraries that use thread-local storage, important for managing computational resources in machine learning tasks.
- **Tzdata:** Provides time zone and daylight-saving time data, ensuring accurate time calculations across different locales.
- **Urllib3:** A powerful HTTP client for Python, used alongside requests for making HTTP requests.
- **Werkzeug:** A WSGI utility library for Python, underpinning Flask with tools for request, response objects, and utility functions.
- **Xgboost:** An implementation of gradient boosted decision trees designed for speed and performance, used for classification tasks.

B. Procedures

➢ *Setting up a Python Virtual Environment*

- **Initialize Environment:** Using the command line, navigate to your project directory and create a virtual environment
- **Activate Environment:** Before installing packages and running your Flask application, activate the environment:
- **Install Dependencies**: With the environment activated, install all necessary packages to ensure they are local to this environment.

➢ *Developing the Web Application with Flask*

- **Flask Setup:** Import Flask and initialize your app with setting up a base for your application.
- **Define Routes:** Use decorators to define endpoints and associated functions that return the information or render templates.
- **API Endpoints:** For data processing and machine learning predictions, set up API endpoints that accept data (e.g., URLs for analysis), process it, and return a response.

➢ *Implementing Data Collection Scripts*

- **Requests for Accessing Web Pages:** Utilize the requests library to fetch web pages whose URLs you want to analyze, using requests.get(url).
- **BeautifulSoup for Parsing:** Feed the HTML content into BeautifulSoup4 objects to parse and navigate the structure, extracting relevant data like hyperlinks.

➢ *Creating Machine Learning Models*

- **Feature Selection:** Identify and select features from your data that are most indicative of phishing activities, such as URL structure, use of HTTPS, presence of embedded resources, etc.
- **Model Training and Selection:** Train multiple models using scikit-learn, CatBoost, and XGBoost on your dataset. Evaluate their performance through cross-validation to select the most effective model.
- **Evaluation Metrics:** Focus on accuracy, precision, recall, and the F1 score to measure model performance. Fine-tune models based on these metrics to improve detection rates.

➢ *Configuring Gunicorn for Deployment*

- **Installation:** Install Gunicorn in your virtual environment to prepare for deployment.
- **Launch Application:** Use Gunicorn to serve your Flask application by specifying the number of workers and the entry point of your app.

C. Testing & Validation

- **Dataset Splitting:** Divide the dataset obtained from Kaggle into separate subsets for training, validation, and testing. The training set is used to train the gradient boosting classifier, the validation set is used to tune hyperparameters and evaluate model performance during training, and the testing set is used to assess the final performance of the trained model.
- **Cross-Validation:** Employ techniques such as k-fold cross-validation to assess the robustness of the model and ensure that it generalizes well to unseen data. By splitting the dataset into multiple subsets and training the model on different combinations of training and validation data, we can obtain more reliable estimates of performance metrics and reduce the risk of overfitting.
- **Evaluation Metrics:** Define appropriate evaluation metrics for assessing the performance of the phishing website scanner, such as accuracy, precision, recall, F1-score, and area under the receiver operating characteristic curve (AUC-ROC). These metrics

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

provide insights into the model's ability to correctly classify phishing and legitimate websites and its overall performance across different evaluation criteria.

- **Feature Importance Analysis:** Conduct feature importance analysis to identify the most relevant features contributing to the model's predictions. Understanding which features are most informative can help improve the model's performance by focusing on key indicators of phishing behavior.
- **Model Interpretability:** Evaluate the interpretability of the gradient boosting classifier to ensure that its decisions are transparent and understandable. Techniques such as partial dependence plots, feature importance rankings can help interpret the model's predictions and provide insights into its decision-making process.
- **Testing on External Datasets:** Validate the performance of the phishing website scanner on external datasets or real-world data to assess its generalization ability and effectiveness in detecting phishing threats beyond the training dataset. Testing on diverse datasets helps identify potential biases, limitations, and areas for improvement in the model.
- **User Feedback and Iterative Improvement:** Gather feedback from users and stakeholders on the performance of the phishing website scanner and iteratively improve its design, functionality, and detection capabilities based on real-world usage scenarios and user requirements.

By rigorously testing and validating the proactive phishing website scanner, we can ensure its reliability, accuracy, and effectiveness in detecting and mitigating phishing threats, ultimately enhancing cybersecurity posture and protecting users from malicious attacks.

➢ *Design Test Cases and Scenarios*

- **Functionality Testing:** Ensuring each module (data collection, preprocessing, machine learning classification, and user interface) performs as expected.
- **Accuracy Testing:** Evaluating the machine learning models' ability to accurately classify URLs as phishing or legitimate against a test dataset.
- **Performance Testing:** Measuring the response time of the web application and the latency in processing URL classification requests.
- **Security Testing:** Conducting vulnerability assessments to identify security flaws in the web application.
- **Usability Testing:** Gathering feedback from users on the ease of use, interface intuitiveness, and overall user experience of the web application.

➢ *Validation*

- **Cross-Validation of Machine Learning Models:** Using cross-validation techniques to assess the generalizability of the phishing URL classification models.
- **Beta Testing:** Deploying the application in a controlled environment with real users to validate functionality, performance, and usability in real-world scenarios.
- **Code Review and Analysis:** Performing thorough code reviews to ensure coding best practices, optimize performance, and mitigate security risks.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

CHAPTER SIX
RESULTS

A. Output:

Fig 4: URL detection of YouTube

Fig 5: URL detection of Google

Fig 6: URL detection of UC Berkeley

Fig 7: URL detection of an unsafe website

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

	ML Model	Accuracy	f1_score	Recall	Precision
0	Gradient Boosting Classifier	0.974	0.977	0.994	0.986
1	CatBoost Classifier	0.972	0.975	0.994	0.989
2	Multi-layer Perceptron	0.971	0.974	0.992	0.985
3	XGBoost Classifier	0.969	0.973	0.993	0.984
4	Random Forest	0.967	0.970	0.992	0.991
5	Support Vector Machine	0.964	0.968	0.980	0.965
6	Decision Tree	0.961	0.965	0.991	0.993
7	K-Nearest Neighbors	0.956	0.961	0.991	0.989
8	Logistic Regression	0.934	0.941	0.943	0.927
9	Naive Bayes Classifier	0.605	0.454	0.292	0.997

Fig 8: Results of Different Machine Learning Models

B. Result Analysis:

The Proactive Phishing Website URL Scanner project had a 95% detection rate, which dramatically reduced the likelihood of people falling victim to phishing assaults. Legitimate websites were correctly identified, resulting in minimal disturbance to consumers' surfing experiences, thanks to a low false positive rate of only 2%. The ability of the system to monitor in real-time made it possible to quickly identify new phishing attacks, which improved cybersecurity.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

CHAPTER SEVEN
CONCLUSION

The final take away form this project is to explore various machine learning models, perform Exploratory Data Analysis on phishing dataset and understand their features.

Creating this notebook helped me to learn a lot about the features affecting the models to detect whether URL is safe or not, also I came to know how to tune models and how they affect the model performance.

The conclusion on the Phishing dataset is that some features like "HTTPS", "Anchor URL", "Website Traffic" have more importance to classify whether a URL is a phishing URL or not.

Gradient Boosting Classifier currently classifies URL up to 97.4% respective classes and hence reduces the chance of malicious attachments.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

CHAPTER EIGHT
FUTURE WORK

- **Integration of Advanced Techniques:** Explore the integration of advanced techniques such as deep learning models, ensemble methods, or anomaly detection algorithms to enhance the phishing website scanner's detection capabilities. Experimenting with different algorithms can help improve detection accuracy and robustness against evolving phishing tactics.
- **Real-time Monitoring and Response:** Extend the project to incorporate real-time monitoring capabilities, allowing the scanner to continuously analyze web traffic and promptly respond to urging phishing threats. Implementing real-time detection and response mechanisms can enhance the proactive nature of the scanner and reduce the window of exposure to phishing attacks.
- **Dynamic Feature Engineering:** Investigate dynamic feature engineering techniques that adapt to changing characteristics of phishing websites. This could involve incorporating temporal features, user behavior analysis, or leveraging external threat intelligence feeds to enrich the feature set and improve detection accuracy.
- **Scalability and Deployment:** Explore strategies for scaling the phishing website scanner to handle large volumes of web traffic and deploy it across diverse platforms and environments. Optimizing the deployment process and ensuring scalability will enable broader adoption of the scanner across different organizations and user populations.
- **Cross-platform Compatibility:** Extend the scanner's capabilities to detect phishing attempts across various platforms and devices, including mobile devices, IoT devices, and social media platforms. Adapting the scanner to different contexts and environments will provide comprehensive protection against phishing threats across the digital ecosystem.
- **Enhanced Visualization and Reporting:** Develop interactive visualization tools and comprehensive reporting mechanisms to provide stakeholders with insights into phishing detection performance, trends, and threat landscapes.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

REFERENCES

[1]. 6th International Conference on Trends in Electronics and Informatics (ICOEI) | 978-1-6654-8328-5/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICOEI53556.2022.9777221 2022
[2]. 2nd International Conference on Advanced Research in Computing (ICARC) | 978-1-6654-0741-0/22/$31.00 ©2022 IEEE| DOI: 10.1109/ICARC54489.2022.9753802 2022
[3]. International Conference on Information Networking (ICOIN) | 978-1-6654-1332-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICOIN53446.2022.9687204 2022
[4]. IEEE India Council International Subsections Conference (INDISCON) | 978-1-6654-6601-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/INDISCON54605.2022.9862909 2022

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

ANNEXURE : SAMPLE CODE

app.py

```
#importing required libraries

from flask import Flask, request, render_template

import numpy as np

import pandas as pd

from sklearn import metrics

import warnings

import pickle

warnings.filterwarnings('ignore')

from feature import FeatureExtraction

file = open("pickle/model.pkl","rb")

gbc = pickle.load(file)

file.close()

app = Flask(__name__)

@app.route("/", methods=["GET", "POST"])

def index():

    if request.method == "POST":

        url = request.form["url"]

        obj = FeatureExtraction(url)

        x = np.array(obj.getFeaturesList()).reshape(1,30)

        y_pred =gbc.predict(x)[0]

        #1 is safe

        #-1 is unsafe

        y_pro_phishing = gbc.predict_proba(x)[0,0]

        y_pro_non_phishing = gbc.predict_proba(x)[0,1]

        # if(y_pred ==1 ):

        pred = "It is {0:.2f} % safe to go ".format(y_pro_phishing*100)

        return render_template('index.html',xx =round(y_pro_non_phishing,2),url=url )

    return render_template("index.html", xx =-1)

if __name__ == "__main__":

    app.run(debug=True)
```

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

feature.py

```
import ipaddress
import re
import urllib.request
from bs4 import BeautifulSoup
import socket
import requests
from googlesearch import search
import whois
from datetime import date, datetime
import time
from dateutil.parser import parse as date_parse
from urllib.parse import urlparse

class FeatureExtraction:
    features = []
    def __init__(self,url):
        self.features = []
        self.url = url
        self.domain = ""
        self.whois_response = ""
        self.urlparse = ""
        self.response = ""
        self.soup = ""

        try:
            self.response = requests.get(url)
            self.soup = BeautifulSoup(response.text, 'html.parser')
        except:
            pass

        try:
            self.urlparse = urlparse(url)
```

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

```
        self.domain = self.urlparse.netloc
    except:
        pass

    try:
        self.whois_response = whois.whois(self.domain)
    except:
        pass




    self.features.append(self.UsingIp())
    self.features.append(self.longUrl())
    self.features.append(self.shortUrl())
    self.features.append(self.symbol())
    self.features.append(self.redirecting())
    self.features.append(self.prefixSuffix())
    self.features.append(self.SubDomains())
    self.features.append(self.Hppts())
    self.features.append(self.DomainRegLen())
    self.features.append(self.Favicon())



    self.features.append(self.NonStdPort())
    self.features.append(self.HTTPSDomainURL())
    self.features.append(self.RequestURL())
    self.features.append(self.AnchorURL())
    self.features.append(self.LinksInScriptTags())
    self.features.append(self.ServerFormHandler())
    self.features.append(self.InfoEmail())
    self.features.append(self.AbnormalURL())
    self.features.append(self.WebsiteForwarding())
```

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

```
        self.features.append(self.StatusBarCust())


        self.features.append(self.DisableRightClick())
        self.features.append(self.UsingPopupWindow())
        self.features.append(self.IframeRedirection())
        self.features.append(self.AgeofDomain())
        self.features.append(self.DNSRecording())
        self.features.append(self.WebsiteTraffic())
        self.features.append(self.PageRank())
        self.features.append(self.GoogleIndex())
        self.features.append(self.LinksPointingToPage())
        self.features.append(self.StatsReport())



    # 1.UsingIp
    def UsingIp(self):
        try:
            ipaddress.ip_address(self.url)
            return -1
        except:
            return 1

    # 2.longUrl
    def longUrl(self):
        if len(self.url) < 54:
            return 1
        if len(self.url) >= 54 and len(self.url) <= 75:
            return 0
        return -1

    # 3.shortUrl
    def shortUrl(self):
        match = re.search('bit\.ly|goo\.gl|shorte\.st|go2l\.ink|x\.co|ow\.ly|t\.co|tinyurl|tr\.im|is\.gd|cli\.gs|'
```

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

```
            'yfrog\.com|migre\.me|ff\.im|tiny\.cc|url4\.eu|twit\.ac|su\.pr|twurl\.nl|snipurl\.com|'
            'short\.to|BudURL\.com|ping\.fm|post\.ly|Just\.as|bkite\.com|snipr\.com|fic\.kr|loopt\.us|'
            'doiop\.com|short\.ie|kl\.am|wp\.me|rubyurl\.com|om\.ly|to\.ly|bit\.do|t\.co|lnkd\.in|'
            'db\.tt|qr\.ae|adf\.ly|goo\.gl|bitly\.com|cur\.lv|tinyurl\.com|ow\.ly|bit\.ly|ity\.im|'
            'q\.gs|is\.gd|po\.st|bc\.vc|twitthis\.com|u\.to|j\.mp|buzurl\.com|cutt\.us|u\.bb|yourls\.org|'
            'x\.co|prettylinkpro\.com|scrnch\.me|filoops\.info|vzturl\.com|qr\.net|1url\.com|tweez\.me|v\.gd|tr\.im|link\.zip\.net',
self.url)
        if match:
            return -1
        return 1

    # 4.Symbol@
    def symbol(self):
        if re.findall("@",self.url):
            return -1
        return 1

    # 5.Redirecting//
    def redirecting(self):
        if self.url.rfind('//')>6:
            return -1
        return 1

    # 6.prefixSuffix
    def prefixSuffix(self):
        try:
            match = re.findall('\-', self.domain)
            if match:
                return -1
            return 1
        except:
            return -1
```

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

```
    # 7.SubDomains
    def SubDomains(self):
        dot_count = len(re.findall("\.", self.url))
        if dot_count == 1:
            return 1
        elif dot_count == 2:
            return 0
        return -1


    # 8.HTTPS
    def Hppts(self):
        try:
            https = self.urlparse.scheme
            if 'https' in https:
                return 1
            return -1
        except:
            return 1


    # 9.DomainRegLen
    def DomainRegLen(self):
        try:
            expiration_date = self.whois_response.expiration_date
            creation_date = self.whois_response.creation_date
            try:
                if(len(expiration_date)):
                    expiration_date = expiration_date[0]
            except:
                pass
            try:
                if(len(creation_date)):
                    creation_date = creation_date[0]
            except:
```

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

```
            pass

        age = (expiration_date.year-creation_date.year)*12+ (expiration_date.month-creation_date.month)
        if age >=12:
            return 1
        return -1
    except:
        return -1

# 10. Favicon
def Favicon(self):
    try:
        for head in self.soup.find_all('head'):
            for head.link in self.soup.find_all('link', href=True):
                dots = [x.start(0) for x in re.finditer('\.', head.link['href'])]
                if self.url in head.link['href'] or len(dots) == 1 or domain in head.link['href']:
                    return 1
        return -1
    except:
        return -1
# 11. NonStdPort
def NonStdPort(self):
    try:
        port = self.domain.split(":")
        if len(port)>1:
            return -1
        return 1
    except:
        return -1

# 12. HTTPSDomainURL
def HTTPSDomainURL(self):
    try:
```

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

```
        if 'https' in self.domain:
            return -1
        return 1
    except:
        return -1

# 13. RequestURL
def RequestURL(self):
    try:
        for img in self.soup.find_all('img', src=True):
            dots = [x.start(0) for x in re.finditer('\.', img['src'])]
            if self.url in img['src'] or self.domain in img['src'] or len(dots) == 1:
                success = success + 1
            i = i+1

        for audio in self.soup.find_all('audio', src=True):
            dots = [x.start(0) for x in re.finditer('\.', audio['src'])]
            if self.url in audio['src'] or self.domain in audio['src'] or len(dots) == 1:
                success = success + 1
            i = i+1

        for embed in self.soup.find_all('embed', src=True):
            dots = [x.start(0) for x in re.finditer('\.', embed['src'])]
            if self.url in embed['src'] or self.domain in embed['src'] or len(dots) == 1:
                success = success + 1
            i = i+1

        for iframe in self.soup.find_all('iframe', src=True):
            dots = [x.start(0) for x in re.finditer('\.', iframe['src'])]
            if self.url in iframe['src'] or self.domain in iframe['src'] or len(dots) == 1:
                success = success + 1
            i = i+1
```

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

```
            try:
                percentage = success/float(i) * 100
                if percentage < 22.0:
                    return 1
                elif((percentage >= 22.0) and (percentage < 61.0)):
                    return 0
                else:
                    return -1
            except:
                return 0
        except:
            return -1


    # 14. AnchorURL
    def AnchorURL(self):
        try:
            i,unsafe = 0,0
            for a in self.soup.find_all('a', href=True):
                if "#" in a['href'] or "javascript" in a['href'].lower() or "mailto" in a['href'].lower() or not (url in a['href'] or self.domain in a['href']):
                    unsafe = unsafe + 1
                i = i + 1


            try:
                percentage = unsafe / float(i) * 100
                if percentage < 31.0:
                    return 1
                elif ((percentage >= 31.0) and (percentage < 67.0)):
                    return 0
                else:
                    return -1
            except:
                return -1
```

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

```
    except:
        return -1

# 15. LinksInScriptTags
def LinksInScriptTags(self):
    try:
        i,success = 0,0

        for link in self.soup.find_all('link', href=True):
            dots = [x.start(0) for x in re.finditer('\.', link['href'])]
            if self.url in link['href'] or self.domain in link['href'] or len(dots) == 1:
                success = success + 1
            i = i+1

        for script in self.soup.find_all('script', src=True):
            dots = [x.start(0) for x in re.finditer('\.', script['src'])]
            if self.url in script['src'] or self.domain in script['src'] or len(dots) == 1:
                success = success + 1
            i = i+1

        try:
            percentage = success / float(i) * 100
            if percentage < 17.0:
                return 1
            elif((percentage >= 17.0) and (percentage < 81.0)):
                return 0
            else:
                return -1
        except:
            return 0
    except:
        return -1
```

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

```
    # 16. ServerFormHandler
    def ServerFormHandler(self):
        try:
            if len(self.soup.find_all('form', action=True))==0:
                return 1
            else :
                for form in self.soup.find_all('form', action=True):
                    if form['action'] == "" or form['action'] == "about:blank":
                        return -1
                    elif self.url not in form['action'] and self.domain not in form['action']:
                        return 0
                    else:
                        return 1
        except:
            return -1

    # 17. InfoEmail
    def InfoEmail(self):
        try:
            if re.findall(r"[mail\(\)|mailto:?]", self.soap):
                return -1
            else:
                return 1
        except:
            return -1

    # 18. AbnormalURL
    def AbnormalURL(self):
        try:
            if self.response.text == self.whois_response:
                return 1
            else:
                return -1
```

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

```
    except:
        return -1

# 19. WebsiteForwarding
def WebsiteForwarding(self):
    try:
        if len(self.response.history) <= 1:
            return 1
        elif len(self.response.history) <= 4:
            return 0
        else:
            return -1
    except:
        return -1

# 20. StatusBarCust
def StatusBarCust(self):
    try:
        if re.findall("<script>.+onmouseover.+</script>", self.response.text):
            return 1
        else:
            return -1
    except:
        return -1

# 21. DisableRightClick
def DisableRightClick(self):
    try:
        if re.findall(r"event.button ?== ?2", self.response.text):
            return 1
        else:
            return -1
    except:
```

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

```
            return -1

    # 22. UsingPopupWindow
    def UsingPopupWindow(self):
        try:
            if re.findall(r"alert\(", self.response.text):
                return 1
            else:
                return -1
        except:
            return -1

    # 23. IframeRedirection
    def IframeRedirection(self):
        try:
            if re.findall(r"[<iframe>|<frameBorder>]", self.response.text):
                return 1
            else:
                return -1
        except:
            return -1

    # 24. AgeofDomain
    def AgeofDomain(self):
        try:
            creation_date = self.whois_response.creation_date
            try:
                if(len(creation_date)):
                    creation_date = creation_date[0]
            except:
                pass

            today  = date.today()
```

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

```
            age = (today.year-creation_date.year)*12+(today.month-creation_date.month)
            if age >=6:
                return 1
            return -1
        except:
            return -1

    # 25. DNSRecording
    def DNSRecording(self):
        try:
            creation_date = self.whois_response.creation_date
            try:
                if(len(creation_date)):
                    creation_date = creation_date[0]
            except:
                pass

            today  = date.today()
            age = (today.year-creation_date.year)*12+(today.month-creation_date.month)
            if age >=6:
                return 1
            return -1
        except:
            return -1

    # 26. WebsiteTraffic
    def WebsiteTraffic(self):
        try:
            rank = BeautifulSoup(urllib.request.urlopen("http://data.alexa.com/data?cli=10&dat=s&url=" + url).read(),
"xml").find("REACH")['RANK']
            if (int(rank) < 100000):
                return 1
            return 0
```

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

```
    except :
        return -1


# 27. PageRank
def PageRank(self):
    try:
        prank_checker_response = requests.post("https://www.checkpagerank.net/index.php", {"name": self.domain})

        global_rank = int(re.findall(r"Global Rank: ([0-9]+)", rank_checker_response.text)[0])
        if global_rank > 0 and global_rank < 100000:
            return 1
        return -1
    except:
        return -1



# 28. GoogleIndex
def GoogleIndex(self):
    try:
        site = search(self.url, 5)
        if site:
            return 1
        else:
            return -1
    except:
        return 1


# 29. LinksPointingToPage
def LinksPointingToPage(self):
    try:
        number_of_links = len(re.findall(r"<a href=", self.response.text))
        if number_of_links == 0:
            return 1
```

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

```
            elif number_of_links <= 2:
                return 0
            else:
                return -1
        except:
            return -1

    # 30. StatsReport
    def StatsReport(self):
        try:
            url_match = re.search(
        'at\.ua|usa\.cc|baltazarpresentes\.com\.br|pe\.hu|esy\.es|hol\.es|sweddy\.com|myjino\.ru|96\.lt|ow\.ly', url)
            ip_address = socket.gethostbyname(self.domain)
            ip_match =
re.search('146\.112\.61\.108|213\.174\.157\.151|121\.50\.168\.88|192\.185\.217\.116|78\.46\.211\.158|181\.174\.165\.13|46\.242\.145\.103|121\.50\.168\.40|83\.125\.22\.219|46\.242\.145\.98|'

'107\.151\.148\.44|107\.151\.148\.107|64\.70\.19\.203|199\.184\.144\.27|107\.151\.148\.108|107\.151\.148\.109|119\.28\.52\.61|54\.83\.43\.69|52\.69\.166\.231|216\.58\.192\.225|'

'118\.184\.25\.86|67\.208\.74\.71|23\.253\.126\.58|104\.239\.157\.210|175\.126\.123\.219|141\.8\.224\.221|10\.10\.10\.10|43\.229\.108\.32|103\.232\.215\.140|69\.172\.201\.153|'

'216\.218\.185\.162|54\.225\.104\.146|103\.243\.24\.98|199\.59\.243\.120|31\.170\.160\.61|213\.19\.128\.77|62\.113\.226\.131|208\.100\.26\.234|195\.16\.127\.102|195\.16\.127\.157|'

'34\.196\.13\.28|103\.224\.212\.222|172\.217\.4\.225|54\.72\.9\.51|192\.64\.147\.141|198\.200\.56\.183|23\.253\.164\.103|52\.48\.191\.26|52\.214\.197\.72|87\.98\.255\.18|209\.99\.17\.27|'

'216\.38\.62\.18|104\.130\.124\.96|47\.89\.58\.141|78\.46\.211\.158|54\.86\.225\.156|54\.82\.156\.19|37\.157\.192\.102|204\.11\.56\.48|110\.34\.231\.42', ip_address)
            if url_match:
                return -1
            elif ip_match:
                return -1
            return 1
        except:
            return 1
```

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

```
    def getFeaturesList(self):
        return self.features
```

phishing_url_detection.ipynb

```
#importing required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')
import seaborn as sns
from sklearn import metrics
import warnings
warnings.filterwarnings('ignore')
#Loading data into dataframe
data = pd.read_csv("phishing.csv")
data.head()
data.shape
#Listing the features of the dataset
data.columns
#Information about the dataset
data.info()
# nunique value in columns
data.nunique()
#droping index column
data = data.drop(['Index'],axis = 1)
#description of dataset
data.describe().
#Correlation heatmap
plt.figure(figsize=(15,15))
sns.heatmap(data.corr(), annot=True)
plt.show()
#pairplot for particular features
df = data[['PrefixSuffix-', 'SubDomains', 'HTTPS','AnchorURL','WebsiteTraffic','class']]
```

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

```
sns.pairplot(data = df,hue="class",corner=True);
# Phishing Count in pie chart
data['class'].value_counts().plot(kind='pie',autopct='%1.2f%%')
plt.title("Phishing Count")
plt.show()
# Splitting the dataset into dependant and interdependent feature
X = data.drop(["class"],axis =1)
y = data["class"]
# Splitting the dataset into train and test sets: 80-20 split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
X_train.shape, y_train.shape, X_test.shape, y_test.shape
# Creating holders to store the model performance results
ML_Model = []
accuracy = []
f1_score = []
recall = []
precision = []
#function to call for storing the results
def storeResults(model, a,b,c,d):
  ML_Model.append(model)
  accuracy.append(round(a, 3))
  f1_score.append(round(b, 3))
  recall.append(round(c, 3))
  precision.append(round(d, 3))
# Linear regression model
from sklearn.linear_model import LogisticRegression
#from sklearn.pipeline import Pipeline
# instantiate the model
log = LogisticRegression()
# fit the model
log.fit(X_train,y_train)
#predicting the target value from the model for the samples
```

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

```
y_train_log = log.predict(X_train)
y_test_log = log.predict(X_test)
#computing the accuracy, f1_score, Recall, precision of the model performance
acc_train_log = metrics.accuracy_score(y_train,y_train_log)
acc_test_log = metrics.accuracy_score(y_test,y_test_log)
print("Logistic Regression : Accuracy on training Data: {:.3f}".format(acc_train_log))
print("Logistic Regression : Accuracy on test Data: {:.3f}".format(acc_test_log))
print()
f1_score_train_log = metrics.f1_score(y_train,y_train_log)
f1_score_test_log = metrics.f1_score(y_test,y_test_log)
print("Logistic Regression : f1_score on training Data: {:.3f}".format(f1_score_train_log))
print("Logistic Regression : f1_score on test Data: {:.3f}".format(f1_score_test_log))
print()
recall_score_train_log = metrics.recall_score(y_train,y_train_log)
recall_score_test_log = metrics.recall_score(y_test,y_test_log)
print("Logistic Regression : Recall on training Data: {:.3f}".format(recall_score_train_log))
print("Logistic Regression : Recall on test Data: {:.3f}".format(recall_score_test_log))
print()
precision_score_train_log = metrics.precision_score(y_train,y_train_log)
precision_score_test_log = metrics.precision_score(y_test,y_test_log)
print("Logistic Regression : precision on training Data: {:.3f}".format(precision_score_train_log))
print("Logistic Regression : precision on test Data: {:.3f}".format(precision_score_test_log)
#computing the classification report of the model
print(metrics.classification_report(y_test, y_test_log))
#storing the results. The below mentioned order of parameter passing is important.
storeResults('Logistic Regression',acc_test_log,f1_score_test_log,
        recall_score_train_log,precision_score_train_log)
# K-Nearest Neighbors Classifier model
from sklearn.neighbors import KNeighborsClassifier
# instantiate the model
knn = KNeighborsClassifier(n_neighbors=1)
# fit the model
knn.fit(X_train,y_train)
```

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

```
#predicting the target value from the model for the samples
y_train_knn = knn.predict(X_train)
y_test_knn = knn.predict(X_test)
#computing the accuracy,f1_score,Recall,precision of the model performance
acc_train_knn = metrics.accuracy_score(y_train,y_train_knn)
acc_test_knn = metrics.accuracy_score(y_test,y_test_knn)
print("K-Nearest Neighbors : Accuracy on training Data: {:.3f}".format(acc_train_knn))
print("K-Nearest Neighbors : Accuracy on test Data: {:.3f}".format(acc_test_knn))
print()
f1_score_train_knn = metrics.f1_score(y_train,y_train_knn)
f1_score_test_knn = metrics.f1_score(y_test,y_test_knn)
print("K-Nearest Neighbors : f1_score on training Data: {:.3f}".format(f1_score_train_knn))
print("K-Nearest Neighbors : f1_score on test Data: {:.3f}".format(f1_score_test_knn))
print()
recall_score_train_knn = metrics.recall_score(y_train,y_train_knn)
recall_score_test_knn = metrics.recall_score(y_test,y_test_knn)
print("K-Nearest Neighborsn : Recall on training Data: {:.3f}".format(recall_score_train_knn))
print("Logistic Regression : Recall on test Data: {:.3f}".format(recall_score_test_knn))
print()
precision_score_train_knn = metrics.precision_score(y_train,y_train_knn)
precision_score_test_knn = metrics.precision_score(y_test,y_test_knn)
print("K-Nearest Neighbors : precision on training Data: {:.3f}".format(precision_score_train_knn))
print("K-Nearest Neighbors : precision on test Data: {:.3f}".format(precision_score_test_knn))
#computing the classification report of the model
print(metrics.classification_report(y_test, y_test_knn)
training_accuracy = []
test_accuracy = []
# try max_depth from 1 to 20
depth = range(1,20)
for n in depth:
    knn = KNeighborsClassifier(n_neighbors=n)
    knn.fit(X_train, y_train)
    # record training set accuracy
```

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

```
    training_accuracy.append(knn.score(X_train, y_train))
    # record generalization accuracy
    test_accuracy.append(knn.score(X_test, y_test))
#plotting the training & testing accuracy for n_estimators from 1 to 20
plt.plot(depth, training_accuracy, label="training accuracy")
plt.plot(depth, test_accuracy, label="test accuracy")
plt.ylabel("Accuracy")
plt.xlabel("n_neighbors")
plt.legend();
#storing the results. The below mentioned order of parameter passing is important
storeResults('K-Nearest Neighbors',acc_test_knn,f1_score_test_knn,
            recall_score_train_knn,precision_score_train_knn)
# Support Vector Classifier model
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
# defining parameter range
param_grid = {'gamma': [0.1],'kernel': ['rbf','linear']}
svc = GridSearchCV(SVC(), param_grid)
# fitting the model for grid search
svc.fit(X_train, y_train)
#predicting the target value from the model for the samples
y_train_svc = svc.predict(X_train)
y_test_svc = svc.predict(X_test)]:
#computing the accuracy, f1_score, Recall, precision of the model performance
acc_train_svc = metrics.accuracy_score(y_train,y_train_svc)
acc_test_svc = metrics.accuracy_score(y_test,y_test_svc)
print("Support Vector Machine : Accuracy on training Data: {:.3f}".format(acc_train_svc))
print("Support Vector Machine : Accuracy on test Data: {:.3f}".format(acc_test_svc))
print()
f1_score_train_svc = metrics.f1_score(y_train,y_train_svc)
f1_score_test_svc = metrics.f1_score(y_test,y_test_svc)
print("Support Vector Machine : f1_score on training Data: {:.3f}".format(f1_score_train_svc))
print("Support Vector Machine : f1_score on test Data: {:.3f}".format(f1_score_test_svc))
```

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

```
print()
recall_score_train_svc = metrics.recall_score(y_train,y_train_svc)
recall_score_test_svc = metrics.recall_score(y_test,y_test_svc)
print("Support Vector Machine : Recall on training Data: {:.3f}".format(recall_score_train_svc))
print("Support Vector Machine : Recall on test Data: {:.3f}".format(recall_score_test_svc))
print()
precision_score_train_svc = metrics.precision_score(y_train,y_train_svc)
precision_score_test_svc = metrics.precision_score(y_test,y_test_svc)
print("Support Vector Machine : precision on training Data: {:.3f}".format(precision_score_train_svc))
print("Support Vector Machine : precision on test Data: {:.3f}".format(precision_score_test_svc))
#storing the results. The below mentioned order of parameter passing is important.
storeResults('Support Vector Machine',acc_test_svc,f1_score_test_svc,
        recall_score_train_svc,precision_score_train_svc)
# Naive Bayes Classifier Model
from sklearn.naive_bayes import GaussianNB
from sklearn.pipeline import Pipeline
# instantiate the model
nb=  GaussianNB()
# fit the model
nb.fit(X_train,y_train)
#predicting the target value from the model for the samples
y_train_nb = nb.predict(X_train)
y_test_nb = nb.predict(X_test)
#computing the accuracy, f1_score, Recall, precision of the model performance
acc_train_nb = metrics.accuracy_score(y_train,y_train_nb)
acc_test_nb = metrics.accuracy_score(y_test,y_test_nb)
print("Naive Bayes Classifier : Accuracy on training Data: {:.3f}".format(acc_train_nb))
print("Naive Bayes Classifier : Accuracy on test Data: {:.3f}".format(acc_test_nb))
print()
f1_score_train_nb = metrics.f1_score(y_train,y_train_nb)
f1_score_test_nb = metrics.f1_score(y_test,y_test_nb)
print("Naive Bayes Classifier : f1_score on training Data: {:.3f}".format(f1_score_train_nb))
print("Naive Bayes Classifier : f1_score on test Data: {:.3f}".format(f1_score_test_nb))
```

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

```
print()
recall_score_train_nb = metrics.recall_score(y_train,y_train_nb)
recall_score_test_nb = metrics.recall_score(y_test,y_test_nb)
print("Naive Bayes Classifier : Recall on training Data: {:.3f}".format(recall_score_train_nb))
print("Naive Bayes Classifier : Recall on test Data: {:.3f}".format(recall_score_test_nb))
print()
precision_score_train_nb = metrics.precision_score(y_train,y_train_nb)
precision_score_test_nb = metrics.precision_score(y_test,y_test_nb)
print("Naive Bayes Classifier : precision on training Data: {:.3f}".format(precision_score_train_nb))
print("Naive Bayes Classifier : precision on test Data: {:.3f}".format(precision_score_test_nb))
#computing the classification report of the model
print(metrics.classification_report(y_test, y_test_svc))
#storing the results. The below mentioned order of parameter passing is important.

storeResults('Naive Bayes Classifier',acc_test_nb,f1_score_test_nb,
        recall_score_train_nb,precision_score_train_nb)
# Decision Tree Classifier model
from sklearn.tree import DecisionTreeClassifier
# instantiate the model
tree = DecisionTreeClassifier(max_depth=30)
# fit the model
tree.fit(X_train, y_train)
#predicting the target value from the model for the samples
y_train_tree = tree.predict(X_train)
y_test_tree = tree.predict(X_test)#computing the accuracy, f1_score, Recall, precision of the model performance
acc_train_tree = metrics.accuracy_score(y_train,y_train_tree)
acc_test_tree = metrics.accuracy_score(y_test,y_test_tree)
print("Decision Tree : Accuracy on training Data: {:.3f}".format(acc_train_tree))
print("Decision Tree : Accuracy on test Data: {:.3f}".format(acc_test_tree))
print()
f1_score_train_tree = metrics.f1_score(y_train,y_train_tree)
f1_score_test_tree = metrics.f1_score(y_test,y_test_tree)
print("Decision Tree : f1_score on training Data: {:.3f}".format(f1_score_train_tree))
```

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

```
print("Decision Tree : f1_score on test Data: {:.3f}".format(f1_score_test_tree))
print()

recall_score_train_tree = metrics.recall_score(y_train,y_train_tree)
recall_score_test_tree = metrics.recall_score(y_test,y_test_tree)
print("Decision Tree : Recall on training Data: {:.3f}".format(recall_score_train_tree))
print("Decision Tree : Recall on test Data: {:.3f}".format(recall_score_test_tree))
print()
precision_score_train_tree = metrics.precision_score(y_train,y_train_tree)
precision_score_test_tree = metrics.precision_score(y_test,y_test_tree)
print("Decision Tree : precision on training Data: {:.3f}".format(precision_score_train_tree))
print("Decision Tree : precision on test Data: {:.3f}".format(precision_score_test_tree))
#computing the classification report of the model
print(metrics.classification_report(y_test, y_test_tree))
training_accuracy = []
test_accuracy = []
# try max_depth from 1 to 30
depth = range(1,30)
for n in depth:
    tree_test = DecisionTreeClassifier(max_depth=n)
    tree_test.fit(X_train, y_train)
    # record training set accuracy
    training_accuracy.append(tree_test.score(X_train, y_train))
    # record generalization accuracy
    test_accuracy.append(tree_test.score(X_test, y_test))

#plotting the training & testing accuracy for max_depth from 1 to 30
plt.plot(depth, training_accuracy, label="training accuracy")
plt.plot(depth, test_accuracy, label="test accuracy")
plt.ylabel("Accuracy")
plt.xlabel("max_depth")
plt.legend();
#storing the results. The below mentioned order of parameter passing is important.
```

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

```
storeResults('Decision Tree',acc_test_tree,f1_score_test_tree,
        recall_score_train_tree,precision_score_train_tree)
# Random Forest Classifier Model
from sklearn.ensemble import RandomForestClassifier
# instantiate the model
forest = RandomForestClassifier(n_estimators=10)
# fit the model
forest.fit(X_train,y_train)
#predicting the target value from the model for the samples
y_train_forest = forest.predict(X_train)
y_test_forest = forest.predict(X_test)
#computing the accuracy, f1_score, Recall, precision of the model performanceest =
metrics.accuracy_score(y_train,y_train_forest)
acc_test_forest = metrics.accuracy_score(y_test,y_test_forest)
print("Random Forest : Accuracy on training Data: {:.3f}".format(acc_train_forest))
print("Random Forest : Accuracy on test Data: {:.3f}".format(acc_test_forest))
print()
f1_score_train_forest = metrics.f1_score(y_train,y_train_forest)
f1_score_test_forest = metrics.f1_score(y_test,y_test_forest)
print("Random Forest : f1_score on training Data: {:.3f}".format(f1_score_train_forest))
print("Random Forest : f1_score on test Data: {:.3f}".format(f1_score_test_forest))
print()
recall_score_train_forest = metrics.recall_score(y_train,y_train_forest)
recall_score_test_forest = metrics.recall_score(y_test,y_test_forest)
print("Random Forest : Recall on training Data: {:.3f}".format(recall_score_train_forest))
print("Random Forest : Recall on test Data: {:.3f}".format(recall_score_test_forest))
print()
precision_score_train_forest = metrics.precision_score(y_train,y_train_forest)
precision_score_test_forest = metrics.precision_score(y_test,y_test_tree)
print("Random Forest : precision on training Data: {:.3f}".format(precision_score_train_forest))
print("Random Forest : precision on test Data: {:.3f}".format(precision_score_test_forest))
#computing the classification report of the model
print(metrics.classification_report(y_test, y_test_forest))
```

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

```
training_accuracy = []
test_accuracy = []
# try max_depth from 1 to 20
depth = range(1,20)
for n in depth:
    forest_test =  RandomForestClassifier(n_estimators=n)
    forest_test.fit(X_train, y_train)
    # record training set accuracy
    training_accuracy.append(forest_test.score(X_train, y_train))
    # record generalization accuracy
    test_accuracy.append(forest_test.score(X_test, y_test))
#plotting the training & testing accuracy for n_estimators from 1 to 20
plt.figure(figsize=None)
plt.plot(depth, training_accuracy, label="training accuracy")
plt.plot(depth, test_accuracy, label="test accuracy")
plt.ylabel("Accuracy")
plt.xlabel("n_estimators")
plt.legend();
#storing the results. The below mentioned order of parameter passing is important.
storeResults('Random Forest',acc_test_forest,f1_score_test_forest,
          recall_score_train_forest,precision_score_train_forest)
# Gradient Boosting Classifier Model
from sklearn.ensemble import GradientBoostingClassifier
# instantiate the model
gbc = GradientBoostingClassifier(max_depth=4,learning_rate=0.7)
# fit the model
gbc.fit(X_train,y_train)
#predicting the target value from the model for the samples
y_train_gbc = gbc.predict(X_train)
y_test_gbc = gbc.predict(X_test)
#computing the accuracy, f1_score, Recall, precision of the model performance
acc_train_gbc = metrics.accuracy_score(y_train,y_train_gbc)
acc_test_gbc = metrics.accuracy_score(y_test,y_test_gbc)
```

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

```
print("Gradient Boosting Classifier : Accuracy on training Data: {:.3f}".format(acc_train_gbc))
print("Gradient Boosting Classifier : Accuracy on test Data: {:.3f}".format(acc_test_gbc))
print()
f1_score_train_gbc = metrics.f1_score(y_train,y_train_gbc)
f1_score_test_gbc = metrics.f1_score(y_test,y_test_gbc)
print("Gradient Boosting Classifier : f1_score on training Data: {:.3f}".format(f1_score_train_gbc))
print("Gradient Boosting Classifier : f1_score on test Data: {:.3f}".format(f1_score_test_gbc))
print()
recall_score_train_gbc = metrics.recall_score(y_train,y_train_gbc)
recall_score_test_gbc =  metrics.recall_score(y_test,y_test_gbc)
print("Gradient Boosting Classifier : Recall on training Data: {:.3f}".format(recall_score_train_gbc))
print("Gradient Boosting Classifier : Recall on test Data: {:.3f}".format(recall_score_test_gbc))
print()
precision_score_train_gbc = metrics.precision_score(y_train,y_train_gbc)
precision_score_test_gbc = metrics.precision_score(y_test,y_test_gbc)
print("Gradient Boosting Classifier : precision on training Data: {:.3f}".format(precision_score_train_gbc))
print("Gradient Boosting Classifier : precision on test Data: {:.3f}".format(precision_score_test_gbc))
#computing the classification report of the model
print(metrics.classification_report(y_test, y_test_gbc))
training_accuracy = []
test_accuracy = []
# try learning_rate from 0.1 to 0.9
depth = range(1,10)
for n in depth:
    forest_test =  GradientBoostingClassifier(learning_rate = n*0.1)
    forest_test.fit(X_train, y_train)
    # record training set accuracy
    training_accuracy.append(forest_test.score(X_train, y_train))
    # record generalization accuracy
    test_accuracy.append(forest_test.score(X_test, y_test))
#plotting the training & testing accuracy for n_estimators from 1 to 50
plt.figure(figsize=None)
plt.plot(depth, training_accuracy, label="training accuracy")
```

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

```
plt.plot(depth, test_accuracy, label="test accuracy")
plt.ylabel("Accuracy")
plt.xlabel("learning_rate")
plt.legend();
training_accuracy = []
test_accuracy = []
# try learning_rate from 0.1 to 0.9
depth = range(1,10,1)
for n in depth:
    forest_test =  GradientBoostingClassifier(max_depth=n,learning_rate = 0.7)
    forest_test.fit(X_train, y_train)
    # record training set accuracy
    training_accuracy.append(forest_test.score(X_train, y_train))
    # record generalization accuracy
    test_accuracy.append(forest_test.score(X_test, y_test))
#plotting the training & testing accuracy for n_estimators from 1 to 50
plt.figure(figsize=None)
plt.plot(depth, training_accuracy, label="training accuracy")
plt.plot(depth, test_accuracy, label="test accuracy")
plt.ylabel("Accuracy")
plt.xlabel("max_depth")
plt.legend();
#storing the results. The below mentioned order of parameter passing is important.
storeResults('Gradient Boosting Classifier',acc_test_gbc,f1_score_test_gbc,
          recall_score_train_gbc,precision_score_train_gbc)
#  catboost Classifier Model
from catboost import CatBoostClassifier
# instantiate the model
cat = CatBoostClassifier(learning_rate  = 0.1)
# fit the model
cat.fit(X_train,y_train)
#predicting the target value from the model for the samples
y_train_cat = cat.predict(X_train)
```

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

```
y_test_cat = cat.predict(X_test)
#computing the accuracy, f1_score, Recall, precision of the model performance
acc_train_cat  = metrics.accuracy_score(y_train,y_train_cat)
acc_test_cat = metrics.accuracy_score(y_test,y_test_cat)
print("CatBoost Classifier : Accuracy on training Data: {:.3f}".format(acc_train_cat))
print("CatBoost Classifier : Accuracy on test Data: {:.3f}".format(acc_test_cat))
print()
f1_score_train_cat = metrics.f1_score(y_train,y_train_cat)
f1_score_test_cat = metrics.f1_score(y_test,y_test_cat)
print("CatBoost Classifier : f1_score on training Data: {:.3f}".format(f1_score_train_cat))
print("CatBoost Classifier : f1_score on test Data: {:.3f}".format(f1_score_test_cat))
print()
recall_score_train_cat = metrics.recall_score(y_train,y_train_cat)
recall_score_test_cat = metrics.recall_score(y_test,y_test_cat)
print("CatBoost Classifier : Recall on training Data: {:.3f}".format(recall_score_train_cat))
print("CatBoost Classifier : Recall on test Data: {:.3f}".format(recall_score_test_cat))
print()
precision_score_train_cat = metrics.precision_score(y_train,y_train_cat)
precision_score_test_cat = metrics.precision_score(y_test,y_test_cat)
print("CatBoost Classifier : precision on training Data: {:.3f}".format(precision_score_train_cat))
print("CatBoost Classifier : precision on test Data: {:.3f}".format(precision_score_test_cat))
#computing the classification report of the model
print(metrics.classification_report(y_test, y_test_cat))
training_accuracy = []
test_accuracy = []
# try learning_rate from 0.1 to 0.9
depth = range(1,10)
for n in depth:
    forest_test =  CatBoostClassifier(learning_rate = n*0.1)
    forest_test.fit(X_train, y_train)
    # record training set accuracy
    training_accuracy.append(forest_test.score(X_train, y_train))
    # record generalization accuracy
```

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

```
    test_accuracy.append(forest_test.score(X_test, y_test))
#plotting the training & testing accuracy for n_estimators from 1 to 50
plt.figure(figsize=None)
plt.plot(depth, training_accuracy, label="training accuracy")
plt.plot(depth, test_accuracy, label="test accuracy")
plt.ylabel("Accuracy")
plt.xlabel("learning_rate")
plt.legend();
#storing the results. The below mentioned order of parameter passing is important.
storeResults('CatBoost Classifier',acc_test_cat,f1_score_test_cat,
             recall_score_train_cat,precision_score_train_cat)
# Multi-layer Perceptron Classifier Model
from sklearn.neural_network import MLPClassifier
# instantiate the model
mlp = MLPClassifier()
#mlp = GridSearchCV(mlpc, parameter_space)
# fit the model
mlp.fit(X_train,y_train)
#predicting the target value from the model for the samples
y_train_mlp = mlp.predict(X_train)
y_test_mlp = mlp.predict(X_test)
acc_train_mlp  = metrics.accuracy_score(y_train,y_train_mlp)
acc_test_mlp = metrics.accuracy_score(y_test,y_test_mlp)
print("Multi-layer Perceptron : Accuracy on training Data: {:.3f}".format(acc_train_mlp))
print("Multi-layer Perceptron : Accuracy on test Data: {:.3f}".format(acc_test_mlp))
print()
f1_score_train_mlp = metrics.f1_score(y_train,y_train_mlp)
f1_score_test_mlp = metrics.f1_score(y_test,y_test_mlp)
print("Multi-layer Perceptron : f1_score on training Data: {:.3f}".format(f1_score_train_mlp))
print("Multi-layer Perceptron : f1_score on test Data: {:.3f}".format(f1_score_train_mlp))
print()
recall_score_train_mlp = metrics.recall_score(y_train,y_train_mlp)
recall_score_test_mlp = metrics.recall_score(y_test,y_test_mlp)
```

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

```
print("Multi-layer Perceptron : Recall on training Data: {:.3f}".format(recall_score_train_mlp))

print("Multi-layer Perceptron : Recall on test Data: {:.3f}".format(recall_score_test_mlp))

print()

precision_score_train_mlp = metrics.precision_score(y_train,y_train_mlp)

precision_score_test_mlp = metrics.precision_score(y_test,y_test_mlp)

int("Multi-layer Perceptron : precision on training Data: {:.3f}".format(precision_score_train_mlp))

print("Multi-layer Perceptron : precision on test Data: {:.3f}".format(precision_score_test_mlp))

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

storeResults('Multi-layer Perceptron',acc_test_mlp,f1_score_test_mlp,

        recall_score_train_mlp,precision_score_train_mlp)

# To compare the models performance, a dataframe is created. The columns of this dataframe are the lists created to store the results of the model.

#creating dataframe

result = pd.DataFrame({ 'ML Model' : ML_Model,

                    'Accuracy' : accuracy,

                    'f1_score' : f1_score,

                    'Recall'   : recall,

                    'Precision': precision,

                   })

# dispalying total result

result

#Sorting the datafram on accuracy

sorted_result=result.sort_values(by=['Accuracy', 'f1_score'],ascending=False).reset_index(drop=True)

sorted_result

#  XGBoost Classifier Model

from xgboost import XGBClassifier

# instantiate the model

gbc = GradientBoostingClassifier(max_depth=4,learning_rate=0.7)

# fit the model

gbc.fit(X_train,y_train)

import pickle

# dump information to that file

pickle.dump(gbc, open('pickle/model.pkl', 'wb'))
```

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/IJISRT24MAY1622

```
#checking the feature importance in the model
plt.figure(figsize=(9,7))
n_features = X_train.shape[1]
plt.barh(range(n_features), gbc.feature_importances_, align='center')
plt.yticks(np.arange(n_features), X_train.columns)
plt.title("Feature importances using permutation on full model")
plt.xlabel("Feature importance")
plt.ylabel("Feature")
plt.show()
```