A Project Report

On

"MALICIOUS URLS DETECTION USING DATA ANALYTICS"

Submitted to

CHADALAWADA RAMANAMMA ENGINEERING COLLEGE (AUTONOMOUS), TIRUPATI

In partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

Submitted by

N. GUNA VARDHAN REDDY	(20P11A0579)
P. DIVYA SREE	(20P11A0598)
V. VIVEKANANDA	(20P11A05C6)
P. SYAMALA	(18P11A0585)

Under the esteemed guidance of

Mrs. N. ANITHA

ASSISTANT PROFESSOR

CHADALAWADA RAMANAMMA ENGINEERING COLLEGE (AUTONOMOUS)

(Accredited by NAAC with 'B' Grade, Approved by AICTE, New Delhi & Affiliated to JNTU Anantapur)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Tirupait-517506, Andhra Pradesh, India

(2020-2024)

CHADALAWADA RAMANAMMA ENGINEERING COLLEGE (AUTONOMOUS)

(Accredited by NAAC with 'B' Grade, Approved by AICTE, New Delhi & Affiliated to JNTU Anantapur)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Certificate

This is to Certify that this project report entitled "**MALICIOUS URLS DETECTION USING DATA ANALYTICS**" is a carried out by **N. Guna Vardhan Reddy (20P11A0579), P. Divya Sree (20P11A0598), V. VIVEKANANDA (20P11A05C6), P. Syamala (18P11A0585)**, in the department of "**COMPUTER SCIENCE AND ENGINEERING**", and is submitted to **Chadalawada Ramanamma Engineering College (Autonomous), Tirupati** is a Project Work carried out by them under my guidance during the academic year 2023-2024.

GUIDE	**HEAD**	**PRINCIPAL**
Mrs. N. Anitha , MTech	**Dr. D. Shobha Rani, Ph.D**	**Dr. P. Ramesh Kumar, Ph.D**
Assistant Professor	Professor & HOD	CREC(A)

Submitted for viva voce exam held on ___________________________________

Internal Examiner **External Examiner**

DECLARATION

I hereby declare that the Project Report titled “**MALICIOUS URLS DETECTION USING DATA ANALYTICS**” submitted to Chadalawada Ramanamma Engineering College(Autonomous), affiliated to Jawaharlal Technological University, Ananthapur (JNTUA) for the award of the Degree of Bachelor of Technology in Computer Science and Engineering is a result of original research carried-out in this report. It is further declared that the Project Report or any part thereof has not been previously submitted to any University or Institute for the award of degree.

N. Guna Vardhan Reddy	**(20P11A0579)**
P. Divya Sree	**(20P11A0598)**
V. Vivekananda	**(20P11A05C6)**
P. Syamala	**(18P11A0585)**

ACKNOWLEDGEMENT

First, we are thankful to our guide **Mrs. N. ANITHA** for her valuable guidance and encouragement. His helping attitude and suggestions have helped us in the successful completion of the project. Successful completion of any project cannot be done without proper support and encouragement.

We sincerely thank the **Chairman Dr. CHADALAWADA KRISHNA MURTHY** for providing all the necessary facilities during the course of study.

I am highly indebted to **Principal Dr. P. RAMESH KUMAR** for the facilities provided to accomplish this project.

I would like to thank my **Dean Dr. C. SUBHAS** for his constructive criticism throughout my project.

I wish to convey my special thanks to **Dr. D. SHOBHA RANI** Head of Computer Science and Engineering in Chadalawada Ramanamma Engineering College, for giving the required information in doing my project.

I would like to thank Project coordinator **Mrs. C. SARASWATHI** for their support and advices to get and complete project work for her guidance and regular schedules. I am extremely great full to my department staff members who helped me in successful completion of this project.

We would like to thank our parents and friends, who have the greatest contributions in all our achievements, for the great care and blessings in making us successful in all our endeavors.

N. Guna Vardhan Reddy	**20P11A0579**
P. Divya Sree	**20P11A0598**
V. Vivekananda	**20P11A05C6**
P. Syamala	**18P11A0585**

ABSTRACT

Phishing is a type of social engineering attack where attackers deceive victims into providing their login information on a form that sends the information to a malicious server. One vs All Classification is a powerful machine learning technique that can detect dangerous URLs by classifying them as belonging to one of several categories, such as malware, phishing, or spam. This algorithm can detect malicious URLs by classifying them as belonging to one of these categories such as malware, phishing, or spam. This method requires training numerous binary classifiers, such as Decision Tree Classifier, Random Forest Classifier, AdaBoost Classifier, KNeighbors Classifier, SGD Classifier, Extra Trees Classifier, Gaussian NB each of which is responsible for differentiating one category from the others. It is possible to train the system using a small dataset that includes only the relevant aspects of the URLs, such as the domain and the path. Our objective of the project is to provide a highly accurate URLs detection system which can be utilized through website.

KEYWORDS: URL, Dataset, Classify, Accuracy, Machine learning

CONTENTS

LIST OF FIGURES

ABBREVIATIONS

ML	Machine Learning
HTML	Hyper Text Markup language
URL	Uniform Resource Locater

CHAPTER 1
INTRODUCTION

In the ever-evolving landscape of cybersecurity, malicious URLs pose a significant threat to individuals and organizations alike. These nefarious links are designed to deceive users into divulging sensitive information, downloading malware, or becoming part of a botnet. The detection and neutralization of such URLs are paramount in safeguarding digital assets and maintaining the integrity of online interactions. Data analytics emerges as a powerful ally in this battle, offering sophisticated tools and methodologies to identify and counteract these cyber threats effectively.

The spread of malicious URLs is fueled by the ease with which attackers can mask their true intentions. Phishing scams, drive-by downloads, and social engineering tactics are just a few of the methods employed to exploit unsuspecting users. Traditional security measures, while necessary, are often reactive and struggle to keep pace with the innovative techniques of cybercriminals. This is where data analytics steps in, transforming the fight against malicious URLs from a reactive stance to a proactive strategy.

Data analytics leverages large datasets, machine learning algorithms, and pattern recognition to discern the characteristics of URLs that may be harmful. By analyzing vast amounts of web traffic data, security systems can learn to detect anomalies and red flags that often indicate a URL's malicious intent. Features such as the URL's length, domain name, the presence of certain keywords, and the use of obfuscation techniques are scrutinized to assess the risk level.

1.1 PROBLEM

To minimize and prevent the cyber-attacks through URLs, a website to detect the URLs and suggests the user whether they should continue opening the URL or to be cautious while opening the URL or avoid opening the URL completely. The website detects the user entered URL with the help of machine learning model which is trained with one v/s all classifier, the one v/s all classifier uses a binary classifier with at most accuracy in output.

1.2 RESEARCH QUESTION

To which extent does "**one v/s all classifier**" outperform the regression and classification models, clustering approaches, hidden Markov models, and various sequential models. The one v/s all classifier is trained with the help of other binary classifiers, we select the binary classifier with at most accuracy, precision, recall and F1 score.

1.3 PURPOSE

The main purpose of this thesis is designing a website to detect malicious URLs using data analytics and make the users aware of dangers in clicking unknown URLs. By using the one v/s all classifier which in turn uses a binary classifier, the binary classifier is selected through data analysis of dataset and how the binary classifier models such as Decision Tree Classifier, Random Forest Classifier, AdaBoost Classifier, KNeighbors Classifier, SGD Classifier, Extra Trees Classifier, Gaussian NB. The binary classifier with highest accuracy, precision, recall and F1 score is selected to train the one v/s all classifier.

1.4 GOALS

The goal of the project is to design a website that detects malicious URLs based on the user input. This has been divided into three sub-goals:

1. Analysing the dataset using data analytics and developing binary classifier model and finding out which binary classifier has highest accuracy.
2. Using the selected binary classifier one v/s all classifier model will be developed
3. Using this model and flask module website is developed with the help of HTML and CSS

1.5 THESIS OUTLINE

Thesis is organized into 6 chapters

Chapter 2 is about Literature Survey

Chapter 3 is about analysis and design

Chapter 4 is about system requirements

Chapter 5 is about coding

Chapter 6 is about results of the project

Chapter 7 is about conclusion

CHAPTER 2
LITERATURE SURVEY

2.1 TYPES OF MALICIOUS URLS

PHISHING URLS:

Phishing URLs are perhaps the most prevalent form of malicious URLs encountered in cyberspace. These URLs masquerade as legitimate websites or services with the intention of tricking users into divulging sensitive information such as login credentials, financial details, or personal data. Phishing URLs often employ social engineering tactics to create a sense of urgency or legitimacy, enticing users to click on links or provide information without suspicion. Common examples include fake login pages for popular websites, bogus financial institutions, or fraudulent emails requesting account verification.

DEFACEMENT URLS:

Defacement URLs represent a category of malicious web addresses that are associated with a specific type of cyber-attack known as website defacement. In a defacement attack, perpetrators gain unauthorized access to a website's server or content management system (CMS) and modify its appearance by replacing legitimate content with their own messages, images, or propaganda. These alterations are often intended to embarrass the website owner, convey a political or ideological message, or simply to demonstrate the attackers' hacking prowess.

MALWARE DISTRIBUTION URLS:

Malware distribution URLs serve as conduits for delivering malicious software (malware) onto unsuspecting users' devices. These URLs may lead to websites hosting malware-infected files, exploit kits, or drive-by download attacks. Upon visiting such URLs, users may inadvertently download and execute malware, leading to a range of adverse consequences including data theft, system compromise, and unauthorized access. Malware distribution URLs often employ deceptive tactics such as enticing offers, fake software updates, or enticing multimedia content to lure users into clicking.

DRIVE-BY DOWNLOAD URLS:

Drive-by download URLs exploit vulnerabilities in web browsers or plugins to initiate automatic downloads of malware onto users' devices without their consent or awareness. These URLs are typically embedded within compromised or malicious websites and can trigger the download of malware simply by visiting the webpage. Drive-by download attacks are particularly insidious as they require minimal user interaction and can infect devices with malware capable of exploiting a wide range of vulnerabilities.

REDIRECTOR URLS:

Redirector URLs are used to redirect users from legitimate websites to malicious or phishing sites. Attackers often compromise legitimate websites and insert malicious scripts or code that redirect users to fraudulent pages controlled by the attackers. Redirector URLs may also be embedded within phishing emails, instant messages, or social media posts, directing users to fake login pages or malware distribution sites. These URLs exploit users' trust in legitimate websites to deceive them into visiting malicious destinations.

SPAM URLS:

Spam URLs are commonly found in unsolicited emails, messages, or comments and are used to promote dubious products, services, or fraudulent schemes. These URLs often lead to scam websites, counterfeit goods, or phishing pages designed to collect personal information. Spam URLs may also be used in conjunction with malware distribution or drive-by download attacks, enticing users to click on links under the guise of legitimate offers or opportunities.

TYPOSQUATTING URLS:

Typosquatting URLs capitalize on typographical errors made by users when entering website addresses. Attackers register domain names that closely resemble popular websites or brands but contain slight misspellings or variations. When users mistype a URL in their browser, they may inadvertently land on a typosquatting site designed to deceive or exploit them. Typosquatting URLs are often used in phishing attacks to impersonate legitimate websites and trick users into disclosing sensitive information.

CROSS-SITE SCRIPTING (XSS) URLS:

Cross-Site Scripting (XSS) URLs exploit vulnerabilities in web applications to inject malicious scripts or code into web pages viewed by other users. These URLs are typically used to steal session cookies, execute unauthorized actions on behalf of users, or deface websites. XSS URLs may be embedded within legitimate websites or shared through social engineering tactics such as phishing emails or malicious advertisements. They pose a significant threat to web security and can result in data breaches, account hijacking, and website defacement.

MALICIOUS SHORTENED URLS:

Malicious shortened URLs leverage URL shortening services to obfuscate the destination of links, making it difficult for users to discern their true nature. Attackers use these shortened URLs in phishing emails, social media posts, or instant messages to conceal malicious destinations such as phishing sites or malware distribution pages. Despite their brevity, malicious shortened URLs can lead to a wide range of cyber threats and are commonly used in conjunction with other malicious tactics to maximize their impact.

2.2 MALICIOUS URL DETECTION:

TRADITIONAL METHODS:

Traditional methods of malicious URL detection rely on static analysis, heuristics, and signature-based techniques to identify known malicious URLs. These methods typically involve the use of blacklists, reputation-based systems, and pattern matching algorithms to classify URLs as either benign or malicious. While effective to some extent, traditional methods are limited in their ability to detect newly emerging threats and sophisticated attack vectors. Additionally, they may suffer from high false positive rates and require frequent updates to maintain accuracy.

BEHAVIORAL ANALYSIS:

Behavioral analysis techniques analyze the behavior of users interacting with URLs to detect anomalies and signs of malicious intent. These methods often involve monitoring web traffic patterns, user interactions, and clickstream data to identify suspicious behavior indicative of phishing attempts, malware downloads, or other malicious activities. Behavioral analysis can provide valuable insights into the context and intent behind URL requests, allowing for more accurate detection and classification of malicious URLs.

CONTENT ANALYSIS:

Content analysis techniques examine the content and structure of URLs to identify signs of malicious activity. This may involve parsing URLs for suspicious keywords, analyzing domain reputation and registration information, or inspecting URL redirects and parameters for indicators of phishing or malware distribution. Content analysis can help identify obfuscated or disguised URLs used in phishing attacks, as well as detect patterns associated with known malicious domains or IP addresses.

MACHINE LEARNING APPROACHES:

Machine learning (ML) approaches have gained prominence in recent years for their ability to effectively detect and classify malicious URLs based on patterns and features extracted from large datasets. ML algorithms, including supervised learning, unsupervised learning, and deep learning techniques, can automatically learn and adapt to evolving threats, making them well-suited for the task of malicious URL detection. These methods leverage features such as URL structure, domain reputation, lexical characteristics, and user behavior to distinguish between benign and malicious URLs with high accuracy.

ENSEMBLE METHODS:

Ensemble methods combine multiple base classifiers to improve the overall performance and robustness of malicious URL detection systems. By aggregating predictions from diverse classifiers, ensemble methods can mitigate the limitations of individual classifiers and achieve higher accuracy and reliability in detecting malicious URLs. Ensemble techniques such as bagging, boosting, and stacking have been successfully applied in the context of malicious URL detection to enhance detection rates and reduce false positive errors.

HYBRID APPROACHES:

Hybrid approaches integrate multiple detection techniques, such as static analysis, behavioral analysis, and machine learning, to create more comprehensive and effective malicious URL detection systems. These approaches leverage the complementary strengths of different detection methods to achieve greater accuracy and coverage in identifying malicious URLs. By combining multiple detection techniques, hybrid approaches can improve detection rates while minimizing false positive errors and adapting to emerging threats in real-time.

The detection of malicious URLs is a critical component of modern cybersecurity efforts, given the pervasive threat posed by malicious activities conducted through URLs. Traditional methods, behavioral analysis, content analysis, machine learning approaches, ensemble methods, and hybrid approaches all play a vital role in detecting and mitigating malicious URLs. By leveraging a combination of these techniques, cybersecurity professionals can develop robust and effective URL detection systems capable of protecting users and organizations from a wide range of cyber threats. However, as the threat landscape continues to evolve, ongoing research and innovation are essential to stay ahead of adversaries and ensure the continued efficacy of malicious URL detection methods.

2.3 MALICIOUS URL PREVENTION:

USER EDUCATION AND AWARENESS:

User education and awareness programs represent a fundamental approach to preventing the dissemination and impact of malicious URLs. By educating users about common phishing tactics, warning signs of malicious URLs, and best practices for safe browsing, organizations can empower users to recognize and avoid potential threats. Training programs may include simulated phishing exercises, interactive tutorials, and regular security awareness campaigns to reinforce safe browsing habits and promote a security-conscious culture.

URL FILTERING AND BLACKLISTING:

URL filtering and blacklisting solutions are designed to proactively block access to known malicious URLs based on predefined criteria, such as domain reputation, URL structure, or content characteristics. These solutions leverage threat intelligence feeds, reputation databases, and machine learning algorithms to identify and categorize malicious URLs in real-time. By blocking access to malicious URLs at the network level, URL filtering and blacklisting solutions help prevent users from inadvertently accessing malicious content and mitigate the risk of infection.

WEB CONTENT FILTERING:

Web content filtering solutions provide granular control over the types of content that users can access on the internet, including websites hosting malicious URLs. These solutions allow organizations to define policies to block access to specific categories of websites, such as gambling, adult content, or known malicious domains. By enforcing content filtering policies at the network perimeter or endpoint level, organizations can reduce the likelihood of users encountering malicious URLs and mitigate the risk of exposure to cyber threats.

SECURE WEB BROWSERS AND EXTENSIONS:

Secure web browsers and browser extensions incorporate built-in security features and capabilities to protect users against malicious URLs and phishing attacks. These features may include anti-phishing filters, malicious URL detection, sandboxing, and browser isolation technologies to prevent unauthorized access to sensitive data and resources. By using secure web browsers and extensions, users can reduce the risk of falling victim to phishing scams, malware downloads, and other malicious activities perpetrated through URLs.

DNS FILTERING AND DNS SECURITY:

DNS filtering solutions intercept and inspect DNS requests to identify and block access to known malicious domains and URLs. By redirecting DNS queries to a filtering service or proxy server, organizations can enforce policies to block access to malicious websites, phishing domains, and other threat sources. DNS security solutions, such as DNSSEC (DNS Security Extensions) and DNS-over-HTTPS (DoH), provide additional layers of protection by encrypting DNS traffic and validating the authenticity of DNS responses, thereby mitigating the risk of DNS-related attacks and DNS spoofing.

ENDPOINT PROTECTION AND ANTIVIRUS SOFTWARE:

Endpoint protection and antivirus software play a critical role in preventing users from accessing malicious URLs and executing malware payloads. These solutions employ signature-based detection, heuristics, and behavior analysis techniques to identify and block known malware strains, phishing attempts, and malicious URLs in real-time. By deploying endpoint protection and antivirus software on endpoints, including desktops, laptops, and mobile devices, organizations can detect and neutralize threats before they can inflict damage.

EMAIL SECURITY GATEWAYS:

Email security gateways are designed to protect organizations against email-based threats, including phishing attacks, malicious attachments, and links to malicious URLs. These solutions employ advanced threat detection capabilities, including machine learning, natural language processing, and behavioral analysis, to identify and block suspicious emails containing malicious URLs. By scanning inbound and outbound email traffic for signs of malicious activity, email security gateways help prevent users from falling victim to phishing scams and malware downloads.

CONTINUOUS MONITORING AND INCIDENT RESPONSE:

Continuous monitoring and incident response capabilities are essential for detecting and mitigating malicious URLs and cyber threats in real-time. By implementing robust monitoring solutions and incident response processes, organizations can quickly identify and respond to security incidents involving malicious URLs, phishing attacks, and malware infections. Proactive monitoring of network traffic, endpoint activity, and user behavior enables organizations to detect anomalous patterns and indicators of compromise, facilitating rapid incident response and containment.

Preventing the proliferation and impact of malicious URLs requires a multi-faceted approach that encompasses user education, proactive security measures, and reactive incident response capabilities. By combining user awareness programs, URL filtering and blacklisting solutions, secure web browsers, DNS filtering, endpoint protection, email security gateways, and continuous monitoring, organizations can effectively mitigate the risk of encountering malicious URLs and protect against a wide range of cyber threats. However, achieving robust URL prevention requires ongoing vigilance, investment in security technologies, and collaboration across organizational boundaries to stay ahead of adversaries and safeguard digital assets.

2.4 EXISTING SYSTEM

Malicious identification systems based on outlier-detection strategies for a CR cooperative sensing device are currently in use. Both sensors transmit their energy transmitter outputs to an access point, which uses a big data and understanding to determine the presence of a primary signal. Throughout each sensing iteration, we explored different rigorous methods for assigning outlier variables to the apps. These outlier variables are used by malicious user identification systems to identify malicious users and reduce their effect on the sensor system's performance.

Disadvantages:

1. This model is based on observational evidence, and requires a large amount of computational work.
2. Sensing data rapidly becomes redundant due to problems such as channel loss and receiver instability, rendering future decision-making impossible.
3. The channel has a lot of interference.

2.5 PROPOSED SYSTEM

The machine is given a dataset of malicious and valid URLs, which is then pre-processed so that the data can be used for analysis. Around 30 attributes of malicious websites are included in the features, which are used to separate them from legitimate ones. Each type has its own set of malicious qualities and standards that must be adhered to. For each URL, the defined characteristics are extracted, and appropriate input ranges are defined. Each malicious website danger is then given one of these values. The ranges for each input vary from 0 to 10, while the output ranges from 0 to 1. The values of malicious attributes are represented by binary numbers 0 and 1, showing whether the attribute is present or not.

After the data has been conditioned, we can make the usage of a machine learn algorithm to analyze the dataset. The algorithms for machine learning have already been discussed in a proper section. Following that, we can use a hybrid classification method in which we combine two classifiers, SVM and Logistic regression, to estimate the precision of the phishing URL detection, resulting in the desired outcome.

CHAPTER 3
ANALYSIS AND DESIGN

3.1 PROBLEM STATEMENT:

The objective is to develop a robust system for detecting and mitigating malicious Uniform Resource Locators (URLs) using data analytics techniques. The system should be capable of accurately distinguishing between benign and malicious URLs in real-time to prevent users from accessing malicious content and falling victim to cyber-attacks.

3.2 DATA COLLECTION AND PREPROCESSING:

- **Data Sources**: Gather a diverse dataset of URLs from various sources, including web traffic logs, threat intelligence feeds, and public repositories.
- **Feature Extraction**: Extract relevant features from the URLs, such as domain reputation, URL structure, lexical characteristics, content semantics, and user behavior.
- **Data Cleaning**: Perform data cleaning and preprocessing steps to remove duplicates, missing values, and irrelevant features. Standardize and normalize the data to ensure consistency and accuracy.

EXPLORATORY DATA ANALYSIS (EDA):

Conduct exploratory data analysis to gain insights into the distribution and characteristics of the dataset. Visualize the distribution of benign and malicious URLs across different features to identify patterns and anomalies.

FEATURE ENGINEERING:

Engineer new features based on domain knowledge and insights gained from EDA to enhance the predictive power of the models. Explore techniques such as domain reputation scoring, URL entropy calculation, n-gram analysis, and user behavior profiling to capture nuanced characteristics of malicious URLs.

MODEL SELECTION AND TRAINING:

Experiment with various machine learning algorithms, including supervised, unsupervised, and semi-supervised learning techniques. Evaluate the performance of different models using metrics such as accuracy, precision, recall, F1-score. Consider ensemble methods, such as bagging, boosting, and stacking, to combine multiple models and improve overall performance.

MODEL EVALUATION AND VALIDATION:

Split the dataset into training, validation, and test sets to assess the generalization performance of the models. Perform cross-validation and hyperparameter tuning to optimize the models and prevent overfitting. Evaluate the robustness of the models against different types of malicious URLs, including phishing, malware distribution, and fraudulent schemes.

SCALABILITY AND PERFORMANCE:

Design the system to be scalable and capable of handling large volumes of URL traffic. Optimize the performance of the detection pipeline to minimize latency and ensure timely response to threats. Consider distributed computing frameworks, such as Apache Spark or TensorFlow, to parallelize computation and improve throughput.

CONTINUOUS LEARNING AND ADAPTATION:

Implement mechanisms for continuous learning and adaptation to stay abreast of evolving threats and attack vectors. Regularly update the models with new data and retrain them to incorporate the latest insights and patterns. Integrate feedback loops to capture user feedback and security incident reports for further model refinement.

WEBPAGE DEVELOPMENT:

Develop a user-friendly webpage that serves as the interface for the malicious URL detection system. Design the webpage with intuitive navigation, clear visualizations, and interactive features to enhance usability and user experience. Incorporate functionality for users to input URLs for analysis, view detection results, and receive actionable insights and recommendations. Implement responsive design principles to ensure compatibility and optimal viewing experience across different devices and screen sizes.

TESTING AND QUALITY ASSURANCE:

Conduct thorough testing of the webpage and backend infrastructure to identify and address any bugs, errors, or performance issues. Perform unit tests, integration tests, and end-to-end tests to validate the functionality and reliability of the system. Implement security testing and vulnerability assessments to identify and mitigate potential security risks and threats.

3.3 UML DIAGRAMS

Unified Modeling Language (UML) is an acronym for "Unified Modeling Language." UML is a standardized general-purpose modelling language used in the field of object-oriented software creation. The Object Management Community in charge of the standard, and it was developed by them.

The ultimate aim is for the UML that becomes a standard of modelling objective-orientedcomputer application. UML has two major components into present form: meta model and notation. Any kind of system and procedure can be applied to, connected with the UML in our future.

The Unified Modeling (UML) is the basic language to describing, visualizing, constructing, reporting software systematic objects, as well the business modelling, other non-software structure.

The UML is collection of validated design principles for modelling large and complex structures. The UML is a critical component of the object-oriented software architecture and the software creation process. To express the architecture of software projects, the UML primarily employs graphical notes.

GOALS:

The first goals in designing of the UML are follows:

1. User should be able to create and share meaningful templates using a ready to usevisual modelling language.
2. To expand the key principles, include frameworks for extendibility and specialization.
3. Be unconstrained by programming languages or implementation processes.
4. With a ready-to-use, expressive digital modelling language, users should be able tobuild and exchange meaningful models.
5. Build a systematic foundation for comprehending the modelling language.
6. Encourage the demand for OO tools to extend.
7. Higher-level programming principles like alliances, models, trends, and modulesshould be supported.
8. Good practices should be included.

3.3.1 USE CASE DIAGRAM:

A use cases diagram is the type of behavior diagrams described by generated from the Use-case studies in the Unified Modeling Language (UML). It is the aim is to provide a graphical representation of a system function in common terms of actor, priorities (represents as the use cases), any dependency between these use cases. A usage cases diagram key aim is to demonstrate what framework is being used will performed for either actor. Roles of these actors in those systems are being depicted.

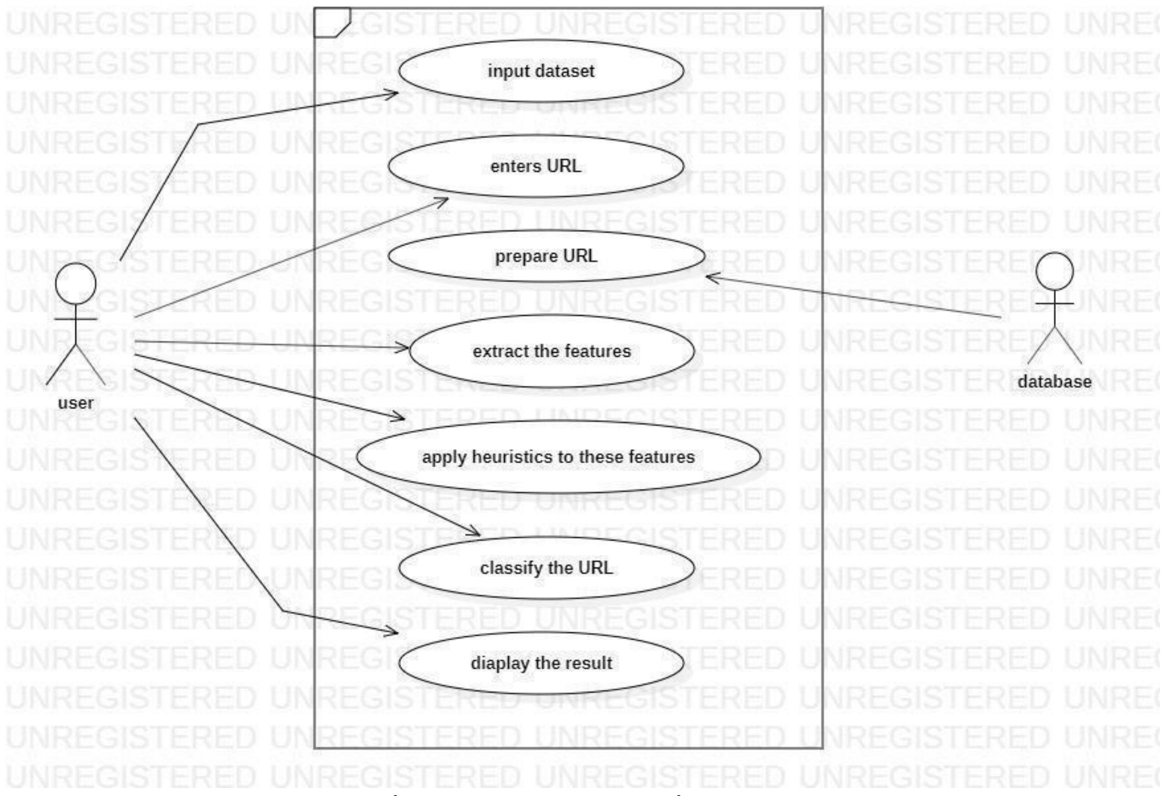


Figure 3.1: Use case Diagram

3.3.2 CLASS DIAGRAM:

A class diagram would be used to describe, characterize, and record numerous elementsof a system, and perhaps to create executable code for a computer system. The attributes and procedures of a class, and perhaps even the system's limitations, are portrayed in a class diagram. Class measures are widely used in the modelling of attribute structures since they are the only UML diagrams that can be explicitly mapped for object-oriented languages.

A class diagram portrays a set of classes, interfaces, alliances, collaborations, and constraints.

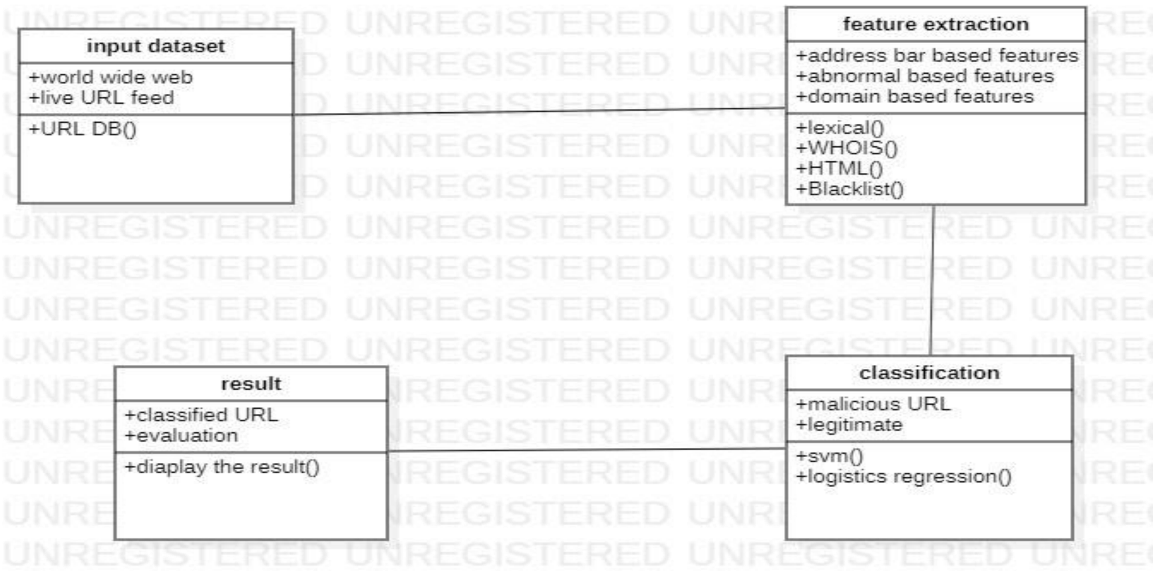


Figure 3.2: Class Diagram

3.3.3 SEQUENCE DIAGRAM:

In the Unified Modeling (UML), a sequence is the type that the activity diagrams that depicts however processes communicate one another and in which else order. It's a Line Chart Map construct. Case diagrams, scatter graph, and timing graphs are all terms usedto describe sequence diagrams.

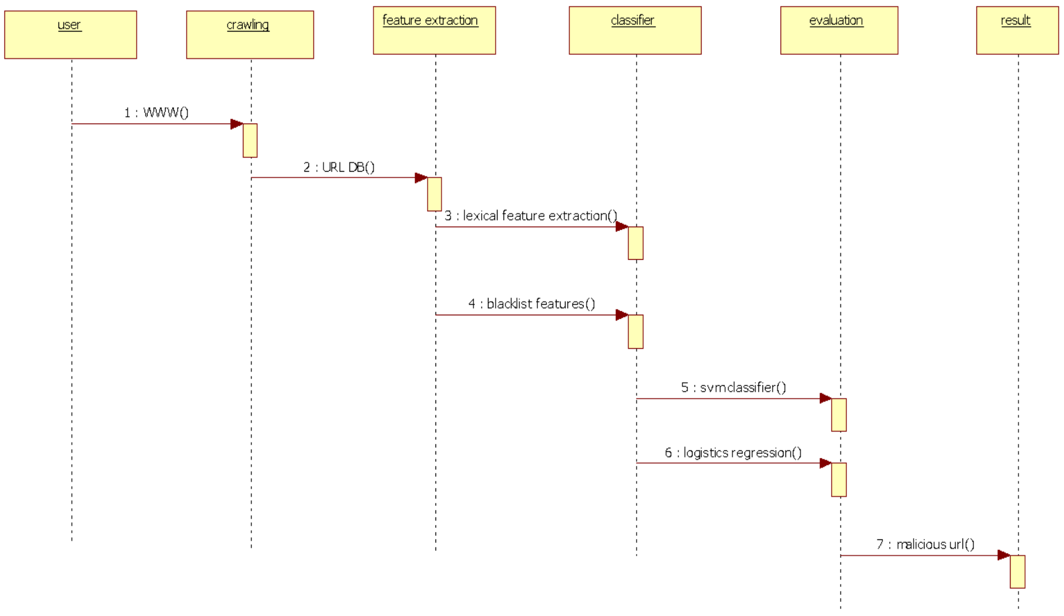


Figure 3.3: Sequence Diagram

3.3.4 ACTIVITY DIAGRAM

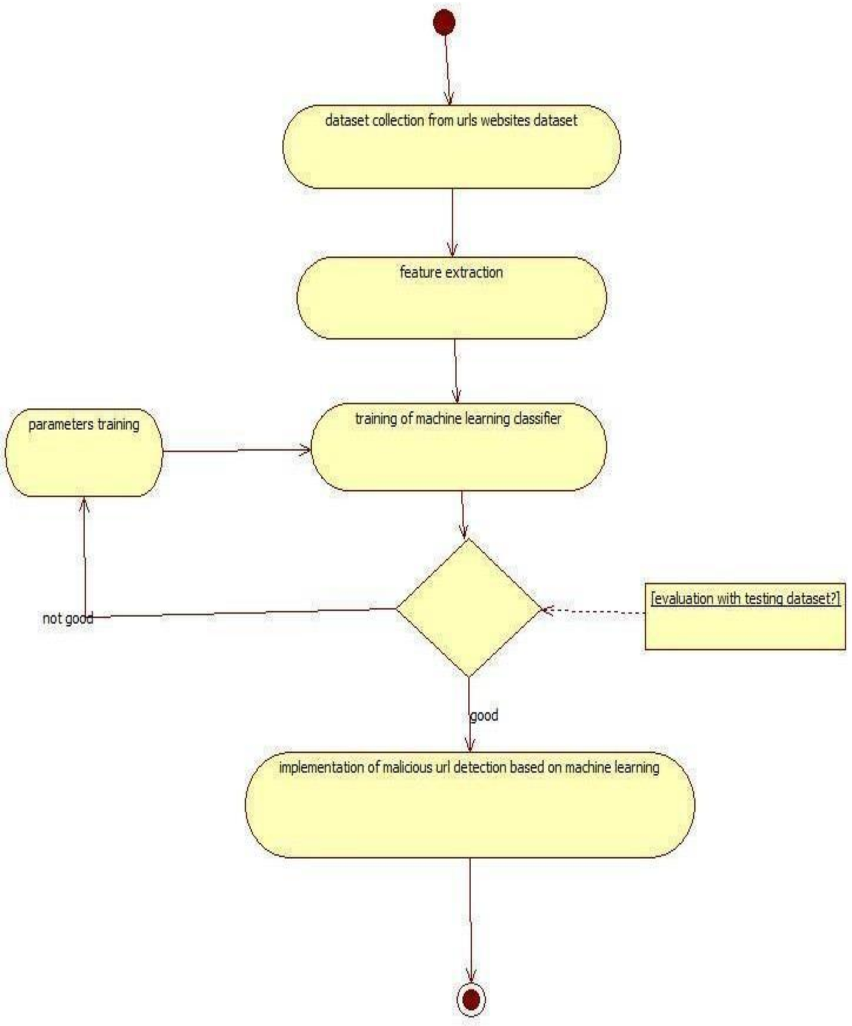

dataset collection from urls websites dataset
feature extraction
parameters training
training of machine learning classifier
not good
[evaluation with testing dataset?]
good
implementation of malicious url detection based on machine learning

Figure 3.4: Activity Diagram

3.3.5 State diagram

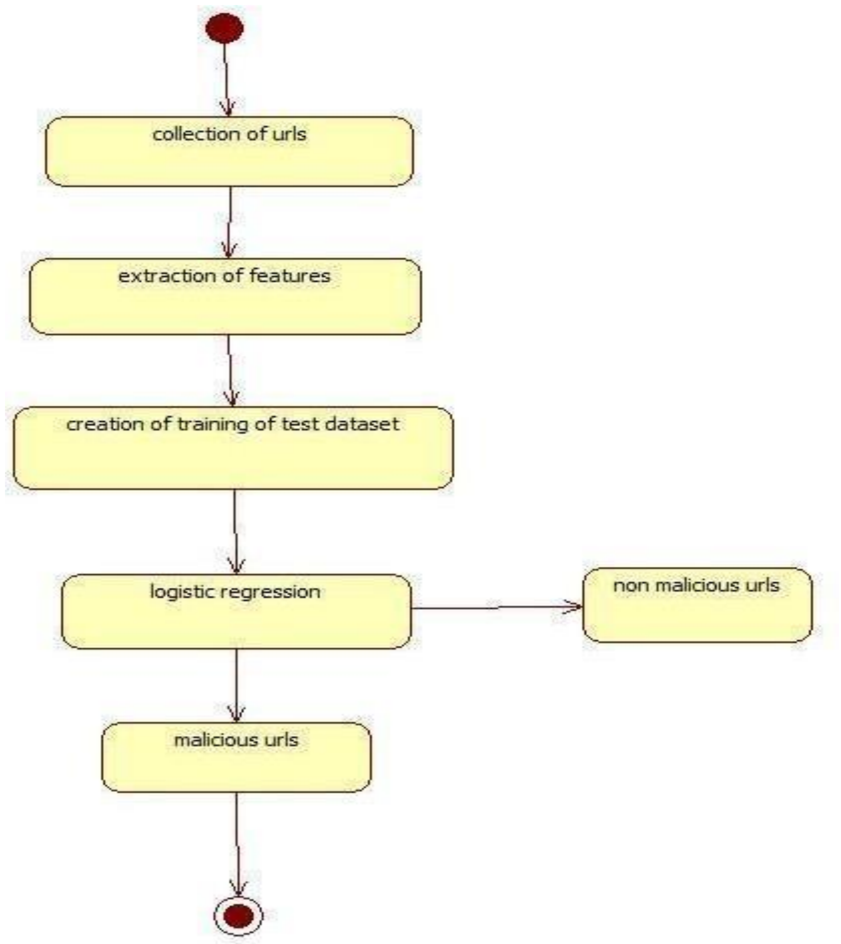


Figure 3.5: State Diagram

3.3.6 DEPLOYMENT DIAGRAM

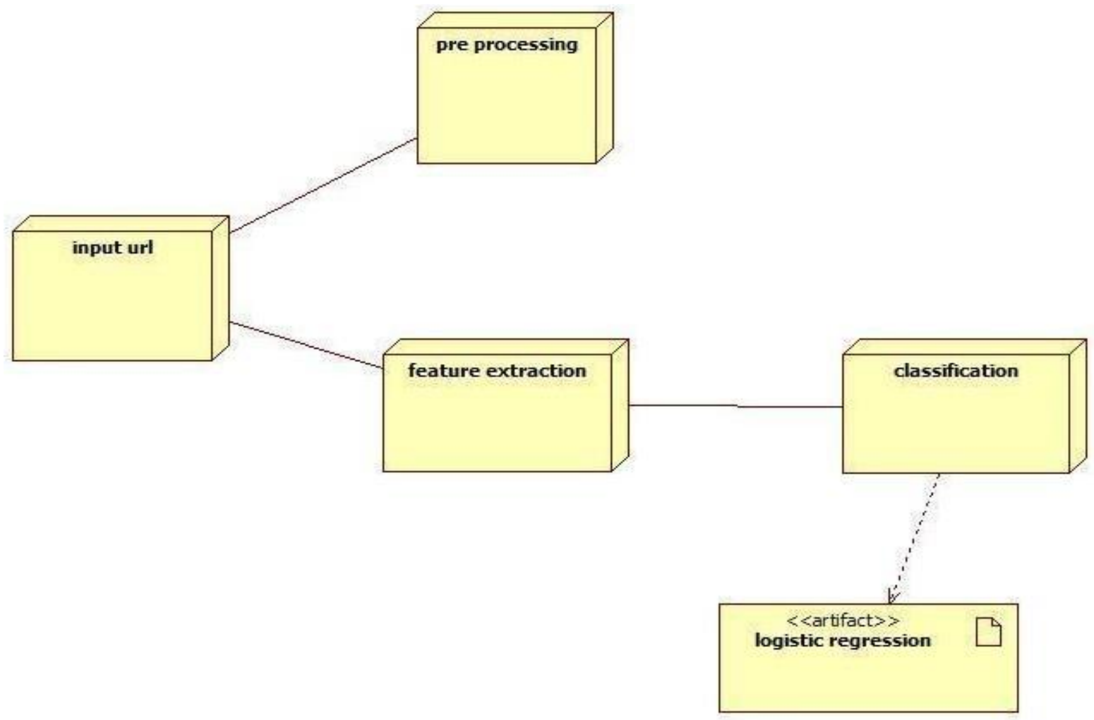


Figure 3.6: Deployment Diagram

SYSTEM IMPLEMENTATION

Software is categorized into modules, which are individually labelled and addressable elements that are combined to fulfil issue specifications. Modularity is the single feature of software that enables a program to be managed mentally. A module is a program's component. Programs are made up of one or more separately created modules that aren't joined together until the end. A single module can contain one or several routines.

MODULES:

- Data Pre-processing
- Feature Extraction
- Model Creation
- Prediction

DATA PRE-PROCESSING

We load the metadata into this pre-processing step, then apply the metadata to the data and replace the transformed data with metadata. The data would then be carried on, with the unnecessary data in the list being removed and the data being divided into train andtest data. To break the data into train and test, we'll need to import train test split from scikit-learn. This will assist the pre-processed data in splitting the data into train and test based into the weights defined on the platform code. The test and train are divided by 0.2 and 0.8, or 20 and 80 percent, respectively.

FEATURE EXTRACTION

Feature extraction is a dimensionality reduction method that reduces a large collection of raw data into smaller categories for processing. The vast number of variables in these large data sets necessitates a lot of computational power to process. Feature extraction refers to methods for selecting and/or combining variables into features into order reduces the quantity of the data that needs to be processed while still correctly and fully describing the existing environment.

MODEL CREATION

We create data into two models:

- Training model
- Testing model

The test and train are divided by 0.2 and 0.8, which equals 20% and 80%, respectively.

For the training portion, we use a machine learning algorithm to assess the model's accuracy. The svm and logistic regression algorithms are used.

PREDICTION

This module is built on the user interface. Bootstrap is used to build a web page. Enter the URL for the web page. We now obtain data from the customer in order to compare the dataset values. Finally, it can determine whether the user is malicious or not.

CHAPTER-4
SYSTEM REQUIREMENTS

4.1 HARDWARE REQUIREMENTS:

Physical computing tools, also known as hardware, are the most basic set of specifications specified by the operating system and the software application. In these cases of operating systems, the hardware specification list is the often followed by the hardware configuration process. The below are the minimum hardware requirements:

1. Processor: pentium iv
2. Ram: 8 Gb
3. Processor: 2.4 ghz
4. Main memory: 8Gb ram
5. Processing speed: 600 mhz
6. Hard disk drive: 1tb
7. Keyboard:104 keys

4.2 SOFTWARE REQUIREMENTS:

Computer specifications are concerned with specifying the tools and prerequisites that must be implemented on a device in order for an application to work. These prerequisites must be installed before the app can be installed. The below are the minimum programme requirements:

1. FRONT END: PYTHON
2. DATASET: CSV
3. IDE: ANACONDA AND PYCHARM
4. OPERATING SYSTEM: WINDOWS 10

METHODOLOGIES

4.3 LANGUAGE SPECIFICATION:

PYTHON LANGUAGE

Guido Rossum developed Python in 1989 as an object-oriented programming language. It'sperfect for quick prototyping of complicated software. It can be extended to C or C++ and has interfaces to several OS device calls and libraries. NASA, Google, YouTube, BitTorrent, and other major organizations use the Python programming language. Python programming is commonly used in specialized areas of computer science such as Artificial Intelligence, Natural Language Generation, Neural Networks, and others. Python puts a heavy emphasis on code readability, and this class will teach you the fundamentals of the language.

APPLICATIONS OF PYTHON PROGRAMMING:

WEB APPLICATIONS

Frameworks and CMS (Content Management Systems) based on Python can be used to build scalable Mobile Applications. Django, Plone, Pyramid, and Django CMS is some other of most common tools to designing Apps. Python used to control websites such as Mozilla, Instagram, PBS, Reddit.

SCIENTIFIC, NUMERIC COMPUTING

Python has a plethora is libraries for the scientifical and numerical programming. In general purpose programming, libraries such as SciPy and NumPy are used. There are also specialised libraries, such as EarthPy for earth science and AstroPy for astronomy, among others. Computer learning, data mining, and deep learning both use the term extensively.

CREATING SOFTWARE PROTOTYPES

When compared to compiled languages like C++ and Java, Python is sluggish. If resources are scarce and performance is needed, it cannot be the best solution. Python, on the other hand, is an excellent language for prototyping. Consider the following scenario: You can start by making a demo for your game using Pygame (a game creation library). If you like the demo, you can make the game using a language like C++.

ANACONDA

Anaconda is the free and open-source Python and R programming language distribution that is simple to set up. Anaconda is a software environment for mathematical computation,computer science, predictive analysis, and deep learning.

Anaconda 5.3 is the most recent distribution, which was launched in October of 2019. It contains the module, an environmental manager, and the library at over 1000 open-source packagers, all of which come with free community support.

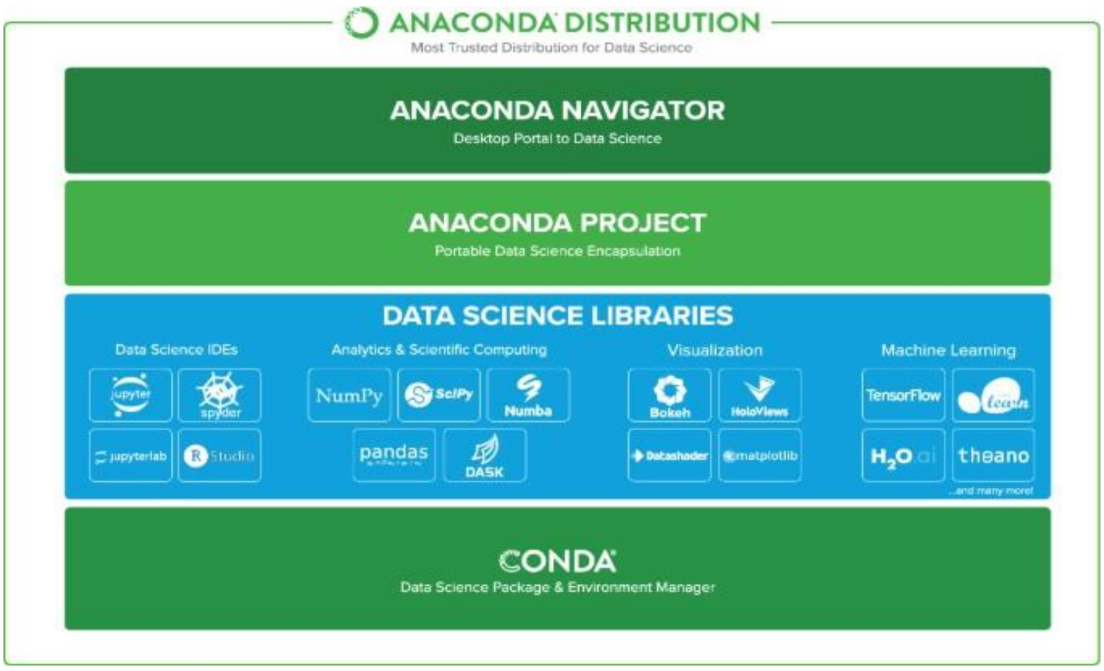


Figure 4.1. Anaconda

Applications Provided in Anaconda Distribution

The Anaconda distribution with the following application with the usage of the Anaconda Navigator.

1. Jupyter Notebook
2. JupyterLab
3. Qt Console
4. Glueviz
5. Spyder
6. RStudio
7. Orange3
8. Visual Studio Code

4.4 MACHINE LEARNING ALGORITHMS:

Machine learning algorithms are computational models that enable machines, particularly computers, to learn from data and make predictions or decisions without being explicitly programmed. These algorithms utilize statistical techniques to recognize patterns, learn from experience, and improve their performance over time as they are exposed to more data.

4.5 ONE V/S ALL ALGORITHM:

One-vs-all Classification, also known as One-vs-rest Classification, is a machine learning technique used for multi-class classification tasks. In multi-class classification, there are more than two classes, and the goal is to assign each input instance to one of the classes.

In One-vs-all Classification, a binary classification model is trained for each class, treating that class as the positive class and the rest of the classes as the negative class. This means that for a dataset with N classes, N binary classifiers will be trained. During prediction, the model with the highest confidence score is selected as the predicted class.

The One-vs-all Classification technique works by transforming a multi-class classification problem into multiple binary classification problems. Each binary classifier is trained to distinguish between one class and the rest of the classes.

During the training phase, the binary classifiers are trained using labelled data, where the positive class is the current class being considered, and the negative class is the combination of all other classes. The classifiers learn to distinguish between the positive and negative classes based on the available features. During the prediction phase, each binary classifier is applied to an unlabelled instance, and the class with the highest confidence score is assigned to that instance.

One-vs-all Classification is an important technique in machine learning because it allows us to tackle multi-class classification problems using binary classifiers, which are typically easier to train and interpret. By breaking down the problem into multiple binary classification tasks, One-vs-all Classification provides a simple and effective way to classify instances into multiple classes. It also allows for the use of any binary classification algorithm, providing flexibility in model selection.

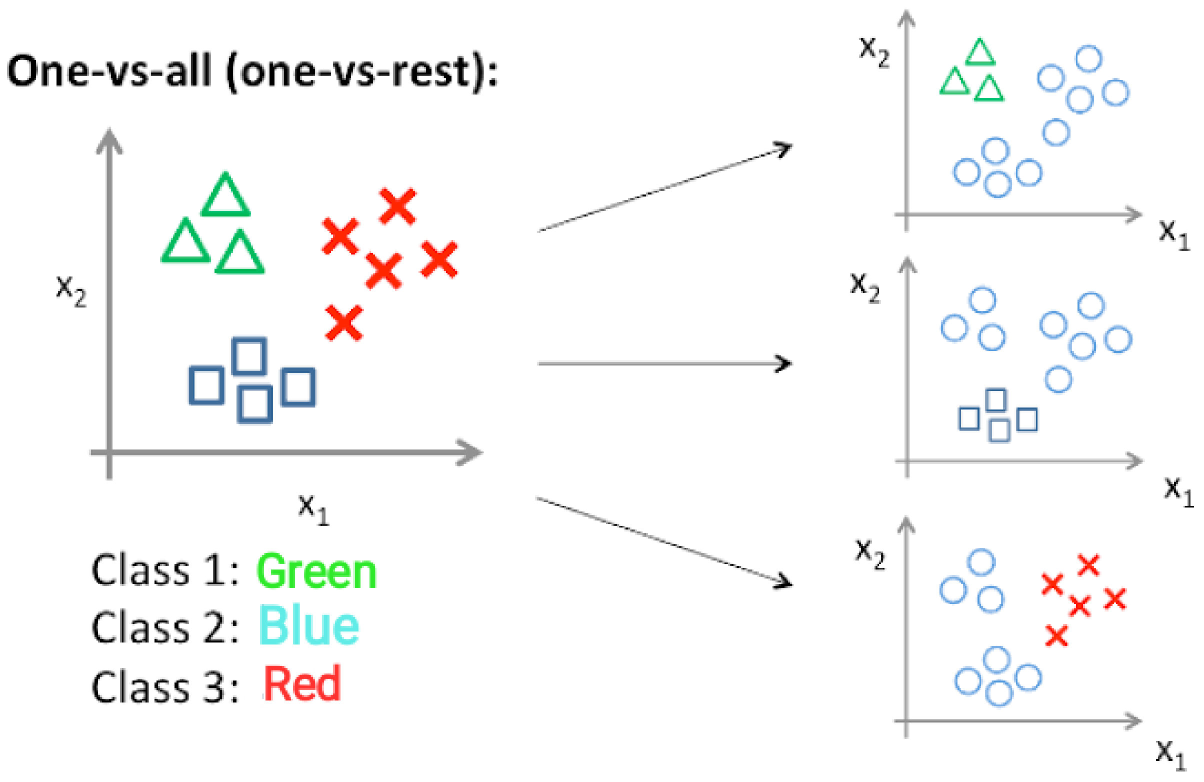


Figure 4.2. One v/s All

PSEUDO CODE FOR ONE V/S ALL ALGORITHM

```
# Train a binary classifier for each class.
for class in classes:
  classifier = train_classifier(class, other_classes)


# Classify a new sample.
def classify(sample):
  predictions = []
  for classifier in classifiers:
    prediction = classifier.predict(sample)
    predictions.append(prediction)


  # Return the class with the highest prediction score.
  return max(predictions, key=lambda x: x[1])
```

4.6 MATPLOTLIB

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, web application servers, and four graphical user interface toolkits. It provides a variety of functions to visualize data and create static, animated, and interactive plots. Matplotlib is widely used for tasks ranging from simple line plots to complex visualizations.

The Visualization Design using matplotlib

- Bar Graph
- Pie Chart
- Box Plot
- Histogram
- Line Chart and Subplots
- Scatter Plot

4.7 BAR GRAPH:

Bar graphs are best used when we need to compare the quantity of categorical values within the same category. Bar graphs should not be used for continuous values.

Bar graph is generated using **plt.bar()** in matplotlib

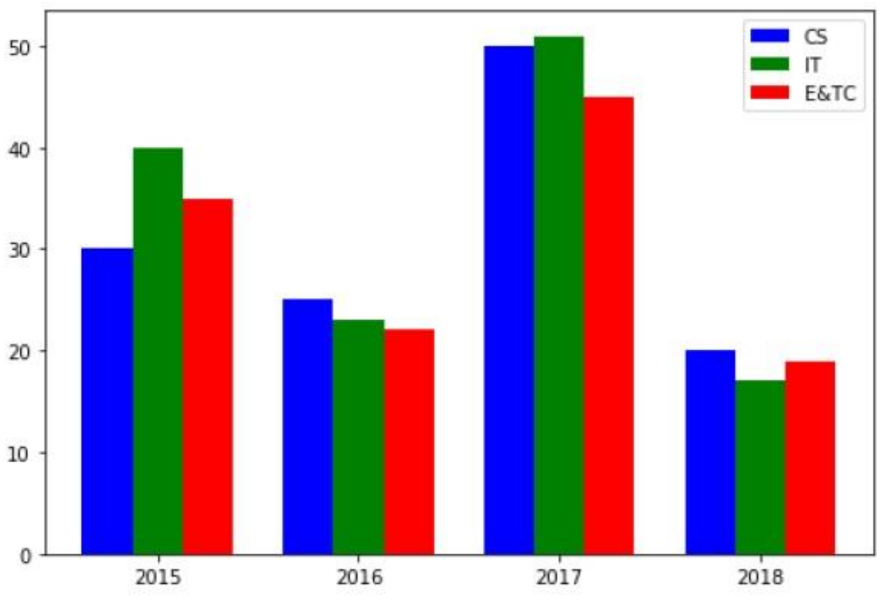


Figure 4.3. Bar Graph

4.8 SEABORN

Seaborn is a statistical data visualization library based on Matplotlib. Seaborn is a powerful and flexible data visualization library in Python that offers an easy-to-use interface for creating informative and aesthetically pleasing statistical graphics. It provides a range of tools for visualizing data, including advanced statistical analysis, and makes it easy to create complex multi-plot visualizations. Seaborn's key benefit lies in its capability to generate attractive plots with minimal coding efforts. It provides a range of default themes and color palettes, which you can easily customize to suit your preferences. Additionally, Seaborn offers a range of built-in statistical functions, allowing users to easily perform complex statistical analysis with their visualizations. Another notable feature of Seaborn is its ability to create complex multi-plot visualizations. With Seaborn, users can create grids of plots that allow for easy comparison between multiple variables or subsets of data. This makes it an ideal tool for exploratory data analysis and presentation.

PLOT TYPES IN SEABORN

- **LINE PLOT**: Line plots are used to visualize trends in data over time or other continuous variables. In a line plot, each data point is connected by a line, creating a smooth curve. In Seaborn, line plots can be created using the lineplot() function.
- **HISTOGRAM**: Histograms visualize the distribution of a continuous variable. In a histogram, the data is divided into bins and the height of each bin represents the frequency or count of data points within that bin. In Seaborn, histograms can be created using the `histplot()` function.
- **BOX PLOT**: Box plots are a type of visualization that shows the distribution of a dataset. They are commonly used to compare the distribution of one or more variables across different categories.
- **VIOLIN PLOT**: A violin plot is a type of data visualization that combines aspects of both box plots and density plots. It displays a density estimate of the data, usually smoothed by a kernel density estimator, along with the interquartile range (IQR) and median in a box plot-like form. The width of the violin represents the density estimate, with wider parts indicating higher density, and the IQR and median are shown as a white dot and line within the violin.

- **HEATMAP**. A heatmap is a graphical representation of data that uses colors to depict the value of a variable in a two-dimensional space. Heatmaps are commonly used to visualize the correlation between different variables in a dataset.

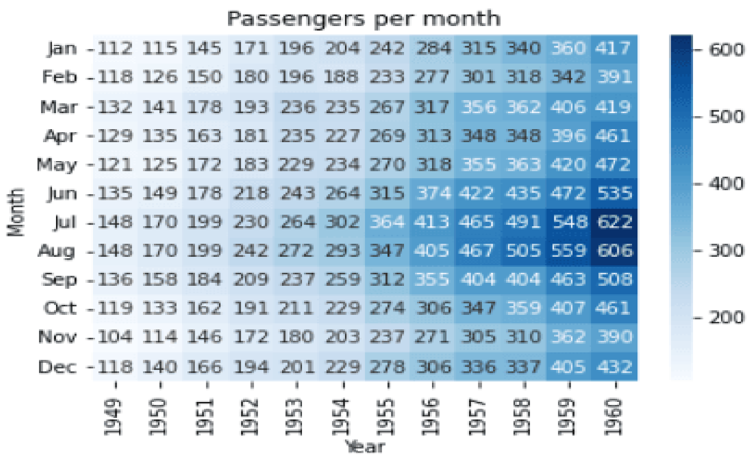


Figure 4.4. Heat Map

CHAPTER 5
CODING

5.1 MODEL.PY

```
import pandas as pd

from tld import get_tld, is_tld

from urllib.parse import urlparse

import re

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.multiclass import OneVsRestClassifier

import pickle



def process_tld(url):

    try:

        res = get_tld(url, as_object=True, fail_silently=False, fix_protocol=True)

        pri_domain = res.parsed_url.netloc

    except:

        pri_domain = None

    return pri_domain
```

```
def abnormal_url(url):
    # Parse the URL to extract the hostname
    parsed_url = urlparse(url)
    hostname = parsed_url.hostname

    # Ensure hostname is not None
    if hostname:
        # Convert hostname to string (in case it's None)
        hostname = str(hostname)

        # Search for the hostname in the URL
        match = re.search(hostname, url)

        # If a match is found, return 1 indicating an abnormal URL
        if match:
            return 1
        else:
            # If no match is found, return 0 indicating a normal URL
            return 0
    else:
        # If hostname is None, return 0 indicating a normal URL
        return 0
```

```
def httpSecure(url):
    htp = urlparse(url).scheme
    match = str(htp)
    if match == 'https':
        # print match.group()
        return 1
    else:
        # print 'No matching pattern found'
        return 0



def digit_count(url):
    digits = 0
    for i in url:
        if i.isnumeric():
            digits = digits + 1
    return digits



def letter_count(url):
    letters = 0
    for i in url:
        if i.isalpha():
            letters = letters + 1
        return letters



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

'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',
                        url)
        if match:
            return 1
        else:
            return 0
```

```
def having_ip_address(url):
    match = re.search(
        '(([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.([01]?\\d\\d?|2[0-4]\\d|25[0-
5])\\.([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.'
        '([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\/)|'  # IPv4
        '(([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.([01]?\\d\\d?|2[0-4]\\d|25[0-
5])\\.([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.'
        '([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\/)|'  # IPv4 with port
        '((0x[0-9a-fA-F]{1,2})\\.(0x[0-9a-fA-F]{1,2})\\.(0x[0-9a-fA-
F]{1,2})\\.(0x[0-9a-fA-F]{1,2})\\/)'  # IPv4 in hexadecimal
        '(?:[a-fA-F0-9]{1,4}:){7}[a-fA-F0-9]{1,4}|'
        '([0-9]+(?:\.[0-9]+){3}:[0-9]+)|'
        '((?:(?:\d|[01]?\d\d|2[0-4]\d|25[0-5])\.){3}(?:25[0-5]|2[0-
4]\d|[01]?\d\d|\d)(?:\/\d{1,2})?)', url)  # Ipv6
    if match:
        return 1
    else:
        return 0



data = pd.read_csv('malicious_phish(1).csv')
data['url'] = data['url'].replace('www.', '', regex=True)
rem = {"Category": {"benign": 0, "defacement": 1, "phishing": 2, "malware": 3}}
data['Category'] = data['type']
data = data.replace(rem)
```

```
data['url_len'] = data['url'].apply(lambda x: len(str(x)))
data['domain'] = data['url'].apply(lambda i: process_tld(i))
feature = ['@', '?', '-', '=', '.', '#', '%', '+', '$', '!', '*', ',', '//']
for a in feature:
    data[a] = data['url'].apply(lambda i: i.count(a))
data['abnormal_url'] = data['url'].apply(lambda i: abnormal_url(i))
data['https'] = data['url'].apply(lambda i: httpSecure(i))
data['digits'] = data['url'].apply(lambda i: digit_count(i))
data['letters'] = data['url'].apply(lambda i: letter_count(i))
data['Shortening_Service'] = data['url'].apply(lambda x: Shortening_Service(x))
data['having_ip_address'] = data['url'].apply(lambda i: having_ip_address(i))
X = data.drop(['url', 'type', 'Category', 'domain'], axis=1)  # ,'type_code'
y = data['Category']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=2)
binaryClassifier = RandomForestClassifier()
ovr = OneVsRestClassifier(binaryClassifier)
ovr.fit(X_train, y_train)
pickle.dump(ovr, open('model.pkl', 'wb'))
```

5.2 FEATUREEXTRACTION.PY

```
import pandas as pd

import model as md

import pickle


model = pickle.load(open('model.pkl', 'rb'))




def url_feature(url):

    #print(url)

    #print(type(url))

    url_dict = {'url_len': len(str(url))}

    feature = ['@', '?', '-', '=', '.', '#', '%', '+', '$', '!', '*', ',', '//']

    for a in feature:

        url_dict[a] = str(url).count(a)

    url_dict['abnormal_url'] = md.abnormal_url(url)

    url_dict['https'] = md.httpSecure(url)

    url_dict['digits'] = md.digit_count(url)

    url_dict['letters'] = md.letter_count(url)

    url_dict['Shortening_Service'] = md.Shortening_Service(url)

    url_dict['having_ip_address'] = md.having_ip_address(url)


    url_features = pd.DataFrame.from_dict([url_dict])
```

```
        X = url_features

        #print(X)

        output = model.predict(X)

        #print(type(output))

        return output


    '''url = "http://www.newtec.ac.uk/courses/health-and-social-care/level-2-diploma-
in-health-a-social-care-.html"

    res = url_feature(url)

    print(res[0])'''
```

5.3 APP.PY

```
from flask import Flask, request, url_for, redirect, render_template
import FeatureExtraction as fe
import pickle
app = Flask(__name__)
model = pickle.load(open('model.pkl', 'rb'))
@app.route('/')
def hello_world():
    return render_template('Home.html')
@app.route('/predict', methods=['POST', 'GET'])
def predict():
    entered_url = request.form.get('url')
    url = str(entered_url)
    #print(url)
    if url == "":
        return render_template('Home.html', pred='Please Enter URL')
    output = fe.url_feature(url)
    if output[0] == 0:
        return render_template('Home.html',pred='Safe to use 👍')
    elif output[0] == 1:
        return render_template('Home.html',pred='Website altered by external factors')
    elif output[0] == 2:
        return render_template('Home.html',pred='This url is a phishing url ⚠️')
    elif output[0] == 3:
        return render_template('Home.html',pred='This URL contains malware ⚠️')

if __name__ == '__main__':
    app.run(debug=True)
```

5.4 HOME.HTML

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Malicious URl Detection</title>
    <link rel="stylesheet" type="text/css" href="{{ url_for('static',
filename='styles/style.css') }}">
  </head>
  <body>
    <h1 >Malicious URL Detection Using Data Analytics</h1>
    <br>
    <form action="/predict" method="post" class="container">
      <input type="text" id="url" name="url" placeholder="Enter URL"
onfocus="this.value=''"><br>
      <button type="submit">Detect</button>
    </form>
    <div>
      <br><h4 align="center"><b>{{pred}}</b></h4>
    </div>
  </body>
</html>
```

5.5 STYLE.CSS

```
body {
    background-color: white; /* Change background color to white */
    font-family: Arial, sans-serif;
}

.container {
    width: 80%;
    margin: auto;
    text-align: center;
}

h1 {
    color: black; /* Change heading color to black */
    padding: 20px 0;
    text-align: center;
    /* Add scrolling animation */
    animation: slide 1s infinite alternate;
}

@keyframes slide {
    0% {
        transform: translateY(0);
    }
    100% {
        transform: translateY(20px);
    }
}
```

```
input[type="text"] {
    width: 50%;
    padding: 10px;
    margin: 20px 0;
    box-sizing: border-box;
    color: black;
}

button {
    background-color: #4c6baf;
    color: white;
    padding: 15px 32px;
    text-align: center;
    text-decoration: none;
    display: inline-block;
    font-size: 16px;
    margin: 4px 2px;
    cursor: pointer;
}

h4 {
    color: black; /* Change color to black */
    padding: 20px 0;
    text-align: center;
}
```

CHAPTER 6

RESULTS

6.1 SCREENSHOT AND OUTPUT

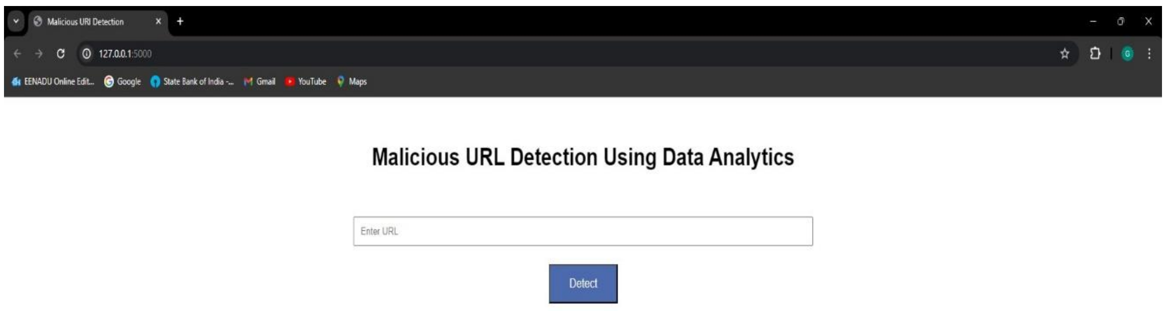


Figure 6.1 Website home page

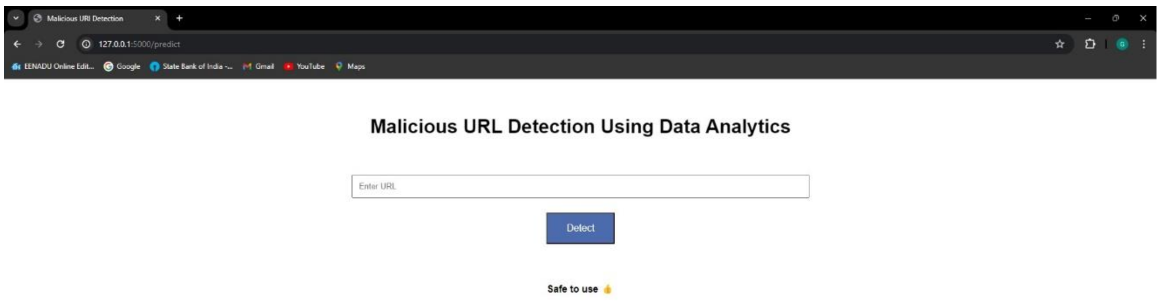
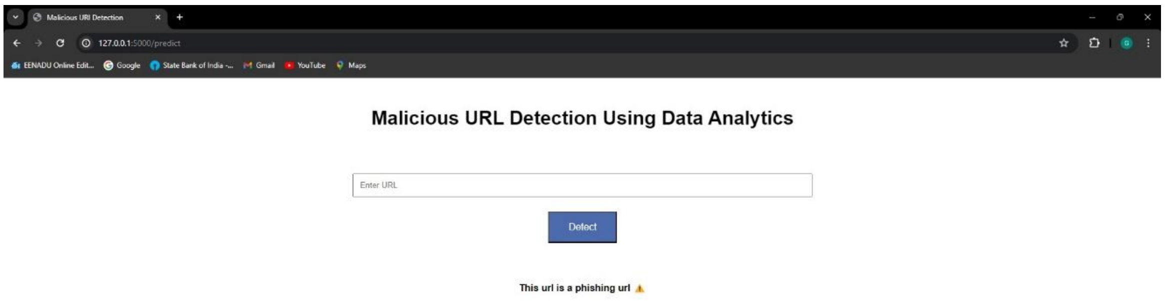


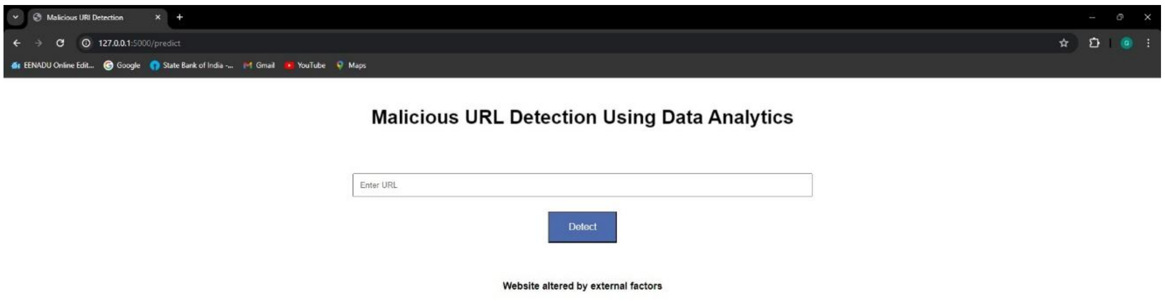
Figure 6.2 Safe URL



Malicious URL Detection Using Data Analytics

This url is a phishing url ⚠

Figure 6.3 Phishing URL



Malicious URL Detection Using Data Analytics

Website altered by external factors

Figure 6.4 Defacement URL

CHAPTER 7
CONCLUSION

Many cyber security applications depend on malicious URL identification, and machine learning techniques are obviously a promising path. We conducted a thorough and ordered analysis of Malicious Detection using AI approaches in the work. We provided the methodical description of Malicious detection from an AI standpoint, followed by nitty gritty information. Current investigations finds malicious URL identification, especially in the types of growing new component portrayals and preparing new learning calculations for determining vindictive URL position assignments We sorted most, if not all, of the existing obligations for malevolent URL position in writing in this overview, as well as acknowledged the requirements and challenges for creating tasks for detecting malicious URLs. In this analysis, we summarize the majority, if not all, of the existing commitments for malignant URL location in writing, as well as the requirements challenges for the develop of Malicious Detection as the Services for the real-world cyber security application. At long last, some featured less useful issue for application space, demonstrated other significant new issues to additional exploration examination.

Specifically, despite extensive research and enormous progress in recent years, automatic detection of spam URLs using AI remains as the challenging open issue. More viable aspect extraction, portrayal learning (example: by profound learning) are expected in the future, as well as more efficient AI calculations for developing predictive models especially for managing idea floats (example: successful internet learning), other arising difficulties (example: area dividing while applying the model to another space), Finally, a clever scheme for protecting named details and client criticism in a closed circle setup (example: coordinating online dynamic learning approach in the genuine system).

BIBLIOGRAPHY

1. Abdelhamid N, Ayesh A, Thabtah F (2014) Phishing detection based associative classification data mining. Science-Direct 41:5948–5959.

2. Chen KT, Chen JY, Huang CR, Chen JY (2009) Fighting phishing with discriminative key point features of webpages. IEEE Internet Comput 13:56–63.

3. Chen X, Bose I, Leung ACM, Guo C (2011) Assessing the severity of phishing attacks: a hybrid data mining approach. Expert Syst Appl 50:662–672.

4. Fu AY, Wenyin L, Deng X (2006) Detecting phishing web pages with visual similarity assessment based on earth mover's distance. IEEE Trans Dependable Secure Comput 3(4):301 321.

5. Islam R, Abawajy J (2013) A multi-tier phishing detection and filtering approach. J Netw Comput Appl 36:324–335.

6. Li Y, Xiao R, Feng J, Zhao L (2013) A semi-supervised learning approach for detection of phishing webpages. Optik 124:6027– 6033.

7. Nishanth KJ, Ravi V, Ankaiah N, Bose I (2012) Soft computing-based imputation and hybrid data and text mining: the case of predicting the severity of phishing alerts. Expert Syst Appl 39:10583–10589.

8. Medvet E, Kirda E, Kruegel C (2008) Visual-similarity-based phishing detection. SecureComm. In: Proceedings of the 4th international conference on Security and privacy in communication netowrks. pp 22–25.

9. Xiang G, Hong J, Rose CP, Cranor L (2011) CANTINA+: a feature-rich machine learning framework for detecting phishing web sites. ACM Trans Inf Syst Secur 14:21.

10. Zhang Y, Hong JI, Cranor LF (2007) CANTINA: a content-based approach to detecting phishing web sites. In: Proceedings of the 16th international conference on world wide web, Banff, p 639–648.

11. R.k. Nepali and Y. Wang "You Look suspicious!!" Leveraging the visible attributes to classify the malicious short URLs on Twitter. in 49th Hawaii International Conference on System Sciences (HICSS) IEEE, 2016, pp. 2648-2655.