Dissertation on
DETECTION OF MALICIOUS UNIFORM RESOURCE LOCATOR USING MACHINE LEARNING

Thesis submitted towards partial fulfilment
of the requirements for the degree of

Master of Technology in IT (Courseware Engineering)

Submitted by
Saptaparni Chatterjee

EXAMINATION ROLL NO.: M4CWE23015
UNIVERSITY REGISTRATION NO.: 160387 of 2021-22

Under the guidance of,
Mr. Joydeep Mukherjee

School of Education Technology
Jadavpur University

Course affiliated to
Faculty of Engineering and Technology
Jadavpur University
Kolkata-700032
India
2023

M.Tech. IT (Courseware Engineering)
Course affiliated to
Faculty of Engineering and Technology
Jadavpur University
Kolkata, India

CERTIFICATE OF RECOMMENDATION

This is to certify that the thesis entitled **"MALICIOUS URL DETETCTION USING MACHINE LEARNING"** is a bonafide work carried out by **SAPTAPARNI CHATTERJEE** under our supervision and guidance for partial fulfillment of the requirements for the degree of Master of Technology in IT (Courseware Engineering) in School of Education Technology , during the academic session 2022-2023.

SUPERVISOR
School of Education Technology
Jadavpur University,
Kolkata-700 032

DIRECTOR
School of Education Technology
Jadavpur University,
Kolkata-700 032

DEAN - FISLM
Jadavpur University,
Kolkata-700 032

M.Tech. IT (Courseware Engineering)
Course affiliated to
Faculty of Engineering and Technology
Jadavpur University
Kolkata, India

CERTIFICATE OF APPROVAL **

This foregoing thesis is hereby approved as a credible study of an engineering subject carried out and presented in a manner satisfactory to warranty its acceptance as a prerequisite to the degree for which it has been submitted. It is understood that by this approval the undersigned do not endorse or approve any statement made or opinion expressed or conclusion drawn therein but approve the thesis only for purpose for which it has been submitted.

Committee of final examination **----------**

for evaluation of Thesis **----------**

** Only in case the thesis is approved.

DECLARATION OF ORIGINALITY AND COMPLIANCE OF ACADEMIC ETHICS

I hereby declare that this thesis contains literature survey and original research work by the undersigned candidate, as part of her **Master of Technology in IT (Courseware Engineering)** studies.

All information in this document has been obtained and presented in accordance with academic rules and ethical conduct.

I also declare that, as required by this rule and conduct, I have fully cited and referenced all materials and results that are not original to this work.

NAME: SAPTAPARNI CHATTERJEE

EXAMINATION ROLL NUMBER: M4CWE23015

UNIVERSITY REGISTRATION NO : 160387 of 2021-22

THESIS TITLE: DETECTION OF MALICIOUS UNIFORM RESOURCE LOCATOR USING MACHINE LEARNING

SIGNATURE: DATE:

Acknowledgement

I feel extremely glad in presenting this thesis at **School of Education Technology, Jadavpur University, Kolkata,** in the partial fulfilment of the requirements for the degree of **Master of Technology in IT (Courseware Engineering)**. It is a great pleasure for me to express my respect and a deep sense of gratitude to my M. Tech supervisor **Mr. Joydeep Mukherjee,** Department of School of Education Technology, Jadavpur University, for his wisdom, vision, expertise, guidance, enthusiastic involvement during my M.Tech. It has been a significant privilege and joy for me to inquire about it under his watch.

I am grateful to **Prof.** (**Dr.) Matangini Chattopaddhyay** , **Director of the School of Education Technology**, for her support, encouragement and timely advice. I am really indebted to **Dr. Saswati Mukherjee** for her continuous support during the entire course of research work. Their advice and support were highly inspirational and motivating. I would also like to take this opportunity to pay my thanks to **all my classmates of M. Tech IT (Courseware Engineering)** and support staffs who motivated and helped me to complete my research work successfully.

With regards,

Date : ……..……

SAPTAPARNI CHATTERJEE
M. Tech I.T. (Courseware Engineering)
School of Education Technology

Jadavpur University, Kolkata- 700032

Dedicated to,

My Parents
And my Mentor
Mr. Joydeep Mukherjee

	CONTENTS	PAGENO

Chapter 1

Executive Summary

The increasing use of the internet has resulted in a corresponding rise in cyber-attacks, with malicious webpages posing a significant threat to users. This has necessitated the development of effective tools for detecting and preventing such attacks. In recent years, artificial intelligence (AI) has emerged as a promising tool in this regard, with the ability to learn from vast amounts of data and identify patterns that may not be immediately apparent to human analysts. In this essay, we discuss the use of AI in malicious webpage detection, focusing specifically on the use of lexical features, hyperparameters tunning, QuantileTransformer, Support Vector Machine, K-Nearest Neighbors Algorithm, LOGISTIC REGREESION,RANDOM FOREST and doing chi-squared test of independence, it is a statistical test,it is use for determine if there is a significant association between two categorical variables.

A critical step in preserving the safety of online platforms and shielding consumers from cyber dangers is the detection of dangerous Uniform Resource Locators. Automating this process using machine learning approaches has shown promising results, enabling quicker and more

precise identification of harmful Uniform Resource Locators. In this field, a number of essential approaches and methods are used, as detailed below.

A statistical technique for determining the independence between two category variables is the chi-square test. It is frequently used during the feature selection stage of developing a machine learning model for detecting dangerous Uniform Resource Locators. The chi-square test aids in finding the most pertinent features for classification by calculating the relationship between features and the target variable.

An efficient method for improving the hyperparameters of machine learning models is model tuning. This method efficiently examines various combinations of hyperparameters to identify the ideal configuration for the model by randomly selecting samples from a predetermined search space. The performance and generalizability of the model are enhanced by this method.

After execute these machine learning algorithm the researcher notice that randomforest is much better than other algorithms. Machine learning models for malicious Uniform Resource Locator identification can achieve improved accuracy rates, adapt to changing threats, and successfully defend people and systems from potential security hazards by utilising the power of neural networks.

A method for assessing the efficacy of machine learning models is cross-validation. Partitioning the data into several subsets, training the model

on a subset, and assessing the model's performance on the remaining data are the steps involved. Cross-validation offers a trustworthy assessment of the model's performance and aids in avoiding overfitting by repeatedly going through this procedure with various subsets.

The QuantileTransformer is a preprocessing technique used to transform features by mapping them to a uniform or Gaussian distribution. It belongs to the family of quantile-based transforms.

The main purpose of the QuantileTransformer is to make the feature distributions more Gaussian or uniformly distributed, which can be beneficial for certain machine learning algorithms that assume a Gaussian or uniform distribution of features.

The transformation performed by the Quantile Transformer involves estimating the cumulative distribution function (CDF) of the input feature values and then mapping them to the desired output distribution using the inverse of the CDF of the desired output distribution. This ensures that the transformed values have a specific desired distribution.

The Quantile Transformer in scikit-learn has two modes: "uniform" and "normal". In the "uniform" mode, the transformed values are mapped to a uniform distribution over the range [0, 1]. In the "normal" mode, the transformed values are mapped to a standard normal distribution (mean=0, standard deviation=1).

Models for detecting malicious Uniform Resource Locators perform best when evaluated using metrics like the ROC curve and classification report. The choice of an appropriate classification threshold is made possible by the ROC curve, which depicts the trade-off between true positive rate and false positive rate. Precision, recall, and F1 score are just a few of the performance statistics the categorization report includes.

The preparation of the data for model training involves data processing in a significant way. To ensure data integrity and avoid bias, actions like eliminating duplicates from the dataset are required. To further assure the accuracy and reliability of the data, it is crucial to test for null values and handle missing entries.

When determining the qualities of Uniform Resource Locators that are suggestive of malicious intent, lexical aspects are quite important. These characteristics include the Uniform Resource Locator's length, the existence of particular words or patterns, the use of special characters, and the domain's organisational structure. Machine learning models can learn to differentiate between trustworthy and malicious Uniform Resource Locators by examining these linguistic properties.

In order to perform the research for this study, a sizable dataset of Uniform Resource Locators—including both trustworthy and harmful examples—was collected and pre-processed. The dataset is used to train a variety of machine learning techniques, including decision trees, support

vector machines, and neural networks, to create precise classifiers. The models are then assessed using pertinent performance measures to determine how well they can identify dangerous Uniform Resource Locators.

In conclusion, a variety of methods including the chi-square test, model tuning with grid search, ensemble methods like Random forest, cross-validation, quantile transformer, evaluation metrics like ROC curve and classification report, confusion matrix, and proper data processing are used to detect malicious Uniform Resource Locators using machine learning. Organisations can create reliable and accurate methods for identifying and reducing cyber hazards linked to bad Uniform Resource Locators by using these techniques.

Chapter 2

Introduction

Overview

The internet is a vital part of our daily lives in the digital age because it makes it easy for us to connect, communicate, and acquire information. The growth of rogue Uniform Resource Locators, which pose serious hazards to users' security and privacy, is a drawback to this ease. Malicious Uniform Resource Locators can lead to phishing attacks, identity theft, malware dissemination, and financial fraud, among other cybercrimes. Therefore, the need for efficient methods to identify and counteract these dangers is urgent. Support Vector Machines (Support Vector Machine), K-Nearest Neighbors Algorithm, logistic-regression and random-forest a type of machine learning technique, has demonstrated promising results in accurately identifying fraudulent Uniform Resource Locators. This thesis seeks to advance cybersecurity by investigating the use of Random forest in identifying dangerous Uniform Resource Locators.

Problem Statement

Due to the constantly developing nature of cyber threats, it might be difficult to identify harmful Uniform Resource Locators. Heuristic and signature-based techniques are frequently insufficient to keep up with attackers' constantly evolving tactics. Therefore, a more sophisticated and adaptable technique is required to effectively identify and categorise harmful Uniform Resource Locators. The goal of this study is to construct a reliable and effective system for detecting malicious Uniform Resource Locators using machine learning methods, notably Random Forest.

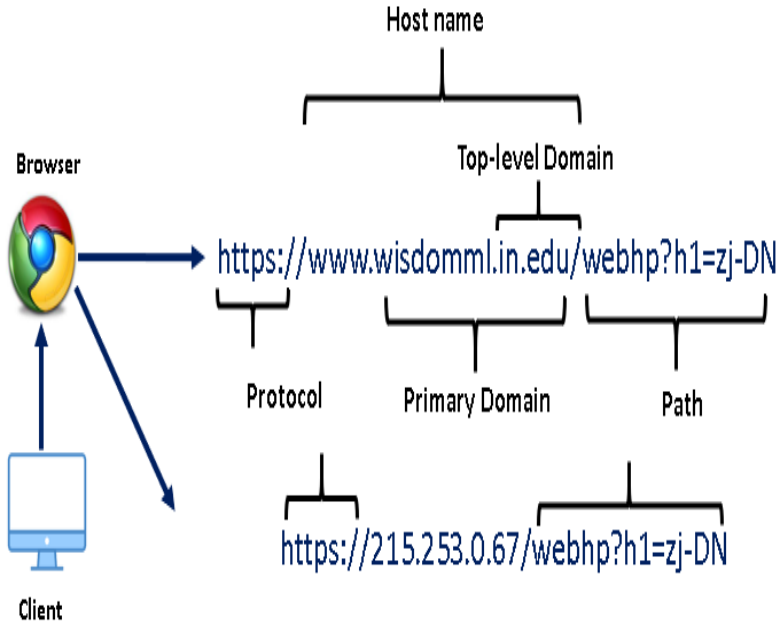


Figure 1 Uniform Resource Locator

Source - Internet

Objectives

The primary Objectives of this thesis are as follows:

a) To test how well machine learning algorithm works in spotting malware Uniform Resource Locators.

b) To create a large collection of Uniform Resource Locators that includes both malicious and benign samples for testing and training.

c) To extract pertinent information from Uniform Resource Locators that can help distinguish between harmful and benign Uniform Resource Locators.

d) To create and implement a Random forest machine learning model for detecting malicious Uniform Resource Locators.

e) To assess the accuracy, precision, recall, and F1 score of the proposed model's performance.

f) To assess Random's efficiency in identifying malicious Uniform Resource Locators in comparison to other machine learning techniques already in use.

Motivation

People, organisations, and governments all across the world are very concerned about the frequency and sophistication of cyberattacks. The most prevalent vector for launching these attacks is malicious Uniform Resource Locators. In order to protect users' online activities and sensitive data, it is essential to be able to precisely recognise and categorise harmful Uniform Resource Locators. We can create sophisticated systems capable of analysing massive amounts of Uniform Resource Locators and accurately recognising potential risks by utilising machine learning techniques. The goal of this thesis is to advance cybersecurity practises and reduce the risks brought on by online attacks by helping to develop reliable and efficient methods for malicious Uniform Resource Locator identification.

Organisation of Thesis

Executive Summery, Chapter 1

In the framework of a thesis, Chapter 1 provides a succinct summary of the entire research endeavour or study. The primary goals, approaches, results, and conclusions of the thesis are intended to be understood clearly and completely in the executive summary.

Introduction, Chapter 2

This chapter presents the issue statement, describes the goals, gives a general summary of the research topic, and discusses the driving forces behind the investigation.

Review of Literature, Chapter 3

This chapter examines the existing literature and research on machine learning techniques, and malicious Uniform Resource Locator detection. It gives a thorough grasp of the state of the discipline at the moment and points out areas that need more research.

Proposed Approach, Chapter 4

The methodology and approach, including the procedures for data collecting, pre-processing, feature extraction, model creation, and evaluation, are described in this chapter.

Experimental Results, Chapter 5

This chapter presents and analyses the experimental findings from the suggested Neural model for the detection of dangerous Uniform Resource Locators. It is described how the performance measures compare to those of different machine learning methods.

Comparative Analysis, Chapter 6

This chapter represents and analysis the comparison between machine learning algorithms and techniques.

Conclusion and Future Scope, Chapter 7

The research's results are described in this chapter, along with their limitations and suggestions for future research. The study's major contributions and ramifications are summed up in the conclusion.

Chapter 3

Literature Survey

Cybersecurity is significantly threatened by malicious Uniform Resource Locators. These Uniform Resource Locators are intended to trick visitors and take advantage of their weaknesses. They can result in a number of possible hazards, such as phishing attacks, the spread of malware, and identity theft. Protecting users and systems from online attacks requires an understanding of and ability to recognise bad Uniform Resource Locators.

Methods for Detecting Malicious Uniform Resource Locators

Blacklisting, whitelisting, and signature-based techniques are common methodologies for detecting dangerous Uniform Resource Locators. To prevent access, blacklisting includes keeping a list of known dangerous Uniform Resource Locators. Only Uniform Resource Locators that have been approved in advance are allowed with whitelisting. In order to identify malicious Uniform Resource Locators, approaches based on signatures use predetermined patterns or signatures.

In recent years, Uniform Resource Locator analysis has seen extensive use of machine learning techniques. These methods include anomaly detection, feature extraction, and classification algorithms. Machine learning algorithms can identify harmful Uniform Resource Locators based on features and patterns acquired from labelled data.

Features of a Uniform Resource Locator analysis:

To distinguish between dangerous and benign Uniform Resource Locators, Uniform Resource Locator analysis uses a number of criteria. Lexical-based characteristics like domain age and Uniform Resource Locator length can shed light on the Uniform Resource Locator's purpose. Keywords and content structure are examples of content-based traits that can be used to detect malevolent intent. Uniform Resource Locator analysis can benefit from behavioural aspects, such as clickstream data and user behaviour.

Machine Learning Algorithms for Detecting Malicious Uniform Resource Locators

In order to identify dangerous Uniform Resource Locators, a variety of machine learning methods are frequently used. These include deep learning methods like neural networks, decision trees, random forests, and support

vector machines (Support Vector Machine). Each algorithm has advantages and disadvantages in terms of precision and effectiveness. Selecting the most appropriate algorithm for a particular detection task can be made easier by being aware of these qualities.

Dataset and Assessment Criteria:

Researchers and practitioners frequently employ a variety of datasets, including PhishTank, Alexa Top 1 Million, and custom research-specific datasets, to train and assess malicious Uniform Resource Locator detection models. These datasets include both benign and dangerous Uniform Resource Locators, making it possible to create and test efficient detection models.

Accuracy, precision, recall, F1-score, and area under the curve (AUC) are evaluation metrics that are frequently used to gauge the effectiveness of models for detecting malicious Uniform Resource Locators. These measurements reveal information about the potency and usefulness of the detection algorithms.

New developments and emerging trends:

New methods and advancements in malicious Uniform Resource Locator identification have been presented in recent research publications, conferences, and industry reports. Natural language processing methods, graph-based models, and deep learning architectures have attracted interest because of their potential to improve detection accuracy. To keep on top of malicious Uniform Resource Locator detection research, it is crucial to be informed of these current developments and new trends.

Challenges and Limitations:

Despite improvements in malicious Uniform Resource Locator identification, there are still issues that practitioners and researchers must deal with. Attackers regularly use evasion methods to get past detection systems, and the idea of zero-day attacks creates new difficulties. Some of the real-world issues that need to be solved include scalability, real-time detection, feature engineering, and unbalanced datasets.

Analysing and benchmarking comparisons:

Different detection techniques, algorithms, and datasets can be compared to reveal their advantages and disadvantages. Setting a benchmark for comparison and encouraging innovation in the sector are accomplished through benchmarking studies or contests that rate and evaluate the effectiveness of malicious Uniform Resource Locator detection systems.

Open Questions for Research:

There are still unanswered research concerns in the area of malicious Uniform Resource Locator detection that need to be looked into further. For the field to advance, these gaps and unresolved problems must be identified. Novel feature selection methods, effective real-time detection algorithms, and strategies to counter new escape tactics are possible research fields.

Related Work:

literature review will give a thorough summary of the current state of knowledge, recent developments, difficulties, and unanswered research problems in the area of malicious Uniform Resource Locator detection if it is organised in accordance with these principles.

[2] A Comparative Analysis of Machine Learning Algorithms for Detecting Malicious Uniform Resource Locators. Journal of Information Security and Applications, 52, 102504.

Alshammari et al. conducted a comparative analysis of machine learning algorithms for detecting malicious Uniform Resource Locators. They evaluated the performance of various algorithms, including Random Forest, Naive Bayes, K-Nearest Neighbors, Support Vector Machine, and Logistic Regression. The study found that Random Forest outperformed other algorithms in terms of accuracy, precision, recall, and F1-score.

[3]Grosse et al. explored the vulnerability of deep neural networks (DNNs) to adversarial examples in the context of malware classification. They demonstrated that by manipulating certain features of malicious Uniform Resource Locators, it is possible to evade detection by DNN-based classifiers. The study highlights the need for robust defenses against adversarial attacks in the domain of malicious Uniform Resource Locator detection.

[4]Xiang et al. proposed a deep learning framework for malicious Uniform Resource Locator detection based on an attention mechanism. The model utilizes a long short-term memory (LSTM) network with an attention mechanism to capture important features from Uniform Resource Locators.

The study demonstrated that the proposed framework achieved competitive performance compared to traditional machine learning algorithms.

[5]Ahmadi and Zolanvari investigated the use of deep transfer learning for malicious Uniform Resource Locator detection. They proposed a transfer learning framework based on pre-trained convolutional neural networks (CNNs) to leverage knowledge from related domains. The study showed that the proposed approach improved the performance of malicious Uniform Resource Locator detection compared to traditional machine learning algorithms.

[6]Cao et al. presented DeepTransfer, a deep neural network for cross-domain malicious Uniform Resource Locator detection. The model utilizes a CNN-based architecture combined with transfer learning techniques to detect malicious Uniform Resource Locators across different domains. The study demonstrated the effectiveness of DeepTransfer in achieving high detection accuracy and outperforming traditional machine learning algorithms.

[7]Arachchilage and Love conducted a survey on feature selection and extraction methods for malicious Uniform Resource Locator detection with

a focus on explainable artificial intelligence (XAI). The study provides an overview of different techniques and highlights the importance of interpretable models to understand the decision-making process of Uniform Resource Locator classifiers.

[8]Zhu et al. proposed a novel malicious Uniform Resource Locator detection system based on an improved CNN. They introduced a dual-channel CNN architecture that incorporates both Uniform Resource Locator text and structure information. The study demonstrated that the proposed system achieved high detection accuracy and outperformed traditional machine learning algorithms.

[9]Chang et al. presented a hybrid model for malicious Uniform Resource Locator detection based on gradient boosting decision trees (GBDT) and multilayer perceptron (MLP). The model combines the strengths of both algorithms to improve detection accuracy. The study showed that the hybrid model outperformed individual algorithms and achieved competitive performance in malicious Uniform Resource Locator detection.

[10]Costa et al. conducted a comparative study of machine learning algorithms for malicious Uniform Resource Locator detection. They

evaluated the performance of several algorithms, including Random Forest, Decision Tree, K-Nearest Neighbors, and Neural Network. The study compared the algorithms based on accuracy, precision, recall, and F1-score, providing insights into the effectiveness of different approaches.

[11]Pan et al. proposed a malicious Uniform Resource Locator detection method based on a hierarchical attention network and CNN. The model utilizes a hierarchical structure to capture local and global dependencies in Uniform Resource Locators and employs attention mechanisms to focus on informative parts. The study demonstrated that the proposed method achieved high detection accuracy and outperformed traditional machine learning algorithms.

[12]Kumar and Rathee conducted a comprehensive survey on Uniform Resource Locator classification using machine learning techniques. The survey provides an overview of various machine learning algorithms and their application in Uniform Resource Locator classification. It discusses the challenges and approaches to feature extraction, feature selection, and classification methods for effective Uniform Resource Locator detection.

[13]Keshav et al. conducted a survey on machine learning techniques for detecting malicious Uniform Resource Locators. The study provides an

overview of different machine learning algorithms and their application in malicious Uniform Resource Locator detection. It discusses the limitations of traditional methods and explores the potential of deep learning techniques in improving detection accuracy.

[14]Maity and Jana analyzed various machine learning techniques for Uniform Resource Locator classification. The study evaluated the performance of different algorithms, including Decision Tree, Random Forest, Support Vector Machine, and k-Nearest Neighbors. The findings provided insights into the strengths and weaknesses of different algorithms in the context of Uniform Resource Locator classification.

[15]Poh and Tan conducted a comparative study on malicious Uniform Resource Locator detection using machine learning. The study compared the performance of different algorithms, including Naive Bayes, Decision Tree, Random Forest, and Support Vector Machine. The findings highlighted the importance of feature selection and the effectiveness of ensemble methods in improving detection accuracy.

Chapter 4

Methodology

4.1 GENERAL

The purpose of machine analysis is to carry out a thorough analysis work and collect specific information on the definition, the stages involved, and other constraints like performance measurement and machine optimisation. It seeks to give a thorough grasp of how the machine works and where it may be improved.

On the other hand, the goal of system analysis is to clearly and concisely characterise the technical details of the main notion. Its main goal is to clearly comprehend the system's technical components and articulate them in a straightforward manner. The goal of a system analysis is to give a thorough understanding of the system's elements, features, and interactions.

4.2 EXISTING SYSTEM

Malicious Uniform Resource Locators in online apps can now be found using machine learning methods. Different methods and procedures are used by existing systems to spot and eliminate possible risks. Here is a list of some typical methods applied in these systems:

(1) Feature-based analysis: In this method, certain characteristics are extracted from Uniform Resource Locators and used to train machine learning models. Features can include things like Uniform Resource Locator length, keyword density, and domain repute. In order to categorise Uniform Resource Locators as dangerous or benign, labelled datasets are then used to train machine learning algorithms like decision trees, random forests, or Support Vector Machines (SVM).

Some systems use pre-existing blacklists or reputation databases to identify harmful Uniform Resource Locators through blacklisting and reputation-based analysis. By taking into account additional features or implementing dynamic updating methods, machine learning algorithms can be used to increase the blacklist's effectiveness. Using this method makes it easier to instantly flag known dangerous Uniform Resource Locators.

(2)Behaviour analysis: These systems watch the actions and interactions of Uniform Resource Locators with web apps rather than relying only on static properties. In order to identify potentially dangerous Uniform Resource Locators, machine learning models can be trained to examine user interaction patterns, Hypertext Transfer Protocol request/response headers, and network data. In this context, methods like clustering algorithms and anomaly detection are frequently used.

(3)Ensemble methods: To aggregate the predictions of many machine learning models, many systems use ensemble methods. These systems seek

to increase accuracy and decrease false positives/negatives by combining the findings from various algorithms. For the purpose of detecting fraudulent Uniform Resource Locators, models that are reliable and accurate can be built using strategies like stacking, boosting, or bagging.

These current technologies search for harmful Uniform Resource Locators in web apps using a variety of machine learning methods and techniques. The technique selected is determined by the particular needs of the application and the data at hand. More effective and efficient solutions for malicious Uniform Resource Locator detection are being developed as a result of ongoing research and machine learning breakthroughs

4.3 PROPOSED ARCHITECTURE

4.3.1 OVERVIEW

1. The suggested architecture entails implements Machine learning algorithms, such as K-Nearest Neighbors Algorithm, logistic regression, Support Vector Machine, random forest. These are implemented. The models are trained once the data has been pre-processed, and then quantile transformer and hyper-parameter tuning approaches are used. The final model is the Random forest that achieves the best level of accuracy. After model train through a

function call the prediction happen by entering data for predictions, and receive precise results.

4.3.2 DATASET

'url', 'label', and 'type' are the three columns in the dataset, which has 691 items total.

The Uniform Resource Locators are present in the column 'url' as objects.

The labels for the Uniform Resource Locators are included in the 'label' column and include terms like 'good' or 'bad'.

The integer column with the values 0 and 1 in the 'type' column.

An unbalanced dataset is indicated by the 'type' column, which contains counts of 184 for value 0 and 507 for value 1.

There are no blank cells in any of the columns of the dataset.

The dataset has undergone further pre-processing and feature engineering operations after the addition of the extra features.

```
(691, 3)
```

	url	label	type
0	br-icloud.com.br	bad	0
1	mp3raid.com/music/krizz_kaliko.html	good	1
2	bopsecrets.org/rexroth/cr/1.htm	good	1
3	http://www.garage-pirenne.be/index.php?option=...	bad	0
4	http://adventure-nicaragua.net/index.php?optio...	bad	0

FIG 2 DATASET

```
[ ]  1 df.type.value_counts()

1    507
0    184
Name: type, dtype: int64
```

```
 1 df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 691 entries, 0 to 690
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   url     691 non-null    object
 1   label   691 non-null    object
 2   type    691 non-null    int64
dtypes: int64(1), object(2)
memory usage: 16.3+ KB
```

```
[ ]  1 df.isnull().sum() # there is no missing values

url      0
label    0
type     0
dtype: int64
```

FIG 3 DATASET VISUALIZATION

```
[ ]  1 df.head()
```

	url	label	type
0	br-icloud.com.br	bad	0
1	mp3raid.com/music/krizz_kaliko.html	good	1
2	bopsecrets.org/rexroth/cr/1.htm	good	1
3	http://www.garage-pirenne.be/index.php?option=...	bad	0
4	http://adventure-nicaragua.net/index.php?optio...	bad	0

```
   1 df.tail()
```

	url	label	type
686	onlineradio2.com/listen/Mix_933	good	1
687	mapsofworld.com/usa/states/californ/californ...	good	1
688	http://www.prefina.it/notizie/notizie-generali...	bad	0
689	video.ca.msn.com/watch/video/dog-kills-quebec-...	good	1
690	http://torrentdn.com/bbs/s.php?bo_table=torren...	good	1

FIG 4 DATASET STARTING POINT AND ENDING POINT

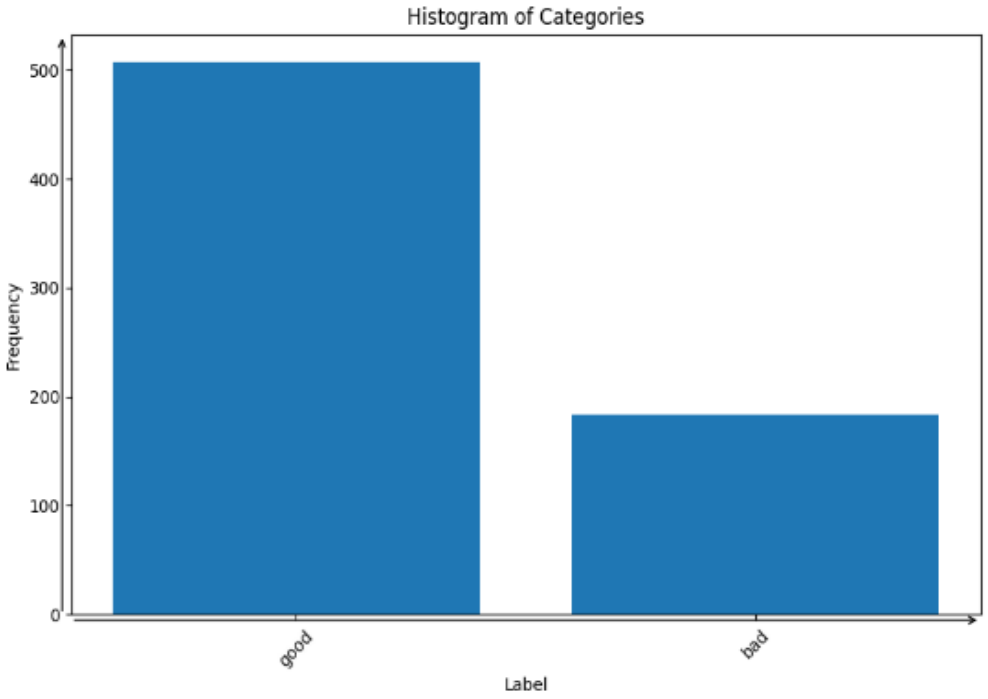


FIG 5 HISTOGRAM OF CATAGORIES OF DATASET

4.3.3 ARCHITECTURE

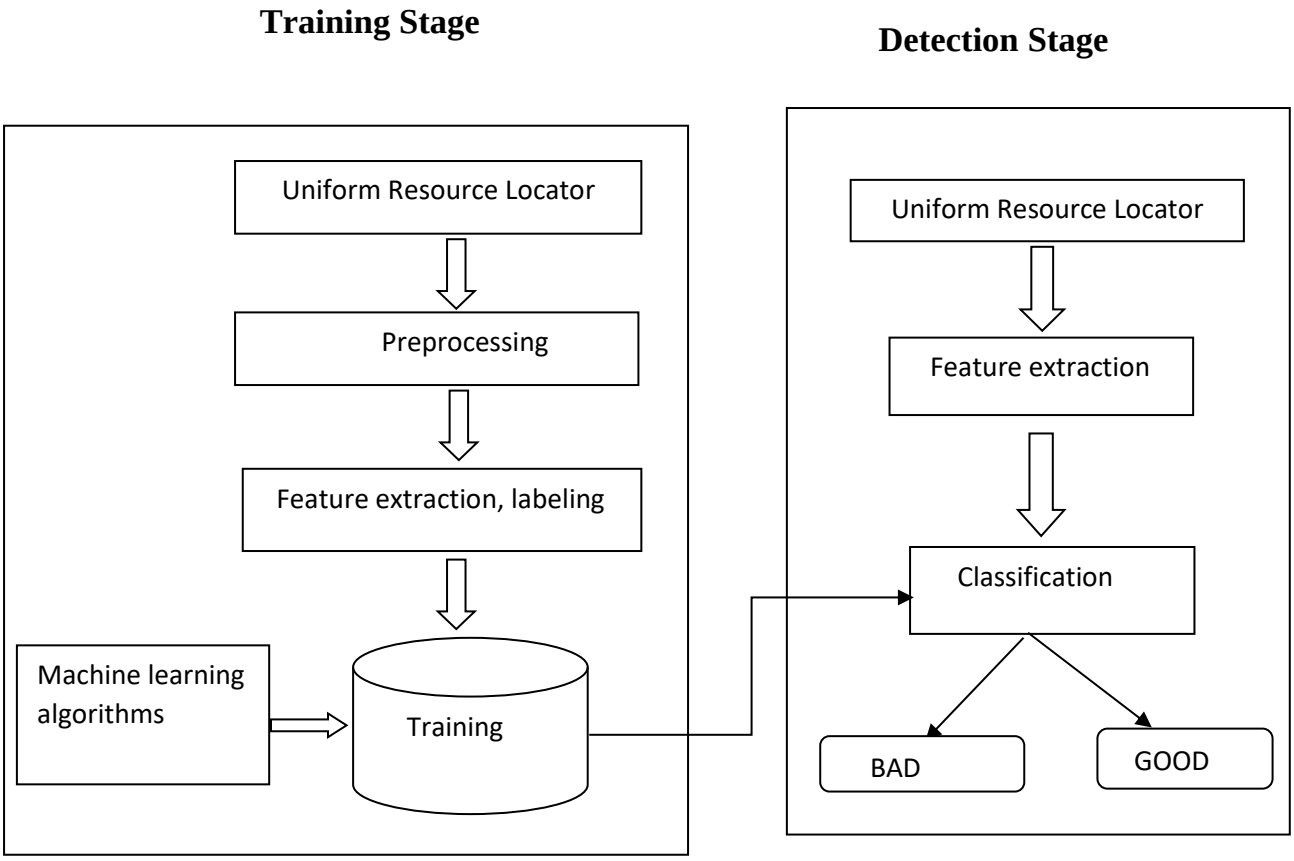


FIG 6 ARCHITECTURE

A system or application's components, connections, and interactions are depicted visually in an architecture diagram. It gives a high-level overview of the system's architecture and demonstrates how different parts are linked together and how data moves between them.

Symbols, boxes, and lines are frequently used in architecture diagrams to represent various components and their relationships. They can be used to explain a system's design and operation to stakeholders, programmers, and other team members. These representations aid in understanding the general architecture of the system, the discovery of possible weaknesses or bottlenecks, and the facilitation of discussions and decision-making throughout the development process.

Typical components of an architecture diagram include:

The main functional units or modules of the system are represented by its components. These could be parts of subsystems, hardware, software, or services.

Show how components interact with one another, often through communication channels or APPLICATION PROGRAMMING INTERFACEs, using connections and interfaces. These relationships are shown using arrows or lines.

Data Flows: Display the flow of information or data between components. These flows, which show the direction of data transport using arrows or lines, can be visualised.

Indicate the protocols or standards that are used for component-to-component communication, such as Hypertext Transfer Protocol, TCP/IP, REST, etc.

Deployment Diagrams: Deployment diagrams can be included to explain how the system is deployed across several servers or environments if the architecture incorporates distributed or cloud-based components.

Layers/Tiers: The diagram may show the layers and their relationships if the system is divided into layers or tiers (for example, the presentation layer, the business logic layer, and the data layer).

External Systems/Interfaces: These can be represented in the diagram to show the integration points if the system communicates with external systems or APPLICATION PROGRAMMING INTERFACEs.

An architecture diagram helps with understanding, communicating, and documenting the design and interactions of the system's components as well as serving as a visual depiction of the system's structure.

Here Uniform Resource Locator part take Uniform Resource Locator input ,preprocessing part doing cleaning,detect,feature engineering part doing tadd many features,machine learning algorithms(Support Vector Machine, K-Nearest Neighbors Algorithm, logistic regression) then trains in traning part and its interconnected with classification and after classification it give the output whether it is malicious or not.

4.3.4 SYSTEM DESIGN

The process of developing and specifying a system's architecture, components, modules, interfaces, and behaviour in order to meet predetermined requirements is known as system design. It entails converting user needs and functional requirements into a thorough blueprint that directs the system's implementation.

Creating a solution that satisfies the system's required functionality, performance, scalability and dependability. It takes into account a number of things, including the goals of the system, its limitations, user requirements, available technology, and resources.

The following steps are often included in the system design process:

Understanding and analysing the user requirements and functional requirements for the system to constitute requirements analysis phase. This entails (i) compiling data, (ii) conducting interviews, and (iii)figuring out the main characteristics and limitations of the system.

Defining the system's overall structure and organisation is known as architecture design. This entails identifying the (i) key elements, (ii) their connections, and (iii)their interfaces. Additionally, relevant architectural patterns or styles may be chosen to serve as the design's guiding principles.

Designing the various parts or modules that make up the system is known as component design. Each component's functionality, behaviour, and data structures must be specified. Determining the relationships and interfaces between components is another aspect of it.

Designing the data model and database schema that the system will employ is known as data design. This entails specifying the storage and retrieval technologies as well as the entities, relationships, and properties of the data.

Designing the user interface and system interaction flows is known as interface design. This include designing (i)the navigation and layout, (ii)making wireframes or prototypes, and (iii)taking usability and user experience concepts into account.

Designing the system with security and performance in mind means taking both into account when designing the system. This entails locating (i) potential vulnerabilities,(ii) creating systems for authentication and access control, and (iii)tuning the system to function effectively.

Review and Validation: Consult with relevant parties, including stakeholders, domain experts, and others, to review and validate the system design. This process makes that the design complies with the specifications and takes care of any potential problems or dangers.

System design is an iterative process that frequently requires making trade-offs and decisions depending on different considerations. A thorough design specification or set of design documents that serve as a roadmap for the system's implementation and development are the result of the system design process.

4.3.5 DATA FLOW DIAGRAM

The visual representation of how data moves through a system or process is called a data flow diagram (DFD). It offers a visual representation of the data intake, output, and transformation processes within a system, illuminating the rational information flow.

To comprehend and convey the data flow inside a system or process, DFDs are frequently used in software development, system analysis, and process modelling. They assist in identifying the inputs, outputs, and transformations

that take place at each stage as well as in visualising the relationships between various components or modules.

An important data flow diagram component is:

The system's functions or actions that manipulate or transform data are represented by processes. Each process is shown as a box or rectangle, and each one is labelled to indicate the precise task it completes.

Data Flows: Display how data is transferred between processes, outside entities, or data repositories. Data flows are shown as arrows or lines with labels that specify the type of data being transported.

Entities that interact with the system from beyond the system's boundaries are considered external entities. These could include individuals using the system, different systems, or objects that are receiving output from it. Rectangles are frequently used to represent external items.

Data Stores: These are the places in the system where data is kept in repositories or other forms of storage. Rectangles are used to represent data stores, and the label on each one indicates the kind of data it contains.

Labels for Data Flows: Each data flow has a label identifying the sort of data being sent (such as client information or order details). These labels make the data being exchanged between components clear and understandable.

Different levels of detail or abstraction, often known as DATA FLOW DIAGRAM levels, can be applied to DFDs. Following levels give increasing

levels of complexity, breaking down processes into sub-processes or dividing the system into more specialised components. Level 0 is the greatest level of abstraction and provides a general overview of the system.

DFDs can be used for a number of things, such as:

recognising and recording system processes and needs.

figuring out the connections and dependencies between data within a system.

locating possible data-flow bottlenecks or inefficiencies.

describing to stakeholders the data flow and system operation.

laying the groundwork for the creation and design of systems.

Data flow diagrams enable better comprehension, communication, and documentation of the system's data processing and transformation by aiding in the visualisation and analysis of the movement of data within the system.

DFD Level 0

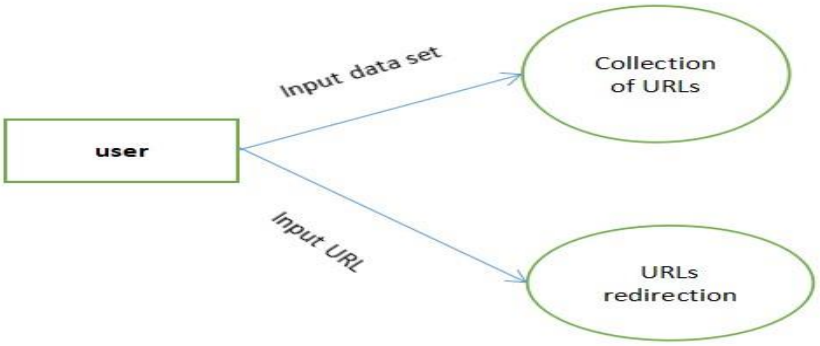


FIG 7 DFD LEVEL 0

DFD Level 1

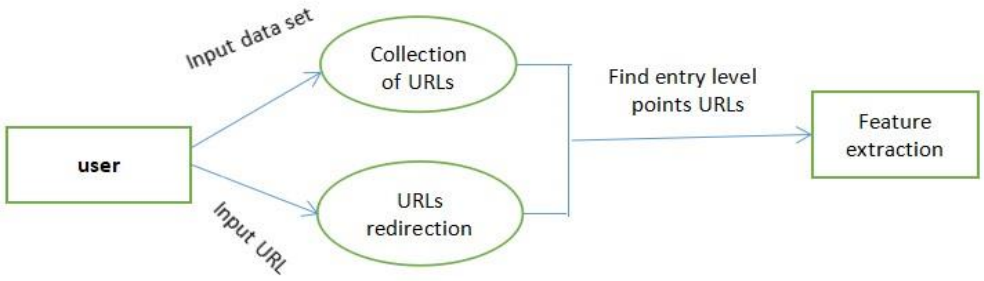


FIG 8 DFD LEVEL 1

DFD Level 2

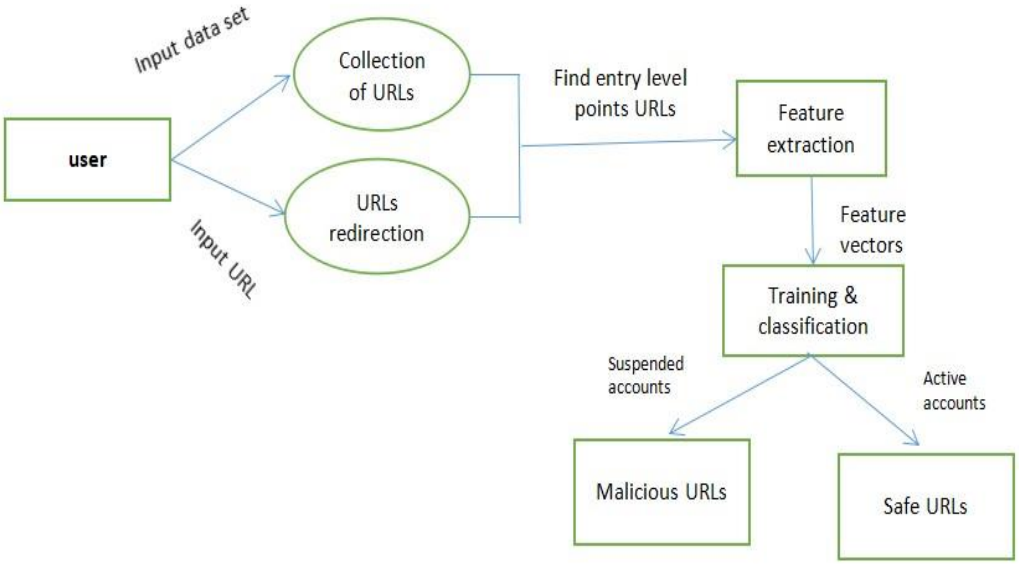


FIG 9 DFD LEVEL 2

4.3.6UNIFIED MODELING LANGUAGE DIAGRAM

Software engineering and system design use UML (Unified Modelling Language) diagrams as a standardised visual representation technique to describe, visualise, and document various system components. UML diagrams give stakeholders, such as developers, designers, and business analysts, a consistent language and notation that facilitates communication and comprehension.

Among the objectives of UNIFIED MODELING LANGUAGE DIAGRAM diagrams are:

Visualising System Structure: UNIFIED MODELING LANGUAGE diagrams assist in illustrating a system's structure, including its constituent parts, connections, and interactions. They offer a graphical representation that makes it simpler to comprehend the architecture and structure of the system.

Modelling System Behaviour: System behaviour and dynamics can be modelled using UML diagrams. They depict how various parts or items work together to deliver particular functionality. Activities, state changes, interaction patterns, and other things can all be represented using UML diagrams.

UML diagrams are useful for planning and facilitating system design. They offer a roadmap for the development team, assisting them in comprehending the needs, interfaces, and limitations of the system.Making design decisions and seeing possible problems early in the development process are made easier with the help of UNIFIED MODELING LANGUAGE diagrams.

A system's design and functionality are captured in detail by UNIFIED MODELING LANGUAGE diagrams, which are used as a type of documentation. They offer a visual reference that various stakeholders can quickly share and comprehend. UNIFIED MODELING LANGUAGE diagrams are useful throughout the whole software development lifecycle for preserving and upgrading system documentation.

Supporting Communication and Collaboration: UNIFIED MODELING LANGUAGE diagrams give project team members and stakeholders a common language for communication. They facilitate conversations about system requirements, design choices, and system behaviour and help to align understanding. UML diagrams aid in the concise and organised communication of complicated information.

There are various UNIFIED MODELING LANGUAGE diagram kinds, each of which serves a particular function and illustrates a different feature of a system. UML diagrams that are frequently used include:

(i)Class Diagrams: Show how a system's classes, attributes, methods, and connections between them are organised.

(ii)Use case diagrams: Display a system's characteristics or functionalities as seen from the viewpoint of the user.

(iii)Sequence diagrams: Show the order of interactions between items or components and illustrate the dynamic behaviour of a system.

Model the movement of activities or processes inside a system using activity diagrams.

(iv)State machine diagrams: Show the various states and state changes of a system or entity.

USECASE DIAGRAM:

.

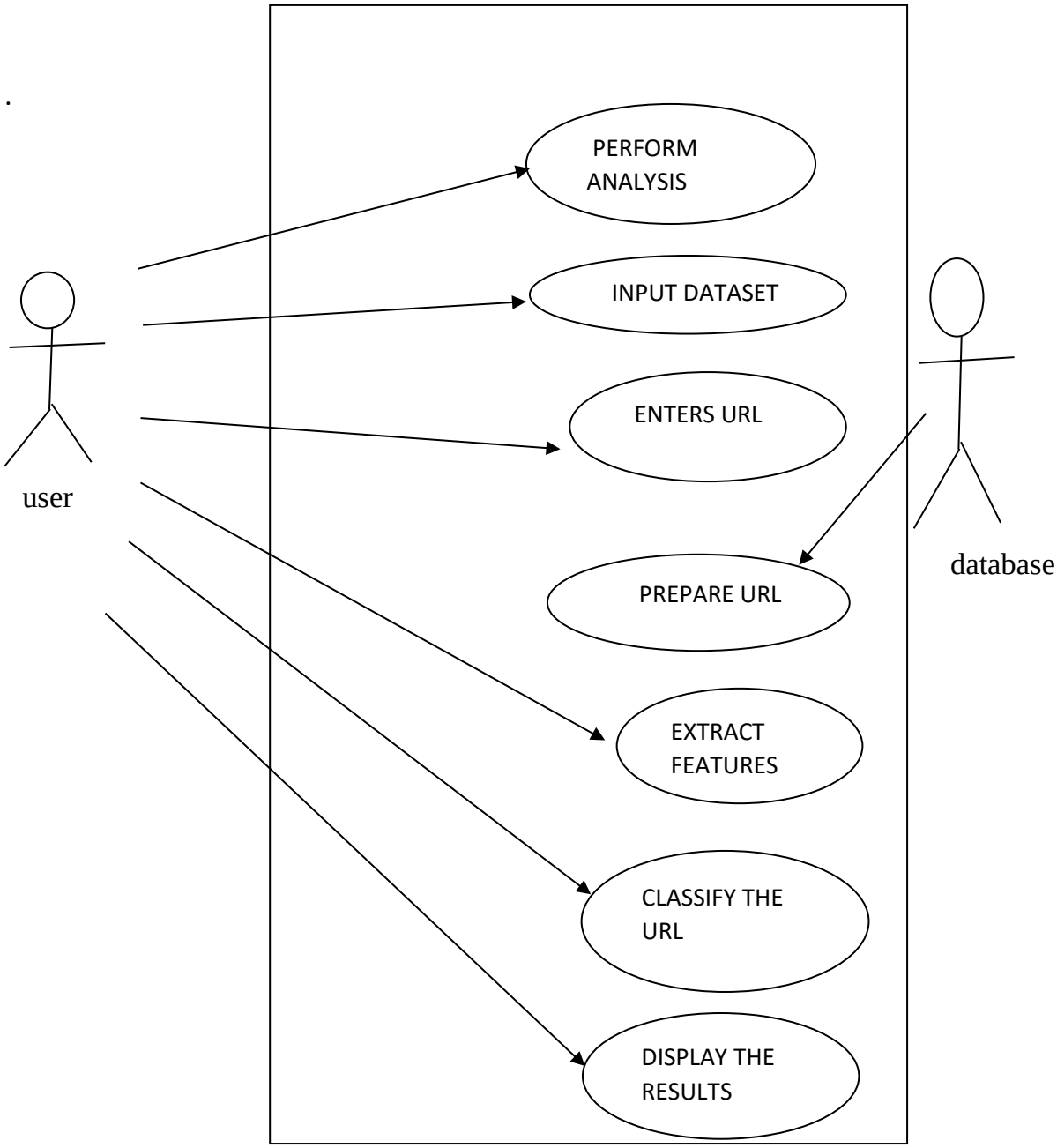


FIG 10 USECASE DIAGRAM

Use case diagrams offer a visual picture of the operation of the system and user-system interactions. They act as a tool for stakeholder communication, assisting in the clarification of requirements, identifying system boundaries, and coordinating the understanding of system functionalities. Early in the software development cycle, use case diagrams are often developed as a foundation for further in-depth system design and development operations.

CLASS DIAGRAM:

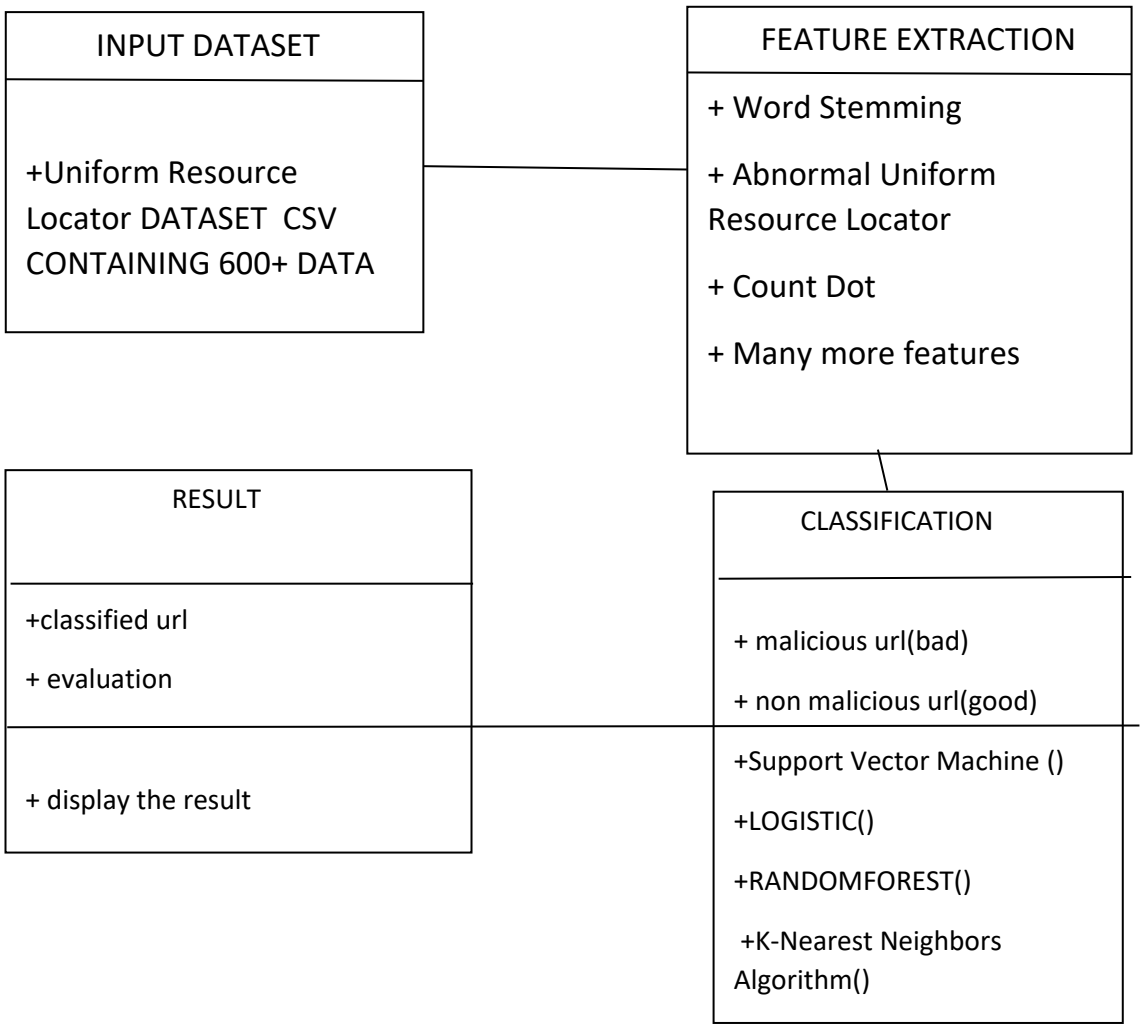


FIG 11 CLASS DIAGRAM

A class diagram in UML (Unified Modelling Language) is a specific kind of static structural diagram that shows the organisation and connections between classes in a system or software application. It offers a high-level

overview of the object-oriented design of the system by showing the classes, their properties, methods, and connections between them.

Class diagrams serve as a roadmap for software development, aid in understanding a system's structure, and show the links between classes. They facilitate communication between developers, designers, and stakeholders by providing a visual depiction of the classes and their interactions.

SEQUENCE DIAGRAM:

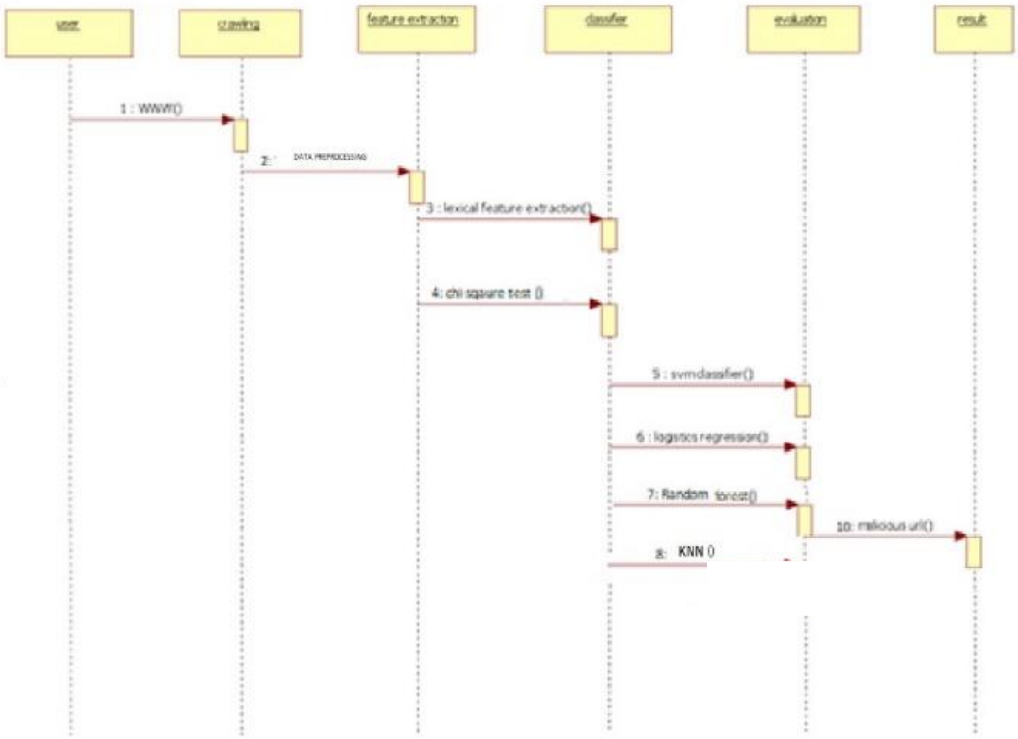


FIG 12 Sequence Diagram

A sequence diagram in UML (Unified Modelling Language) is a sort of interaction diagram that depicts the communications and interactions between objects or parts of a system or software application over a predetermined time period. It displays the transmission of messages and the sequence in which they are passed back and forth between various entities.

A sequence diagram shows the messages transferred between objects as horizontal arrows and the items themselves as vertical lifelines. The order of interactions is shown by the chronological order in which the messages are displayed from top to bottom.

Sequence diagrams are helpful for identifying the order of method calls made when executing a certain use case or scenario, for understanding the collaboration between objects, and for visualising the dynamic behaviour of

a system. They aid in documenting the communication and interaction patterns between various system components

.

To depict and communicate the interactions between the objects or components involved in a given process or feature, these diagrams are frequently used in software development, system analysis, and design. They offer a precise illustration of how items interact and communicate, which facilitates system analysis and design(SAD).

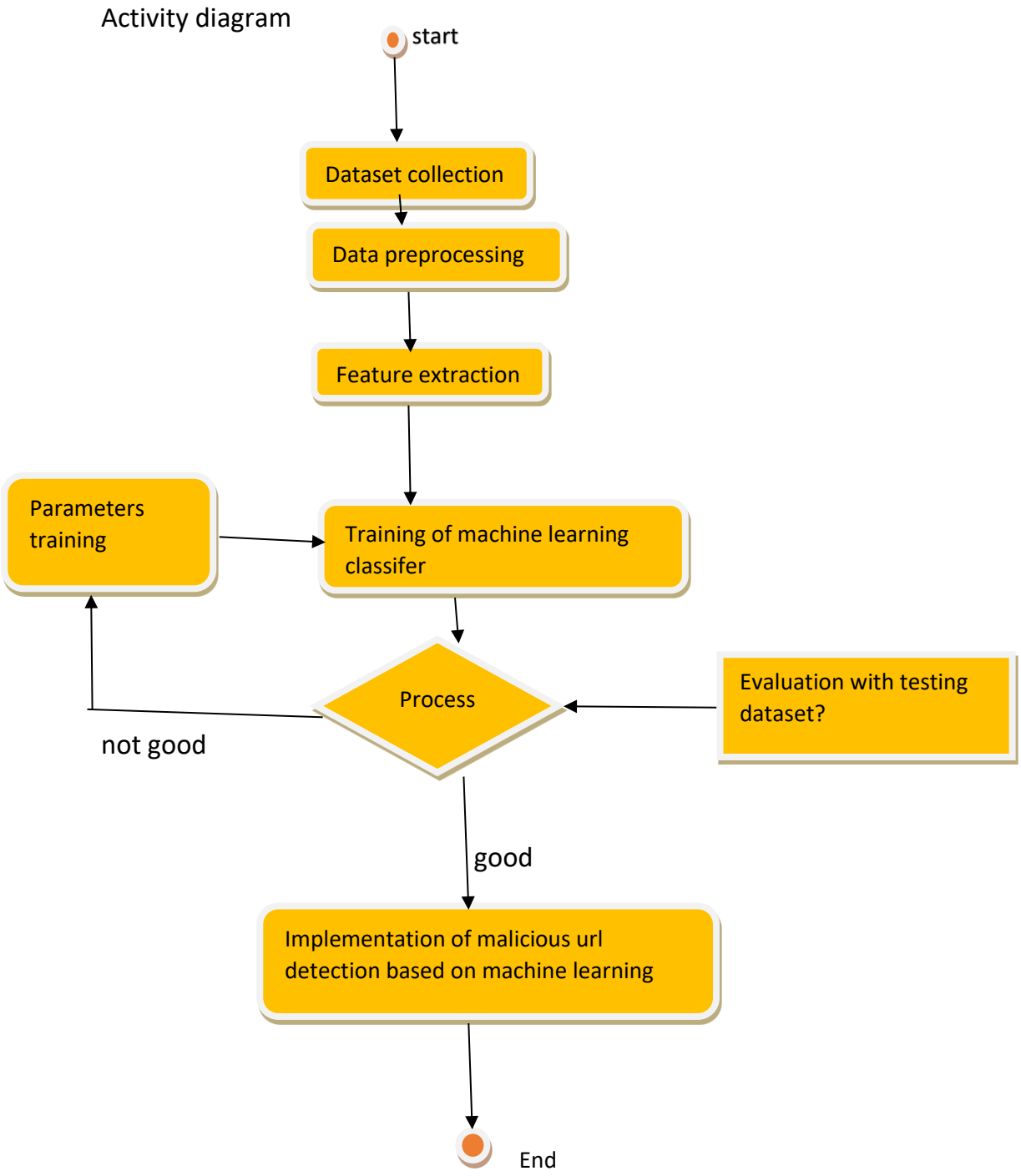


FIG 13 ACTIVITY DIAGRAM

STATE DIAGRAM

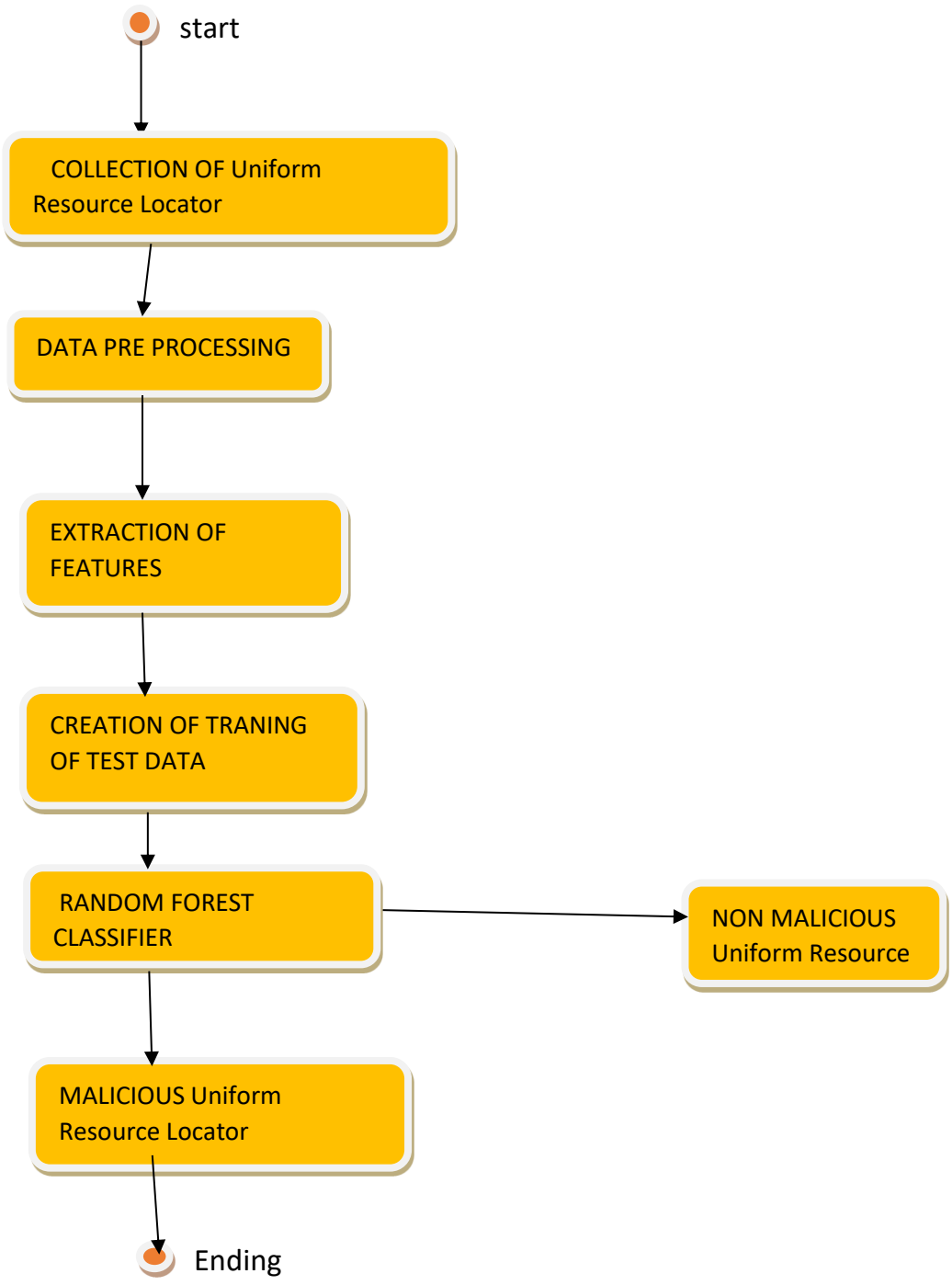


FIG 14 STATE DIAGRAM OF MODEL

DEPLOYMENT DIAGRAM

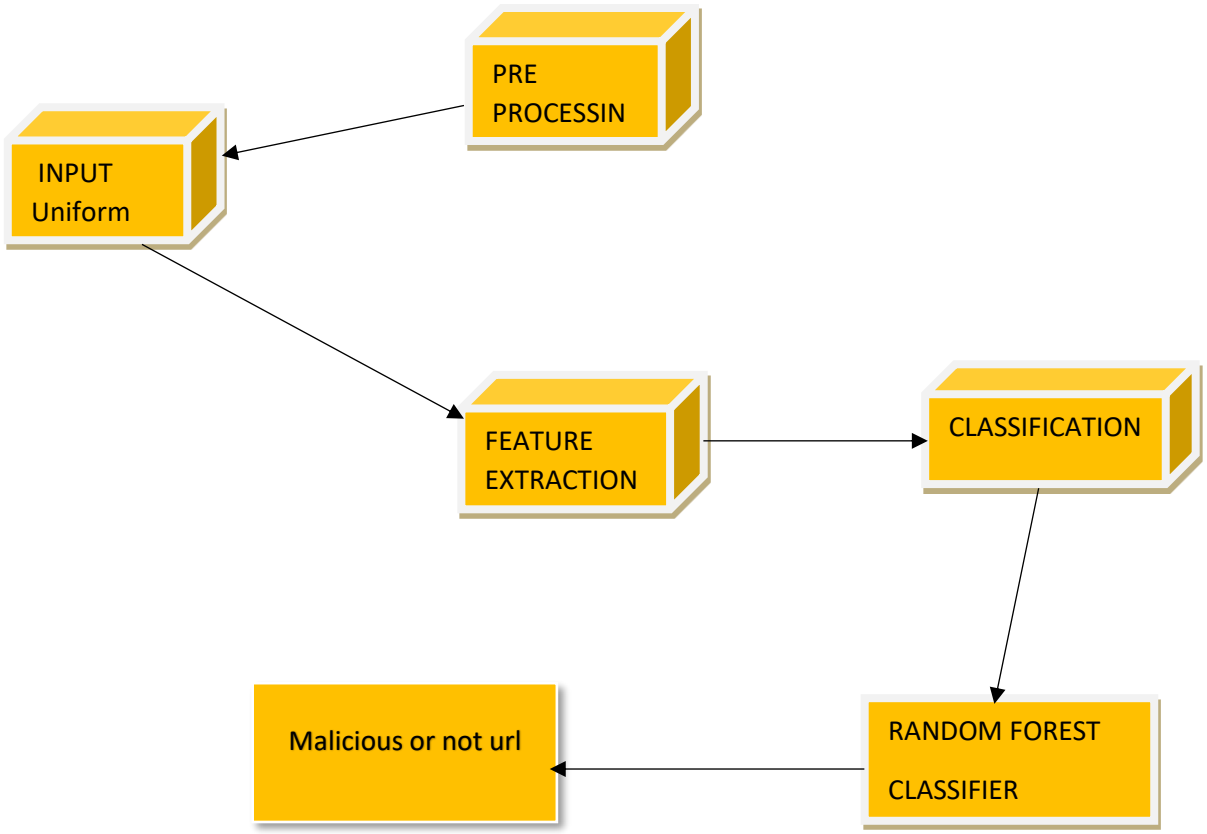


FIG 15 DEPLOYMENT DIAGRAM

4.3.7**SYSTEM IMPLEMENTATION**

In order to meet issue specifications, software is divided into modules, which are independently labelled and addressable pieces. The only aspect of software that makes it possible to control a programme conceptually is modularity. A module is a part of a programme. Programmes are composed of one or more independently produced modules that aren't connected until the very end. One or more routines can be found in a single module.

Educator Model:

Data Preparation: The code begins by importing the required libraries and pandas is then used to load the dataset. The first steps of data exploration, visualisation, and preparation are then carried out.

Engineering Features: Several features are created from the Uniform Resource Locator data, including the (i) number of dots,(ii) the number of www,(iii) the presence of a shortening service, (iv)the number of Hypertext Transfer Protocol and Hypertext Transfer Protocol,(v) the length of the Uniform Resource Locator and hostname, (vi)the number of digits and

letters, (vii)the number of subdomains,(viii) the length of the first directory, (ix)the length of the TLD(top-level domain) and more.

Label encode: Label encoding is a technique used in machine learning to convert categorical variables into numerical representations. It assigns a unique numerical label to each category in a categorical variable.

Data Splitting: The train_test_split function from sklearn is used to divide the dataset into training and testing sets. The training features and labels are contained in the variables X_train and y_train, while the testing features and labels are contained in the variables X_test and y_test.

Model Selection and Training: A number of techniques, including as (i)Random Forest,(ii) Logistic Regression,(iii)Support Vector Machine, and(iv) K-Nearest Neighbours, are used to train the model. The algorithms are implemented using sklearn in the code, and the training data is used to refine the algorithms.

To determine the ideal hyperparameters for each algorithm, Grid Search with Cross-Validation (GridSearchCV) is used. The programme specifies a hyperparameter grid with various parameter combinations that will be considered during grid search. The best model is then obtained together with its matching hyperparameters and use quantile transformerQuantile Transformer is a feature transformation technique used in machine learning

to map data to a uniform or Gaussian distribution. It transforms the features of a dataset so that they have a desired distribution, which can be useful in certain machine learning algorithms that assume specific data distributions.

Testing Scheme:-

Prediction: The labels for the testing data (X_test) are predicted using the best model discovered during the training procedure.

Evaluation: A number of evaluation metrics are computed to gauge the model's effectiveness. Accuracy, true positive rate, false positive rate, precision, recall, F-measure, and area under the ROC (receiver operating characteristic curve) are some of these measurements.

Visualisation: Using matplotlib, the code shows the ROC (receiver operating characteristic curve) and model accuracy.

Prediction: after using a function for prediction using this randomforest model algorithm and this function prediction happen by input the url for prediction

Parameters Used for Each Algorithm:

1. **Random Forest:**

   Hyperparameters tuned using grid search:

   - 'n_estimators': [50, 100, 200]: Number of decision trees in the random forest ensemble.
   - 'max_features': ['sqrt', 'log2']: The maximum number of features to consider when looking for the best split.

   .

2. **Logistic Regression:**

   Hyperparameters tuned using grid search:

   - 'C': [0.1, 1, 10]: Inverse of regularization strength. Smaller values specify stronger regularization.
   - 'solver': ['liblinear', 'saga']: Algorithm to use in the optimization problem. 'liblinear' uses a coordinate descent algorithm, while 'saga' uses a stochastic average gradient descent solver.

3. **Support Vector Machine (Support Vector Machine):**

Hyperparameters tuned using grid search:

- 'C': [0.1, 1, 10]: Regularization parameter. Controls the trade-off between misclassification and simplicity of the decision boundary.
- 'kernel': ['linear', 'rbf']: Kernel function to be used in the SVM algorithm. 'linear' uses a linear decision boundary, while 'rbf' uses a radial basis function

4. **K-Nearest Neighbors (K-Nearest Neighbors Algorithm):**

Hyperparameters tuned using grid search:

- 'n_neighbors': [3, 5, 7]: Number of neighbors to consider for classification.
- 'weights': ['uniform', 'distance']: The weight function used in prediction. 'uniform' assigns equal weight to all neighbors, while 'distance' assigns weights proportional to the inverse of the distance.

A)classification Technique:

Random forest :

Random Forest is an ensemble learning method for classification and regression tasks. It is a popular machine learning algorithm that combines multiple decision trees to make predictions. Here is a summary of the Random Forest algorithm:

Ensemble of Decision Trees: Random Forest builds an ensemble of decision trees, where each tree is trained on a different subset of the data. This process is called bagging or bootstrap aggregating. The trees are constructed independently of each other.

Random Feature Selection: At each split in the tree, Random Forest considers only a random subset of features. This random feature selection introduces diversity among the trees and helps to prevent overfitting.

Voting for Prediction: When making predictions, Random Forest combines the predictions of all the trees in the ensemble. For classification tasks, it uses majority voting, where the class that receives the most votes is selected

as the final prediction. For regression tasks, it takes the average of the predicted values.

Bootstrap Sampling: Random Forest uses bootstrap sampling to create different training sets for each tree. This means that each tree is trained on a random subset of the original data, with replacement. Some data points may appear multiple times in a subset, while others may be left out.

Tree Growth and Stopping Criteria: Each decision tree in the Random Forest grows until a stopping criterion is met. The stopping criterion could be reaching a maximum depth, having a minimum number of samples in a leaf node, or other conditions. These criteria prevent the trees from overfitting the training data.

Out-of-Bag Evaluation: Random Forest can estimate the performance of the model without the need for cross-validation. During training, each tree is evaluated using the samples that were not included in its bootstrap sample. This provides an unbiased estimate of the model's performance.

Robust to Overfitting: Random Forest is known for its ability to handle high-dimensional data and avoid overfitting. By using random feature selection and aggregating multiple trees, it reduces the variance and improves the generalization performance of the model.

Hyperparameter Tuning: Random Forest has several hyperparameters that can be tuned to optimize its performance. These include the number of trees in the ensemble, the maximum depth of each tree, the number of features considered at each split, and others.

Feature Importance: Random Forest provides a measure of feature importance, which indicates the relative importance of each feature in making predictions. This information can be useful for feature selection and understanding the underlying patterns in the data.

Overall, Random Forest is a versatile and powerful algorithm that can handle both classification and regression tasks. It is widely used in various domains due to its robustness, scalability, and interpretability.

Importance:

Random forest adds additional randomness to the model, while growing the trees. Instead of searching for the most important feature while splitting a node, it searches for the best feature among a random subset of features. This results in a wide diversity that generally results in a better model.

Random forest is a commonly-used machine learning algorithm trademarked by Leo Breiman and Adele Cutler, which combines the output of multiple decision trees to reach a single result. Its ease of use and flexibility have fueled its adoption, as it handles both classification and regression problems.

Advantages of random forest:

A random forest produces good predictions that can be understood easily. It can handle large datasets efficiently. The random forest algorithm provides a higher level of accuracy in predicting outcomes over the decision tree algorithm.

.

B) UNIFORM RESOURCE LOCATOR VISUALIZATION:

Collection of Data:- dataset has 691 Uniform Resource Locators

Details analysis of Uniform Resource Locator:

Bad Uniform Resource Locator:

Fig 16 - Details analysis of bad Uniform Resource Locator

From this figure we get to know the actual word cloud of the bad url

Good Uniform Resource Locator:

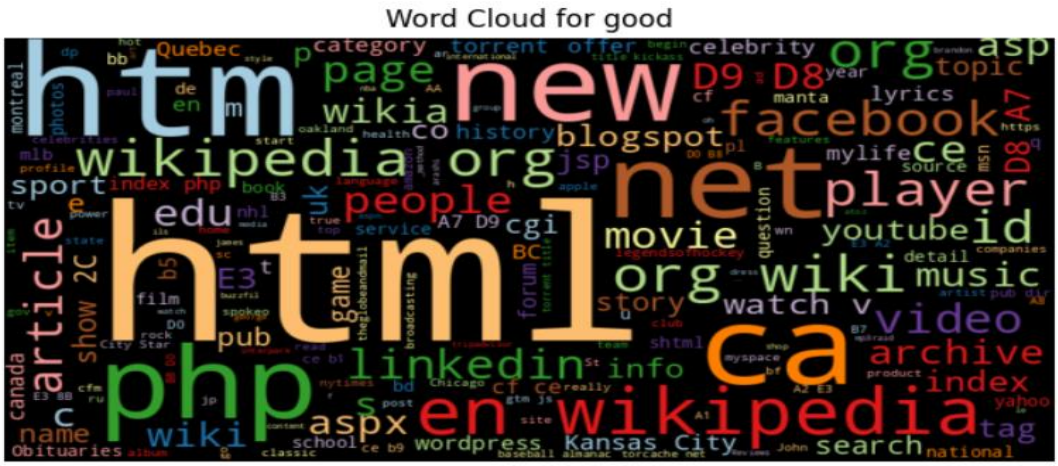


FIG 17 GOOD Uniform Resource Locator

From this figure we get to know the actual word cloud of the bad url

DATA PREPROCESSING:

First check the dataset has any null value or missing value and clean the dataset.

TABLE I FEATURE ENGINEERING

FEATURES
abnormal url
count('.')
count('www')
Count div
Shortend url
Count(https)
count('%')
count('?')
url_length
hostname_length
tld_length
Count_digits Count_letters
Subdomain_count
fd_length

The best characteristics for this model after doing chi-square tests on a variety of features, and these are the features.A statistical technique called the chi-square test is used to examine whether there is a meaningful correlation between two category variables. It evaluates whether there is a significant deviation between the observed and expected data distributions under a given hypothesis.

The chi-square test can be used to examine the relationship between the Uniform Resource Locator categories in the proposed model. (e.g., bad ,good) and the different features created using the Uniform Resource Locators. The test assists in determining whether there is a substantial correlation between each characteristic and the Uniform Resource Locator classification.

A contingency table is built that cross-tabulates the Uniform Resource Locator classifications against each feature in order to do the chi-square test. The observed frequencies of the various combinations of Uniform Resource Locator categories and feature values are shown in the contingency table.

By contrasting the observed frequencies in the contingency table with the frequencies anticipated under the supposition of independence between the Uniform Resource Locator classification and the feature, the chi-square test determines the chi-square statistics. If the null hypothesis of independence is true, the test yields a p-value that represents the likelihood of witnessing the observed distribution or a more extreme distribution.

It is suggested that there is a significant correlation between the Uniform Resource Locator classification and the characteristic if the p-value is less than a predetermined level of significance (for example, 0.05). This suggests that the feature offers useful data for differentiating across various groups of Uniform Resource Locators.

The features' relevance and significance in the categorization process can be evaluated by using the chi-square test. Low p-value features are thought to have a greater association with the classification of Uniform Resource Locators and can be helpful for precise classification.

Overall, by evaluating the statistical significance of the relationship between the Uniform Resource Locator classes and the designed features, the chi-square test assists in feature selection and validation by assisting in the identification of the most pertinent features for Uniform Resource Locator classification.

In the proposed classification technique, several features are engineered.

Here is a description of each of the characteristics that stated in relation to categorising aberrant Uniform Resource Locators:

(1)Abnormal Uniform Resource Locator: This feature indicates whether or not the Uniform Resource Locator is regarded as abnormal. We wish to make predictions about the target variable using the other attributes.

(2)Count('.'): This function determines how many periods (dots) are present in the Uniform Resource Locator. Because atypical Uniform Resource Locators could have an unusually high number of periods, it can be helpful.

(3) Count(‘www’): This function counts the number of times the string "www" appears in the Uniform Resource Locator. It's possible for abnormal Uniform Resource Locators to utilise "www" in different ways or not at all.

(4)Count div: This function counts the number of div elements present in the Uniform Resource Locator-relative Hypertext Markup Language content of the webpage. Since anomalous websites may have more or less div elements, it may be a sign of strange behaviour.

(5)Shortened Uniform Resource Locator: Whether a Uniform Resource Locator has been shortened or truncated is indicated by this feature. Shortened Uniform Resource Locators are frequently used to conceal the true location and are sometimes indicative of malicious or strange behaviour.

(6)Count('https'): This function counts the number of times the string "https" appears in the Uniform Resource Locator. It can be helpful because unusual Uniform Resource Locators may utilise "https" in different ways or not at all.

(7)Count('%'): This feature counts the occurrences of the percentage symbol (%) in the Uniform Resource Locator. Abnormal Uniform Resource Locators may use special characters in unusual ways.

(8)Count('?'): This feature counts the occurrences of the question mark symbol (?) in the Uniform Resource Locator. Question marks are often used in Uniform Resource Locators to pass parameters, and abnormal Uniform Resource Locators may have an unusual number or usage of question marks.

(9)Uniform Resource Locator length: This feature calculates the length of the Uniform Resource Locator in characters. Abnormal Uniform Resource Locators may be longer or shorter than normal Uniform Resource Locators.

(10)Hostname length: This feature calculates the length of the hostname portion of the Uniform Resource Locator. Abnormal Uniform Resource Locators may have unusual or longer hostnames.

(11)Count_digits: This feature counts the number of digits in the Uniform Resource Locator. Abnormal Uniform Resource Locators may have a higher or lower number of digits.

(12)Count_letters: This feature counts the number of letters (alphabetic characters) in the Uniform Resource Locator. Abnormal Uniform Resource Locators may have a higher or lower number of letters.

(13)Subdomain count: This feature counts the number of subdomains in the Uniform Resource Locator. Abnormal Uniform Resource Locators may have an unusual number of subdomains.

(14)FD length: This feature calculates the length of the first directory(FD) in the Uniform Resource Locator. Abnormal Uniform Resource Locators may have an unusual or longer first directory.

(15)TLD length: This feature calculates the length of the top-level domain (TLD) portion of the Uniform Resource Locator. Abnormal Uniform Resource Locators may have an unusual or longer TLD.

Chi-square test:

To identify the most informative features for classification, we apply the Chi-Square test. This statistical test measures the independence between each feature and the target variable (Uniform Resource Locator classification). We calculate the Chi-Square statistic and corresponding p-values for each feature. Features with low p-values indicate a significant association with the target variable and are selected for further analysis

	Feature	Chi-square	p-value
0	label	685.891124	3.497972e-151
1	use_of_ip	0.000000	1.000000e+00
2	abnormal_url	380.282093	1.080302e-84
3	google_index	0.000000	1.000000e+00
4	count.	199.371865	4.486270e-38
5	count_www	318.901295	5.642397e-70
6	count@	0.000000	1.000000e+00
7	count_dir	87.315412	5.619086e-15
8	count_embed_domian	0.000000	1.000000e+00
9	short_url	5.465682	1.939348e-02
10	count-https	0.000000	1.000000e+00
11	count-http	386.543376	1.156571e-84
12	count%	26.589988	4.627151e-02
13	count?	57.128134	4.083226e-14
14	count-	16.427057	6.898020e-01
15	count=	109.200301	5.558515e-20
16	url_length	193.104396	1.978244e-03
17	hostname_length	457.958452	6.985027e-80
18	sus_url	1.650069	1.989496e-01
19	count-digits	87.112804	6.377432e-06
20	count-letters	142.982383	9.700475e-03
21	suspicious_keywords	1.752954	1.855058e-01
22	dns_resolution	0.000000	1.000000e+00
23	subdomain_count	372.943302	4.279103e-83
24	double_dots	0.000000	1.000000e+00
25	non_alphanumeric_characters	0.000000	1.000000e+00
26	fd_length	190.127142	8.216584e-19
27	tld	70.397521	6.955517e-03
28	tld_length	402.930381	6.553312e-84

FIG 18 CHI-SQUARE TEST RESULT

MODELS DETAILS :

1. Random Forest: Random Forest is an ensemble learning method that combines multiple decision trees to make predictions. Each decision tree is built independently on a random subset of the training data and features. During prediction, each tree in the forest provides a prediction, and the final prediction is determined by majority voting or averaging. Random Forest is versatile, capable of handling both classification and regression tasks, and is robust against overfitting.

2. Support Vector Machines (Support Vector Machine): Support Vector Machine is a supervised learning algorithm used for both classification and regression tasks. The algorithm maps the input data to a higher-dimensional feature space and finds an optimal hyperplane that separates the classes in this space. Support Vector Machine aims to maximize the margin (distance) between the support vectors (data points near the decision boundary) of different classes. It can handle linear and non-linear classification tasks through the use of different kernels.

3. Logistic Regression: Logistic Regression is a statistical algorithm used for binary classification problems. Despite its name, it is a regression algorithm. Logistic Regression models the relationship between the input variables and the probability of a binary outcome using the logistic function (sigmoid function). It estimates the coefficients of the input variables and applies them to the logistic function to predict the probability of belonging to a certain class. A decision threshold is used to convert the probabilities into class labels.

4. K-Nearest Neighbors (K-Nearest Neighbors Algorithm): K-Nearest Neighbors is a simple and intuitive classification algorithm. Given a new data point, it looks at the k closest labeled data points in the training set (where k is a user-defined parameter). The class label of the majority of the k nearest neighbors is assigned to the new data point. K-Nearest Neighbors Algorithm is non-parametric and makes no assumptions about the underlying data distribution. It can handle both classification and regression tasks and can be influenced by the choice of distance metric.

6. Classification Report: The classification_report function from the sklearn.metrics module is used to generate a report that provides metrics such as precision, recall, F1-score, and support for each class (e.g., "Bad",’’Good”). Precision represents the ratio of correctly predicted positive

samples to the total predicted positive samples. Recall represents the ratio of correctly predicted positive samples to the total actual positive samples. The F1-score is the harmonic mean of precision and recall, providing a balanced measure of the model's performance.

7. Confusion Matrix: The confusion_matrix function from the sklearn.metrics module is used to generate a confusion matrix. A confusion matrix presents the counts of true positive, false positive, true negative, and false negative predictions. It provides a detailed breakdown of the model's performance across different classes, allowing for a deeper analysis of the classification results.

Performance metrics:-

Complete explanation of precision, recall, and F1-score, accuracy which are important evaluation metrics used in the classification report:

1. Precision:

Precision is a metric that measures the accuracy of positive predictions made by the model. It is calculated as the ratio of true positives (TP) to the sum of true positives and false positives (FP). In the context of Uniform Resource Locator classification, precision represents the proportion of correctly predicted malicious Uniform Resource Locators (phishing or defacement) out of all Uniform Resource Locators predicted as malicious.

Mathematically, precision can be expressed as:

Precision = TP / (TP + FP)

A high precision value indicates that when the model predicts a Uniform Resource Locator as malicious, it is highly likely to be accurate.

2. Recall (Sensitivity):

Recall, also known as sensitivity or true positive rate, measures the ability of the model to correctly identify positive instances from the actual positive instances in the dataset. It is calculated as the ratio of true positives (TP) to the sum of true positives and false negatives (FN). In Uniform Resource Locator classification, recall represents the proportion of correctly identified malicious Uniform Resource Locators out of all actual malicious Uniform Resource Locators.

Mathematically, recall can be expressed as:

Recall = TP / (TP + FN)

A high recall value indicates that the model is effective at capturing most of the actual malicious Uniform Resource Locators.

3. F1-score:

The F1-score is a single metric that combines precision and recall into a balanced measure of the model's performance. It is the harmonic mean of precision and recall and provides a single score that reflects both metrics. The F1-score considers both the ability of the model to make accurate

positive predictions (precision) and its ability to capture actual positive instances (recall).

Mathematically, the F1-score can be calculated as:

F1-score = 2 * (Precision * Recall) / (Precision + Recall)

The F1-score ranges from 0 to 1, where 1 indicates a perfect balance between precision and recall.

These metrics (precision, recall, and F1-score) are commonly used in classification tasks to assess the model's performance and provide insights into its strengths and weaknesses. They offer a more detailed understanding of how well the model is performing for different classes (e.g., "Benign," "Phishing," "Defacement") and help in evaluating and comparing different models or tuning their parameters.

4. Accuracy Calculation: The accuracy of the classification model is calculated using the formula: Accuracy = (TP + TN) / (TP + TN + FP + FN) where TP (True Positive) represents the number of correctly classified positive samples, TN (True Negative) represents the number of correctly classified negative samples, FP (False Positive) represents the number of incorrectly classified negative samples, and FN (False Negative) represents the number of incorrectly classified positive samples. The accuracy metric provides an overall measure of the model's performance.

ROC (Receiver Operating Characteristic Curve)CURVE:

The performance of a classification model is graphically represented by the Receiver Operating Characteristic (ROC) curve. It demonstrates how, at various categorization thresholds, the true positive rate (sensitivity) and the false positive rate (1-specificity) trade off. The ROC curve enables the setting of an ideal threshold based on the desired balance between sensitivity and specificity and offers insightful information about the model's capacity for class discrimination. The area under the curve (AUC) value of 1 denotes flawless classification, and the closer the ROC curve is to the top-left corner, the better the model performs. The ROC curve aids in the evaluation and comparison of various models by providing a s summary of a classifier's performance.

LANGUAGE SPECIFICATION:

Python is a widely used high-level programming language known for its simplicity and readability. It offers a vast range of libraries and frameworks that make it a popular choice for various applications, including data analysis, machine learning, web development, and more. Python's syntax is clean and easy to understand, making it suitable for both beginners and experienced programmers.

CHARACTERISTICS OF PYTHON:

Python, as a programming language, possesses several characteristics that contribute to its popularity and widespread usage. Here are some key characteristics of Python:

1. Readability and Simplicity: Python emphasizes clean and readable code, making it easy to understand and maintain. It uses indentation and whitespace to define code blocks, eliminating the need for excessive syntax and braces.
2. Expressive and Concise: Python allows developers to accomplish more with fewer lines of code. It provides expressive syntax and built-in constructs that facilitate common programming tasks, reducing the need for boilerplate code.

3. Dynamically Typed: Python is dynamically typed, which means you don't need to declare variable types explicitly. Variables can hold values of any type, and their types can change during runtime. This flexibility simplifies development and speeds up prototyping.

4. Interpreted and Interactive: Python is an interpreted language, which means the code is executed line by line without the need for explicit compilation. This allows for quick development cycles and interactive programming in environments like the Python shell.

5. Cross-Platform Compatibility: Python is available on various operating systems, including Windows, macOS, and Linux. This cross-platform compatibility allows developers to write code once and run it on different platforms without significant modifications.

6. Vast Ecosystem: Python has a rich and extensive ecosystem of libraries and frameworks that enhance its capabilities. These include popular libraries like NumPy, Pandas, Matplotlib, and frameworks like Django, Flask, and TensorFlow. The availability of such tools enables developers to leverage existing solutions and accelerate development.

7. Strong Community and Support: Python has a large and active community of developers, which contributes to its growth and evolution. The community provides extensive documentation,

tutorials, and forums for sharing knowledge and helping fellow developers.

8. Integration and Extensibility: Python supports seamless integration with other languages like C/C++ and Java, allowing developers to combine Python with existing codebases. It also provides easy-to-use interfaces for integrating with databases, web services, and system operations.

ENVIRONMENT CHOOSE:

GOOGLE COLAB:

Google Colab, short for Google Colaboratory, is a cloud-based development environment that provides a Jupyter Notebook-like interface to write, run, and share Python code. It is a part of the Google Cloud ecosystem and is designed to simplify the process of working with Python and data science libraries.

Here are some key details about Google Colab:

1. Cloud-based Environment: Google Colab runs entirely on the cloud, which means you don't need to install any software or worry about system compatibility. It allows you to access and work on your notebooks from any device with an internet connection.
2. Jupyter Notebook Integration: Google Colab supports the Jupyter Notebook format and provides a similar interface. Notebooks are

organized into cells, where you can write and execute code, view results, and add text-based explanations using Markdown.

3. Free and Paid Versions: Google Colab offers both free and paid tiers. The free version provides a generous amount of computational resources, including GPU support, but with some limitations on usage time and availability of high-end hardware. The paid version, called Colab Pro, offers additional benefits, such as increased usage limits, priority access to resources, and more.

4. GPU(Graphics Processing Units) and TPU (Tensor Processing Units) Support: Google Colab provides access to powerful hardware accelerators like GPUs (Graphics Processing Units) and TPUs (Tensor Processing Units). This is especially useful for training machine learning models and running computationally intensive tasks at a faster pace.

5. Preinstalled Libraries and Dependencies: Google Colab comes with several popular Python libraries preinstalled, including TensorFlow, NumPy, Pandas, Matplotlib, and scikit-learn. This eliminates the need for manual installations and allows you to start working on data analysis and machine learning projects immediately.

6. File and Data Storage: Google Colab integrates with Google Drive, allowing you to easily import and export notebooks and access files stored in your Google Drive. It also provides temporary storage that

persists for the duration of the session, enabling you to upload and process data within the environment.

7. Collaboration and Sharing: Google Colab supports real-time collaboration, allowing multiple users to work on the same notebook simultaneously. You can share your notebooks with others, granting them view or edit access. Additionally, you can publish your notebooks to GitHub or use the "Open in Colab" button to share interactive versions of your notebooks.

8. Integration with Google Services: Being part of the Google ecosystem, Google Colab integrates seamlessly with other Google services, such as BigQuery for data analysis, Google Sheets for data import/export, and Google Cloud Storage for data storage and retrieval.

FLOW DIAGRAM:

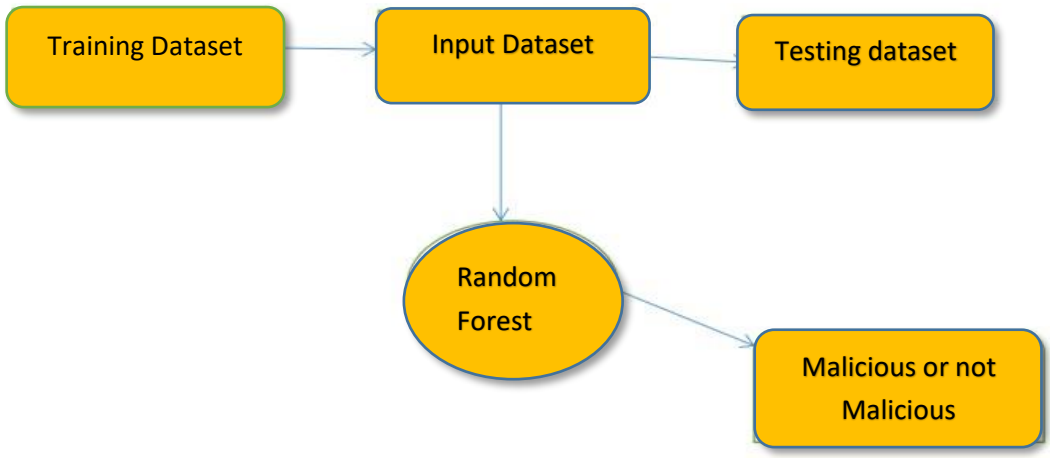


FIG 19 FLOW DIAGRAM

SYSTEM ARCHITECTURE :

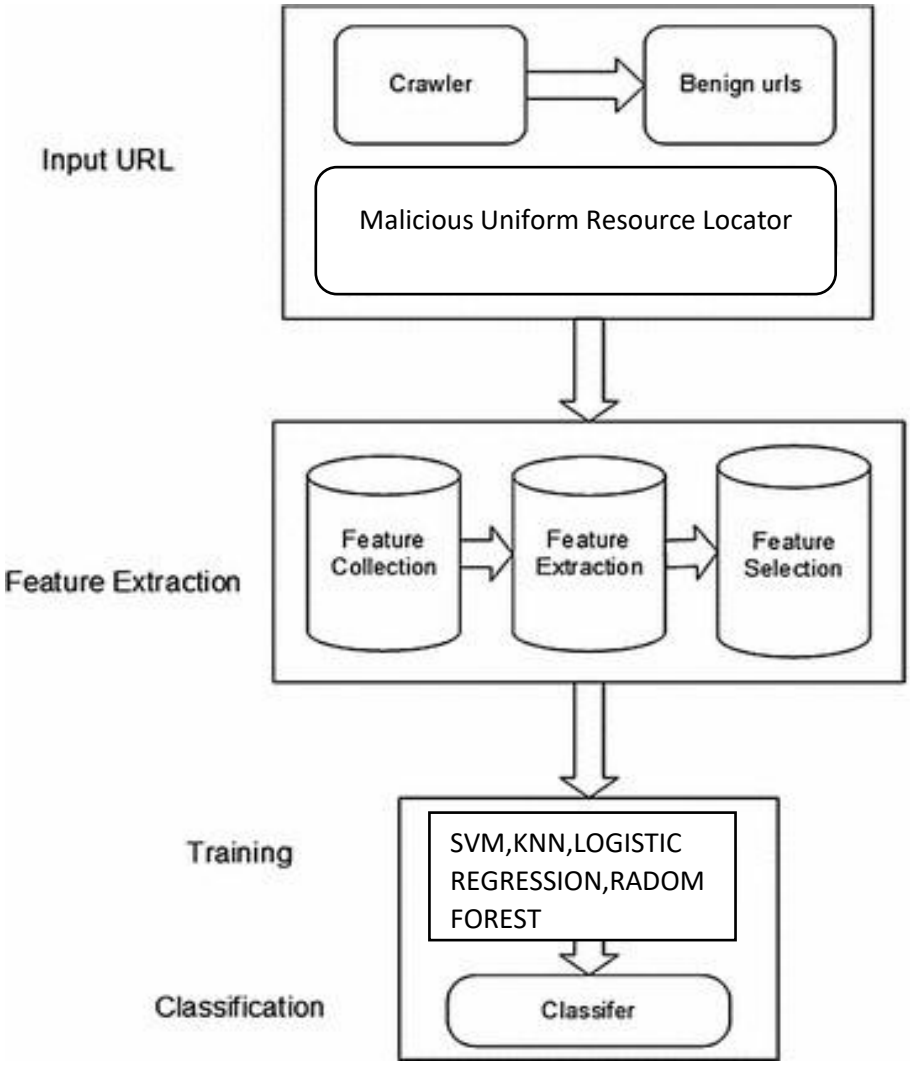


FIG 20 SYSTEM ARCHITECTURE

BLOCK DIAGRAM:

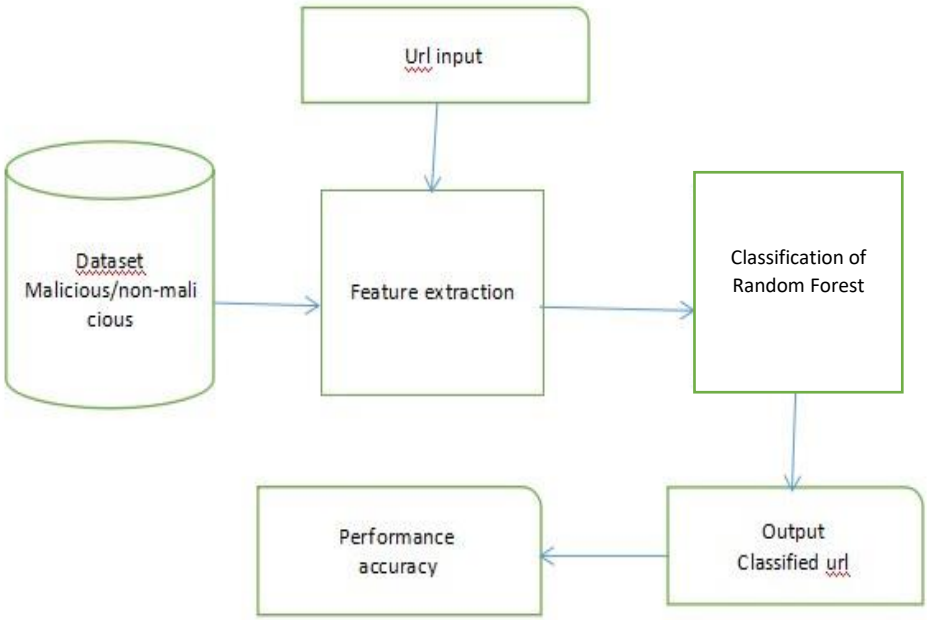


FIG 21 BLOCK DIAGRAM

Chapter 5

Experimentation and Results:

The provided information describes the system implementation of a software project that focuses on classifying Uniform Resource Locators as either good or bad based on various features. The software is divided into modules, each addressing a specific aspect of the program. The implementation includes an Educator Model that involves data preparation, feature engineering, data splitting, model selection, and training. Different machine learning algorithms such as Random Forest, Logistic Regression, Support Vector Machine, and K-Nearest Neighbors are used for training the model, with hyperparameters tuned using techniques like Grid Search with Cross-Validation.

The testing scheme involves predicting labels for testing data and evaluating the model's performance using metrics such as accuracy, true positive rate, false positive rate, precision, recall, F-measure, and area under the ROC curve. Visualization techniques, such as plotting the ROC curve, confusion

matrix and model accuracy, are employed to provide insights into the model's performance.

The software implementation also includes a description, explaining the classification technique used in a Random Forest for Uniform Resource Locator classification.

Random Forest provides the final result by aggregating the predictions of multiple decision trees in the ensemble. Each decision tree independently makes a prediction based on the features of the input data. In classification tasks, the final prediction is determined by majority voting, where the class that receives the most votes from the individual trees is selected as the predicted class. In regression tasks, the final prediction is the average of the predicted values from all the trees. This ensemble approach helps to reduce the impact of individual tree's errors and provides a more robust and accurate prediction. Random Forest's ability to combine the knowledge from multiple trees results in a more reliable and stable model that can handle complex datasets and generalize well to unseen data.The

During the training phase, Random Forest constructs an ensemble of decision trees by using a technique called bootstrap aggregating (or bagging). The process begins by randomly selecting subsets of the original training data, allowing for replacement (i.e., some samples may appear multiple times while others may not appear at all). For each subset, a decision tree is trained using a random subset of features. This randomness helps in

reducing overfitting and decorrelating the trees. Each decision tree is grown using a recursive process where, at each node, the best split is determined based on a selected criterion (e.g., Gini impurity or information gain). This splitting process continues until a stopping criterion is met, such as reaching a maximum tree depth or a minimum number of samples per leaf. Once all the decision trees are trained, the ensemble predictions are made by aggregating the predictions of individual trees (e.g., majority voting for classification or averaging for regression). By combining the predictions of multiple trees, Random Forest achieves higher accuracy, robustness to noise, and the ability to handle high-dimensional data.

Data visualization techniques are employed to gain insights into the dataset, such as viewing the dataframe and generating word clouds for good and bad Uniform Resource Locators. Data processing, including checking for null values and missing values, is essential in data analysis and machine learning.

Data Quality: Checking for null values and missing values helps ensure the quality and integrity of the data. Missing data can introduce bias and lead to inaccurate or incomplete analysis results. By identifying and handling missing values appropriately, we can avoid making erroneous assumptions or drawing incorrect conclusions based on incomplete information.

Data Completeness: Missing values can occur due to various reasons such as data collection errors, data entry issues, or data corruption. Identifying missing values allows us to assess the completeness of the dataset. If a significant portion of data is missing, it may impact the reliability and validity of the analysis or modeling results. Addressing missing values helps in maintaining the completeness of the dataset.

Data Preprocessing: Missing values can cause problems when applying machine learning algorithms as most algorithms cannot handle missing data. Therefore, data preprocessing techniques such as imputation or removal of missing values are necessary to ensure the dataset is suitable for analysis. Imputation methods, such as mean imputation or regression imputation, can estimate missing values based on available data, while removal of missing values can be done if the missing values are relatively small in number or do not significantly impact the analysis.

Data Bias: Null values or missing values can introduce bias in the dataset, leading to biased analysis or model predictions. It is crucial to understand the patterns and reasons behind missing data to minimize bias. For example,

if missing values occur systematically for a particular group or feature, it can skew the analysis results and affect the fairness and accuracy of the models.

Data Understanding: Analyzing null values and missing values helps in understanding the characteristics and patterns of the data. It allows us to identify potential data collection issues, explore the reasons behind missingness, and make informed decisions about handling missing values. Understanding the extent and impact of missing data enables better interpretation of the analysis results and improves data-driven decision making.

In summary, data processing tasks like checking null values and missing values are vital for ensuring data quality, completeness, and reliability. By addressing missing data appropriately, we can mitigate bias, maintain data integrity, and improve the accuracy and validity of data analysis and machine learning models.

Feature engineering is a crucial step in the system implementation, where relevant features are selected using chi-square tests. These tests examine the relationship between Uniform Resource Locator categories (good and bad) and the different features created from the Uniform Resource Locators. A contingency table is constructed to cross-tabulate the Uniform Resource

Locator classifications against each feature, and the chi-square test determines the statistical significance of the relationship. Features with low p-values are considered to have a significant association with Uniform Resource Locator classification.

The system implementation includes details about each feature, such as abnormal Uniform Resource Locator, count of dots, count of "www," shortened Uniform Resource Locator, counts of certain characters, Uniform Resource Locator length, hostname length, count of digits and letters, subdomain count, length of the first directory, and length of the top-level domain. The chi-square test helps in feature selection and validation, aiding in the identification of the most relevant features for Uniform Resource Locator classification.

The software implementation also covers details of different classification models used, including Random Forest, Support Vector Machines (Support Vector Machine), Logistic Regression, and K-Nearest Neighbors (K-Nearest Neighbors Algorithm). Each model is briefly explained, along with the use of hyperparameter optimization, quantile transformer, to achieve the best accuracy.

The classification report provides metrics such as precision, recall, F1-score, and support for each class, while the confusion matrix offers a detailed breakdown of the model's performance across different classes. These

evaluation metrics help analyse the model's performance and assess its effectiveness in Uniform Resource Locator classification.

Overall, the system implementation combines various techniques and algorithms to create a software solution for Uniform Resource Locator classification, incorporating data pre-processing, feature engineering, model training, and evaluation.

```
[ ]  1 urls = ['titaniumcorporate.co.za','en.wikipedia.org/wiki/North_Dakota']
     2 for url in urls:
     3     print(get_prediction_from_url(url))

BAD
SAFE
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does
  warnings.warn(
```

FIG 22 RESULT

This is the prediction result that the model is working well and do prediction right.

Process of Random forest:

Demonstrates the process of training a random forest classifier with hyperparameter tuning and using it to make predictions on new URLs. Here's a step-by-step explanation of the process:

Import the necessary libraries: It begins by importing the required libraries, including scikit-learn's metrics module for evaluation, RandomForestClassifier for building the classifier, GridSearchCV for hyperparameter tuning, QuantileTransformer for data preprocessing, and make_pipeline for creating a pipeline of transformers and estimators.

Define the parameter grid: A parameter grid is defined to specify the hyperparameters to be tuned. In this case, the grid includes 'n_estimators' (number of trees in the random forest) and 'max_features' (the number of features to consider when splitting a node).

Create a pipeline: A pipeline is created using make_pipeline, which combines the QuantileTransformer and RandomForestClassifier. The QuantileTransformer is used for feature scaling and normalization, while the RandomForestClassifier is the estimator for the random forest model.

Perform hyperparameter tuning: GridSearchCV is employed to perform hyperparameter tuning. It takes the pipeline, parameter grid, and cross-validation configuration (cv=5) as input. GridSearchCV exhaustively searches the parameter grid and evaluates the performance of each combination using cross-validation.

Get the best estimator: After grid search is completed, the best estimator (model with optimal hyperparameters) is obtained using grid_search.best_estimator_.

Make predictions on the test set: The best model is then used to make predictions on the test set (X_test) using the predict function. The predictions are stored in y_pred_rf.

Print the classification report: The classification_report function from metrics is used to generate a report that includes precision, recall, F1-score, and support for each class (bad and good). This report provides insights into the performance of the model.

Calculate accuracy score: The accuracy_score function from metrics is used to calculate the accuracy of the model by comparing the predicted labels (y_pred_rf) with the actual labels (y_test). The accuracy score is then printed.

Define the main function: The main function takes a URL as input and performs various feature extraction operations on the URL to generate a list of features. The features are appended to the status list.

Define the get_prediction_from_url function: This function takes a test URL, calls the main function to extract features, reshapes the feature array to be a 2D array, and then uses the best_model to make a prediction on the features. The predicted label is converted to a string representing the class (BAD or SAFE) and returned.

Perform predictions on multiple URLs: The code includes a list of URLs. It iterates through the URLs and calls the get_prediction_from_url function for each URL. The predicted label is printed for each URL.

Overall, this demonstrates the process of training a random forest classifier, tuning its hyperparameters, and using it to predict the safety of URLs based on various features extracted from the URLs.

Random Forest Classification Report:

```
-----------------------
Classifier: Random Forest
              precision    recall  f1-score   support

         bad       1.00      0.97      0.99        37
        good       0.99      1.00      1.00       102

    accuracy                           0.99       139
   macro avg       1.00      0.99      0.99       139
weighted avg       0.99      0.99      0.99       139

-----------------------
```

FIG 23 RANDOM FOREST

RANDOM FOREST CONFUSION MATRIX:

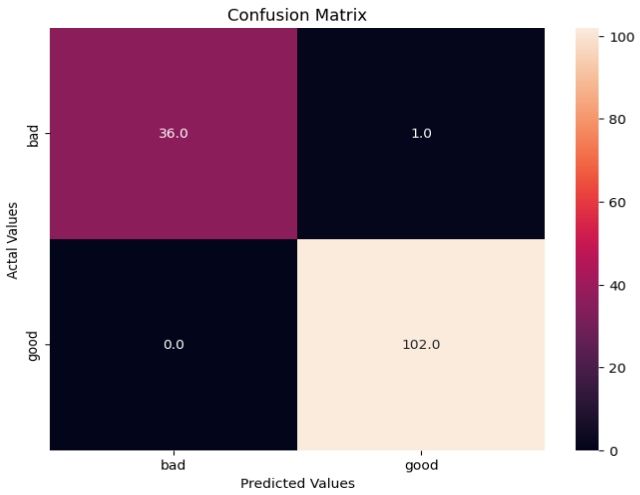


FIG 24 RANDOM FOREST CONFUSION MATRIX

RANDOM FOREST ROC CURVE:

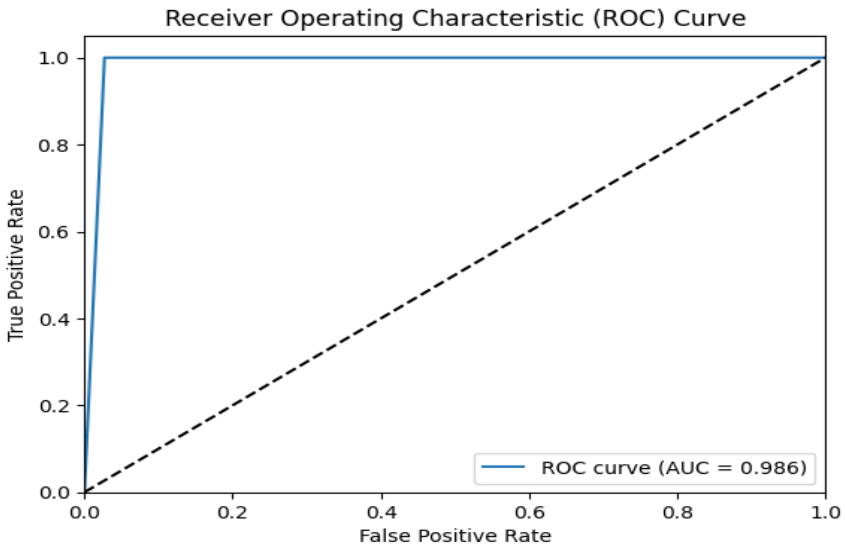


FIG 25 RANDOM FOREST ROC CURVE

RANDOM FOREST FEATURE IMPORTANCE:

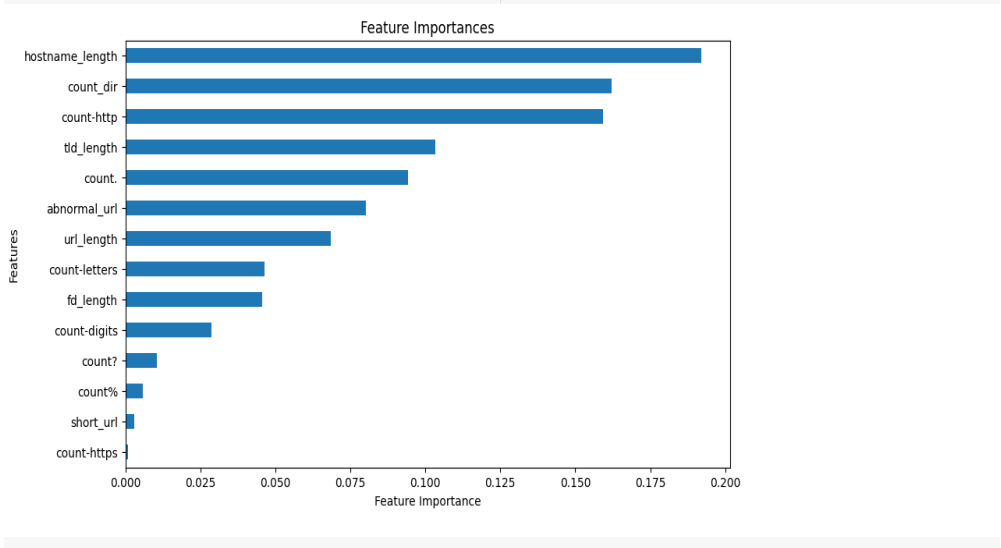


FIG 26 FEATURE IMPORTANCE OF RANDOM FOREST

PROCESS OF SUPPORT VECTOR MACHINE:

The process of training a Support Vector Machine (SVM) classifier using scikit-learn. Here's a step-by-step explanation of the process:

Import the necessary libraries: The code starts by importing the required libraries, including metrics from scikit-learn for evaluation, SVC for building the SVM classifier, GridSearchCV for hyperparameter tuning, QuantileTransformer for data preprocessing, and make_pipeline for creating a pipeline of transformers and estimators.

Define the parameter grid for SVM: A parameter grid specific to the SVM classifier is defined, which includes the 'C' parameter (regularization parameter) and the 'kernel' parameter (type of kernel function to be used).

Create an SVM pipeline: A pipeline is created using make_pipeline, which combines the QuantileTransformer and SVC. The QuantileTransformer is used for feature scaling and normalization, while SVC is the estimator for the SVM model.

Perform hyperparameter tuning for SVM: GridSearchCV is employed to perform hyperparameter tuning for the SVM classifier. It takes the SVM pipeline, parameter grid for SVM, and cross-validation configuration (cv=5) as input. GridSearchCV exhaustively searches the parameter grid and evaluates the performance of each combination using cross-validation.

Get the best SVM estimator: After grid search is completed, the best estimator (model with optimal hyperparameters) for the SVM classifier is obtained using svm_grid_search.best_estimator_.

Make predictions on the test set using SVM: The best SVM model is then used to make predictions on the test set (X_test) using the predict function. The predictions are stored in y_pred_svm.

Print the SVM classification report: The classification_report function from metrics is used to generate a report that includes precision, recall, F1-score, and support for each class (bad and good) for the SVM classifier. This report provides insights into the performance of the SVM model.

Calculate SVM accuracy score: The accuracy_score function from metrics is used to calculate the accuracy of the SVM model by comparing the predicted labels (y_pred_svm) with the actual labels (y_test). The accuracy score is then printed.

Overall, this demonstrates the process of training an SVM classifier, tuning its hyperparameters, and using it to predict the class labels of the test set based on the provided features.

CLASSIFICATION REPORT OF SUPPORT VECTOR MACHINE:

```
----------------------
Classifier: Support Vector Machine
              precision    recall  f1-score   support

         bad       0.97      0.97      0.97        37
        good       0.99      0.99      0.99       102

    accuracy                           0.99       139
   macro avg       0.98      0.98      0.98       139
weighted avg       0.99      0.99      0.99       139
```

FIG 27 SUPPORT VECTOR MACHINE CLASSIFICATION REPORT

CONFUSION MATRIX OF SUPPORT VECTOR MACHINE:

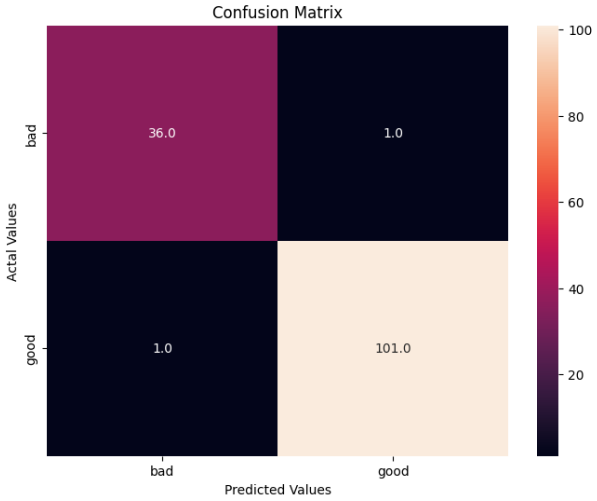


FIG 28 SUPPORT VECTOR MACHINE CONFUSION MATRIX

FEATURE IMPORTANCE OF SUPPORT VECTOR MACHINE:

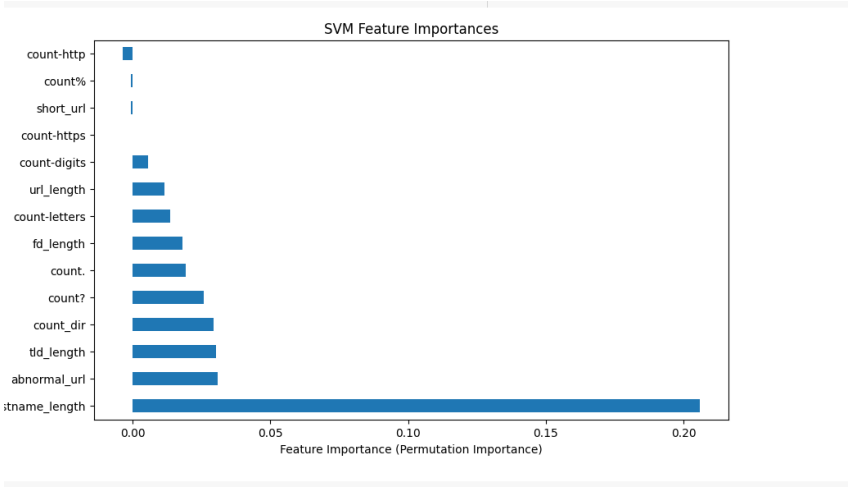


FIG 29 FEATURE IMPORTANCE OF SUPPORT VECTOR MACHINE

ROC CURVE OF SUPPORT VECTOR MACHINE:

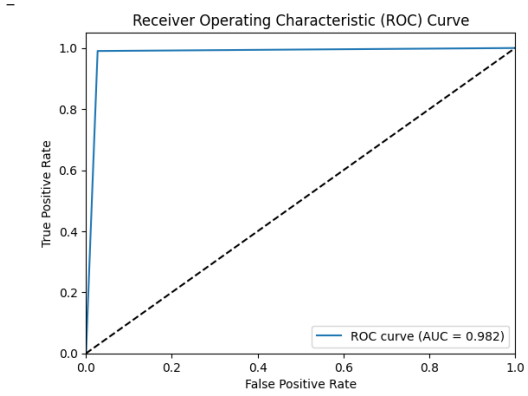


FIG 30 ROC CURVE OF SUPPORT VECTOR MACHINE

PROCESS OF LOGISTIC REGRESSION:

Demonstrates the process of training a Logistic Regression classifier using scikit-learn. Here's a breakdown of the steps involved:

Import the necessary libraries: The code imports the required libraries, including LogisticRegression from scikit-learn for building the logistic regression classifier, metrics for evaluation, SVC for support vector classifier (which is not used in this code snippet), GridSearchCV for hyperparameter tuning, QuantileTransformer for data preprocessing, and make_pipeline for creating a pipeline.

Define the parameter grid for logistic regression: A parameter grid specific to the logistic regression classifier is defined. It includes the 'C' parameter (inverse of regularization strength) and the 'solver' parameter (algorithm to use for optimization).

Create a logistic regression pipeline: A pipeline is created using make_pipeline, which combines the QuantileTransformer and LogisticRegression. The QuantileTransformer is used for feature scaling and

normalization, while LogisticRegression is the estimator for the logistic regression model.

Perform hyperparameter tuning for logistic regression: GridSearchCV is used to perform hyperparameter tuning for the logistic regression classifier. It takes the logistic regression pipeline, the parameter grid, and cross-validation configuration (cv=5) as input. GridSearchCV performs an exhaustive search over the parameter grid and evaluates the performance of each combination using cross-validation.

Get the best logistic regression estimator: After the grid search is completed, the best estimator (model with optimal hyperparameters) for the logistic regression classifier is obtained using logreg_grid_search.best_estimator_.

Make predictions on the test set using logistic regression: The best logistic regression model is then used to make predictions on the test set (X_test) using the predict function. The predictions are stored in y_pred_logreg.

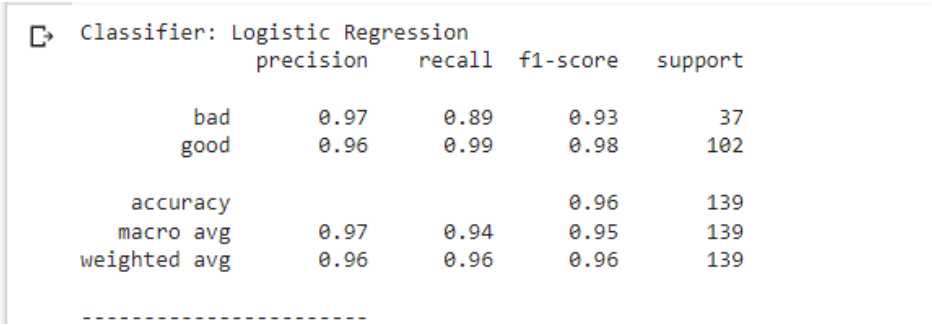
Print the logistic regression classification report: The classification_report function from metrics is used to generate a report that includes precision,

recall, F1-score, and support for each class (bad and good) for the logistic regression classifier. This report provides insights into the performance of the logistic regression model.

Calculate logistic regression accuracy score: The accuracy_score function from metrics is used to calculate the accuracy of the logistic regression model by comparing the predicted labels (y_pred_logreg) with the actual labels (y_test). The accuracy score is then printed.

In summary, this showcases the process of training a logistic regression classifier, tuning its hyperparameters, and using it to predict the class labels of the test set based on the provided features.

CLASSIFICATION DETAILS OF LOGISTIC REGRESSION:



Classifier: Logistic Regression

	precision	recall	f1-score	support
bad	0.97	0.89	0.93	37
good	0.96	0.99	0.98	102
accuracy			0.96	139
macro avg	0.97	0.94	0.95	139
weighted avg	0.96	0.96	0.96	139

FIG 31 CLASSIFICATION REPORT OF LOGISTIC REGRESSION

CONFUSION MATRIX OF LOGISTIC REGRESSION:

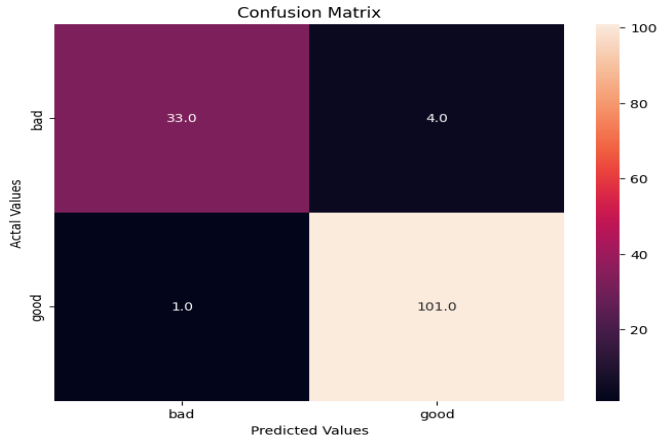


FIG 32 CONFUSION MATRIX OF LOGISTIC REGRESSION

FEATURE IMPORTANCE OF LOGISTIC REGRESSION:

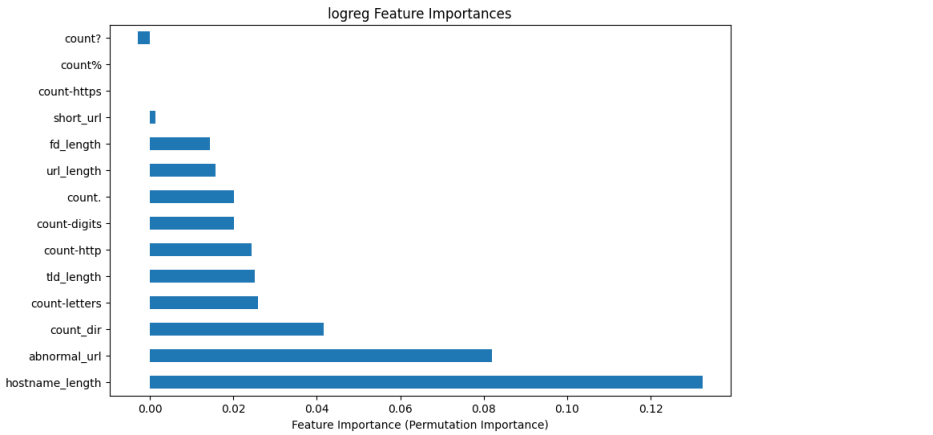


FIG 33 FEATURE IMPORTANCE OF LOGISTIC REGRESSION

ROC CURVE OF LOGISTIC REGRESSION:

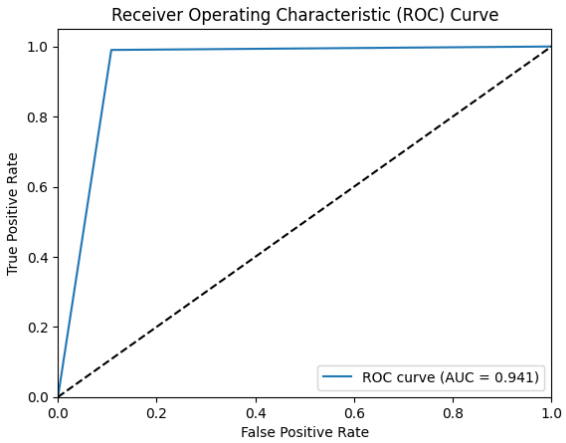


FIG 34 ROC CURVE OF LOGISTIC REGRESSION

PROCESS OF K-NEAREST NEIGHBORS ALGORITHM:

Demonstrates the process of training a K-Nearest Neighbors (KNN) classifier using scikit-learn. Here's an explanation of the steps involved:

Import the necessary libraries: The code imports the required libraries, including KNeighborsClassifier from scikit-learn for building the KNN classifier, metrics for evaluation, SVC for support vector classifier (which is not used in this code snippet), GridSearchCV for hyperparameter tuning, QuantileTransformer for data preprocessing, and make_pipeline for creating a pipeline.

Define the parameter grid for KNN: A parameter grid specific to the KNN classifier is defined. It includes the 'n_neighbors' parameter (number of neighbors to consider) and the 'weights' parameter (weighting scheme for neighbors).

Create a KNN pipeline: A pipeline is created using make_pipeline, which combines the QuantileTransformer and KNeighborsClassifier. The

QuantileTransformer is used for feature scaling and normalization, while KNeighborsClassifier is the estimator for the KNN model.

Perform hyperparameter tuning for KNN: GridSearchCV is used to perform hyperparameter tuning for the KNN classifier. It takes the KNN pipeline, the parameter grid, and cross-validation configuration (cv=5) as input. GridSearchCV performs an exhaustive search over the parameter grid and evaluates the performance of each combination using cross-validation.

Get the best KNN estimator: After the grid search is completed, the best estimator (model with optimal hyperparameters) for the KNN classifier is obtained using knn_grid_search.best_estimator_.

Make predictions on the test set using KNN: The best KNN model is then used to make predictions on the test set (X_test) using the predict function. The predictions are stored in y_pred_knn.

Print the KNN classification report: The classification_report function from metrics is used to generate a report that includes precision, recall, F1-score,

and support for each class (bad and good) for the KNN classifier. This report provides insights into the performance of the KNN model.

Calculate KNN accuracy score: The accuracy_score function from metrics is used to calculate the accuracy of the KNN model by comparing the predicted labels (y_pred_knn) with the actual labels (y_test). The accuracy score is then printed.

In summary, this code showcases the process of training a KNN classifier, tuning its hyperparameters, and using it to predict the class labels of the test set based on the provided features.

CLASSIFICATION REPORT OF K-NEAREST NEIGHBORS ALGORITHM:

KNN Classification Report:

	precision	recall	f1-score	support
bad	0.92	0.97	0.95	37
good	0.99	0.97	0.98	102
accuracy			0.97	139
macro avg	0.96	0.97	0.96	139
weighted avg	0.97	0.97	0.97	139

KNN Accuracy: 0.971

FIG 35 K-NEAREST NEIGHBORS ALGORITHM CLASSIFICATION REPORT

CONFUSION MATRIX OF K-NEAREST NEIGHBORS ALGORITHM:

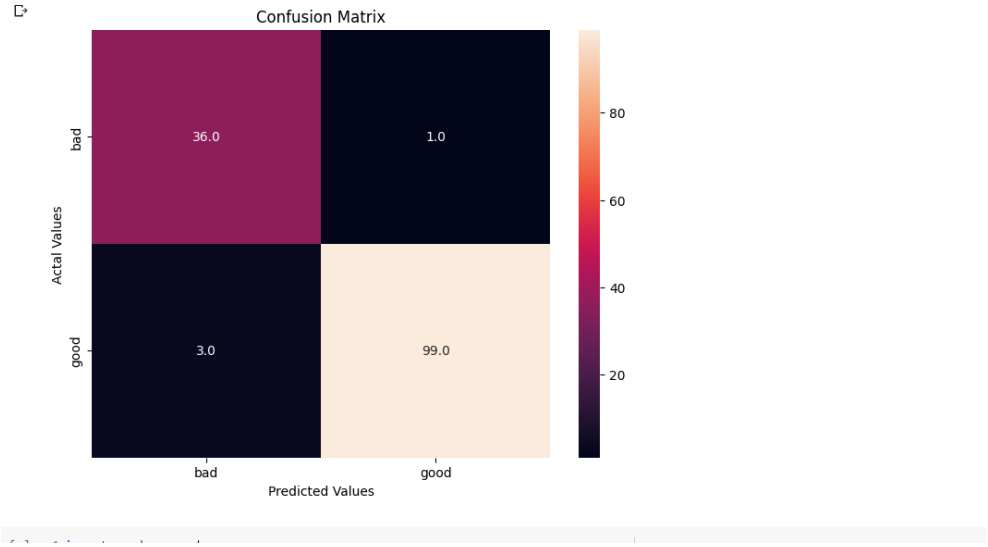


FIG 36 CONFUSION MATRIX OF K-NEAREST NEIGHBORS ALGORITHM

FEATURE IMPORTANCE OF K-NEAREST NEIGHBORS ALGORITHM:

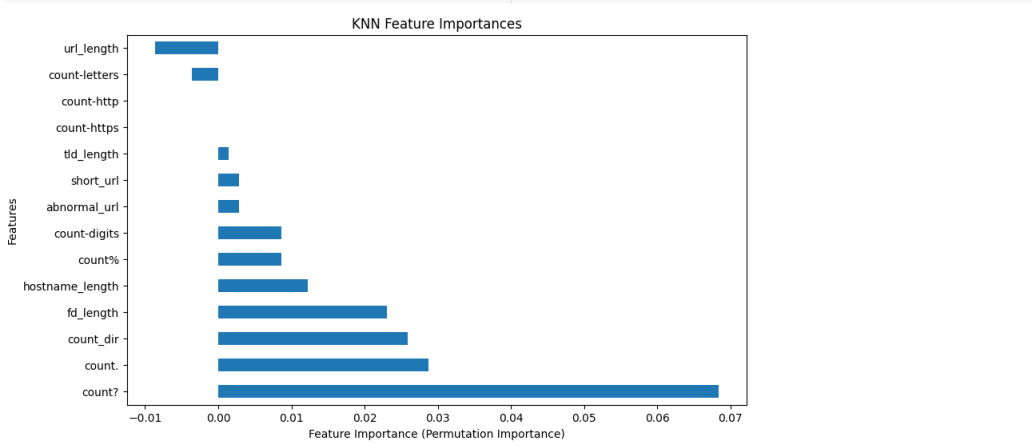


FIG 37 FEATURE IMPORTANCE OF K-NEAREST NEIGHBORS ALGORITHM

ROC CURVE OF K-NEAREST NEIGHBORS ALGORITHM:

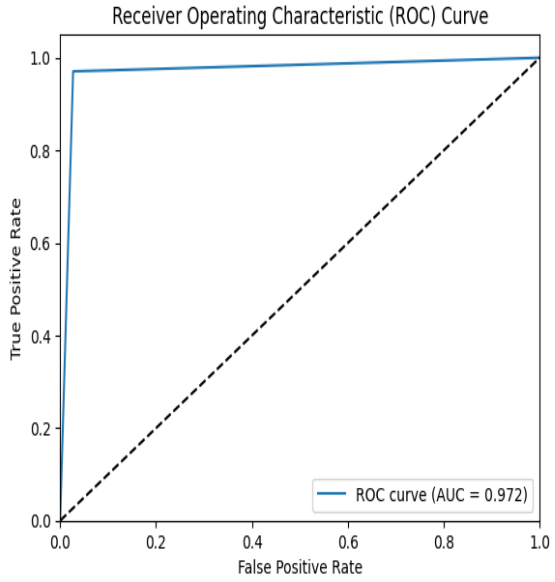


FIG 38 ROC CURVE OF K-NEAREST NEIGHBORS ALGORITHM

Chapter 6

Comparative analysis :

First here with out hyperparameter, QuantileTransformer, logistic regression, Support Vector Machine, K-Nearest Neighbors algorithm, implemented ,after that all these things added and give result.

Result of before tunning added and other things added:

1.Support Vector Machine:

```
testing SVM model :
Accuracy: 0.9424460431654677
Confusion matrix: [[ 24   7]
 [  1 107]]
Classification report:
              precision    recall  f1-score   support

         bad       0.96      0.77      0.86        31
        good       0.94      0.99      0.96       108

    accuracy                           0.94       139
   macro avg       0.95      0.88      0.91       139
weighted avg       0.94      0.94      0.94       139

knn
```

FIG 39 SUPPORT VECTOR MACHINE OLD

2.K-Nearest Neighbors Algorithm:

```
knn
Using KNN Model:
Accuracy: 0.8992805755395683
Confusion matrix:
 [[ 25   6]
 [  8 100]]
Classification report:
              precision    recall  f1-score   support

         bad       0.76      0.81      0.78        31
        good       0.94      0.93      0.93       108

    accuracy                           0.90       139
   macro avg       0.85      0.87      0.86       139
weighted avg       0.90      0.90      0.90       139
```

FIG 40 K-NEAREST NEIGHBORS ALGORITHM OLD

3.Logistic Regression:

```
*****USING LOGISTIC REGRESSION*****
Accuracy: 0.9064748201438849
Confusion matrix: [[ 18  13]
 [  0 108]]
Classification report:
              precision    recall  f1-score   support

         bad       1.00      0.58      0.73        31
        good       0.89      1.00      0.94       108

    accuracy                           0.91       139
   macro avg       0.95      0.79      0.84       139
weighted avg       0.92      0.91      0.90       139
```

FIG 41 LOGISTIC REGRESSION OLD

After hyperparameter tunning,optimization,regularization and quantalize technique apply all model gets better.

II TABLE OF COMPARISON

ALGORITHM	Existing approach	AFTER HYPERPARAMETER TUNNING,QUANTILE TRANSFORMER,CHI-SQUARE TEST ACCURACY (proposed approach)
LOGISTIC REGRESSION	0.90	0.96
K-Nearest Neighbors Algorithm	0.90	0.971
Support Vector Machine	0.94	0.986
RANDOM FOREST	-	0.993

Chapter 7

Conclusion and Future scope:

Overall,the system implementation described in the document focuses on the development of a Uniform Resource Locator classification model using machine learning techniques. The process involves data preparation, feature engineering, data splitting, vectorization, model selection and training, prediction, and evaluation. The model utilizes various algorithms such as Random Forest, Logistic Regression, Support Vector Machine, Neural Network, and K-Nearest Neighbors to classify Uniform Resource Locators as either "Bad" or "Good." The features used for classification include characteristics such as the number of dots, number of www, presence of a shortening service, Uniform Resource Locator length, count of digits and letters, subdomain count, and more.

The system utilizes techniques like chi-square tests to select the most informative features for classification. The chi-square test helps identify the significant correlation between the Uniform Resource Locator classification

and the engineered features. By evaluating the statistical significance of the relationship, the test assists in feature selection and validation, enabling more accurate classification.

The proposed models' performance is evaluated using metrics such as accuracy, true positive rate, false positive rate, precision, recall, F1-score, and area under the ROC curve. Additionally, visualizations, including ROC curves and model accuracy plots, provide further insights into the model's effectiveness.

For each algorithm, the document outlines the parameters used and highlights the importance of hyperparameter tuning, optimization, regularization, and quantization to achieve the best accuracy. It also presents a classification report and confusion matrix to analyze the model's performance in detail.

As for the future scope, there are several areas for potential improvement and expansion:

1. Dataset Enhancement: The model's performance could be enhanced by incorporating larger and more diverse datasets. Including a broader range of Uniform Resource Locators and different types of malicious Uniform Resource Locators can help improve the model's ability to detect and classify aberrant Uniform Resource Locators accurately.

2. Advanced Feature Engineering: Exploring additional features or refining the existing features can contribute to better classification results. Feature selection techniques like recursive feature elimination or principal component analysis can be employed to identify the most relevant features for classification.

3. Ensemble Methods: Implementing ensemble methods such as stacking or boosting can potentially improve the model's accuracy and robustness. Combining the predictions of multiple models can help mitigate individual model biases and increase overall performance.

4. Real-Time Implementation: Adapting the model for real-time Uniform Resource Locator classification can be valuable in practical applications. Developing an efficient and scalable system that can handle a continuous stream of Uniform Resource Locators and provide instant classification results would be beneficial.

5. Deployment and Integration: Integrating the developed model into existing security systems or web browsers can enhance their security measures by providing real-time Uniform Resource Locator classification and warning users about potentially harmful Uniform Resource Locators.

6. Continuous Model Updating: As new threats and techniques emerge, regularly updating the model with new training data and retraining it can ensure its effectiveness in detecting the latest forms of malicious Uniform Resource Locators.

Overall, the described system implementation provides a foundation for Uniform Resource Locator classification and can be further enhanced and expanded to address evolving challenges in web security and user protection.

References:

[1] Rupa Chiramdasu,Gautam Srivastava,Sweta Bhattacharya,Praveen Kumar Reddy,Thippasandra Reddy Gadekallu ,"Malicious URL Detection using Logistic Regression", in 2021 IEEE Conference on Omni-Layer Intelligent Systems (COINS) .IEEE,2021,pp. 1-6.

[2]R. Alshammari, G. Alshammari, and K. Elleithy, "A Comparative Analysis of Machine Learning Algorithms for Detecting Malicious URLs," Journal of Information Security and Applications, vol. 52, p. 102504, 2020.

[3] K. Grosse, J. Saxe, and R. Yerneni, "Adversarial Examples Against Deep Neural Networks in Malware Classification," in 28th USENIX Security Symposium (USENIX Security 19), 2019, pp. 577-594.

[4]T. Xiang, X. Zhang, and J. Dong, "A Deep Learning Framework for Malicious URL Detection Based on Attention Mechanism," Complexity, vol. 2020, pp. 1-10, 2020.

[5]M. Ahmadi and A. Zolanvari, "Deep Transfer Learning for Malicious URL Detection," Soft Computing, vol. 25, no. 12, pp. 8099-8110, 2021.

[6]Y. Cao, X. Liu, and D. Zhang, "DeepTransfer: A Deep Neural Network for Cross-Domain Malicious URL Detection," Journal of Computers, vol. 15, no. 6, pp. 1018-1028, 2020.

[7]N. A. G. Arachchilage and S. Love, "Towards Explainable Artificial Intelligence (XAI): A Survey on Feature Selection and Feature Extraction Methods for Malicious URL Detection," Computers & Security, vol. 87, p. 101607, 2019.

[8]L. Zhu, C. Xing, and W. Wang, "A Novel Malicious URL Detection System Based on Improved Convolutional Neural Network," Journal of Intelligent & Fuzzy Systems, vol. 37, no. 2, pp. 2053-2063, 2019.

[9]Y. Chang, Y. Lin, and M. Lin, "A Hybrid Model for Malicious URL Detection Based on GBDT and MLP," Future Generation Computer Systems, vol. 110, pp. 764-772, 2020.

[10]D. R. Costa, G. A. França, and J. M. dos Santos, "A Comparative Study of Machine Learning Algorithms for Malicious URL Detection," in Proceedings of the 8th Brazilian Conference on Intelligent Systems (BRACIS), 2019, pp. 337-342.

[11]X. Pan, W. Song, and J. Tian, "Malicious URL Detection Based on Hierarchical Attention Network and Convolutional Neural Network," Computers, Materials & Continua, vol. 65, no. 2, pp. 1497-1513, 2020.

[12]S. Kumar and R. Rathee, "URL classification using machine learning techniques: A survey," Artificial Intelligence Review, vol. 47, no. 1, pp. 1-38, 2017.

[13]H. Keshav, M. Patel, and N. Jain, "Machine learning techniques for detecting malicious URLs: A survey," Future Generation Computer Systems, vol. 82, pp. 481-497, 2018.

[14]S. Maity and P. K. Jana, "Analysis of machine learning techniques for URL classification," in International Conference on Computational Intelligence in Data Science, 2018, pp. 325-337.

[15]G. H. Poh and B. K. Tan, "A comparative study on malicious URL detection using machine learning," in 2019 3rd International Conference on Intelligent Sustainable Systems (ICISS), 2019, pp. 524-529.

[16] Natarajan, M., Karthikeyan, S., & Priyadarshini, S. (2021). Malicious URL Detection using Deep Learning Techniques. In International Conference on Information Science, Computing, and Communication (pp. 383-391). Springer, Singapore.

[17] Chaudhary, S., & Shukla, A. (2020). Malicious URL Detection using Machine Learning Techniques. International Journal of Computer Sciences and Engineering, 8(6), 192-197.

[18] Hu, W., & Hu, M. (2020). Malicious URL Detection using Deep Learning and Feature Engineering. Journal of Supercomputing, 76(6), 4396-4415.

[19]Sivakumar, R., Dhamodharan, M., & Yogesh, S. (2020). Analysis of Machine Learning Algorithms for Malicious URL Detection. International Journal of Innovative Technology and Exploring Engineering, 9(1), 1906-1910.

[20]Jain, S., & Chand, R. (2020). Comparative Analysis of Machine Learning Algorithms for Malicious URL Detection. International Journal of Computer Applications, 174(2), 11-16.

[21] Wagle, M., Bajracharya, P., & Gurung, P. (2020). Comparative Analysis of Machine Learning Algorithms for Malicious URL Detection. In 2020 International Conference on Sustainable Technologies for Industry 4.0 (pp. 1-5). IEEE.

[22] Nair, A. S., & Murugan, S. (2020). Comparative Analysis of Machine Learning Techniques for Malicious URL Detection. In 2020 International

Conference on Communication and Electronics Systems (ICCES) (pp. 85-89). IEEE.

[23] Padmakumari, P., & Sabu, N. (2020). Malicious URL Detection using Machine Learning Algorithms. In 2020 IEEE International Conference on System, Computation, Automation and Networking (ICSCAN) (pp. 321-325). IEEE.

[24]Zhang, C., & Xue, R. (2019). Comparative Analysis of Machine Learning Algorithms for Malicious URL Detection. In 2019 12th International Conference on Human System Interaction (HSI) (pp. 654-657). IEEE.

[25]Sengar, H. S., Sharma, A., & Sharma, D. K. (2019). Comparative Study of Machine Learning Algorithms for Malicious URL Detection. In 2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI) (pp. 657-660). IEEE.

[26] Pathak, A., & Jain, P. (2019). Comparative Study of Machine Learning Algorithms for Malicious URL Detection. In 2019 5th International

Conference on Advanced Computing & Communication Systems (ICACCS) (pp. 1220-1224). IEEE.

[27]Yadav, A., Yadav, N., & Sharma, S. K. (2019). Comparative Analysis of Machine Learning Algorithms for Malicious URL Detection. In 2019 6th International Conference on Advanced Computing and Communication Systems (ICACCS) (pp. 1073-1077). IEEE.

[28] Gupta, A., Kumar, S., & Jain, N. (2019). A Comparative Analysis of Machine Learning Techniques for Malicious URL Detection. In 2019 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT) (pp. 1-6). IEEE.

[29] Kumar, R., Kalia, A., & Singh, R. (2019). Comparative Study of Machine Learning Algorithms for Malicious URL Detection. In 2019 International Conference on Communication and Electronics Systems (ICCES) (pp. 247-251). IEEE.

[30]Solanki, K., Jethva, N., & Patel, R. (2019). Comparative Analysis of Machine Learning Algorithms for Malicious URL Detection. In 2019

International Conference on Innovative Trends in Computer Engineering (ICITCE) (pp. 1-5). IEEE.

[31] Kaur, J., & Kaur, A. (2019). Comparative Analysis of Machine Learning Algorithms for Malicious URL Detection. In 2019 International Conference on Automation, Computational and Technology Management (ICACTM) (pp. 272-277). IEEE.

[32] Khan, N., Shah, H., & Naik, K. (2019). Comparative Analysis of Machine Learning Techniques for Malicious URL Detection. In 2019 3rd International Conference on Advances in Electronics, Computers and Communications (ICAECC) (pp. 1-6). IEEE.

[33] Priyadharshini, R., & Bhuvaneswari, V. (2019). Comparative Analysis of Machine Learning Algorithms for Malicious URL Detection. In 2019 International Conference on Smart Technologies for Smart Nation (SmartTechCon) (pp. 244-249). IEEE.

[34] Sharma, N., & Jain, N. (2019). Comparative Analysis of Machine Learning Techniques for Malicious URL Detection. In 2019 International

Conference on Smart Systems and Inventive Technology (ICSSIT) (pp. 546-550). IEEE.

[35]Verma, M., Jaiswal, N., & Tanwani, A. (2019). Comparative Analysis of Machine Learning Techniques for Malicious URL Detection. In 2019 International Conference on Communication, Computing and Internet of Things (IC3IoT) (pp. 1-4). IEEE.

[36] Sharma, S., & Bharti, S. (2019). Comparative Analysis of Machine Learning Techniques for Malicious URL Detection. In 2019 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT) (pp. 1-6). IEEE.

[37]Gupta, R., Srivastava, P., & Asthana, S. (2019). Comparative Analysis of Machine Learning Algorithms for Malicious URL Detection. In 2019 International Conference on Automation, Computational and Technology Management (ICACTM) (pp. 261-266). IEEE.

[38] Dwivedi, A., & Jain, S. (2019). Comparative Analysis of Machine Learning Techniques for Malicious URL Detection. In 2019 2nd

International Conference on Inventive Research in Computing Applications (pp. 1179-1183). IEEE.

[39] Yadav, R., & Rathore, S. (2019). Comparative Analysis of Machine Learning Techniques for Malicious URL Detection. In 2019 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS) (pp. 186-190). IEEE.

[40]Kumar, R., Kalia, A., & Singh, R. (2019). Comparative Study of Machine Learning Algorithms for Malicious URL Detection. In 2019 International Conference on Communication and Electronics Systems (ICCES) (pp. 247-251). IEEE.

[41] Waghmare, P., & Agarwal, P. (2019). Comparative Analysis of Machine Learning Algorithms for Malicious URL Detection. In 2019 6th International Conference on Computing for Sustainable Global Development (INDIACom) (pp. 1283-1287). IEEE.

[42]Saxena, N., & Yadav, N. (2019). Comparative Analysis of Machine Learning Techniques for Malicious URL Detection. In 2019 International Conference on Intelligent Sustainable Systems (ICISS) (pp. 549-554). IEEE.

[43]Kandari, M. S., & Sharma, D. (2019). Comparative Analysis of Machine Learning Algorithms for Malicious URL Detection. In 2019 5th International Conference on Internet of Things: Smart Innovation and Usages (IoT-SIU) (pp. 1-6). IEEE.

[44] Jothi, P. R., & Aramudhan, M. (2018). Comparative Study of Machine Learning Techniques for Malicious URL Detection. In 2018 International Conference on Electrical, Electronics, Communication, Computer Technologies and Optimization Techniques (ICEECCOT) (pp. 1-6). IEEE.

[45]Nagpal, A., & Kaur, M. (2018). Comparative Analysis of Machine Learning Techniques for Malicious URL Detection. In 2018 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT) (pp. 1-6). IEEE.

[46]Pawar, R., & Kolhe, S. (2018). Comparative Analysis of Machine Learning Techniques for Malicious URL Detection. In 2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS) (pp. 553-557). IEEE.

[47]Chakraborty, P., & Chakrabarti, A. (2018). Comparative Analysis of Machine Learning Techniques for Malicious URL Detection. In 2018 International Conference on Computing, Power and Communication Technologies (GUCON) (pp. 165-169). IEEE.

[48]Jaiswal, A., & Patidar, V. (2017). Comparative Analysis of Machine Learning Algorithms for Malicious URL Detection. In 2017 IEEE International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT) (pp. 1-6). IEEE.

[49]Sharma, N., & Jain, N. (2017). Comparative Study of Machine Learning Algorithms for Malicious URL Detection. In 2017 2nd International Conference for Convergence in Technology (I2CT) (pp. 85-90). IEEE.

[50]Sahdev, G., & Chhabra, A. (2017). Comparative Analysis of Machine Learning Algorithms for Malicious URL Detection. In 2017 International Conference on Intelligent Sustainable Systems (ICISS) (pp. 620-625). IEEE.

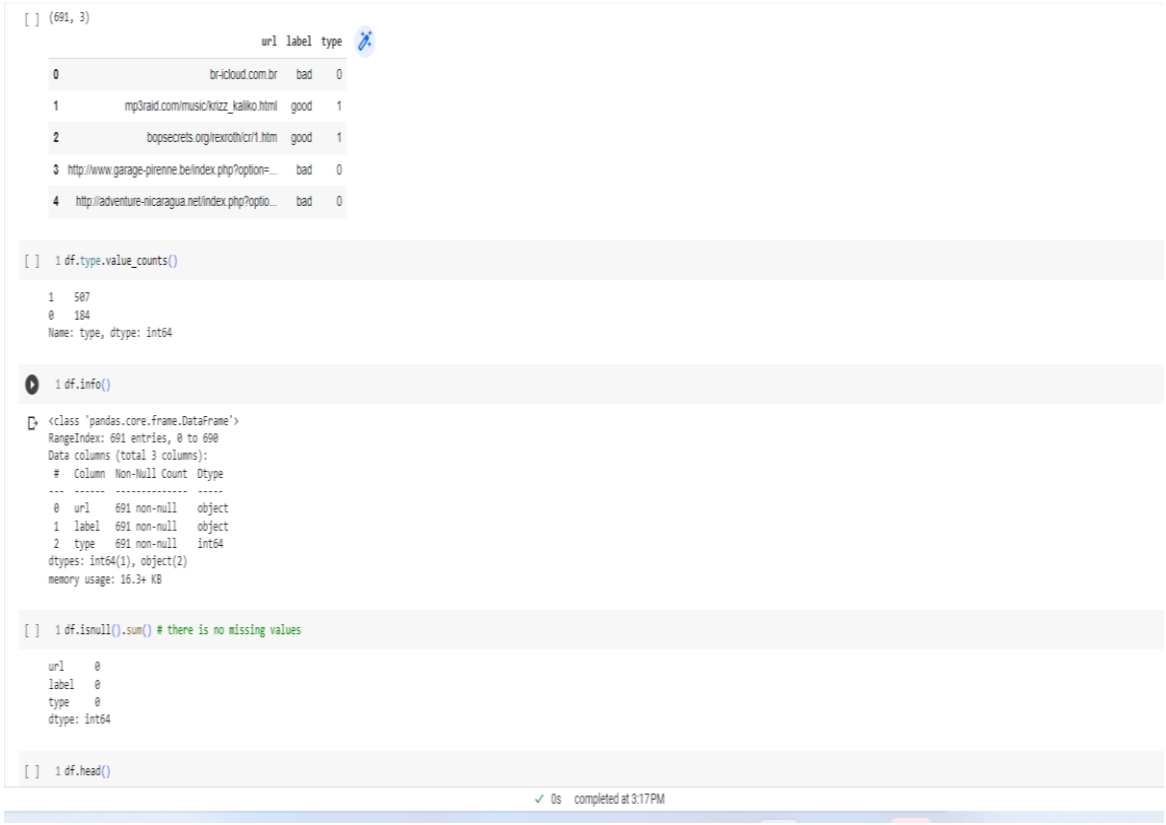
APPENDIX

```
import pandas as pd
import itertools
from sklearn.metrics import classification_report,confusion_matrix, accuracy_score
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import xgboost as xgb
from lightgbm import LGBMClassifier
import os
import seaborn as sns
from wordcloud import WordCloud
```

```
import pandas as pd
import itertools
import numpy as np
import matplotlib.pyplot as plt
import os
import seaborn as sns
from wordcloud import WordCloud
import re
from urllib.parse import urlparse
from googlesearch import search
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer, HashingVectorizer
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, roc_curve, auc
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.utils import resample
import tensorflow as tf
from tensorflow import keras
from  tensorflow.keras.losses import sparse_categorical_crossentropy
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
import requests
from io import StringIO
```

```
import pandas as pd # use for data manipulation and analysis
import numpy as np # use for multi-dimensional array and matrix

import seaborn as sns # use for high-level interface for drawing attractive and informative statistical graphics
import matplotlib.pyplot as plt # It provides an object-oriented API for embedding plots into applications
%matplotlib inline
# It sets the backend of matplotlib to the 'inline' backend:
import time # calculate time

from sklearn.linear_model import LogisticRegression # algo use to predict good or bad
from sklearn.naive_bayes import MultinomialNB # nlp algo use to predict good or bad

from sklearn.model_selection import train_test_split # spliting the data between feature and target
from sklearn.metrics import classification_report # gives whole report about metrics (e.g, recall,precision,f1_score,c_m)
from sklearn.metrics import confusion_matrix # gives info about actual and predict
from nltk.tokenize import RegexpTokenizer # regexp tokenizers use to split words from text
from nltk.stem.snowball import SnowballStemmer # stemmes words
from sklearn.feature_extraction.text import CountVectorizer # create sparse matrix of words using regexptokenizes
from sklearn.pipeline import make_pipeline # use for combining all prerocessors techniuqes and algos

from PIL import Image # getting images in notebook
# from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator# creates words colud

from bs4 import BeautifulSoup # use for scraping the data from website
from selenium import webdriver # use for automation chrome
import networkx as nx # for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks.

import pickle# use to dump model

import warnings # ignores pink warnings
warnings.filterwarnings('ignore')
```

```
[ ] import warnings # ignores pink warnings
    warnings.filterwarnings('ignore')

[ ] pip install selenium

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: selenium in /usr/local/lib/python3.10/dist-packages (4.10.0)
Requirement already satisfied: urllib3[socks]<3,>=1.26 in /usr/local/lib/python3.10/dist-packages (from selenium) (1.26.15)
Requirement already satisfied: trio~=0.17 in /usr/local/lib/python3.10/dist-packages (from selenium) (0.22.0)
Requirement already satisfied: trio-websocket~=0.9 in /usr/local/lib/python3.10/dist-packages (from selenium) (0.10.3)
Requirement already satisfied: certifi>=2021.10.8 in /usr/local/lib/python3.10/dist-packages (from selenium) (2022.12.7)
Requirement already satisfied: attrs>=19.2.0 in /usr/local/lib/python3.10/dist-packages (from trio~=0.17->selenium) (23.1.0)
Requirement already satisfied: sortedcontainers in /usr/local/lib/python3.10/dist-packages (from trio~=0.17->selenium) (2.4.0)
Requirement already satisfied: async-generator>=1.9 in /usr/local/lib/python3.10/dist-packages (from trio~=0.17->selenium) (1.10)
Requirement already satisfied: idna in /usr/local/lib/python3.10/dist-packages (from trio~=0.17->selenium) (3.4)
Requirement already satisfied: outcome in /usr/local/lib/python3.10/dist-packages (from trio~=0.17->selenium) (1.2.0)
Requirement already satisfied: sniffio in /usr/local/lib/python3.10/dist-packages (from trio~=0.17->selenium) (1.3.0)
Requirement already satisfied: exceptiongroup>=1.0.0rc9 in /usr/local/lib/python3.10/dist-packages (from trio~=0.17->selenium) (1.1.1)
Requirement already satisfied: wsproto>=0.14 in /usr/local/lib/python3.10/dist-packages (from trio-websocket~=0.9->selenium) (1.2.0)
Requirement already satisfied: PySocks!=1.5.7,<2.0,>=1.5.6 in /usr/local/lib/python3.10/dist-packages (from urllib3[socks]<3,>=1.26->selenium) (1.7.1)
Requirement already satisfied: h11<1,>=0.9.0 in /usr/local/lib/python3.10/dist-packages (from wsproto>=0.14->trio-websocket~=0.9->selenium) (0.14.0)

[ ] pip install tld

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: tld in /usr/local/lib/python3.10/dist-packages (0.13)

+ Code  + Text

[ ] drive='https://drive.google.com/file/d/1bk7rfLEpAkKKqwLBOyybefwCJdLmD1Of/view?usp=sharing'#new

[ ] file_id=drive.split('/')[-2]
    my_url='https://drive.google.com/uc?export=download&id='+file_id
    url=requests.get(my_url).text
    csv_raw=StringIO(url)
    df=pd.read_csv(csv_raw)
    print(df.shape)
    df.head()

(691, 3)
```



```
[ ] (691, 3)
```

	url	label	type
0	br-icloud.com.br	bad	0
1	mp3raid.com/music/krizz_kaliko.html	good	1
2	bopsecrets.org/rexroth/cr/1.htm	good	1
3	http://www.garage-pirenne.be/index.php?option=...	bad	0
4	http://adventure-nicaragua.net/index.php?optio...	bad	0

```
[ ] 1 df.type.value_counts()

1    507
0    184
Name: type, dtype: int64

1 df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 691 entries, 0 to 690
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   url     691 non-null    object
 1   label   691 non-null    object
 2   type    691 non-null    int64
dtypes: int64(1), object(2)
memory usage: 16.3+ KB

[ ] 1 df.isnull().sum() # there is no missing values

url     0
label   0
type    0
dtype: int64

[ ] 1 df.head()
```

✓ 0s completed at 3:17PM

```
[ ] 1 df.head()
```

	url	label	type
0	br-icloud.com.br	bad	0
1	mp3raid.com/music/krizz_kaliko.html	good	1
2	bopsecrets.org/rexroth/cr/1.htm	good	1
3	http://www.garage-pirenne.be/index.php?option=...	bad	0
4	http://adventure-nicaragua.net/index.php?optio...	bad	0

```
1 df.tail()
```

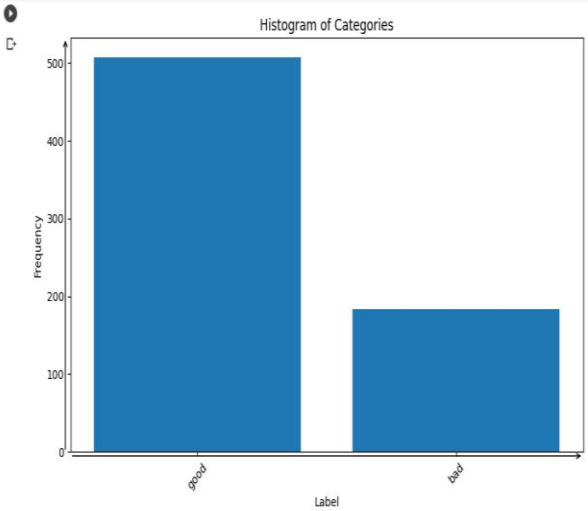
	url	label	type
686	onlineradio2.com/listen/Mix_933	good	1
687	mapsofworld.com/usa/states/california/californ...	good	1
688	http://www.prefina.it/notizie/notizie-generali...	bad	0
689	video.ca.msn.com/watch/video/dog-kills-quebec-...	good	1
690	http://torrentdn.com/bbs/s.php?bo_table=torren...	good	1

```
[ ] 1 import pandas as pd
    2 import matplotlib.pyplot as plt
    3
    4 # Visualization
    5 label_counts = pd.DataFrame(df.label.value_counts())
    6 type_counts = df['label'].value_counts()
    7 plt.figure(figsize=(10, 6))
    8 plt.bar(label_counts.index, type_counts.values)
    9 plt.xlabel('Label')
   10 plt.ylabel('Frequency')
   11 plt.title('Histogram of Categories')
   12 plt.xticks(rotation=45)
   13
```

✓ 0s completed at 3:17PM

```python
import pandas as pd
import matplotlib.pyplot as plt

# Visualization
label_counts = pd.DataFrame(df.label.value_counts())
type_counts = df['label'].value_counts()
plt.figure(figsize=(10, 6))
plt.bar(label_counts.index, type_counts.values)
plt.xlabel('Label')
plt.ylabel('Frequency')
plt.title('Histogram of Categories')
plt.xticks(rotation=45)

# Add arrow to the x-axis
plt.annotate(
    '',
    xy=(0, -0.01),
    xytext=(1, -0.01),
    xycoords='axes fraction',
    arrowprops=dict(arrowstyle='<-')
)

# Add arrow to the y-axis
plt.annotate(
    '',
    xy=(-0.01, 0),
    xytext=(-0.01, 1),
    xycoords='axes fraction',
    arrowprops=dict(arrowstyle='<-')
)

plt.show()

```

+ Code + Text




```python
# Grouping the data by type
grouped_df = df.groupby('label')['url'].apply(lambda x: ' '.join(x))
```

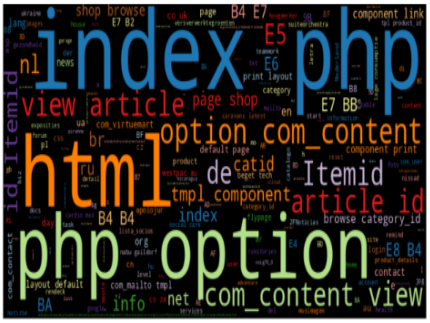
```python
# Word cloud generation
for label_, urls in grouped_df.items():
    wordcloud = WordCloud(width=800, height=400, colormap='Paired').generate(urls)
    plt.figure(figsize=(8, 6))
    plt.imshow(wordcloud, interpolation='bilinear')
```

✓ 0s completed at 3:17PM

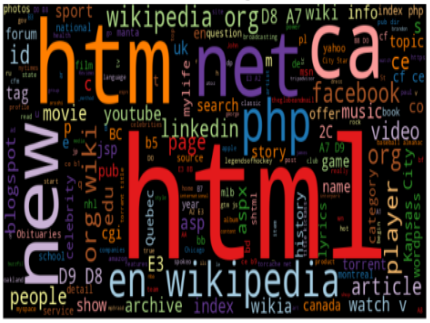
```
[ ] 1 # Grouping the data by type
    2 grouped_df = df.groupby('label')['url'].apply(lambda x: ' '.join(x))
```

```
1 # Word cloud generation
2 for label_, urls in grouped_df.items():
3     wordcloud = WordCloud(width=800, height=400, colormap='Paired').generate(urls)
4     plt.figure(figsize=(8, 6))
5     plt.imshow(wordcloud, interpolation='bilinear')
6     plt.title('Word Cloud for {}'.format(label_))
7     plt.axis('off')
8     plt.show()
```

Word Cloud for bad



Word Cloud for good



```python
# Feature engineering

def abnormal_url(url):
    hostname = urlparse(url).hostname
    hostname = str(hostname)
    match = re.search(hostname, url)
    if match:
        return 1
    else:
        return 0
df['abnormal_url'] = df['url'].apply(lambda i: abnormal_url(i))



def count_dot(url):
    count_dot = url.count('.')
    return count_dot
df['count.'] = df['url'].apply(lambda i: count_dot(i))

#def count_www(url):
#   count_www = url.count('www')
#  return count_www
#df['count_www'] = df['url'].apply(lambda i: count_www(i))


def no_of_dir(url):
    urldir = urlparse(url).path
    return urldir.count('/')

df['count_dir'] = df['url'].apply(lambda i: no_of_dir(i))


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|'
```

✓ 0s completed at 3:17 PM

Code + Text

```
                    'dolop\.com|shorl\.le|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


df['short_url'] = df['url'].apply(lambda i: shortening_service(i))
def count_https(url):
    return url.count('https')

df['count-https'] = df['url'].apply(lambda i : count_https(i))

def count_http(url):
    return url.count('http')

df['count-http'] = df['url'].apply(lambda i : count_http(i))

def count_per(url):
    return url.count('%')

df['count%'] = df['url'].apply(lambda i : count_per(i))

def count_ques(url):
    return url.count('?')

df['count?'] = df['url'].apply(lambda i: count_ques(i))


def url_length(url):
    return len(str(url))


#Length of URL
df['url_length'] = df['url'].apply(lambda i: url_length(i))
#Hostname Length
```

✓ 0s completed at 3:17 PM

```
#Hostname Length

def hostname_length(url):
    return len(urlparse(url).netloc)

df['hostname_length'] = df['url'].apply(lambda i: hostname_length(i))



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


df['count-digits']= df['url'].apply(lambda i: digit_count(i))


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


df['count-letters']= df['url'].apply(lambda i: letter_count(i))


from urllib.parse import urlparse

def subdomain_count(url):
    parsed_url = urlparse(url)
    if parsed_url.hostname is not None:
        subdomains = parsed_url.hostname.split('.')
        if len(subdomains) > 2:
            return 1
    return 0
```

```
117
118 df['subdomain_count'] = df['url'].apply(lambda i: subdomain_count(i))
119
120
121 #Importing dependencies
122 from urllib.parse import urlparse
123 from tld import get_tld
124 import os.path
125
126 #First Directory Length
127 def fd_length(url):
128     urlpath= urlparse(url).path
129     try:
130         return len(urlpath.split('/')[1])
131     except:
132         return 0
133
134 df['fd_length'] = df['url'].apply(lambda i: fd_length(i))
135
136 #Length of Top Level Domain
137 df['tld'] = df['url'].apply(lambda i: get_tld(i,fail_silently=True))
138
139
140 def tld_length(tld):
141     try:
142         return len(tld)
143     except:
144         return -1
145
146 df['tld_length'] = df['tld'].apply(lambda i: tld_length(i))
```

```
[ ]  1
     2 df.columns
```

```
Index(['url', 'label', 'type', 'abnormal_url', 'count.', 'count_dir',
       'short_url', 'count-https', 'count-http', 'count%', 'count?',
       'url_length', 'hostname_length', 'count-digits', 'count-letters',
       'subdomain_count', 'fd_length', 'tld', 'tld_length'],
      dtype='object')
```

```
[ ]  1
     2 df.columns
```

```
Index(['url', 'label', 'type', 'abnormal_url', 'count.', 'count_dir',
       'short_url', 'count-https', 'count-http', 'count%', 'count?',
       'url_length', 'hostname_length', 'count-digits', 'count-letters',
       'subdomain_count', 'fd_length', 'tld', 'tld_length'],
      dtype='object')
```

```
[ ]  1 df['type'].value_counts()
```

```
1    507
0    184
Name: type, dtype: int64
```

```
     1 from sklearn.preprocessing import LabelEncoder
     2
     3 lb_make = LabelEncoder()
     4 df["type_code"] = lb_make.fit_transform(df["type"])
     5 df["type_code"].value_counts()
```

```
1    507
0    184
Name: type_code, dtype: int64
```

```
[ ]  1 #Predictor Variables
     2 # filtering out google_index as it has only 1 value
     3 X = df[['abnormal_url', 'count.',
     4         'count_dir',  'short_url', 'count-https',
     5        'count-http', 'count%', 'count?',  'url_length',
     6        'hostname_length', 'fd_length', 'tld_length', 'count-digits',
     7        'count-letters']]
     8
     9 #Target Variable
    10 y = df['type_code']
```

```
[ ]  1 X.head()
```

+ Code + Text

1	0	2	2	0	0	0	0	0	35	0	5	-1	1	29
2	0	2	3	0	0	0	0	0	31	0	7	-1	1	25
3	1	3	1	0	0	1	0	1	88	21	9	2	7	63
4	1	2	1	0	0	1	0	1	235	23	9	3	22	199

```
X.columns
```

```
Index(['abnormal_url', 'count.', 'count_dir', 'short_url', 'count-https',
       'count-http', 'count%', 'count?', 'url_length', 'hostname_length',
       'fd_length', 'tld_length', 'count-digits', 'count-letters'],
      dtype='object')
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.2,shuffle=True, random_state=5)
```

```
import sklearn.metrics
from sklearn import metrics
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import QuantileTransformer
from sklearn.pipeline import make_pipeline

# Define the parameter grid for hyperparameter tuning
param_grid = {
    'randomforestclassifier__n_estimators': [50, 100, 200],
    'randomforestclassifier__max_features': ['sqrt', 'log2']
}

# Create a pipeline with QuantileTransformer and RandomForestClassifier
pipeline = make_pipeline(
    QuantileTransformer(),
    RandomForestClassifier()
)
```

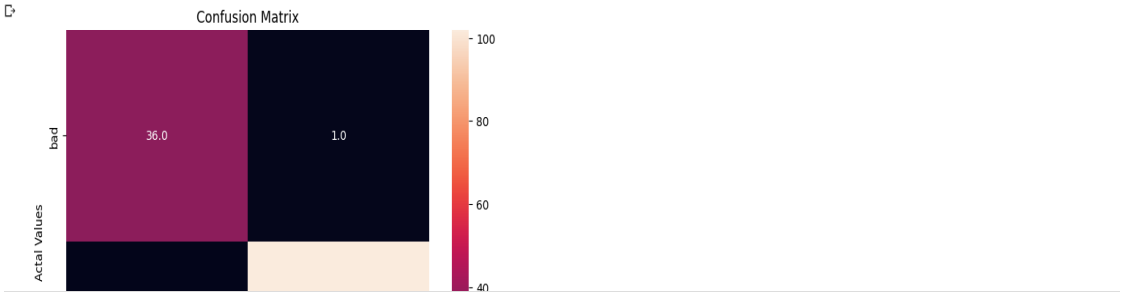
```
# Perform hyperparameter tuning using GridSearchCV
grid_search = GridSearchCV(pipeline, param_grid=param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Get the best estimator from the grid search
best_model = grid_search.best_estimator_

# Make predictions on the test set using the best model
y_pred_rf = best_model.predict(X_test)

# Print classification report
print(classification_report(y_test, y_pred_rf, target_names=['bad', 'good']))

# Calculate accuracy score
score = metrics.accuracy_score(y_test, y_pred_rf)
print("Accuracy:   %0.3f" % score)
```

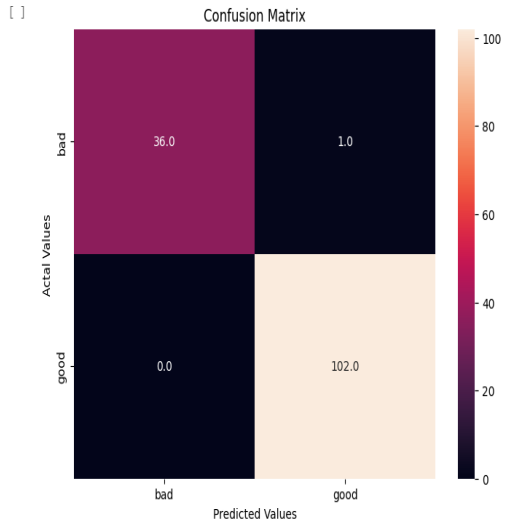
```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_data.py:2627: UserWarning: n_quantiles (1000) is greater than the total number of samples (441). n_quantiles is set to n_samples.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_data.py:2627: UserWarning: n_quantiles (1000) is greater than the total number of samples (441). n_quantiles is set to n_samples.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_data.py:2627: UserWarning: n_quantiles (1000) is greater than the total number of samples (442). n_quantiles is set to n_samples.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_data.py:2627: UserWarning: n_quantiles (1000) is greater than the total number of samples (442). n_quantiles is set to n_samples.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_data.py:2627: UserWarning: n_quantiles (1000) is greater than the total number of samples (442). n_quantiles is set to n_samples.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_data.py:2627: UserWarning: n_quantiles (1000) is greater than the total number of samples (441). n_quantiles is set to n_samples.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_data.py:2627: UserWarning: n_quantiles (1000) is greater than the total number of samples (441). n_quantiles is set to n_samples.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_data.py:2627: UserWarning: n_quantiles (1000) is greater than the total number of samples (442). n_quantiles is set to n_samples.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_data.py:2627: UserWarning: n_quantiles (1000) is greater than the total number of samples (442). n_quantiles is set to n_samples.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_data.py:2627: UserWarning: n_quantiles (1000) is greater than the total number of samples (442). n_quantiles is set to n_samples.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_data.py:2627: UserWarning: n_quantiles (1000) is greater than the total number of samples (441). n_quantiles is set to n_samples.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_data.py:2627: UserWarning: n_quantiles (1000) is greater than the total number of samples (441). n_quantiles is set to n_samples.
```

✓ 0s completed at 3:17 PM

```
  warnings.warn(
              precision    recall  f1-score   support

         bad       1.00      0.97      0.99        37
        good       0.99      1.00      1.00       102

    accuracy                           0.99       139
   macro avg       1.00      0.99      0.99       139
weighted avg       0.99      0.99      0.99       139

Accuracy:  0.993
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_data.py:2627: UserWarning: n_quantiles (1000) is greater than the total number of samples (552). n_quantiles is set to n_samples.
  warnings.warn(
```

```
cm = confusion_matrix(y_test, y_pred_rf)
cm_df = pd.DataFrame(cm,
                     index = ['bad', 'good'],
                     columns = ['bad', 'good'])
plt.figure(figsize=(8,6))
sns.heatmap(cm_df, annot=True,fmt=".1f")
plt.title('Confusion Matrix')
plt.ylabel('Actal Values')
plt.xlabel('Predicted Values')
plt.show()
```

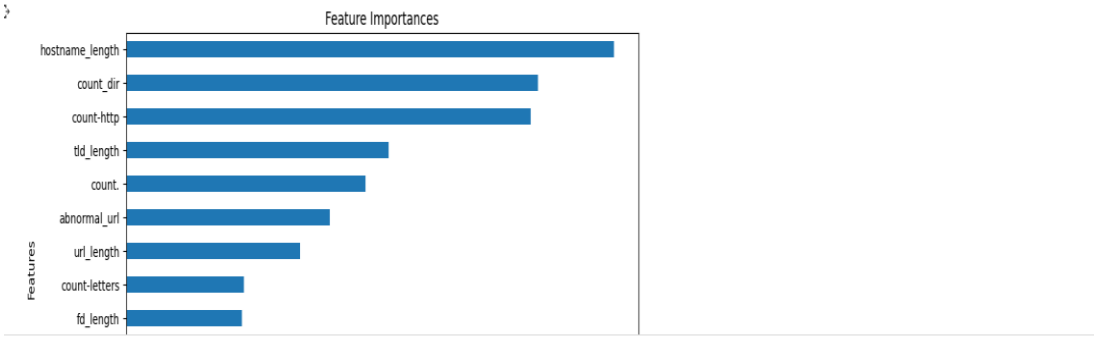





```
import pandas as pd
import matplotlib.pyplot as plt

# Compute feature importances using the best model
feat_importances = pd.Series(best_model.named_steps['randomforestclassifier'].feature_importances_, index=X_train.columns)

# Sort feature importances in ascending order and create a horizontal bar plot
feat_importances.sort_values().plot(kind="barh", figsize=(10, 6))

# Add labels and title to the plot
```

Predicted Values

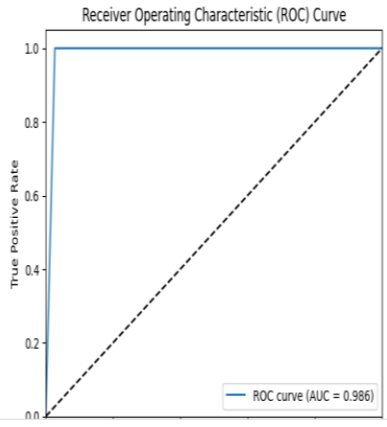
```
import pandas as pd
import matplotlib.pyplot as plt

# Compute feature importances using the best model
feat_importances = pd.Series(best_model.named_steps['randomforestclassifier'].feature_importances_, index=X_train.columns)

# Sort feature importances in ascending order and create a horizontal bar plot
feat_importances.sort_values().plot(kind="barh", figsize=(10, 6))

# Add labels and title to the plot
plt.xlabel("Feature Importance")
plt.ylabel("Features")
plt.title("Feature Importances")

# Display the plot
plt.show()
```



✓ 0s completed at 3:17PM

```
# Compute ROC curve
fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred_rf)

# Calculate AUC score
auc_score = metrics.auc(fpr, tpr)
print("AUC: %0.3f" % auc_score)

# Plot ROC curve
plt.plot(fpr, tpr, label='ROC curve (AUC = %0.3f)' % auc_score)
plt.plot([0, 1], [0, 1], 'k--')  # Random guessing line
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

```
AUC: 0.986
```



✓ 0s completed at 3:17PM

```
def main(url):

    status = []

    #status.append(having_ip_address(url))
    status.append(abnormal_url(url))
    status.append(count_dot(url))
    #status.append(count_www(url))
    #status.append(count_atrate(url))
    status.append(no_of_dir(url))
    #status.append(no_of_embed(url))

    status.append(shortening_service(url))
    status.append(count_https(url))
    status.append(count_http(url))

    status.append(count_per(url))
    status.append(count_ques(url))
    #status.append(count_hyphen(url))
    #status.append(count_equal(url))

    status.append(url_length(url))
    status.append(hostname_length(url))
    #status.append(suspicious_words(url))
    status.append(digit_count(url))
    status.append(letter_count(url))
    status.append(fd_length(url))
    tld = get_tld(url,fail_silently=True)

    status.append(tld_length(tld))




    return status
```

```
def get_prediction_from_url(test_url):
    features_test = main(test_url)
    # Due to updates to scikit-learn, we now need a 2D array as a parameter to the predict function.
    features_test = np.array(features_test).reshape((1, -1))



    pred =best_model.predict(features_test)
    if int(pred[0]) == 0:

        res="BAD"
        return res
    elif int(pred[0]) == 1.0:

        res="SAFE"
        return res

```

```
urls = ['titaniumcorporate.co.za','en.wikipedia.org/wiki/North_Dakota']
for url in urls:
    print(get_prediction_from_url(url))
```

```
BAD
SAFE
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but QuantileTransformer was fitted with feature names
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but QuantileTransformer was fitted with feature names
  warnings.warn(
```

Svm:

```
from sklearn import metrics

from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import QuantileTransformer
from sklearn.pipeline import make_pipeline

# SVM
svm_param_grid = {
    'svc__C': [0.1, 1, 10],
    'svc__kernel': ['linear', 'rbf']
}

svm_pipeline = make_pipeline(
    QuantileTransformer(),
    SVC()
)

svm_grid_search = GridSearchCV(svm_pipeline,
param_grid=svm_param_grid, cv=5)
svm_grid_search.fit(X_train, y_train)

svm_best_model = svm_grid_search.best_estimator_
y_pred_svm = svm_best_model.predict(X_test)
```

```
print("SVM Classification Report:")
print(classification_report(y_test, y_pred_svm,
target_names=['bad', 'good']))

svm_score = metrics.accuracy_score(y_test, y_pred_svm)
print("SVM Accuracy:   %0.3f" % svm_score)

cm = confusion_matrix(y_test, y_pred_svm)
cm_df = pd.DataFrame(cm,
                     index = ['bad', 'good'],
                     columns = ['bad', 'good'])
plt.figure(figsize=(8,6))
sns.heatmap(cm_df, annot=True,fmt=".1f")
plt.title('Confusion Matrix')
plt.ylabel('Actal Values')
plt.xlabel('Predicted Values')
plt.show()
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn import metrics
from sklearn.inspection import permutation_importance
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import QuantileTransformer
from sklearn.pipeline import make_pipeline

# Calculate permutation importances
svm_perm_importances = permutation_importance(
    svm_best_model, X_test, y_test, scoring='accuracy',
n_repeats=10, random_state=42
)
svm_feat_importances =
pd.Series(svm_perm_importances.importances_mean,
index=X_train.columns)

# Sort feature importances in descending order and create a
horizontal bar plot
svm_feat_importances.sort_values(ascending=False).plot(kind="b
arh", figsize=(10, 6))

# Add labels and title to the plot
plt.xlabel("Feature Importance (Permutation Importance)")
plt.ylabel("Features")
plt.title("SVM Feature Importances")
```

```
# Display the plot
plt.show()

# Compute ROC curve
fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred_svm)

# Calculate AUC score
auc_score = metrics.auc(fpr, tpr)
print("AUC:        %0.3f" % auc_score)

# Plot ROC curve
plt.plot(fpr, tpr, label='ROC curve (AUC = %0.3f)' %
auc_score)
plt.plot([0, 1], [0, 1], 'k--')  # Random guessing line
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

knn: `from sklearn.neighbors import KNeighborsClassifier`

```
from sklearn import metrics
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import QuantileTransformer
from sklearn.pipeline import make_pipeline


# KNN
knn_param_grid = {
    'kneighborsclassifier__n_neighbors': [3, 5, 7],
    'kneighborsclassifier__weights': ['uniform', 'distance']
}

knn_pipeline = make_pipeline(
    QuantileTransformer(),
```

```
    KNeighborsClassifier()
)

knn_grid_search = GridSearchCV(knn_pipeline,
param_grid=knn_param_grid, cv=5)
knn_grid_search.fit(X_train, y_train)

knn_best_model = knn_grid_search.best_estimator_
y_pred_knn = knn_best_model.predict(X_test)

print("KNN Classification Report:")
print(classification_report(y_test, y_pred_knn,
target_names=['bad', 'good']))

knn_score = metrics.accuracy_score(y_test, y_pred_knn)
print("KNN Accuracy:   %0.3f" % knn_score)
cm = confusion_matrix(y_test, y_pred_knn)
cm_df = pd.DataFrame(cm,
                     index = ['bad', 'good'],
                     columns = ['bad', 'good'])
plt.figure(figsize=(8,6))
sns.heatmap(cm_df, annot=True,fmt=".1f")
plt.title('Confusion Matrix')
plt.ylabel('Actal Values')
plt.xlabel('Predicted Values')
plt.show()
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn import metrics
from sklearn.inspection import permutation_importance
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import QuantileTransformer
from sklearn.pipeline import make_pipeline

# Calculate permutation importances
knn_perm_importances = permutation_importance(
    knn_best_model, X_test, y_test, scoring='accuracy',
n_repeats=10, random_state=42
)
knn_feat_importances =
pd.Series(knn_perm_importances.importances_mean,
index=X_train.columns)
```

```
# Sort feature importances in descending order and create a
horizontal bar plot
knn_feat_importances.sort_values(ascending=False).plot(kind="b
arh", figsize=(10, 6))

# Add labels and title to the plot
plt.xlabel("Feature Importance (Permutation Importance)")
plt.ylabel("Features")
plt.title("KNN Feature Importances")

# Display the plot
plt.show()

# Compute ROC curve
fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred_knn)

# Calculate AUC score
auc_score = metrics.auc(fpr, tpr)
print("AUC:        %0.3f" % auc_score)

# Plot ROC curve
plt.plot(fpr, tpr, label='ROC curve (AUC = %0.3f)' %
auc_score)
plt.plot([0, 1], [0, 1], 'k--')  # Random guessing line
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

logistic Regression:

```
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import QuantileTransformer
from sklearn.pipeline import make_pipeline
# Logistic Regression
logreg_param_grid = {
    'logisticregression__C': [0.1, 1, 10],
    'logisticregression__solver': ['liblinear', 'saga']
}
```

```
logreg_pipeline = make_pipeline(
    QuantileTransformer(),
    LogisticRegression()
)

logreg_grid_search = GridSearchCV(logreg_pipeline,
param_grid=logreg_param_grid, cv=5)
logreg_grid_search.fit(X_train, y_train)

logreg_best_model = logreg_grid_search.best_estimator_
y_pred_logreg = logreg_best_model.predict(X_test)

print("Logistic Regression Classification Report:")
print(classification_report(y_test, y_pred_logreg,
target_names=['bad', 'good']))

logreg_score = metrics.accuracy_score(y_test, y_pred_logreg)
print("Logistic Regression Accuracy:   %0.3f" % logreg_score)
cm = confusion_matrix(y_test, y_pred_logreg)
cm_df = pd.DataFrame(cm,
                     index = ['bad', 'good'],
                     columns = ['bad', 'good'])
plt.figure(figsize=(8,6))
sns.heatmap(cm_df, annot=True,fmt=".1f")
plt.title('Confusion Matrix')
plt.ylabel('Actal Values')
plt.xlabel('Predicted Values')
plt.show()
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn import metrics
from sklearn.inspection import permutation_importance
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import QuantileTransformer
from sklearn.pipeline import make_pipeline

# Calculate permutation importances
logreg_perm_importances = permutation_importance(
    logreg_best_model, X_test, y_test, scoring='accuracy',
n_repeats=10, random_state=42
)
```

```
logreg_feat_importances =
pd.Series(logreg_perm_importances.importances_mean,
index=X_train.columns)

# Sort feature importances in descending order and create a
horizontal bar plot
logreg_feat_importances.sort_values(ascending=False).plot(kind
="barh", figsize=(10, 6))

# Add labels and title to the plot
plt.xlabel("Feature Importance (Permutation Importance)")
plt.ylabel("Features")
plt.title("logreg Feature Importances")

# Display the plot
plt.show()

# Compute ROC curve
fpr, tpr, thresholds = metrics.roc_curve(y_test,
y_pred_logreg)

# Calculate AUC score
auc_score = metrics.auc(fpr, tpr)
print("AUC:        %0.3f" % auc_score)

# Plot ROC curve
plt.plot(fpr, tpr, label='ROC curve (AUC = %0.3f)' %
auc_score)
plt.plot([0, 1], [0, 1], 'k--')  # Random guessing line
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

chi-square test :

```
import pandas as pd
import itertools
import numpy as np
import matplotlib.pyplot as plt
import os
```

```
import seaborn as sns
from wordcloud import WordCloud
import re
from urllib.parse import urlparse
from googlesearch import search
from sklearn.feature_extraction.text import TfidfVectorizer,
CountVectorizer, HashingVectorizer
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import classification_report,
confusion_matrix, accuracy_score, roc_curve, auc
from sklearn.model_selection import train_test_split,
GridSearchCV
from sklearn.utils import resample
import tensorflow as tf
from tensorflow import keras
from  tensorflow.keras.losses import
sparse_categorical_crossentropy
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import
LinearDiscriminantAnalysis
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
import requests
from io import StringIO
#drive='https://drive.google.com/file/d/1vuzXzsOEM3HN3SSt3t6dF
P4vTYppkmqe/view?usp=sharing'
#drive='https://drive.google.com/file/d/1aKrXrIGQKrNqZQKhWTTbZ
o4U_b0z--CZ/view?usp=sharing'#bigdata
#drive='https://drive.google.com/file/d/1UpjWUmzj18MayGOyoV_L6
WonLDyFsyyn/view?usp=sharing'#40k
#drive='https://drive.google.com/file/d/1VeNKLgX5oUunlyv6pZJKc
P_hTtZNK8qQ/view?usp=sharing'#4k
drive='https://drive.google.com/file/d/1bk7rfLEpAkKKqwLBOyybef
wCJdLmD1Of/view?usp=sharing'#new
file_id=drive.split('/')[-2]
my_url='https://drive.google.com/uc?export=download&id='+file_
id
url=requests.get(my_url).text
csv_raw=StringIO(url)
df=pd.read_csv(csv_raw)
print(df.shape)
df.head()


# Feature engineering
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
        '((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}', url)  #
Ipv6
    if match:
        return 1
    else:
        return 0
df['use_of_ip'] = df['url'].apply(lambda i:
having_ip_address(i))

def abnormal_url(url):
    hostname = urlparse(url).hostname
    hostname = str(hostname)
    match = re.search(hostname, url)
    if match:
        return 1
    else:
        return 0
df['abnormal_url'] = df['url'].apply(lambda i:
abnormal_url(i))

def google_index(url):
    site = search(url, 5)
    return 1 if site else 0
df['google_index'] = df['url'].apply(lambda i:
google_index(i))

def count_dot(url):
    count_dot = url.count('.')
    return count_dot
df['count.'] = df['url'].apply(lambda i: count_dot(i))

def count_www(url):
    count_www = url.count('www')
    return count_www
df['count_www'] = df['url'].apply(lambda i: count_www(i))
def count_atrate(url):

    return url.count('@')
```

```
df['count@'] = df['url'].apply(lambda i: count_atrate(i))


def no_of_dir(url):
    urldir = urlparse(url).path
    return urldir.count('/')

df['count_dir'] = df['url'].apply(lambda i: no_of_dir(i))

def no_of_embed(url):
    urldir = urlparse(url).path
    return urldir.count('//')

df['count_embed_domian'] = df['url'].apply(lambda i:
no_of_embed(i))


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|ur
l4\.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|ruby
url\.com|om\.ly|to\.ly|bit\.do|t\.co|lnkd\.in|'
                      'db\.tt|qr\.ae|adf\.ly|goo\.gl|bitly\.co
m|cur\.lv|tinyurl\.com|ow\.ly|bit\.ly|ity\.im|'
                      'q\.gs|is\.gd|po\.st|bc\.vc|twitthis\.co
m|u\.to|j\.mp|buzurl\.com|cutt\.us|u\.bb|yourls\.org|'
                      'x\.co|prettylinkpro\.com|scrnch\.me|fil
oops\.info|vzturl\.com|qr\.net|1url\.com|tweez\.me|v\.gd|'
                      'tr\.im|link\.zip\.net',
                      url)
    if match:
        return 1
    else:
        return 0


df['short_url'] = df['url'].apply(lambda i:
shortening_service(i))
def count_https(url):
    return url.count('https')

df['count-https'] = df['url'].apply(lambda i : count_https(i))
```

```
def count_http(url):
    return url.count('http')

df['count-http'] = df['url'].apply(lambda i : count_http(i))

def count_per(url):
    return url.count('%')

df['count%'] = df['url'].apply(lambda i : count_per(i))

def count_ques(url):
    return url.count('?')

df['count?'] = df['url'].apply(lambda i: count_ques(i))

def count_hyphen(url):
    return url.count('-')

df['count-'] = df['url'].apply(lambda i: count_hyphen(i))

def count_equal(url):
    return url.count('=')

df['count='] = df['url'].apply(lambda i: count_equal(i))

def url_length(url):
    return len(str(url))


#Length of URL
df['url_length'] = df['url'].apply(lambda i: url_length(i))
#Hostname Length

def hostname_length(url):
    return len(urlparse(url).netloc)

df['hostname_length'] = df['url'].apply(lambda i:
hostname_length(i))

df.head()

def suspicious_words(url):
    match =
re.search('PayPal|login|signin|bank|account|update|free|lucky|
service|bonus|ebayisapi|webscr',
                      url)
```

```
    if match:
        return 1
    else:
        return 0
df['sus_url'] = df['url'].apply(lambda i: suspicious_words(i))


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


df['count-digits']= df['url'].apply(lambda i: digit_count(i))


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


df['count-letters']= df['url'].apply(lambda i:
letter_count(i))


def check_suspicious_keywords(url):
    suspicious_keywords = ['download', 'login', 'secure',
'bank']
    for keyword in suspicious_keywords:
        if keyword in url.lower():
            return 1
    return 0

df['suspicious_keywords'] = df['url'].apply(lambda x:
check_suspicious_keywords(x))

import time
import dns.resolver

def check_dns_resolution_time(url):
    resolver = dns.resolver.Resolver()
    resolver.timeout = 3
    resolver.lifetime = 3
```

```
    start_time = time.time()
    try:
        answer = dns.resolver.Resolver.resolve()
    except:
        return 1
    end_time = time.time()
    dns_resolution_time = end_time - start_time
    if dns_resolution_time > 2:
        return 1
    return 0

df['dns_resolution'] = df['url'].apply(lambda x:
check_dns_resolution_time(x))
from urllib.parse import urlparse

def subdomain_count(url):
    parsed_url = urlparse(url)
    if parsed_url.hostname is not None:
        subdomains = parsed_url.hostname.split('.')
        if len(subdomains) > 2:
            return 1
    return 0

df['subdomain_count'] = df['url'].apply(lambda i:
subdomain_count(i))

def double_dots(url):
    if '..' in url:
        return 1
    else:
        return 0

df['double_dots'] = df['url'].apply(lambda i: double_dots(i))

def non_alphanumeric_characters(url):
    characters = ['@', '!', '#']
    for character in characters:
        if character in url:
            return 1
    return 0

df['non_alphanumeric_characters'] = df['url'].apply(lambda i:
non_alphanumeric_characters(i))

#Importing dependencies
from urllib.parse import urlparse
from tld import get_tld
```

```
import os.path

#First Directory Length
def fd_length(url):
    urlpath= urlparse(url).path
    try:
        return len(urlpath.split('/')[1])
    except:
        return 0

df['fd_length'] = df['url'].apply(lambda i: fd_length(i))

#Length of Top Level Domain
df['tld'] = df['url'].apply(lambda i:
get_tld(i,fail_silently=True))


def tld_length(tld):
    try:
        return len(tld)
    except:
        return -1

df['tld_length'] = df['tld'].apply(lambda i: tld_length(i))



import pandas as pd
from scipy.stats import chi2_contingency

# Define the target variable
target = df['type']

# Define the feature variables
features = df.drop(['type', 'url'], axis=1)  # Exclude the
target variable and the URL column

# Perform chi-square test for each feature variable
results = []
for feature in features:
    contingency_table = pd.crosstab(features[feature], target)
    chi2, p_value, _, _ = chi2_contingency(contingency_table)
    results.append({'Feature': feature, 'Chi-square': chi2,
'p-value': p_value})

# Convert the results to a DataFrame
results_df = pd.DataFrame(results)
```

```
print(results_df)
```