A Project Report

on

Detection of Malicious URLs

Submitted in partial fulfillment of the requirements

for the award of degree of

BACHELOR OF TECHNOLOGY

in

Information Technology

by

Alluri Meher Gayatri Devi (18WH1A1206)

Thondepu Gowri Siva Naga Sai Chandana (18WH1A1207)

Javvaji Anuhya (18WH1A1240)

Under the esteemed guidance of

Dr.Aruna Rao S L

Professor & HoD

Department of Information Technology

BVRIT HYDERABAD College of Engineering for Women

Rajiv Gandhi Nagar, Nizampet Road, Bachupally, Hyderabad – 500090

(Affiliated to Jawaharlal Nehru Technological University Hyderabad)

(NAAC 'A' Grade & NBA Accredited- ECE, EEE, CSE IT)

June, 2022

DECLARATION

We hereby declare that the work presented in this project entitled "**Detection of Malicious URLs**" submitted towards completion of the project in IV year II sem of B.Tech Department of IT at "BVRIT HYDERABAD College of Engineering for Women", Hyderabad is an authentic record of our original work carried out under the esteem guidance of **Dr. Aruna Rao S L, Professor & HoD** , IT Department.

Alluri Meher Gayatri Devi (18WH1A1206)

Thondepu Gowri Siva Naga Sai Chandana (18WH1A1207)

Javvaji Anuhya (18WH1A1240)

BVRIT HYDERABAD

College of Engineering for Women

Rajiv Gandhi Nagar, Nizampet Road, Bachupally, Hyderabad – 500090

(Affiliated to Jawaharlal Nehru Technological University Hyderabad)

(NAAC 'A' Grade & NBA Accredited- ECE, EEE, CSE IT)

CERTIFICATE

This is to certify that the project report on **"Detection Of Malicious URLs"** is a bonafide work carried out by **Alluri Meher Gayatri Devi (18WH1A1206), Thondepu Gowri Siva Naga Sai Chandana (18WH1A1207)** and **Javvaji Anuhya (18WH1A1240)** in the partial fulfillment for the award of B.Tech degree in **Information Technology, BVRIT HYDERABAD College of Engineering for Women, Bachupally, Hyderabad** affiliated to Jawaharlal Nehru Technological University, Hyderabad under my guidance and supervision. The results embodied in the project work have not been submitted to any other university or institute for the award of any degree or diploma.

Internal Guide
Dr. Aruna Rao S L
Professor & HoD
Department of IT

Head of the Department
Dr. Aruna Rao S L
Professor & HoD
Department of IT

External Examiner

ACKNOWLEDGEMENT

We would like to express our profound gratitude and thanks to **Dr. K. V. N. Sunitha, Principal, BVRIT HYDERABAD** for providing the working facilities in the college.

Our sincere thanks and gratitude to our guide **Dr. Aruna Rao S L, Professor & Head, Department of IT, BVRIT HYDERABAD** for all the timely support, constant guidance and valuable suggestions during the period of our project.

Finally, we would also like to thank our Project Coordinators **Dr. P. Kayal, Associate Professor, Dr. P. S Latha Kalyampudi, Associate Professor**, all the faculty and staff of the IT Department who helped us directly or indirectly, parents and friends for their cooperation in completing the project work.

Alluri Meher Gayatri Devi(18WH1A1206)

Thondepu Gowri Siva Naga Sai Chandana (18WH1A1207)

Javvaji Anuhya (18WH1A1240)

ABSTRACT

The World Wide Web has become an application on the Internet which brought in an immense risk of cyber-attacks. The more we grow in technology over internet the more we get exposed to malicious data which leads to cyber-attacks. With the rise in the underground Internet economy, automated malicious programs popularly known as malwares have become a major threat to computers and information systems connected to the internet. Malicious URLs have been widely used to mount various cyber-attacks including spamming, phishing and malware. To avoid this we train a machine learning model that takes a dataset which contains 6,51,191 records of various URLs and then distinguish benign, phishing, malware and defacement URLs. The user gets an alert whether an entered URL is malicious or benign.

LIST OF FIGURES

LIST OF ABBREVIATIONS

Acronym	Abbreviation
URL	Uniform Resource Locator
UML	Unified Modelling Language
BEP	Browser Exploit Pack
AdaBoost	Adaptive Boosting
SVM	Support Vector Machine
LightGBM	Light Gradient Boosting Machine
CatBoost	Categorical Boosting
XGB	Extreme Gradient Boosting
UI	User Interface
TLD	Top level Domain
FLD	Full length Domain
IP	Internet Protocol
WWW	World Wide Web
RAM	Random Access Memory
API	Application programming interface
UML	Unified Modeling Language
SMO	Sequential Minimal Optimization
KNN	K-Nearest Neighbours

CONTENTS

1. Introduction

Malicious URL detection is useful for users which prevents the users from clicking the malicious links and cautions them about the threats that can occur by clicking the link. Hackers often use spam and phishing to trick users into clicking malicious URL, the trojans will be implanted into the victims computers, or the victims sensitive information will be leaked. The technology of malicious URL detection can help users identify malicious URL and prevent users from being attacked by malicious URL.

Currently, the web is a widely used platform that supports an increasing number of daily activities. However, the security of web applications is unfortunately, becoming a serious problem as web-based services become increasingly prevalent. URL is the abbreviation of Uniform Resource Locator, which is the global address of documents and other resources on the World Wide Web.

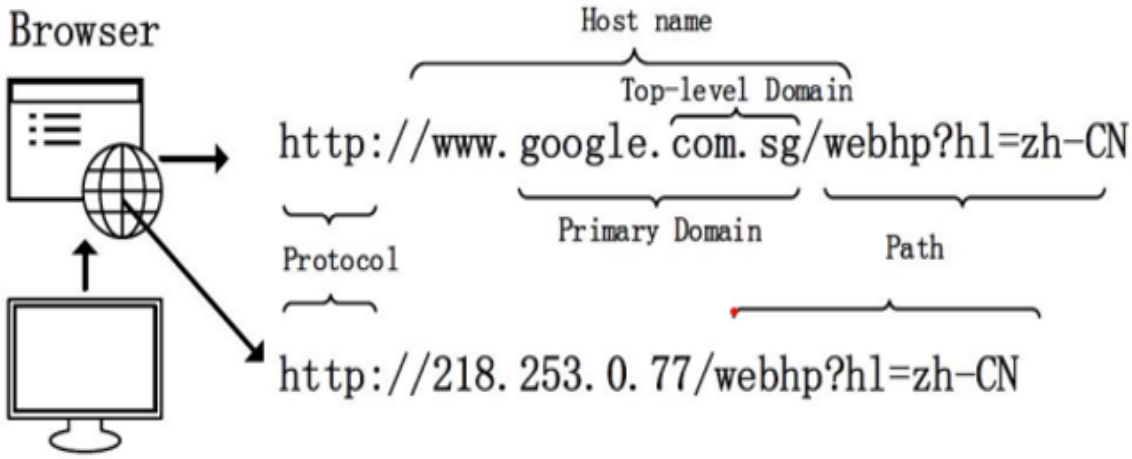

Figure 1.1: Example of URL - 'Uniform Resource Locator'

URLs are text strings that are human readable but not directly usable by client programs. Through a multi-step resolution process, the browser translates a URL into instructions on how to locate

the server that is hosting the site, as well as where within that host the site or resource is placed. Because a URL must go through this machine translation process, it has the following standard syntax:

protocol://[hostname][path]

Most business enterprises rely on the Internet services available on the WWW. These services are however susceptible to cyberattacks. Most cyberattacks usually occur when users click on malicious URLs. URLs are used on the WWW to access legitimate resources and when they are used for other reasons other than that they pose threats to data availability, controllability, confidentiality and integrity. Evidently, there is a continuous increase in web attacks launched through phishing, spam and malware.

Phishing URLs are usually used to gain unauthorised access to personal information for example banking details, passwords and names. When attackers obtain this information, they proceed to stealing money or logging into private institutions like government websites, school websites and hospital medical report databases. Spam URLs normally intend to perform unauthorized advertisements especially using well known brands like Amazon, Taobao, Alibaba and PayPal. Malware URLs usually spread malicious software that once downloaded will infect the host. Malicious software is spread with the intention to launch silent attacks on a user's computer by installing malicious programs.

Many web applications suffer from various web attacks due to the lack of awareness concerning security. Malicious URLs host unsolicited content (spam, phishing, drive-by downloads, etc.) and lure unsuspecting users to become victims of scams (monetary loss, theft of private information, and malware installation), and cause losses of billions of dollars every year. It is imperative to detect and act on such threats in a timely manner.

Now, it's important to note that malicious links can be created on fake and legitimate websites. Cybercriminals can create entirely fake and malicious websites, or they may opt to create malicious URLs for legitimate domains. Malicious URLs are delivered via many methods, including emails, websites, and advertisements. They host a variety of unwanted content ranging from spam-advertised products, to phishing sites, to dangerous 'drive-by' exploits that infect a visitor's machine with malware. Cybercriminals trick users into downloading malicious software that can used be to spy on victims or take over their devices. Creating these malicious URLs to carry out phishing attacks to gain access to users personal information to carry out identity theft or other types of fraud. And gain access to users login credentials to access their personal or professional accounts ,get into victims computers to encrypt their files for a ransomware attack.

Instead of sending malicious attachments, attackers embed malicious links in spear phishing emails and distribute them to their target audience. Clicking the link redirects the user's browser to a malicious domain running the BEP . BEP then scans the details of the browser, including various components such as plugins, to detect vulnerabilities that could be exploited in malware downloads. This attack, known as a drive-by download attack, forces targeted users to access malicious domains through social engineering .

An attacker can create a custom malicious domain to avoid exploiting legitimate websites to host malware. A custom malicious domain is a domain registered by an attacker that is unknown to evade detection and remains active for a short period of time. This design is primarily used for widespread infections, not targeted infections. However, attacks are more targeted due to changes in the attack patterns used in drive-by downloads. The context of malware infection remains the same, but the approach is different.

A safe way to know whether the link is malicious is to check which URL the link takes you to. If you hover your mouse over the link, you will be able to see the URL. If the URL shows a dif-

Figure 1.2: Malicious URL

ferent domain than the one it claims to lead to, then you'll know that it's a phishing or malicious URL.

An user can be attacked in different ways with malicious URLs, one such way is sending through E-mails. Check the sender's email address, most fraudulent emails have spoofed email addresses. The email might claim to be from a well-known company, but the sender's email address might give away their maligned intentions. The sender's email address can be completely different from the company the email claims to have originated from. The email address does not match the sender's display name. This is a glaring red flag of a potentially dangerous email, and you should refrain from taking any actions that the sender asks or demands.

This includes clicking on links or calling any phone numbers mentioned in the email. Check the subject of the email online. If you're not sure whether the email contains a malicious link or a genuine one, check the subject line on any search engine. There are high chances that the same malicious emails are sent to thousands of other people and somebody has posted examples of them online to help others. This method can help you figure out whether an email is fraudulent within seconds.

1.1 Objective

This application ensures all the users who request web resources via URLs are safe. URLs also provide an easy point of entry for attackers, who embed executable code into the website content or inject malicious code into the URL, creating a web attack. Our main objective is to improve the reliability of web applications by accurately detecting malicious URLs using machine learning techniques.

1.2 Problem Definition

In order to detect malicious URLs the user can enter the URL as an input to the application to identify whether an URL is malicious or benign. Our project is focusing on three main modules.

- Feature Extraction Module
- Detection Module
- Interface Module

In feature extraction module, we collect the relevant information about the URL like general features, count features, IP address, etc.,. In detection module, we train and test the model using machine learning algorithms. In interface module, user can know whether the URL is malicious or benign.

2. Literature Survey

2.1 Related Work

Adrian S, Tefan Popescu, Dumitru Bogdan Prelipcean and Dragos¸ Teodor Gavrilut[1] focuses on the usage of different machine learning techniques and unsupervised learning methods for detecting malicious URLs with respect to memory footprint. A fingerprint algorithm has the purpose to generate an idealized form. Two similar URLs have the same idealized form of an URL. All URLs with the same idealized form are grouped into the same cluster. Using more than one fingerprint algorithm has the advantage of covering multiple aspects of the same URL.This paper focused on the usage of different machine learning techniques like unsupervised learning algorithm that requires less human intervention and reliable in terms of detection rate, response time and false positives. These algorithms can be used in an integrated solution for detection of malicious web resources.

Mohammed Nazim Feroz, Susan Mengel[2] focuses on the malicious data for which a counter measure must be taken that generalizes across web services to protect the user from phishing host URLs. An approach is used to classify URLs automatically based on their lexical and hostbased features. Clustering is performed on the entire dataset and a cluster ID (or label) is derived for each URL, which in turn is used as a predictive feature by the classification system. Online URL reputation services are used in order to categorize URLs and the categories returned are used as a supplemental source of information that would enable the system to rank URLs. The classifier achieves 93-98 accuracy by detecting a large number of phishing hosts, while maintaining a modest false positive rate.

Sunil B. Rathod, Tareek M. Pattewar[3] focuses on the malicious data which is being sent through emails using URL's.Thus to stop such activity a spam and malicious URLs detection system is required which benefits users by removing spam content and malicious URLs in Email.They used data mining approach like supervised classification which improves the systems accuracy and detects more amount of spam and malicious URLs. They used combination approach of Bayesian classifier and Decision Tree. Combination approach has improved the results in terms of Accuracy and It

became the efficient method for classification of content based spam detection and malicious URL detection in integrated form.

In [4], they proposed a multi-layer model for detecting malicious URL. The filter can directly determine the URL by training the threshold of each layer filter when it reaches the threshold. Otherwise, the filter leaves the URL to next layer. The model is mainly composed of 4- layer classifier, where these classifiers are also called filters in this model, so the model consists of 4 Layer filter composition. In the multi-layer filter model, Naive Bayesian filter determined 308 URLs, decision tree filter determined the 1370 URLs, another 322 URLs. Every layer of the model plays a beneficial role for classification of the URLs and ultimately improves the detection of malicious URL in terms of accuracy.

T. Manyumwa, P. F. Chapita, H. Wu and S. Ji,[5] have proposed an approach to detect malicious URLs using four datasets containing different URLs. They extracted URL based features such as word-based features, special character features, KL divergence features, bag of words segmentation features, domain name features and short URL features and evaluated the performance of four ensemble machine learning classifiers and reported their average accuracy, micro and macro precision, micro and macro recall and F1-Score.The four different algorithms used are XGBoost, AdaBoost, LightGBM and CatBoost. They trained these algorithms on 1,26,983 URLs from benchmark datasets and all four learners returned an overall accuracy above 0.95.

Baojiang Cui[6] have proposed statistical analysis based on gradient learning and feature extraction using a sigmoidal threshold level are combined to propose a new detection approach based on machine learning techniques. They used naïve Bayes, decision tree and SVM classifiers and validated the accuracy and efficiency of the model. The dataset used is collected from real time data from different network security companies. An automatic feature extraction algorithm based on a sigmoidal threshold level is proposed to validate the effectiveness of each feature.

To identify the malicious URLs, Machine Learning - based classifiers draws features from web-pages and contents like lexical, visual,URL features, redirect paths, host-based features, or combinations of them. Such classifiers acts as knowledge base which are usually in browser URLs or from web service providers. If the classifier is fed with URL-based features, an aggregator is pre-processed before extracting features. Supervised learning paradigms like Naive Bayes, SVM with different kernels, and Logistic Regression are popular Machine Learning classifiers used for filtering spam and phishing URLs. To solve this problem they used a multi-layer filtering model to classify and handle relative data.

Alazab, M. Venkatraman, S. Watters, P.Alazab, M focuses[12] on unknown malware that are created using code obfuscation techniques that can modify the parent code to produce offspring copies which have the same functionality but with different signatures. In this paper, they have proposed and evaluated a novel method of employing several data mining techniques like KNN, SMO, Naive Bayes which are used to detect and classify zero-day malware with high levels of accuracy and efficiency based on the frequency of Windows API calls. This paper describes the methodology employed for the collection of large data sets to train the classifiers, and analyses the performance results of the various data mining algorithms adopted for the study using a fully automated tool developed in this research to conduct the various experimental investigations and evaluation. Based on the performance results of the algorithms, comparative analysis was drawn for accurately detecting zero-day malware successfully.

3. System Design

3.1 UML Diagrams

UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The goal of UML is to become a common language for creating models of object oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. UML is a very important part of developing objects oriented software and the software development process. It uses mostly graphical notations to express the design of software projects.

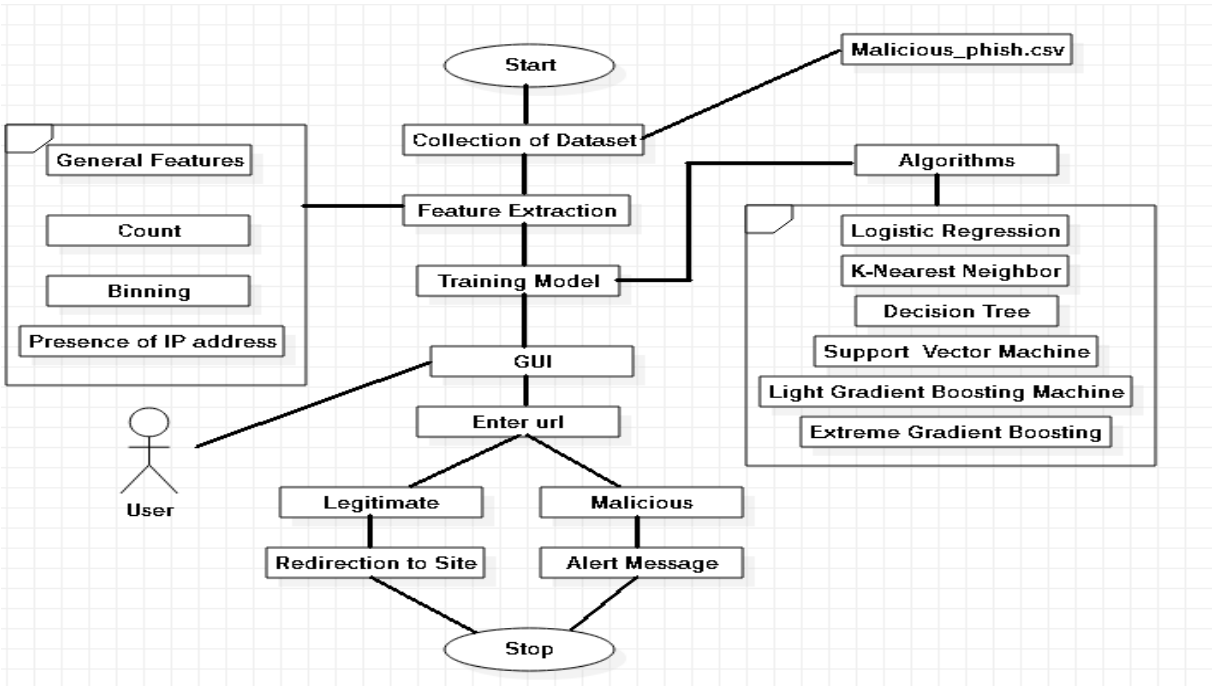

Figure 3.1: Use Case Diagram for Detection of Malicious URL

Figure 3.1 shows the use case diagram for detection of malicious URLs to help capture requirements of the system. Here, user is the actor who interacts with the UI. The process starts by collecting the dataset and the features are extracted from the URLs in the dataset. The dataset is divided into

testing and training. Different machine learning algorithms are used to train the model. After training, the user can enter the URLs as input to detect the type of URLs.

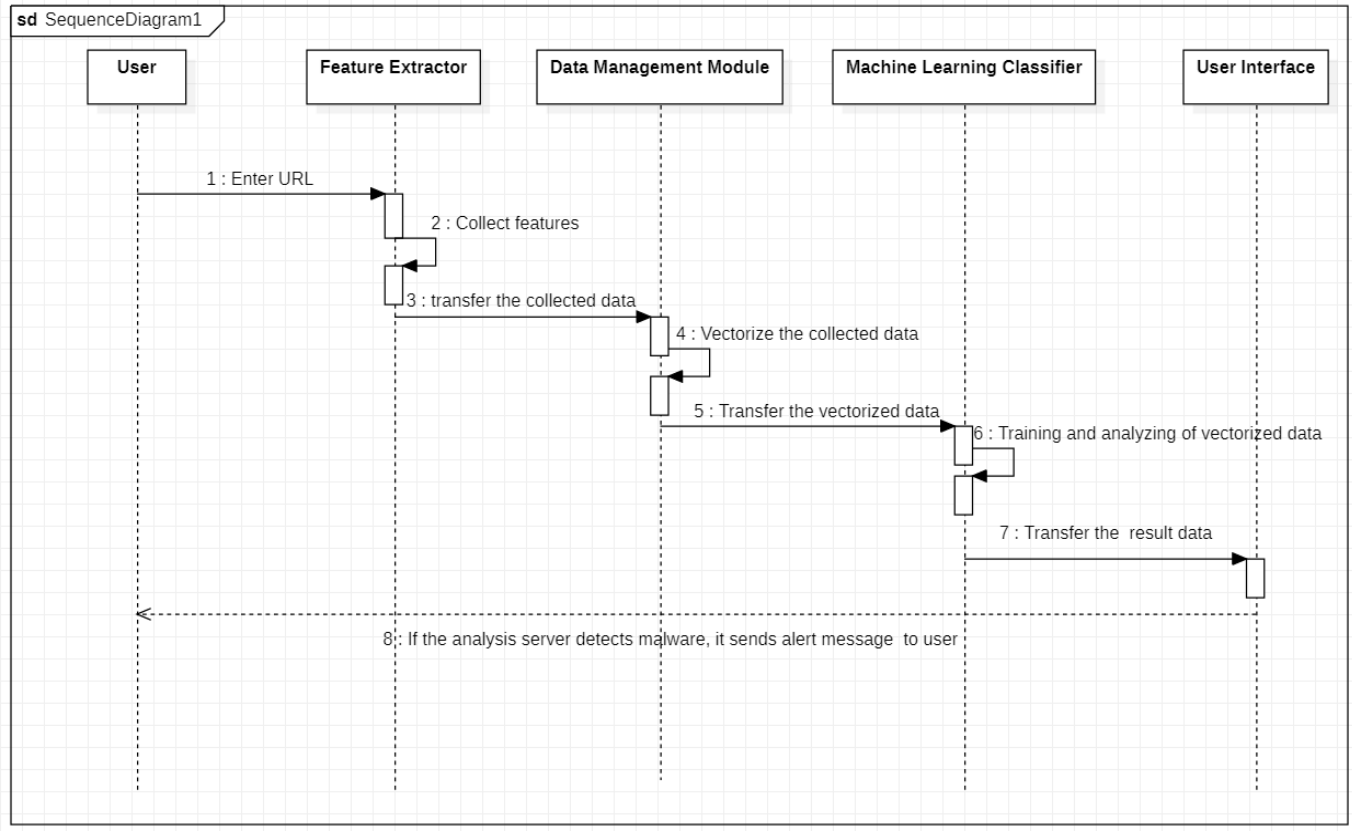

Figure 3.2: Sequence Diagram for Detection of Malicious URL

Figure 3.2 shows the sequence of interactions between lifelines. The lifelines here are User, Feature Extractor, Data Management Module, Machine Learning Classifier and User Interface. When the user enters an URL, the system extracts the features of the URL and compare the results with the trained model. If the URL is legitimate, the system redirects to entered site. If the URL is malicious, an alert message is popped in the interface.

3.2 Software and Hardware Requirements

Hardware Requirements:

- Intel x86 or compatible processor
- Minimum of 512 MB RAM
- Compatible operating systems:
    - An x86 Linux operating system.
    - A 64-bit Windows operating system such as Windows Vista, Windows 7, Windows 8, Windows 10, Windows Server 2008 or Windows Server 2012.

Software Requirements:

- Tools:
    - Anaconda
    - Python 3
- Libraries:
    - Numpy
    - Pandas
    - Matplotlib
    - Sklearn
    - Pickle
- Framework:
    - Flask

3.3 Libraries Used

- **Numpy:** NumPy, which stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed. We have used numpy to convert the comprehensive data like categories of URLs into numerical array representation.

- **Pandas:** Pandas is an open source Python package that is most widely used for data science/data analysis and machine learning tasks. It is built on top of another package named Numpy, which provides support for multi-dimensional arrays. pandas is a software library written for data manipulation and analysis. We have used pandas library for reading the dataset.

- **Matplotlib:** Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It is a comprehensive library for creating static, animated, and interactive visualizations in Python. To represent the categories of URLs graphically matplotlib is used.

- **Sklearn:** Sklearn is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python. Different Machine learning algorithms are imported using sklearn library.

- **Pickle:** In python pickle is used for serializing and de-serializing python object structures. “Pickling” is the process whereby a Python object hierarchy is converted into a byte stream. We have used pickle library to store the trained model.

4. Methodology

4.1 Architecture

Malicious URL Detection using machine learning approach tries to analyze the information of a URL and its corresponding websites or webpages, by extracting good feature representations of URLs like general features, count features, binned features and ratio features. In the UI an user enters the URL and the results will display the type of URL i.e, benign or malware.

Our Proposed architecture contains three main phases, they are Feature extraction phase, Detection phase and Interface phase. In feature extraction phase general features and additional features of an URL are extracted. In detection phase, the data is splitted into test and train and machine learning algorithms are used to train the data. In interface phase, UI is implemented using flask framework.

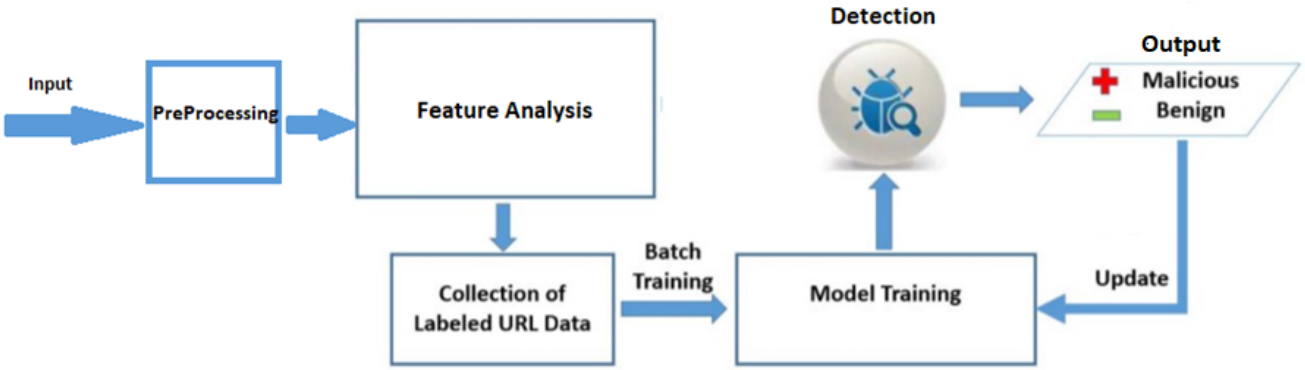

Figure 4.1: Framework for Malicious URL Detection

In figure 4.1 first the data is given as input and is preprocessed. The preprocessing procedure makes sure the data are in a usable format and transfers the values to vector features. Feature extraction is the core of the processing; it identifies the essential characteristics of malicious URLs. From the Feature Analysis, labeled URL data is collected and the features are fed into an algorithm. The machine learning algorithms like Decision Tree, Logistic Regression, LightGBM, XGB and SVM are

used. The machine learning module contains two independent processes. The training process uses the machine learning algorithm to build the detection model. The testing process loads the detection model to test unknown URLs and outputs the test result. The evaluation module is devoted to estimate whether the given URL is malicious or not and the result is updated to the model.

4.2 Modules

Figure 4.2 shows three modules i.e Feature Extraction Module, Detection Module and Interface Module. These modules will be discussed further .

Figure 4.2: Modules

4.2.1 Feature Extraction Module

This module is engineering oriented, which aims to collect relevant information about the URL.The unstructured information about the URL (e.g. textual description) is appropriately formatted, and converted to a numerical vector so that it can be fed into machine learning algorithms. For example, the numerical information can be used as is, and Bag-of-words is often used for representing textual or lexical content. , an automatic feature extraction algorithm is proposed to validate the effectiveness of each feature. This module proposes a new method to evaluate features automatically and extract more effective features to detect malicious URLs.The features are extracted from the URLs

and categorised based on background information.

4.2.2 Detection Module

This detection framework can be divided into three main modules: the processing procedure, the machine learning procedure and the evaluation procedure. The processing procedure makes sure the data are in a usable format and transfers the values to vector features. The machine learning module is where the core algorithms reside; depending on the algorithm, it feeds the algorithm valid clean data from the processing module and extracts knowledge or information. The evaluation module is to estimate whether the preceding processing can result in good performance and be put to use. Different algorithms are adopted at analysing different data types. To obtain the best performance, multiple classification models are tested to obtain the most appropriate algorithm.

4.2.3 Interface Module

In this module we introduce a UI using flask framework where the user can enter the URL to check if it is a malicious or not. If the URL is legitimate or benign, the URL will redirect to the entered site. If the URL is malicious, phishing or defacement, an alert message is displayed.

4.3 Algorithms

The core idea of detection is to feed valid clean data to the machine learning classification model to improve the detection precision. Many machine learning classification models have good performance and could be applied; therefore, determining how best to process the web data is the most significant procedure. The classifiers used in the classification framework include Logistic Regression, KNN using cosine metric, SVM, Decision Tree, Random Forest, and XGBoost. These boosting ensembles were initially used for binary classification.

4.3.1 Logistic Regression

Logistic regression is basically a supervised classification algorithm. In a classification problem,

the target variable(or output), y, can take only discrete values for a given set of features(or inputs), X. Contrary to popular belief, logistic regression is a regression model. Logistic regression uses the concept of predictive modeling as regression; therefore, it is called logistic regression, but is used to classify samples; Therefore, it falls under the classification algorithm. figure 4.3 shows the logistic function.

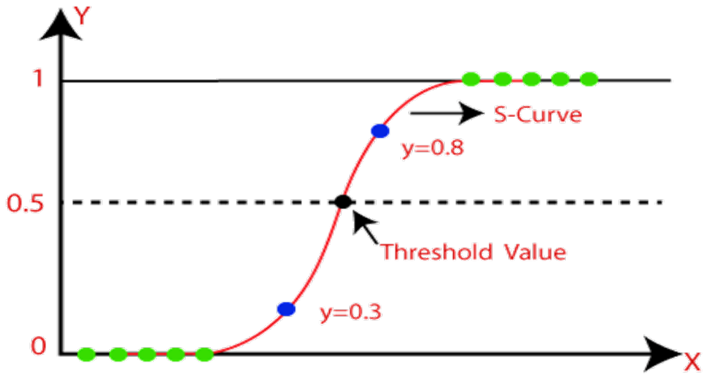

Figure 4.3: Representation of Binomial Logistic Regression

4.3.2 KNN Using Cosine Metric

The KNN algorithm is a simple, easy-to-implement supervised machine learning algorithm that can be used to solve both classification and regression problems. Machine learning models use a set of input values to predict output values. KNN is one of the simplest forms of machine learning algorithms mostly used for classification. It classifies the data point on how its neighbor is classified.

The distance metric is mainly used to calculate the similarity between two vectors by the cosine of the angle between two vectors and determines whether two vectors are pointing in the same direction.Therefore the range of the Cosine Distance ranges from 0 to 1 as well. If the Cosine Distance is zero (0), that means the items are identical. If the Cosine Distance is one (1) that means the items are definitely different.

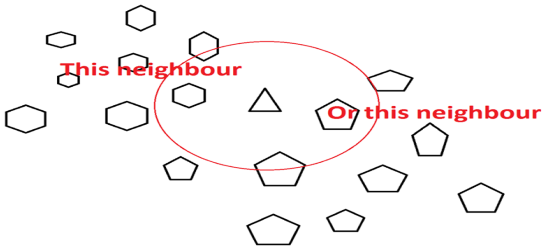

Figure 4.4: Nearest Neighbour Based on Cosine Metric

4.3.3 SVM

SVM is a supervised machine learning algorithm which can be used for classification or regression problems. The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane. SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine.

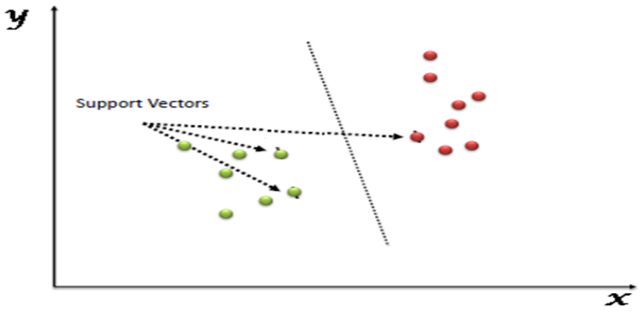

Figure 4.5: Hyperplane Drawn for 2-Dimentional Space

4.3.4 Decision Tree

Decision Tree algorithm belongs to the family of supervised learning algorithms. Unlike other supervised learning algorithms, the decision tree algorithm can be used for solving regression and classification problems too. The goal of using a Decision Tree is to create a training model that can use to predict the class or value of the target variable by learning simple decision rules inferred from prior data(training data). In Decision Trees, for predicting a class label for a record we start from the root of the tree. We compare the values of the root attribute with the record's attribute.

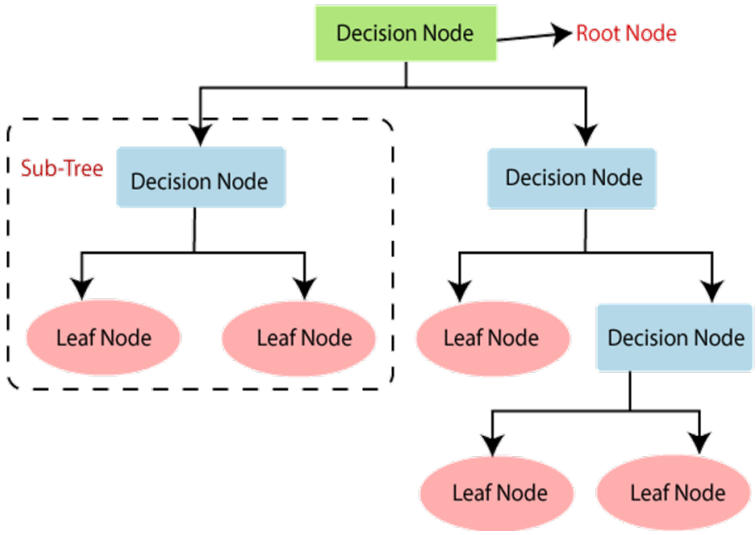

Figure 4.6: Work Flow of Decision Tree Algorithm

4.3.5 Random Forest

Random forest is a Supervised Machine Learning Algorithm that is used widely in Classification and Regression problems. It builds decision trees on different samples and takes their majority vote for classification and average in case of regression. One of the most important features of the Random Forest Algorithm is that it can handle the data set containing continuous variables as in the case of regression and categorical variables as in the case of classification. It performs better results for

classification problems.

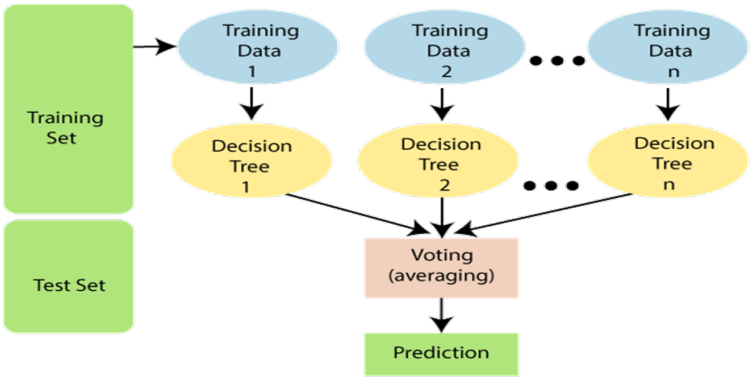

Figure 4.7: Work Flow of Random Forest

4.3.6 XGBoost

The XGBoost algorithm provides a gradient boosting framework designed for better computational speed and performance. The boosting technique has the ability to handle missing values and perform better regularization. XGBoost is widely used due to its loss minimization abilities. In this algorithm, decision trees are created in sequential form. Weights play an important role in XGBoost. Weights are assigned to all the independent variables which are then fed into the decision tree which predicts results. The weight of variables predicted wrong by the tree is increased and these variables are then fed to the second decision tree. These individual classifiers/predictors then ensemble to give a strong and more precise model.

5. Implementation

5.1 Dataset

The dataset we used is **Malicious_phish.csv**. It has 6,51,191 records consisting of two columns namely URL and type. Figure 5.1 shows the sample data.

```
[ ]  data.head()
```

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

Figure 5.1: Sample Data

Dataset taken into consideration contains four different types of URLs such as Benign - 66%, Defacement - 15%, Phishing and Malware - 19%.

Benign URLs are the safe URLs. Benign URLs were collected from Alexa top websites. The domains have been passed through a Heritrix web crawler to extract the URLs.

Defacement is an attack in which malicious parties penetrate a website and replace content on the site with their own messages.

Malware in short a “malicious software”, a file or code, typically delivered over a network, that infects, explores, steals or conducts virtually any behavior an attacker wants. And because malware comes in so many variants, there are numerous methods to infect computer systems.

Phishing is a type of social engineering attack in which cyber criminals trick victims into handing over sensitive information or installing malware.

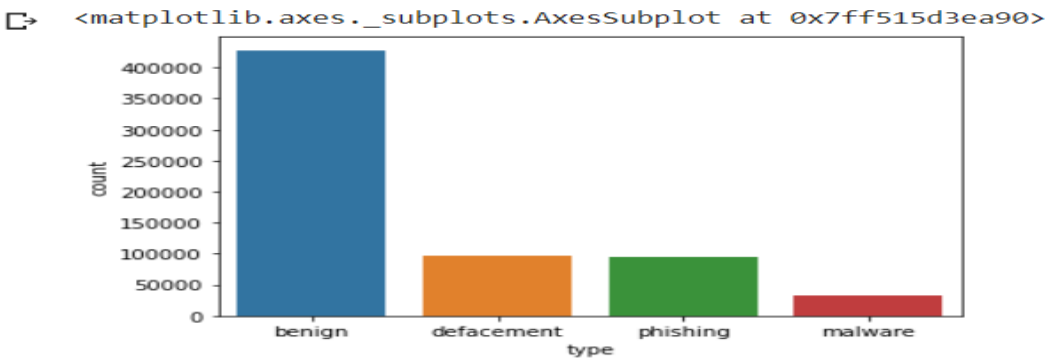

Figure 5.2: Distribution of Types of URLs in the Dataset

5.2 Code

In dataset, URL's are categorized into four types. They include Benign, Defacement, Phishing and Malware. In preprocessing the text data is converted into array by using numpy library. The result is shown in binary values in figure 5.3.

```
from sklearn import preprocessing

# label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'species'.
dataset['type']= label_encoder.fit_transform(dataset['type'])
dataset['type'].unique()
```

```
array([3, 0, 1, 2])
```

Figure 5.3: Preprocessed Values

We split the whole dataset for testing and training. As shown in figure 5.4, 25 % of dataset is used for testing and remaining 75 % of dataset is used for training.

```
# splitting the url dataset using 25 percent for testing
url_train, url_test, target_train, target_test = model_selection.train_test_split(url_data, target_label, test_size=0.2)
```

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)
```

Figure 5.4: Splitting of Data for Testing and Training

Machine Learning Algorithms like logistic regression, KNN using cosine metric, SVM, decision tree, random forest are implemented and their accuracy scores are shown in the Figure 5.5.

```
models = pd.DataFrame({
    'Model': [ 'Logistic Regression', 'KNN','Naive Bayes','SVM','Decision Tree','Random Forest'],
    'Test Score': [ log_reg,knn_cos,nb,svm_rbf,dec_tree,rf]})
models.sort_values(by='Test Score', ascending=False)
```

	Model	Test Score
4	Decision Tree	0.958842
1	KNN	0.950701
3	SVM	0.947083
5	Random Forest	0.942108
0	Logistic Regression	0.926730
2	Naive Bayes	0.591135

Figure 5.5: Algorithms and their Accuracy Scores

XgBoost Algorithm is implemented in the figure 5.6. This algorithm gives the highest accuracy compared to other machine learning algorithms implemented before.

```
import xgboost as xgb
model = xgb.XGBClassifier(n_estimators= 1000)
model.fit(x_train,y_train)
y_pred = model.predict(x_test)
from sklearn.metrics import accuracy_score
xgboost =accuracy_score(y_test,y_pred6)
xgboost
```

```
0.9606512890094979
```

Figure 5.6: Implemention of XgBoost Algorithm

Notebook.ipynb

import pandas as pd

import numpy as np

from string import printable

from sklearn import model_selection

from tensorflow.keras.preprocessing import sequence

from tensorflow.keras.layers import Dropout, Dense

from tensorflow.keras import regularizers

from tensorflow.keras.layers import Input, Embedding, Convolution1D, ELU, MaxPooling1D, LSTM

from tensorflow.keras.models import Model

from tensorflow.keras.optimizers import Adam

dataset = pd.read_csv('OneDrive/Desktop/Malicious-Url-Detection-Updatedt (2)/data/malicious_phish.csv')

```
dataset.info()
class 'pandas.core.frame.DataFrame'
RangeIndex: 651191 entries, 0 to 651190
Data columns (total 2 columns):
Column Non-Null Count Dtype
— —— ———— —
0 url 651191 non-null object
1 type 651191 non-null object
dtypes: object(2)
memory usage: 9.9+ MB

missing_data = dataset.isnull().sum()
print(missing_data)

url 0
type 0
dtype: int64

dataset.head()

from sklearn import preprocessing

# label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()
```

```
# Encode labels in column 'species'.
dataset['type']= label_encoder.fit_transform(dataset['type'])
dataset['type'].unique()
array([3, 0, 1, 2])

len(dataset)
651191

printable
'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!"$%&()*+,-./:;¡=¿?@'

def urlPrep(url):
    return [printable.index(char) + 1 for char in url if char in printable]

url_tokens = []
for url in dataset.url:
    url_tokens.append(urlPrep(url))

max_length = 75
url_data = sequence.pad_sequences(url_tokens, maxlen=max_length)

target_label = np.array(dataset.type)
target_label
array([3, 0, 0, ..., 3, 3, 3])

url_data.shape
(651191, 75)
```

```
target_label.shape
```

(651191,)

```
# splitting the url dataset using 25 percent for testing
url_train, url_test, target_train, target_test = model_selection.train_test_split(url_data, target_label, test_size=0.2)

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

ds= pd.read_csv("data/dataset_website.csv")
ds.head()
ds.isnull().any()

x=ds.iloc[:,1:31].values
y=ds.iloc[:,-1].values
print(x,y)
```

```
[[ 1  1  1 ...  1  1  1]
 [ 1  0  1 ...  1  0 -1]
 [ 1  0  1 ...  1 -1  1]
 ...
 [ 1 -1  1 ...  1  0  1]
 [-1 -1  1 ...  1  1  1]
 [-1 -1  1 ... -1  1 -1]] [-1 -1 -1 ... -1 -1 -1]
```

```
y=ds.iloc[:,-1].values
y
```

array([-1, -1, -1, ..., -1, -1, -1], dtype=int64)

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)
```

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
lr=LogisticRegression()
lr.fit(x_train,y_train)
y_pred1=lr.predict(x_test)
from sklearn.metrics import accuracy_score
log_reg=accuracy_score(y_test,y_pred1)
log_reg
```

0.926729986431479

KNeighborsClassifier using Cosine metric

```
from sklearn.neighbors import KNeighborsClassifier
kn1=KNeighborsClassifier(n_neighbors=5,metric='cosine')
kn1.fit(x_train,y_train)
y_pred2=kn1.predict(x_test)
from sklearn.metrics import accuracy_score
knn_cos=accuracy_score(y_test,y_pred2)
knn_cos
```

0.9507010402532791

SVM

```
from sklearn.svm import SVC
svm1=SVC(kernel='rbf', gamma='auto')
svm1.fit(x_train,y_train)
y_pred4=svm1.predict(x_test)
from sklearn.metrics import accuracy_score
svm_rbf=accuracy_score(y_test,y_pred4)
svm_rbf
```

0.9470827679782904

Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
dt=DecisionTreeClassifier()
dt.fit(x_train,y_train)
y_pred6=dt.predict(x_test)
from sklearn.metrics import accuracy_score
dec_tree=accuracy_score(y_test,y_pred6)
dec_tree
```

0.9611035730438715

Random Forest

```
from sklearn.ensemble import RandomForestRegressor
Rf=RandomForestRegressor(n_estimators=100,random_state=0,n_jobs=-1)
Rf.fit(x_train,y_train)
y_pred7=Rf.predict(x_test)
from sklearn.metrics import accuracy_score
rf=accuracy_score(y_test,y_pred7.round())
rf
```

0.942107643600181

GaussianNB

```
from sklearn.naive_bayes import GaussianNB
gb=GaussianNB()
gb.fit(x_train,y_train)
y_pred8=gb.predict(x_test)
from sklearn.metrics import accuracy_score
nb=accuracy_score(y_test,y_pred8)
nb
```

0.5911352329262777

Xg Boosting Algorithm

```
import xgboost as xgb
model = xgb.XGBClassifier(n_estimators= 1000)
model.fit(x_train,y_train)
y_pred = model.predict(x_test)
from sklearn.metrics import accuracy_score
xgboost =accuracy_score(y_test,y_pred6)
xgboost
```

0.9606512890094979

```
test_url_mal = ”naureen.net/etisalat.ae/index2.php”
test_url_benign = ”sixt.com/php/reservation?language=en_US”
#url = ”naureen.net/etisalat.ae/index2.php”
url = ”sixt.com/php/reservation?language=en_US”
#url = ”http://www.approvi.com.br/ck.htm”

url_int_tokens = [[printable.index(x) + 1 for x in test_url_benign if x in printable]]
max_len=75
X = sequence.pad_sequences(url_int_tokens, maxlen=max_len)
```

```
target_proba = model.predict(X, batch_size = 1)
def print_result(proba):
    if proba   0.5:
        return ”malicious”
    else:
        return ”benign”
print('Test URL:', url, 'is', print_result(target_proba))
```

Test URL: sixt.com/php/reservation?language=en_US is malicious

```
import pickle
pickle.dump(xgboost,open('Phising_website.pkl','wb'))
```

feature.py

```
import regex
from tldextract import extract
import ssl
import socket
from bs4 import BeautifulSoup
import urllib.request
import whois
import datetime
import requests
import favicon
import re
import google
from googlesearch import search

#checking if URL contains any IP address. Returns -1 if contains else returns 1
def having_IPhaving_IP_Address(url):
    match=regex.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 1
```

```
#Checking for the URL length. Returns 1 (Legitimate) if the URL length is less than 54 characters
#Returns 0 if the length is between 54 and 75
#Else returns -1;

def URLURL_Length (url):
    length=len(url)        if(length=75):
        if(length54):
            return 1
        else:
            return 0
    else:
        return -1

#Checking with the shortening URLs.
#Returns -1 if any shortening URLs used.
#Else returns 1
def Shortining_Service (url):
    match=regex.search('bit/.ly goo/.gl shorte/.st—go2l/.ink x/.co ow/.ly t/.co tinyurl tr/.im is/.gd cli/.gs ' 'yfrog/.com migre/.me—ff/.im tiny/.cc url4/.eu twit/.ac su/.pr twurl/.nl snipurl/.com—' 'short/.to—BudURL/.com ping/.fm post/.ly Just/.as bkite/.com snipr/.com fic/.kr loopt/.us' 'doiop/.com short/.ie kl/.am wp/.me rubyurl/.com om/.ly to/.ly bit/.do t/.co lnkd/.in' 'db/.tt qr/.ae adf/.ly goo/.gl bitly/.com cur/.lv tinyurl/.com ow/.ly bit/.ly ity/.im' 'q/.gs is/.gd po/.st bc/.vc twitthis/.com u/.to j/.mp buzurl/.com cutt/.us u/.bb yourls/.org' 'x/.co prettylinkpro/.com scrnch/.me filoops/.info vzturl/.com qr/.net 1url/.com tweez/.me v/.gd tr/.im link/.zip/.net',url)
```

```
    if match:
        return -1
    else:
        return 1
```

Checking for @ symbol. Returns 1 if no @ symbol found. Else returns 0.

```
def having_At_Symbol(url):
    symbol=regex.findall(r'@',url)
    if(len(symbol)==0):
        return 1
    else:
        return -1
```

Checking for Double Slash redirections. Returns -1 if // found. Else returns 1

```
def double_slash_redirecting(url):
    for i in range(8,len(url)):
        if(url[i]=='/'):
            if(url[i-1]=='/'):
                return -1
    return 1
```

#Checking for - in Domain. Returns -1 if '-' is found else returns 1.

```
def Prefix_Suffix(url):
    subDomain, domain, suffix = extract(url)
    if(domain.count('')):
        return -1
    else:       return 1
```

```
#checking the Subdomain. Returns 1 if the subDomain contains less than 1 '.'
#Returns 0 if the subDomain contains less than 2 '.'
#Returns -1 if the subDomain contains more than 2 '.'
def having_Sub_Domain(url):
    subDomain, domain, suffix = extract(url)
    if(subDomain.count('.')¡=2):
        if(subDomain.count('.')¡=1):
            return 1
        else:
            return 0
    else:
        return -1

#Checking the SSL. Returns 1 if it returns the respomse code and -1 if exceptions are thrown.
def SSLfinal_State(url):
    try:
        response = requests.get(url)
        return 1
    except Exception as e:
        return -1

#domains expires on  1 year returns -1, otherwise returns 1
def Domain_registeration_length(url):
    try:
        domain = whois.whois(url)
        exp=domain.expiration_date[0]
```

```
            up=domain.updated_date[0]
            domainlen=(exp-up).days
            if(domainlen¡=365):
                  return -1
            else:
                  return 1
      except:
            return -1
```

#Checking the Favicon. Returns 1 if the domain of the favicon image and the URL domain match else returns -1.

```
def Favicon(url):
      subDomain, domain, suffix = extract(url)
      b=domain
      try:
            icons = favicon.get(url)
            icon = icons[0]
            subDomain, domain, suffix =extract(icon.url)
            a=domain
            if(a==b):
                  return 1
            else:
                  return -1
      except:
            return -1
```

```
#Checking the Port of the URL. Returns 1 if the port is available else returns -1.
def port(url):
    try:
    a_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        location=(url[7:],80)
        result_of_check = a_socket.connect_ex(location)
        if result_of_check == 0:
            return 1
        else:
            return -1
        a_socket.close
    except:
        return -1


# HTTPS token in part of domain of URL returns -1, otherwise returns 1 def HTTPS_token(url):
    match=re.search('https://—http://',url)
    if (match.start(0)==0):
        url=url[match.end(0):]
    match=re.search('http https',url)
    if match:
        return -1
    else:
        return 1
```

```
#% of request URL¡22% returns 1, otherwise returns -1
def Request_URL(url):
    try:
        subDomain, domain, suffix = extract(url)
        websiteDomain = domain
        opener = urllib.request.urlopen(url).read()
        soup = BeautifulSoup(opener, 'lxml')
        imgs = soup.findAll('img', src=True)
        total = len(imgs)
        linked_to_same = 0
        avg =0
        for image in imgs:
            subDomain, domain, suffix = extract(image['src'])
            imageDomain = domain
            if(websiteDomain==imageDomain or imageDomain==''):
                linked_to_same = linked_to_same + 1
        vids = soup.findAll('video', src=True)
        total = total + len(vids)
        for video in vids:
            subDomain, domain, suffix = extract(video['src'])
            vidDomain = domain
            if(websiteDomain==vidDomain or vidDomain==''):
                linked_to_same = linked_to_same + 1
        linked_outside = totallinked_to_same
        if(total!=0):
            avg = linked_outside/total
        if(avg¡0.22):
```

```
                return 1
        else:
                return -1
    except:
        return 0
```

 # % of URL of anchor¡31% returns 1,% of URL of anchor 31% and 67% returns 0, otherwise returns -1

```
def URL_of_Anchor(url):
    try:
        subDomain, domain, suffix = extract(url)
        websiteDomain = domain
        opener = urllib.request.urlopen(url).read()
    soup = BeautifulSoup(opener, 'lxml')
        anchors = soup.findAll('a', href=True)
        total = len(anchors)
        linked_to_same = 0
        avg = 0
        for anchor in anchors:
            subDomain, domain, suffix = extract(anchor['href'])
            anchorDomain = domain
            if(websiteDomain==anchorDomain or anchorDomain==''):
                linked_to_same = linked_to_same + 1
        linked_outside = totallinked_to_same
        if(total!=0):
            avg = linked_outside/total
        if(avg¡0.31):
```

```
            return 1
        elif(0.31¡=avg¡=0.67):
            return 0
        else:
            return -1
    except:
        return 0
```

#:% of links in ¡meta¿, ¡script¿and¡link¿tags ¡ 25% returns 1,% of links in ¡meta¿, #¡script¿ and ¡link¿ tags 25% and 81% returns 0, otherwise returns -1 def Links_in_tags(url):

```
    try:
        opener = urllib.request.urlopen(url).read()
        soup = BeautifulSoup(opener, 'lxml')
        no_of_meta = 0
        no_of_link = 0
        no_of_script = 0
        anchors = 0
        avg = 0
        for meta in soup.find_all('meta'):
            no_of_meta = no_of_meta+1
        for link in soup.find_all('link'):
            no_of_link = no_of_link + 1
        for script in soup.find_all('script'):
            no_of_script = no_of_script + 1
        for anchor in soup.find_all('a'):
            anchors = anchors + 1
        total = no_of_meta + no_of_link + no_of_script + anchors
```

```
            tags = no_of_meta + no_of_link + no_of_script
            if(total!=0):
                avg = tags/total
            if(avg¡0.25):
                return -1
            elif(0.25¡=avg¡=0.81):
                return 0
            else:
                return 1
        except:
            return 0

#Server Form Handling
#SFH is 'about: blank' or empty → phishing, SFH refers to a different domain → suspicious, otherwise → legitimate
def SFH(url):
    return -1

#:using 'mail()' or 'mailto:' returning -1, otherwise returns 1
def Submitting_to_email(url):
    try:
        opener = urllib.request.urlopen(url).read()
        soup = BeautifulSoup(opener, 'lxml')
        if(soup.find('mailto:','mail():')):
            return -1
        else:
            return 1
```

```
    except:
        return -1

#Host name is not in URL returns -1, otherwise returns 1 def Abnormal_URL(url):
    subDomain, domain, suffix = extract(url)
    try:
        domain = whois.whois(url)
        hostname=domain.domain_name[0].lower()
        match=re.search(hostname,url)
        if match:
            return 1
        else:
            return -1
    except:
        return -1

#number of redirect page  1 returns 1, otherwise returns 0
def Redirect(url):
    try:
        request = requests.get(url)
        a=request.history
        if(len(a)¡=1):
            return 1
        else:
            return 0
    except:
        return 0
```

```
#onMouseOver changes status bar returns -1, otherwise returns 1
def on_mouseover(url):
    try:
        opener = urllib.request.urlopen(url).read()
        soup = BeautifulSoup(opener, 'lxml')
        no_of_script = 0
        for meta in soup.find_all(onmouseover=True):
            no_of_script = no_of_script + 1
        if(no_of_script==0):
            return 1
        else:
            return -1
    except:
        return -1

#right click disabled returns -1, otherwise returns 1
def RightClick(url):
    try:
        opener = urllib.request.urlopen(url).read()
        soup = BeautifulSoup(opener, 'lxml')
        if(soup.find_all('script',mousedown = True)) :
            return − 1
        else :
            return1
    except :
        return − 1
```

```
#popup window contains text field → phishing, otherwise → legitimate
def popUpWidnow(url):
    return 1

#using iframe returns -1, otherwise returns 1
def Iframe(url):
    try:
        opener = urllib.request.urlopen(url).read()
        soup = BeautifulSoup(opener, 'lxml')
        nmeta=0
        for meta in soup.findAll('iframe',src=True):
            nmeta= nmeta + 1
        if(nmeta!=0):
            return -1
        else:
            return 1
    except:
        return -1

#:age of domain  6 months returns 1, otherwise returns -1
def age_of_domain(url) :
    try :          w = whois.whois(url).creation_date[0].year
        if (w <= 2018) :
            return1
        else :
            return − 1
```

```
    exceptExceptionase :
        return − 1

#no DNS record for domain returns -1, otherwise returns 1
def DNSRecord(url):
    subDomain, domain, suffix = extract(url)
    try:
        dns = 0
        domain_name = whois.whois(url)
    except:
        dns = 1
    if(dns == 1):
        return -1
    else:
        return 1

#website rank ¡ 100.000 returns 1, website rank ¿ 100.000 returns 0, otherwise returns -1 def web_traffic(url):
    try:
        rank = BeautifulSoup(urllib.request.urlopen(”http://data.alexa.com/data?cli=10dat=surl=-
url).read(), ”xml”).find(REACH”)[’RANK’]
    except TypeError:
        return -1
    rank = int(rank)
    if (rank¡100000):
        return 1
    else:
        return 0
```

```
# PageRank ¡ 0,2 → phishing, otherwise → legitimate
def Page_Rank(url):
    return 1

#webpage indexed by Google returns 1, otherwise returns -1
def Google_Index(url):
    try:
        subDomain, domain, suffix = extract(url)
        a=domain + '.' + suffix
        query = url
        for j in search(query, tld=ço.in", num=5, stop=5, pause=2):
            subDomain, domain, suffix = extract(j)
            b=domain + '.' + suffix
        if(a==b):
            return 1
        else:
            return -1
    except:
        return -1

#:number of links pointing to webpage = 0 returns 1, number of links pointing to webpage¿ 0
#and  2 returns 0, otherwise returns -1 def Links_pointing_to_page (url):
    try:
        opener = urllib.request.urlopen(url).read()
        soup = BeautifulSoup(opener, 'lxml')
        count = 0
```

```
            for link in soup.find_all('a'):
                count += 1
            if(count¿=2):
                return 1
            else:
                return 0
        except:
            return -1

#:host in top 10 phishing IPs or domains returns -1, otherwise returns 1
def Statistical_report (url):
    hostname = url
    h = [(x.start(0), x.end(0)) for x in regex.finditer('https://—http://—www.—https://www.—http://www.',
hostname)]
    z = int(len(h))
    if z != 0:
        y = h[0][1]
        hostname = hostname[y:]
        h = [(x.start(0), x.end(0)) for x in regex.finditer('/', hostname)]
        z = int(len(h))
        if z != 0:
            hostname = hostname[:h[0][0]]
    url_match=regex.search('at/.ua usa/.cc baltazarpresentes/.com/.br pe/.hu esy/.es hol/.es swedd-
y/.com myjino/.ru 96/.lt ow/.ly',url)
    try:
        ip_address = socket.gethostbyname(hostname)
        ip_match=regex.search('146/.112/.61/.108 213/.174/.157/.151 121/.50/.168/.88 192/.185/.217/.116
```

78/.46/.211/.158 181/.174/.165/.13 46/.242/.145/.103 121/.50/.168/.40 83/.125/.22/.219 46/.242/.145/.98 107/.151/.148/.44 107/.151/.148/.107 64/.70/.19/.203 199/.184/.144/.27 107/.151/.148/.108 107/.151/.148/.109 119/.28/.52/.61 54/.83/.43/.69 52/.69/.166/.231 216/.58/.192/.225 118/.184/.25/.8667/.208/.74/.71 23/.253/.126/.58 104/.239/.157/.210 175/.126/.123/.219 141/.8/.224/.221 10/.10/.10/.10 43/.229/.108/.32 103/.232/.215/.140 69/.172/.201/.153 216/.218/.185/.162 54/.225/.104/.146 103/.243/.24/.98 199/.59/.243/.120 31/.170/.160/.61 213/.19/.128/.77 62/.113/.226/.131 208/.100/.26/.234 195/.16/.127/.102 195/.16/.127/.157 34/.196/.13/.28 103/.224/.212/.222 172/.217/.4/.225 54/.72/.9/.51 192/.64/.147/.141 198/.200/.56/.183 23/.253/.164/.103 52/.48/.191/.26 52/.214/.197/.72 87/.98/.255/.18 209/.99/.17/.27 216/.38/.62/.18 104/.130/.124/.96 47/.89/.58/.141 78/.46/.211/.158 54/.86/.225/.156 54/.82/.156/.19 37/.157/.192/.102 204/.11/.56/.48 110/.34/.231/.42',ip_address)

```
        except:         return -1
        if url_match:         return -1
        else:         return 1

#returning scrapped data to calling function in app.py
def main(url):
        check = [[having_IPhaving_IP_Address (url),URLURL_Length(url),Shortining_Service(url),
                having_At_Symbol(url),double_slash_redirecting(url),Prefix_Suffix(url),having_Sub_Domain(url),
                SSLfinal_State(url),Domain_registeration_length(url),Favicon(url),port(url),HTTPS_token(url),
                Request_URL(url),URL_of_Anchor(url),Links_in_tags(url),SFH(url),Submitting_to_email(url),
                Abnormal_URL(url),Redirect(url),on_mouseover(url),RightClick(url),popUpWidnow(url),
                Iframe(url),age_of_domain(url),DNSRecord(url),web_traffic(url),Page_Rank(url),Google_Index(url),
                Links_pointing_to_page(url),Statistical_report(url)]]
        print(check)
        return check
```

app.py

```
import numpy as np
from flask import Flask, request, jsonify, render_template
import pickle
import pandas as pd
import feature
import sqlite3
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

app = Flask(__name__)
model = pickle.load(open('Phising_website.pkl','rb'))

ds= pd.read_csv("data/dataset_website.csv")
x=ds.iloc[:,1:31].values
y=ds.iloc[:,1].values
print(x,y)
y=ds.iloc[:,1].values
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)

dt=XGBClassifier()
dt.fit(x_train,y_train)
y_pred6=dt.predict(x_test)
score = round((accuracy_score(y_test,y_pred6) * 100),2)
print(score)
```

```
@app.route('/')
def home():
    return render_template('home.html')

@app.route('/predict')
def predict():
    return render_template('index.html')

@app.route('/y_predict', methods=['POST'])
def y_predict():
    url = request.form['URL']
    checkprediction = feature.main(url)
    prediction = dt.predict(checkprediction)
    print(prediction)
    output=prediction[0]
    if(output==1):
        import webbrowser
        pred='The Result : You are safe!! This is a legitimate Website.'
        webbrowser.open(url, new=2)
        return render_template('index.html',bns=pred, val = score)

    else:
        pred='The Result : You are on the wrong site. Be cautious!'
        return render_template('index.html',ans=pred, val = score)

if __name__ == '__main__':
    app.run(host='127.0.0.1', debug=True)
```

6. Results and Discussion

Figure 6.1 shows the home page. To check with the URL type, click on predict hyperlink on top of home page or the button-let's go at the bottom of the page.

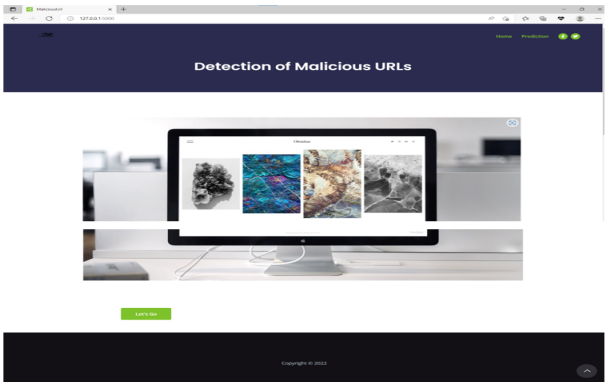

Figure 6.1: HomePage

After clicking on the hyperlink or button, the user will be redirected to the page shown in figure 6.2. In this page the user can check the type by entering the URL in the input field.

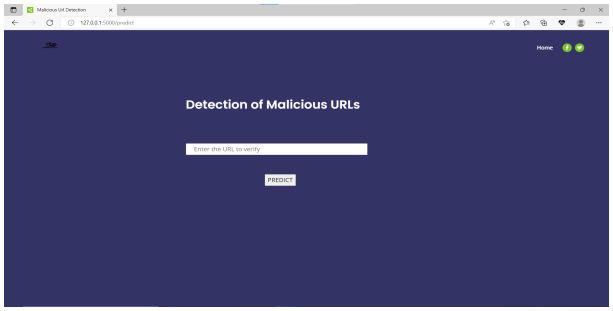

Figure 6.2: Detection Page

Figure 6.3 shows the detection page when the entered URL is a malicious URL, a warning message is shown in red.

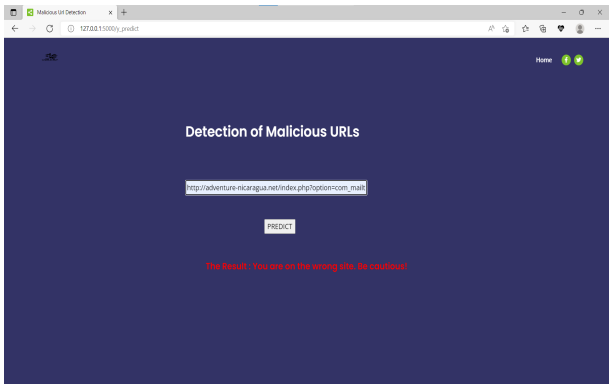

Figure 6.3: Identification of Malicious URL

Figure 6.4 shows detection page when the entered URL is benign. If the entered URL is benign /legitimate, A new tab is opened with the given URL.

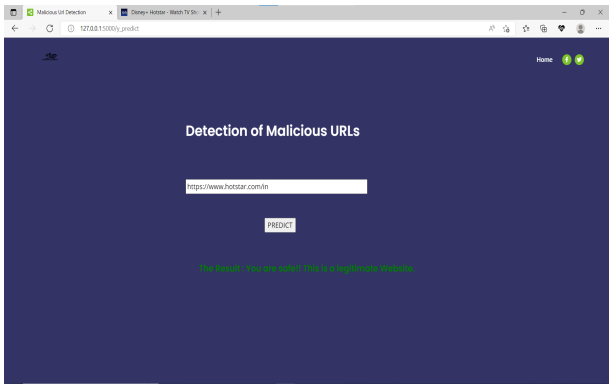

Figure 6.4: Identification of Benign URL

Since the enter URL is benign the redirected page is shown in figure 6.5.

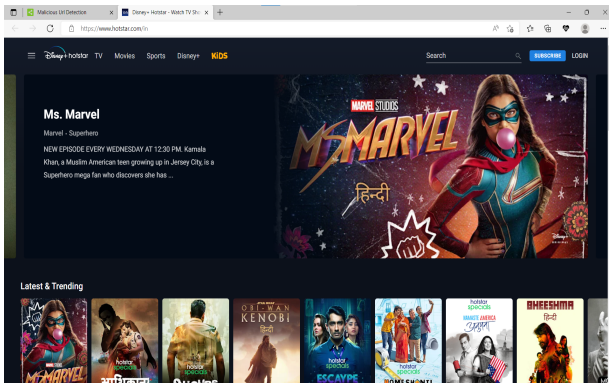

Figure 6.5: Redirection to Benign Site

7. Conclusion and Future Scope

We have proposed a new approach for automatically classifying URLs as either malicious or benign based on machine learning techniques. By analysing the features extracted from an URL our experiment shows an better approach than blacklist methods, which cannot predict the status of previously unseen malicious patterns. Furthermore, we have compared with different classification algorithms like logistic regression, SVM, KNN, random forest, decision tree and found that the Extreme Boosting algorithm has the best performance.

In future work, the detection can be made more specific by analysing the behaviours of malicious URLs. The precision may be improved to 99.99% by perfecting the malicious key words and extracting more features. We would also like to implement chrome extension for easy use of the application. This can be considered as one of the significant directions for future work.

References

1. A. S. Popescu, D. B. Prelipcean and D. T. Gavrilut, .A Study on Techniques for Proactively Identifying Malicious URLs,"2015 17th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2015, pp. 204-211
   DOI: 10.1109/SYNASC.2015.40.

2. M. N. Feroz and S. Mengel, "Phishing URL Detection Using URL Ranking,"2015 IEEE International Congress on Big Data, 2015, pp. 635-638
   DOI: 10.1109/BigDataCongress.2015.97.

3. S. B. Rathod and T. M. Pattewar, .A comparative performance evaluation of content based spam and malicious URL detection in E-mail,"2015 IEEE International Conference on Computer Graphics, Vision and Information Security (CGVIS), 2015, pp. 49-54
   DOI: 10.1109/CGVIS.2015.7449891.

4. R. Kumar, X. Zhang, H. A. Tariq and R. U. Khan, "Malicious URL detection using multi-layer filtering model,"2017 14th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP), 2017, pp. 97-100
   DOI: 10.1109/ICCWAMTIP.2017.8301457.

5. T. Manyumwa, P. F. Chapita, H. Wu and S. Ji, "Towards Fighting Cybercrime: Malicious URL Attack Type Detection using Multiclass Classification,"2020 IEEE International Conference on Big Data (Big Data), 2020, pp. 1813-1822
   doi: 10.1109/BigData50022.2020.9378029.

6. Cui, Baojiang He, Shanshan Yao, Xi Shi, Peilin. (2018). Malicious URL detection with feature extraction based on machine learning. International Journal of High Performance Computing and Networking. 12. 166. 10.1504/IJHPCN.2018.094367.

7. Hugh A. Chipman, Edward I. George, and Robert E. McCulloch. “Bayesian CART Model Search.” Journal of the American Statistical Association, Vol. 93(443), pp 935–948, September 1998.

8. Fadi Thabtah Maher Aburrous, M.A.Hossain, Keshav Dahal. “Intelligent phishing detection system for ebanking using fuzzy data mining.” Expert Systems with Applications, Vol. 37(12), pp 7913–7921, Dec 2010.

9. Dan Steinberg and Phillip Colla. “CART: Classification and Regression Trees.” The Top Ten Algorithms in Data Mining, pp 179–201, 2009.

10. Ying Yang and Geoffrey I. Webb. “Discretization for Naive-Bayes learning: managing a discretization bias and variance.” Machine Learning, Vol. 74(1), pp 39–74, Jan 2009

11. Vinayakumar, R., Alazab, M., Soman, K. P., Poornachandran, P., Al-Nemrat, A., Venkatraman, S. (2019). Deep learning approach for intelligent intrusion detection system. IEEE Access, 7, 41525-41550.

12. Alazab, M., Layton, R., Broadhurst, R., Bouhours, B. (2013, November). Malicious spam emails developments and authorship attribution. In Cybercrime and Trustworthy Computing Workshop (CTC), 2013 Fourth (pp. 58-68). IEEE.