PHISHING WEBSITE DETECTION USING DEEP LEARNING

Report submitted to the SASTRA deemed to be university
as the requirement of the course

CSE 300 - MINI PROJECT

Submitted by

NAFIZE AHAMED F
(Reg. No:124003188,B.Tech CSE)
NITHARSHAN V
(Reg. No:124003204,B.Tech CSE)
VISHVAA K
(Reg. No:124003367,B.Tech CSE)

May 2023

SCHOOL OF COMPUTING

THANJAVUR, TAMIL NADU, INDIA - 613401

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 of the UGC Act,1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA
THANJAVUR | KUMBAKONAM | CHENNAI

SCHOOL OF COMPUTING

THANJAVUR - 613401

Bonafide certificate

This is to certify that the report titled **"Phishing website detection using deep learning"** submitted as a requirement for the course, **CSE 300 - MINI PROJECT** for B.Tech. is a bonafide record of the work done by **Nafize Ahamed F(Reg.No:124003188, B.Tech CSE) ,Nitharshan V(Reg.No:124003204, B.Tech CSE), Vishvaa K(Reg.No:124003367, B.Tech CSE)** during the academic year of 2022-2023 in the School of Computing, under my supervision.

Signature of Project Supervisor: B Kannan .

Name with Affiliation: Dr. Kannan Balasubramanian

Date: 11.05.2023

Mini project viva voce held on _____

Examiner 1 **Examiner 2**

ACKNOWLEDGEMENT

We would like to thank our Honorable Chancellor **Prof. R. Sethuraman** for providing us with an opportunity and the necessary infrastructure for carrying out this project as a part of our curriculum.

We would like to thank our Honorable Vice-Chancellor **Dr. S. Vaidhyasubramaniam** and **Dr. S. Swaminathan**, Dean, Planning & Development, for the encouragement and strategic support at every step of our college life.

We extend our sincere thanks to **Dr. R. Chandramouli**, Registrar, SASTRA Deemed to be University for providing the opportunity to pursue this project.

We extend our heartfelt thanks to **Dr. A. Umamakeswari**, Dean, School of Computing, **Dr. S. Gopalakrishnan**, Associate Dean, Department of Computer Application, **Dr. B.Santhi**, Associate Dean, Research, **Dr. V. S. Shankar Sriram**, Associate Dean, Department of Computer Science and Engineering, **Dr. R. Muthaiah**, Associate Dean, Department of Information Technology and Information & Communication Technology

Our guide **Dr.Kannan Balasubramanian**, Department of CSE , School of Computing was the driving force behind this whole idea from the start. His deep insight in the field and invaluable suggestions helped us in making progress throughout our project work. We also thank the project review panel members for their valuable comments and insights which made this project better.

We would like to extend our gratitude to all the teaching and non-teaching faculties of the School of Computing who have either directly or indirectly helped us in the completion of the project.

We gratefully acknowledge all the contributions and encouragement from our family and friends resulting in the successful completion of this project. We thank you all for providing us an opportunity to showcase our skills through a project.

List of Figures

Abbreviations

LSTM	Long Short-Term Memory
GRU	Gated Recurrent Unit
ML	Machine Learning
DL	Deep Learning
AI	Artificial Intelligence

Abstract

Phishing is a social engineering cyber-attack where criminals deceive users to obtain their credentials through a login form that submits the data to a server. The risk of a cyber attack is continually increasing as a result of attackers' creative methods. They could utilize their social engineering abilities to persuade users to visit phishing websites and enter sensitive personal data and the stolen data could then be used against users. They are difficult to spot because hackers can create phishing messages that appear real to users. With the development and applications of machine learning technology, many machine learning-based solutions for detecting phishing have been proposed. This project proposes a deep learning-based framework for detecting phishing websites. We will implement the framework as a browser plug-in capable of determining whether there is a phishing risk in real time when the user visits a web page and gives a warning message. The real-time prediction service combines multiple strategies to improve accuracy, reduce false alarm rates, and reduce calculation time, including white list filtering, blacklist interception, and machine learning prediction. We compared multiple machine-learning models in the Machine Learning prediction module using several datasets. In this project, we evaluated the accuracy of machine learning models to predict phishing websites.

Keywords: Phishing, Browser plug-in, Prediction

Table of contents

CHAPTER 1

SUMMARY OF THE BASE PAPER

Title: A Deep Learning-Based Framework for Phishing Website Detection

Journal Name: IEEE Access

Author: Tang, Lizhen and Mahmoud, Qusay H.

Year: 2022

1.1 INTRODUCTION:

Phishing is a type of cyberattack that tricks people into giving up their login credentials that send the information to the hacker. Because of the innovative techniques used by attackers, the risk of a cyber assault is constantly rising. Attacks like phishing and spear phishing happen to people from time to time,but when all of it is started the answer is not too long ago these attacks emerged between people in a mere decade (10 years). They might use their social engineering skills to get users to enter sensitive personal information on phishing websites, and then use that information against them. Until now, the technologies that can block or prevent phishing attacks effectively are lower in number to the point you can't see them.There are many ways to send phishing links to the target by email,by social media platforms like Discord, Facebook, and Instagram,but where ever it came from it is only accessed through the browser. This Project proposes a machine learning and deep learning-based framework to analyze an URL and give a report on whether it is a phishing link or not. We developed a browser plug-in to retrieve the URL when a client visits a web page.it takes the URL and sends it to the ML model and displays the report to the user.

- The highest accuracy among deep learning algorithms is 99.41% and we achieved it through the RNN-GRU model.
- The highest accuracy among machine learning algorithms is 99.52% and we achieved it through the Logistic Regression model.

1.2 BACKGROUND RESEARCH

Phishing links are commonly spread by e-mails, text messages, and social media platforms.When the URL opened in the browser is a phishing URL, we can detect the risk of the URL to the user with the help of computer network security technology and alert the user about the situation they are in which they are going to lose their personal information.we are

going to parse the URL for lexical features and use them to detect whether it is phishing or not. “Nearly 1.2% of all emails sent are malicious” Phishing email statistics suggested this. In numbers, There are 3.4 billion phishing emails daily.in India the top most trusted pro. We will be dividing the related work into two subdivisions: (1) deep learning-based methods for detecting phishing URLs and (2) frameworks with prototype implementations.

1.3 PROPOSED METHOD

Train and test the machine learning model with datasets where the benign URL is labeled as ‘0’ and the malicious URL is labeled as ‘1’. Save the model in a file. Create a web framework to handle web requests and responses and bind the saved model to this web framework. Deploy the whole application in the cloud. A web browser plugin is created to get the current tab URL of the user and send it to the cloud application. If the return value was malicious, show a popup message or block the page. If any misprediction occurs by the model, the users can report it through the report page.

1.4 PLAN ARCHITECTURE

One possible architecture for a machine learning model deployed in the cloud and accessed through a browser plugin is as follows. The architecture consists of three main components: the cloud service, the browser plugin, and the machine learning model. The machine learning model is the core of the system that performs the desired task, such as classification, regression, or generation. The model is hosted on a cloud platform that provides scalability, security, and reliability. The browser plugin is a lightweight extension communicating with the model via an API. The plugin can send requests to the model with the user's input data and receive predictions or recommendations from the model. The plugin can also display the results in a user-friendly way and allow the user to provide feedback on the model. This architecture enables the user to leverage the power of machine learning without installing any software or hardware on their device.We must manually check the reported URLs because there is a possibility of false reports. If we directly add them to datasets, then it will affect the function of our model.This architecture may face technical challenges such as latency, reliability, and performance issues, depending on the quality of the network and the model.

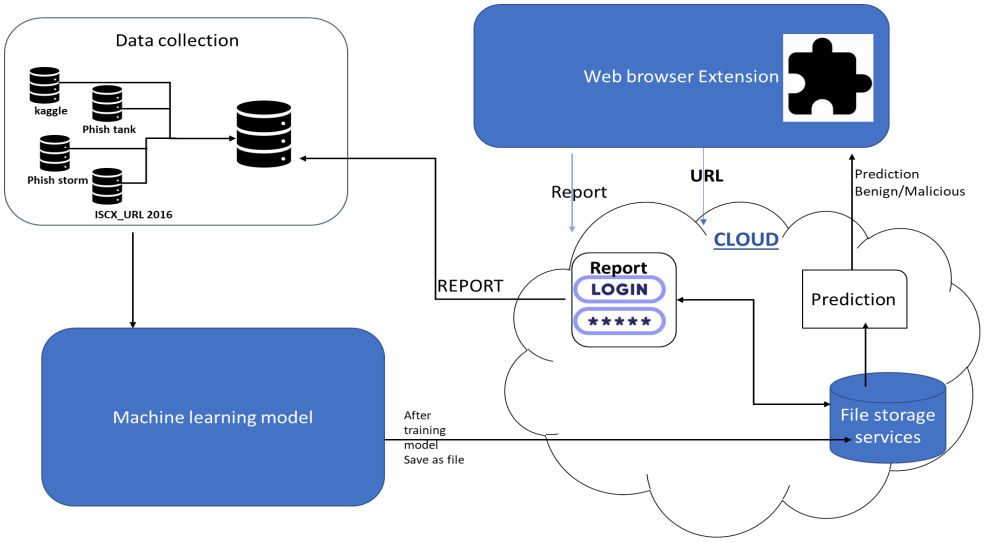


Fig 1.1 Architectural Design

1.5 DATA COLLECTION:

One of the best aspects of developing a machine learning model is choosing an appropriate dataset. A dataset collects data points representing the problem domain and the desired outcomes. Choosing a suitable dataset can significantly impact the performance and generalization of the machine learning model. A poor dataset can lead to overfitting, underfitting, bias, or variance issues. Therefore, it is essential to carefully select and pre-process the dataset before applying any machine learning algorithms or techniques.

For our problem statement, the dataset contains a list of URLs and the status of the URL which may be benign(legitimate) or malicious. The benign URLs may be marked or labeled as '0' and the malicious URL is marked '1'. therefore, the dataset is classified binarily.

URL(https://domainname/path)	label(0/1)

The source of the dataset is also important because the data have to be trustworthy. our dataset is given by the following sources:

1. Kaggle
2. Phish tank
3. Phish storm
4. ISCX-URL 2016

Structure of raw datasets

	A	B	C	D
1		url	label	result
2	0	https://wv	benign	0

Fig 1.2 Kaggle Dataset

	A	B	C	D	E	F	G	H
1	phish_id	url	phish_deta	submissior	verified	verificatio	online	target
2	8051731	https://ac	http://ww	2023-02-2	yes	2023-02-2	yes	Other

Fig 1.3 Phish Tank

ISCX-URL 2016

	A
1	http://v2.email-marketing.adminsimple.com
2	http://bid.openx.net/json?amp;amp;amp;an

Fig 1.4 ISCX-URL 2016

As you can see the URL from different sources have different column structures

We can combine it into one data set with the help of Microsoft Excel by importing data from other datasets. the unnecessary columns will be removed from the dataset to produce the result.

1	url	result
2	https://www.google.com	0

Fig 1.5 Resultant Dataset

Now we can pre-process the result data set.

Pre-processing the dataset:

- Handling missing values in the dataset.
- Handling unbalanced datasets.
- Merge all the datasets.
- Feature Extraction

1.6 MACHINE LEARNING

The main responsibility of the ML module is to train and test the models. The ML module is mainly responsible for the prediction purpose. In this approach, user feedback will be used to update the training model's data on a regular basis. Thus we have developed machine learning models like Logistic Regression, Random Forest, and deep learning models like RNN-GRU and LSTM and we have deployed one of the models with highest accuracy for prediction. To extract features from URLs, we employ natural language processing (NLP) technology. The tokenization procedure in the ml and dl models breaks down a URL string into a list of characters (Character-level tokens). The system will save the model to the given directory once model training is finished.

1.7 WEB BROWSER EXTENSION

We have developed a Chrome browser extension. The extension is on the client side where the URL will be fetched from the browser and sent to the cloud. The return response will be compared and if the return value was malicious then it will show a popup message to the user. Otherwise, nothing will happen. whenever an extension is created a unique id will be given by the browser to that extension. This ID will have to be added inside the web framework to get permission.

The browser plugin is composed of four significant files,

1. Manifest.json
2. Popup HTML
3. Background scripts
4. Content scripts

Manifest.json

A manifest.json file is a JSON-formatted file that provides important information about a web application or an extension. It specifies the name, version, icons, permissions, content scripts, background scripts, and other details of the application or extension. The manifest.json file must be placed in the root directory of the application or extension and must follow a specific schema. The manifest.json file allows the browser to recognize and load the application or extension correctly.

Popup HTML

It is an HTML file and It is the access point of the application where the user and the extension communicate. It will display the details about the URL path, port, host,ip, and country. The popup of an extension is a small window that appears when you click on the

extension in your toolbar. It allows the user to access the features like the report page. The extension's popup is designed to be simple and intuitive, so you can quickly and easily manage your extension

Background scripts

Background scripts are JavaScript files that run in the background of an extension. They can perform tasks such as listening for browser events, modifying web pages, communicating with other scripts, and accessing browser APIs. Background scripts are registered in the manifest.json file of an extension, and they can be persistent or event-driven. Persistent background scripts run continuously in a separate process, while event-driven background scripts are loaded only when needed by an event listener. Background scripts can use most of the browser extension APIs, except for those that require a user interface or a tab context. Whenever a new tab is opened or a new link is opened and the current page is reloaded the background script obtains the current tab URL and sends it to the cloud application then gets the response in JSON format. the response will be compared and if it is malicious, it will show an alert message to the user.

Content scripts

Content scripts of an extension are JavaScript files that run in the context of web pages. They can access and modify the DOM of the web pages that match the specified pattern JSON file. Content scripts can also communicate with the background script of the extension using Chrome. runtime API. Content scripts are useful for adding functionality or enhancing the user experience of web pages. Whenever the extension is clicked the content script will obtain the current tab URL and send it to the cloud application and get the response.The response will either be legitimate or malicious. whatever the response is, it will add an image and display the URL and the label of the URL which will show whether it is malicious or not and the image to the popup HTML and show it to the user.

Difference between content script and background scripts.

The main difference between them is that a background script can communicate with other parts of the extension, such as popups, options pages, or other background scripts, while a content script can only communicate with its parent extension through messages. A background script can also listen to events that occur in the browser, such as clicks, tabs, or navigation, while a content script can only react to events that occur on its web page.

1.8 CLOUD APPLICATION

The whole web framework along with the ml model deployed in the cloud application. So it can be accessed through the internet. When a false alarm occurs in the prediction service, the user can take the initiative to report the current falsely detected URL. We have developed a website and managed an SQL database at the backend to receive these reports.

CHAPTER 2

MERITS AND DEMERITS OF THE BASE PAPER

2.1 RELATED WORK

This paper addresses several related studies in phishing URL detection, including machine learning (ML) and deep learning (DL) techniques. Phishing URL detection frequently employs machine learning (ML) techniques like random forest (RF) and Logistic Regression Models. These strategies can be used with ensemble strategies to build reliable detection classifiers. It is recommended to handle large datasets with DL techniques. The paper highlights the potential of DL and ML algorithms for identifying phishing URLs and sets a standard for further investigation in this field. Furthermore, the study stresses the importance of data pre-processing in the ML process.

2.2 MERITS OF THIS BASE PAPER

- This base paper provides a clear architecture plan of the solution which is helpful for a better understanding of the project
- This base paper has used datasets from different sources
- Detection of phishing attacks through browser extension
- A Report is provided for the users for decreasing the misprediction of ML algorithm
- Information about the URL is provided in the web browser extension

2.3 DEMERITS OF THIS BASE PAPER

- Preprocessing Techniques has not been defined clearly
- Dataset references have not been updated
- Feature extraction has not been discussed elaborately

CHAPTER 3

CODE

Logistic Regression :

```
#import packages
import pandas as pd
import numpy as np
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
#Preprocessing Data
phishing_data = pd.read_csv('/content/urldata.csv')
p1 = pd.read_csv('/content/phishing_dataset.csv')
p2 = pd.read_csv('/content/verified_online.csv')
p3 = pd.read_csv('/content/legitimate url-iscx-url 2016.csv')
p4 = pd.read_csv('/content/urlset.csv')#,encoding='ISO-8859-9')
unnamed_cols  =  phishing_data.columns.str.contains('Unnamed')
phish_data = phishing_data.drop(phishing_data[phishing_data.columns[unnamed_cols]],
axis=1)
phish_data = phish_data.drop(columns='label',axis=1)
p1.columns = ["url"]
p1['result'] = 1
p2=p2.drop(columns=['phish_id','phish_detail_url','submission_time','verified','verification_ti
me','online','target'],axis=1)
p2['result'] = 1
p3.columns = ["url"]
p3['result'] = 0
p4.isnull().sum()
p4 = p4.dropna(how='any')
p4=p4.drop(columns=['ranking','mld_res','mld.ps_res','card_rem','ratio_Rrem','ratio_Arem','ja
ccard_RR','jaccard_RA','jaccard_AR','jaccard_AA','jaccard_ARrd','jaccard_ARrem'],axis=1)
```

```
p4.rename(columns={"label": "result","domain":"url"}, inplace=True)
p4['result'] = p4['result'].astype(int)
p4 = p4.loc[p4["result"] == 1 ]
merged_dataset = pd.concat([phish_data,p1,p2,p3,p4])
# Training And Testing
X = merged_dataset.drop(columns=['result'],axis=1).values
Y = merged_dataset['result'].values
vectorizer = TfidfVectorizer()
vectorizer.fit(X.ravel())
X = vectorizer.transform(X.ravel())
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.2,random_state=2)
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(solver='lbfgs', max_iter=1000).fit(X_train,Y_train)
from sklearn.metrics import accuracy_score
X_train_prediction = classifier.predict(X_train)
training_data_accuracy = accuracy_score(Y_train,X_train_prediction)
print(training_data_accuracy)
Output : 0.9902508828066239

X_test_prediction = classifier.predict(X_test)
test_data_accuracy = accuracy_score(Y_test,X_test_prediction)
print(test_data_accuracy)
Output : 0.9846130311181367
#Predictive System
input_url = input("Enter the url : ")
df1 = np.array(input_url)
print(df1)
#vectorizer1 = TfidfVectorizer()
#vectorizer1.fit(df1.ravel())
vect_url = vectorizer.transform(df1.ravel())
X_pred = vect_url
prediction = classifier.predict(X_pred)
print(prediction)
if(prediction == 1):
  print("The url is malicious")
else:
  print("The url is legitimate")
```

Random Forest :

```
import pandas as pd
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.preprocessing import MinMaxScaler

# Load data from CSV file
data = pd.read_csv('/content/clean.csv')
del data['Unnamed: 0']
X = data.iloc[:, :-1]
# Apply min-max scaling to X
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
# Select target variable
y = data.iloc[:, -1]
# Feature selection with SelectKBest
selector = SelectKBest(chi2, k=20)
selector.fit(X_scaled, y)
# Transform your data to select the top k features
X_new = selector.transform(X_scaled)
# Get the selected column names
selected_cols = X.columns[selector.get_support()]
import time
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
import pandas as pd
# Split the data into training and testing sets
X_train,X_test,Y_train,Y_test = train_test_split(X,y,test_size=0.2,random_state=2)

model = RandomForestClassifier(n_estimators=300)
start_time = time.time()
model.fit(X_train.values, Y_train.values)
from sklearn.metrics import accuracy_score
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(Y_train,X_train_prediction)
print(training_data_accuracy)
Output : 0.9982923307602599
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(Y_test,X_test_prediction)
print(test_data_accuracy)
```

Output : 0.9746101559376249

```
import re
from urllib.parse import urlparse
from tld import get_tld
def having_ip_address(urls):
    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}', urls)  # Ipv6
    if match:
        return 1
    else:
        return 0
def count_random_words(urls):
    li1 = re.findall("[A-Za-z]+",urls)
    return len(li1)

def avg_len_random(urls):
    sum = 0
    li2 = re.findall("[A-Za-z]+",urls)
    for u in li2:
        sum = sum + len(u)
    return (sum/len(li2))
def semicolon_in_URL(urls):
  count = urls.count(';')
  return count
def AND_in_URL(urls):
  count = urls.count('&')
  return count
def URL_length(urls):
  return len(str(urls))
def uppercase_count(url):
    letters1 = 0
```

```
    for i in url:
        if i>='A' and i<='Z':
            letters1 = letters1 + 1
    return letters1
def lowercase_ratio(url):
    url_len = len(str(url))
    letters2 = 0
    for i in url:
        if i>='a' and i<='z':
            letters2 = letters2 + 1
    return (letters2/url_len)
def spl_char_count(url):
    url_len = len(str(url))
    chars = 0
    for i in url:
        if i=='!' or i=='@' or i=='#' or i=='%' or i=='^' or i=='&' or i=='*' or i=='(' or i==')':
            chars = chars + 1
    return chars
def number_ratio(url):
    url_len = len(str(url))
    nums = 0
    for i in url:
        if i>='0' and i<='9':
            nums = nums + 1
    return (nums/url_len)
def alphabet_ratio(url):
    url_len = len(str(url))
    letters = 0
    for i in url:
        if i.isalpha():
            letters = letters + 1
    return (letters/url_len)
def has_server_in_string(url):
    if 'server' in url.lower():
        return 1
    else:
        return 0
def has_login_in_string(url):
    if 'login' in url.lower():
```

```
        return 1
    else:
        return 0
def has_client_in_string(url):
    if 'client' in url.lower():
        return 1
    else:
        return 0
def has_admin_in_string(url):
    if 'admin' in url.lower():
        return 1
    else:
        return 0
def no_of_dir(url):
    urldir = urlparse(url).path
    return urldir.count('/')
def domain_len(url):
    domain = urlparse(str(url)).netloc
    return len(domain)
def dots_in_domain(url):
    count1 = url.count('.')
    return count1
def number_subdomain(url):
    count = 0
    domain = urlparse(url).netloc
    dom = '.'.join(domain.split('.')[:-2])
    for i in dom:
        if i=='.':
            count = count+1
    return (count-1)
def hyphens_in_domain(url):
    count1 = url.count('-')
    return count1
def letter_count(url):
    letters = 0
    for i in url:
        if i.isalpha():
            letters = letters + 1
    return letters
```

```
def fd_length(url):
    urlpath= urlparse(url).path
    try:
        return len(urlpath.split('/')[1])
    except:
        return 0
def lowercase_count(url):
    letters2 = 0
    for i in url:
        if i>='a' and i<='z':
            letters2 = letters2 + 1
    return letters2
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\.g
d|'
                    'tr\.im|link\.zip\.net',
                    url)
    if match:
        return -1
    else:
        return 1
def tld_length(tld):
    try:
        return len(tld)
    except:
        return -1
def uppercase_ratio(url):
```

```
    url_len = len(str(url))
    letters1 = 0
    for i in url:
        if i>='A' and i<='Z':
            letters1 = letters1 + 1
    return (letters1/url_len)
```

LSTM.pynb

```
from google.colab import drive
 drive.mount('/content/drive')
```

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

```
 import numpy as np
 import pandas as pd

 import matplotlib.pyplot as plt
 import seaborn as sns

 import os

urldata = pd.read_csv("/content/drive/MyDrive/x/z.csv")

urldata.head()
```

	url	result
0	https://www.google.com	0
1	https://www.youtube.com	0
2	https://www.facebook.com	0
3	https://www.baidu.com	0
4	https://www.wikipedia.org	0

```
urldata.head()
```

	url	result
0	https://www.google.com	0
1	https://www.youtube.com	0
2	https://www.facebook.com	0

```
3        https://www.baidu.com          0
4  https://www.wikipedia.org            0
```

```
urldata.shape
```

```
(525433, 2)
```

```
urldata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
 RangeIndex: 525433 entries, 0 to 525432
 Data columns (total 2 columns):
  #   Column  Non-Null Count   Dtype
 ---  ------  --------------   -----
  0   url           525433 non-null  object
  1   result  525433 non-null  int64
 dtypes: int64(1), object(1)
 memory usage: 8.0+ MB
```

Checking Missing Values

```
urldata.isnull().sum()
```

```
url       0
 result  0
 dtype: int64
```

No missing values in any column.

1. DATA PREPROCESSING

1.1 Length Features

```
!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)
```

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

```
from tld import get_tld
import os.path

#Length of URL
urldata['url_length'] = urldata['url'].apply(lambda i: len(str(i)))

#Hostname Length
urldata['hostname_length'] = urldata['url'].apply(lambda i: len(urlparse(i).netloc))

#Path Length
urldata['path_length'] = urldata['url'].apply(lambda i: len(urlparse(i).path))

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

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

#Length of Top Level Domain
urldata['tld'] = urldata['url'].apply(lambda i: get_tld(i,fail_silently=True))
def tld_length(tld):
        try:
        return len(tld)
        except:
        return -1

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

urldata.head()
```

	url	result	url_length	hostname_length	\
0	https://www.google.com	0	22	14	
1	https://www.youtube.com	0	23	15	
2	https://www.facebook.com	0	24	16	
3	https://www.baidu.com	0	21	13	
4	https://www.wikipedia.org	0	25	17	

	path_length	fd_length	tld	tld_length
0	0	0	com	3
1	0	0	com	3
2	0	0	com	3
3	0	0	com	3
4	0	0	org	3

```
urldata = urldata.drop("tld",1)
```

```
<ipython-input-18-4fbc22699343>:1: FutureWarning: In a future version of pandas all arguments of DataFrame.drop except for the argument 'labels' will be keyword-only.
  urldata = urldata.drop("tld",1)
```

```
urldata.head()
```

	url	result	url_length	hostname_length	\
0	https://www.google.com	0	22	14	
1	https://www.youtube.com	0	23	15	
2	https://www.facebook.com	0	24	16	
3	https://www.baidu.com	0	21	13	
4	https://www.wikipedia.org	0	25	17	

	path_length	fd_length	tld_length
0	0	0	3
1	0	0	3
2	0	0	3
3	0	0	3
4	0	0	3

1.2 Count Features

```
urldata['count-'] = urldata['url'].apply(lambda i: i.count('-'))
```

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

```
urldata['count?'] = urldata['url'].apply(lambda i: i.count('?'))
```

```
urldata['count%'] = urldata['url'].apply(lambda i: i.count('%'))
```

```
urldata['count.'] = urldata['url'].apply(lambda i: i.count('.'))
```

```
urldata['count='] = urldata['url'].apply(lambda i: i.count('='))

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

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

urldata['count-www'] = urldata['url'].apply(lambda i: i.count('www'))

def digit_count(url):
        digits = 0
        for i in url:
        if i.isnumeric():
        digits = digits + 1
        return digits
 urldata['count-digits']= urldata['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
 urldata['count-letters']= urldata['url'].apply(lambda i: letter_count(i))

def no_of_dir(url):
        urldir = urlparse(url).path
        return urldir.count('/')
 urldata['count_dir'] = urldata['url'].apply(lambda i: no_of_dir(i))
```

Data after extracting Count Features

```
urldata.head()
```

	url	result	url_length	hostname_length	\
0	https://www.google.com	0	22	14	
1	https://www.youtube.com	0	23	15	
2	https://www.facebook.com	0	24	16	
3	https://www.baidu.com	0	21	13	
4	https://www.wikipedia.org	0	25	17	

path_length	fd_length	tld_length	count-	count@	count?	count%	count.	\

0	0	0	3	0	0	0	0	2
1	0	0	3	0	0	0	0	2
2	0	0	3	0	0	0	0	2
3	0	0	3	0	0	0	0	2
4	0	0	3	0	0	0	0	2

	count=	count-http	count-https	count-www	count-digits	count-letters	\
0	0	1	1	1	0	17	
1	0	1	1	1	0	18	
2	0	1	1	1	0	19	
3	0	1	1	1	0	16	
4	0	1	1	1	0	20	

	count_dir
0	0
1	0
2	0
3	0
4	0

1.3 Binary Features

```
import re
 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 1
urldata['use_of_ip'] = urldata['url'].apply(lambda i: having_ip_address(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\.g
d|'
                'tr\.im|link\.zip\.net',
                url)
        if match:
        return -1
        else:
        return 1
urldata['short_url'] = urldata['url'].apply(lambda i: shortening_service(i))
```

Data after extracting Binary Features

```
urldata.head()

                url  result  url_length  hostname_length  \
0       https://www.google.com       0       22               14
1       https://www.youtube.com      0       23               15
2   https://www.facebook.com         0       24               16
3       https://www.baidu.com        0       21               13
4  https://www.wikipedia.org         0       25               17

   path_length  fd_length  tld_length  count-  count@  count?  ...  count.  \
0        0         0         3         0       0          0  ...     2
1        0         0         3         0       0          0  ...     2
```

```
2       0       0       3       0       0          0  ...      2
3       0       0       3       0       0          0  ...       2
4       0       0       3       0       0          0  ...      2

   count=  count-http  count-https  count-www  count-digits  count-letters  \
0       0           1           1          1             0             17
1       0           1           1          1             0             18
2       0           1           1          1             0             19
3       0           1           1          1             0             16
4       0           1           1          1             0             20

   count_dir  use_of_ip  short_url
0          0          1          1
1          0          1          1
2          0          1          1
3          0          1          1
4          0          1          1

[5 rows x 21 columns]


x = urldata[['hostname_length',
        'path_length', 'fd_length', 'tld_length', 'count-', 'count@', 'count?',
        'count%', 'count.', 'count=', 'count-http','count-https', 'count-www', 'count-digits',
        'count-letters', 'count_dir', 'use_of_ip']]



y = urldata['result']


x_train, x_test, y_train, y_test = train_test_split(x, y, train_size=0.2, random_state=42)

import numpy as np
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout, GRU, Bidirectional
```

```
from keras.optimizers import SGD
import math
from sklearn.metrics import mean_squared_error
import tensorflow as tf
```

```
x_train.shape
```

```
(105086, 17)
```

```
x_train.shape[1],1
```

```
(17, 1)
```

```
from keras import backend as K
```

```
def recall_m(y_true, y_pred):
        true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
        possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
        recall = true_positives / (possible_positives + K.epsilon())
        return recall

def precision_m(y_true, y_pred):
        true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
        predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
        precision = true_positives / (predicted_positives + K.epsilon())
        return precision

def f1_m(y_true, y_pred):
        precision = precision_m(y_true, y_pred)
        recall = recall_m(y_true, y_pred)
        return 2*((precision*recall)/(precision+recall+K.epsilon()))
```

```
regressor = Sequential()
# First LSTM layer with Dropout regularisation
regressor.add(LSTM(units=50, return_sequences=True, input_shape=(x_train.shape[1],1)))
regressor.add(Dropout(0.2))
# Second LSTM layer
regressor.add(LSTM(units=50, return_sequences=True))
regressor.add(Dropout(0.2))
# Third LSTM layer
```

```
regressor.add(LSTM(units=50, return_sequences=True))
regressor.add(Dropout(0.2))
#fourth layer
regressor.add(LSTM(units=50, return_sequences=True))
regressor.add(Dropout(0.2))
# Fifth LSTM layer
regressor.add(LSTM(units=50))
regressor.add(Dropout(0.2))
# The output layer
regressor.add(Dense(units=1))

# Compiling the LSTM
regressor.compile(optimizer='adam',loss='mean_squared_error',
metrics=['acc',f1_m,precision_m, recall_m])



regressor.fit(x_train,y_train,epochs=10,batch_size=32)

Epoch 1/10
3284/3284 [==============================] - 201s 58ms/step - loss: 0.0353 - acc:
0.9549 - f1_m: 0.8929 - precision_m: 0.9150 - recall_m: 0.8823
Epoch 2/10
3284/3284 [==============================] - 181s 55ms/step - loss: 0.0088 - acc:
0.9924 - f1_m: 0.9882 - precision_m: 0.9948 - recall_m: 0.9829
Epoch 3/10
3284/3284 [==============================] - 178s 54ms/step - loss: 0.0082 - acc:
0.9930 - f1_m: 0.9891 - precision_m: 0.9953 - recall_m: 0.9841
Epoch 4/10
3284/3284 [==============================] - 178s 54ms/step - loss: 0.0080 - acc:
0.9931 - f1_m: 0.9894 - precision_m: 0.9950 - recall_m: 0.9848
Epoch 5/10
3284/3284 [==============================] - 178s 54ms/step - loss: 0.0078 - acc:
0.9933 - f1_m: 0.9896 - precision_m: 0.9955 - recall_m: 0.9848
Epoch 6/10
3284/3284 [==============================] - 178s 54ms/step - loss: 0.0077 - acc:
0.9934 - f1_m: 0.9898 - precision_m: 0.9953 - recall_m: 0.9853
Epoch 7/10
3284/3284 [==============================] - 179s 54ms/step - loss: 0.0076 - acc:
```

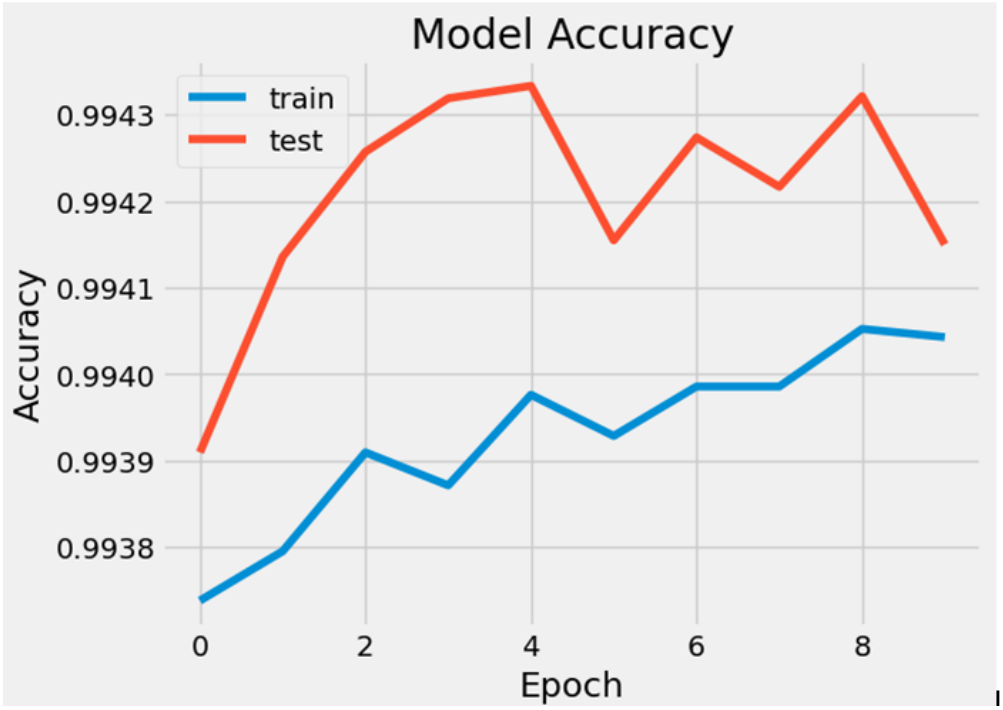
```
0.9934 - f1_m: 0.9899 - precision_m: 0.9952 - recall_m: 0.9855
 Epoch 8/10
 3284/3284 [==============================] - 177s 54ms/step - loss: 0.0075 - acc:
0.9935 - f1_m: 0.9896 - precision_m: 0.9954 - recall_m: 0.9849
 Epoch 9/10
 3284/3284 [==============================] - 178s 54ms/step - loss: 0.0074 - acc:
0.9937 - f1_m: 0.9903 - precision_m: 0.9956 - recall_m: 0.9859
 Epoch 10/10
 3284/3284 [==============================] - 177s 54ms/step - loss: 0.0073 - acc:
0.9937 - f1_m: 0.9902 - precision_m: 0.9955 - recall_m: 0.9859

<keras.callbacks.History at 0x7f531e0bb2b0>

# Calculate accuracy
 accuracy = regressor.evaluate(x_test, y_test)[1]

13136/13136 [==============================] - 216s 16ms/step - loss: 0.0057 -
acc: 0.9941 - f1_m: 0.9908 - precision_m: 0.9957 - recall_m: 0.9869

print(accuracy)

0.9941096305847168

regressor.save('/content/drive/MyDrive/urldata.csv (1)')

WARNING:absl:Found untraced functions such as _update_step_xla,
lstm_cell_5_layer_call_fn, lstm_cell_5_layer_call_and_return_conditional_losses,
lstm_cell_6_layer_call_fn, lstm_cell_6_layer_call_and_return_conditional_losses while
saving (showing 5 of 11). These functions will not be directly callable after loading.

reconstructed_model = keras.models.load_model("/content/drive/MyDrive/urldata.csv (1)",
custom_objects={'f1_m': f1_m, 'precision_m': precision_m, 'recall_m': recall_m})

import numpy as np
 import tensorflow as tf
 from tensorflow import keras
 reconstructed_model = keras.models.load_model("/content/drive/MyDrive/urldata.csv (1)")

calculation
```

```
tf.keras.utils.plot_model(
        reconstructed_model,
        to_file='model.png',
        show_shapes=False,
        show_dtype=False,
        show_layer_names=True,
        rankdir='TB',
        expand_nested=False,
        dpi=96,
        layer_range=None,
        show_layer_activations=False,
        show_trainable=False
)

import matplotlib.pyplot as plt

 # train the model and save the history
 history = regressor.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=10,
batch_size=32)

 # plot the accuracy curve
 plt.plot(history.history['acc'])
 plt.plot(history.history['val_acc'])
 plt.title('Model Accuracy')
 plt.ylabel('Accuracy')
 plt.xlabel('Epoch')
 plt.legend(['train', 'test'], loc='upper left')
 plt.show()

 # plot the loss curve
 plt.plot(history.history['loss'])
 plt.plot(history.history['val_loss'])
 plt.title('Model Loss')
 plt.ylabel('Loss')
 plt.xlabel('Epoch')
 plt.legend(['train', 'test'], loc='upper left')
 plt.show()
```

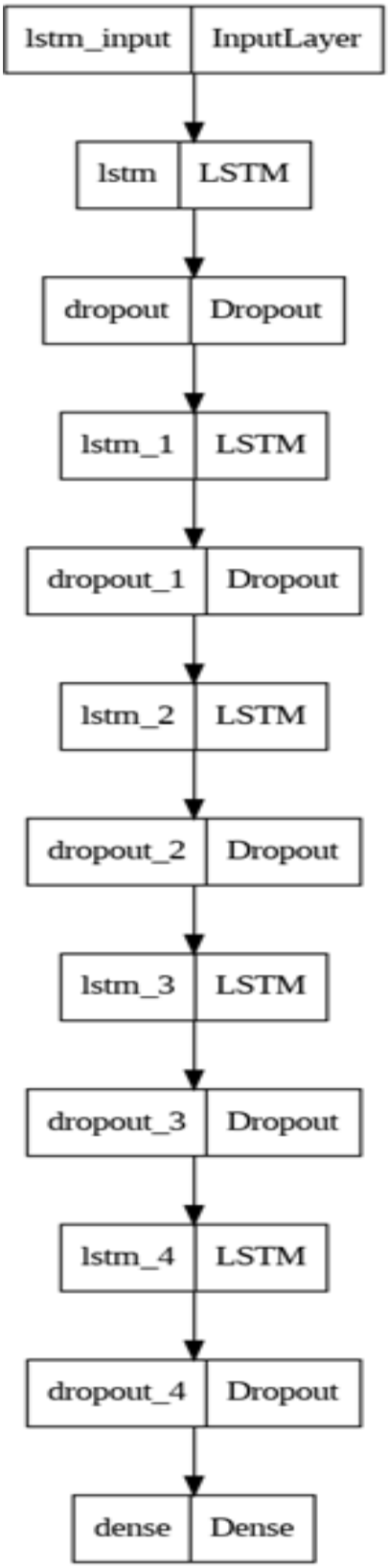
```
Epoch 1/10
 3284/3284 [==============================] - 378s 115ms/step - loss: 0.0073 -
acc: 0.9937 - f1_m: 0.9904 - precision_m: 0.9953 - recall_m: 0.9864 - val_loss: 0.0059 -
val_acc: 0.9939 - val_f1_m: 0.9905 - val_precision_m: 0.9959 - val_recall_m: 0.9862
 Epoch 2/10
 3284/3284 [==============================] - 378s 115ms/step - loss: 0.0072 -
acc: 0.9938 - f1_m: 0.9903 - precision_m: 0.9955 - recall_m: 0.9861 - val_loss: 0.0057 -
val_acc: 0.9941 - val_f1_m: 0.9909 - val_precision_m: 0.9959 - val_recall_m: 0.9868
 Epoch 3/10
 3284/3284 [==============================] - 379s 115ms/step - loss: 0.0071 -
acc: 0.9939 - f1_m: 0.9903 - precision_m: 0.9954 - recall_m: 0.9862 - val_loss: 0.0056 -
val_acc: 0.9943 - val_f1_m: 0.9911 - val_precision_m: 0.9960 - val_recall_m: 0.9871
 Epoch 4/10
 3284/3284 [==============================] - 377s 115ms/step - loss: 0.0071 -
acc: 0.9939 - f1_m: 0.9905 - precision_m: 0.9955 - recall_m: 0.9865 - val_loss: 0.0055 -
val_acc: 0.9943 - val_f1_m: 0.9912 - val_precision_m: 0.9957 - val_recall_m: 0.9875
 Epoch 5/10
 3284/3284 [==============================] - 377s 115ms/step - loss: 0.0071 -
acc: 0.9940 - f1_m: 0.9905 - precision_m: 0.9956 - recall_m: 0.9864 - val_loss: 0.0055 -
val_acc: 0.9943 - val_f1_m: 0.9912 - val_precision_m: 0.9959 - val_recall_m: 0.9874
 Epoch 6/10
 3284/3284 [==============================] - 378s 115ms/step - loss: 0.0070 -
acc: 0.9939 - f1_m: 0.9905 - precision_m: 0.9955 - recall_m: 0.9865 - val_loss: 0.0057 -
val_acc: 0.9942 - val_f1_m: 0.9909 - val_precision_m: 0.9950 - val_recall_m: 0.9878
 Epoch 7/10
 3284/3284 [==============================] - 386s 117ms/step - loss: 0.0070 -
acc: 0.9940 - f1_m: 0.9907 - precision_m: 0.9954 - recall_m: 0.9869 - val_loss: 0.0057 -
val_acc: 0.9943 - val_f1_m: 0.9911 - val_precision_m: 0.9958 - val_recall_m: 0.9873
 Epoch 8/10
 3284/3284 [==============================] - 377s 115ms/step - loss: 0.0070 -
acc: 0.9940 - f1_m: 0.9907 - precision_m: 0.9957 - recall_m: 0.9866 - val_loss: 0.0056 -
val_acc: 0.9942 - val_f1_m: 0.9910 - val_precision_m: 0.9958 - val_recall_m: 0.9872
 Epoch 9/10
 3284/3284 [==============================] - 380s 116ms/step - loss: 0.0069 -
acc: 0.9941 - f1_m: 0.9907 - precision_m: 0.9956 - recall_m: 0.9868 - val_loss: 0.0055 -
val_acc: 0.9943 - val_f1_m: 0.9912 - val_precision_m: 0.9958 - val_recall_m: 0.9875
 Epoch 10/10
 3284/3284 [==============================] - 377s 115ms/step - loss: 0.0069 -
```

```
acc: 0.9940 - f1_m: 0.9906 - precision_m: 0.9956 - recall_m: 0.9867 - val_loss: 0.0058 -
val_acc: 0.9942 - val_f1_m: 0.9909 - val_precision_m: 0.9960 - val_recall_m: 0.9868

reconstructed_model.evaluate(
        x=x_test,
        y=y_test,
        batch_size=32,
        verbose="auto",
        sample_weight=None,
        steps=None,
        callbacks=None,
        max_queue_size=10,
        workers=1,
        use_multiprocessing=False,
        return_dict=False
 )

reconstructed_model.fit(x_test,y_test)

url=input("url  :")
 length_url=len(url)
 hostname_length=len(urlparse(url).netloc)
 path_length=len(urlparse(url).path)
 #print(path_length)
 fd_len=fd_length(url)
 tld_len= tld_length(url)
 minus=url.count('-')
 #print(minus)
 at=url.count('@')
 qm=url.count('?')
 mod=url.count('%')
 dot=url.count('.')
 eq=url.count('=')
 htp=url.count('http')
 htps=url.count('https')
 ww=url.count('www')
 numdigit= digit_count(url)
 #print(numdigit)
 numletters=letter_count(url)
```

```
numdir= no_of_dir(url)
ipuse= having_ip_address(url)
shorturl=  shortening_service(url)


urlfeat = [hostname_length,
        path_length, fd_len, tld_len, minus, at, qm,
        mod, dot, eq, htp,https, ww, numdigit,
        numletters, numdir, ipuse]
print(urlfeat)

urlfeat=[urlfeat]

url
:https://www.youtube.com/watch?v=KRc3vcWpuSA&ab_channel=OriginalSNLCompilation
s
 [15, 6, 5, 78, 0, 0, 1, 0, 2, 2, 1, 1, 1, 1, 66, 1, 1]

predict=reconstructed_model.predict(urlfeat)
 p=predict[0]
 q=p[0]
 print(q)

1/1 [==============================] - 0s 42ms/step
 -0.00017100573

if(q>0.9):
   print("malicious")
 else:
   print("benign")


Benign
```


Model Accuracy
train
test
0.9943
0.9942
0.9941
0.9940
0.9939
0.9938
Accuracy
0
2
4
6
8
Epoch


Model Loss
train
test
0.00725
0.00700
0.00675
0.00650
0.00625
0.00600
0.00575
0.00550
Loss
0
2
4
6
8
Epoch


lstm_input InputLayer
lstm LSTM
dropout Dropout
lstm_1 LSTM
dropout_1 Dropout
lstm_2 LSTM
dropout_2 Dropout
lstm_3 LSTM
dropout_3 Dropout
lstm_4 LSTM
dropout_4 Dropout
dense Dense

GRU

```
from google.colab import drive
 drive.mount('/content/drive')
Mounted at /content/drive

 import numpy as np
 import pandas as pd

 import matplotlib.pyplot as plt
 import seaborn as sns


urldata = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/Updated datasets/z.csv")
urldata.head()
                    url  result
 0        https://www.google.com       0
 1        https://www.youtube.com      0
 2   https://www.facebook.com          0
 3        https://www.baidu.com        0
 4  https://www.wikipedia.org          0

urldata.head()
                    url  result
 0        https://www.google.com       0
 1        https://www.youtube.com      0
 2   https://www.facebook.com          0
 3        https://www.baidu.com        0
 4  https://www.wikipedia.org          0
urldata.shape
(525433, 2)
urldata.info()
<class 'pandas.core.frame.DataFrame'>
 RangeIndex: 525433 entries, 0 to 525432
 Data columns (total 2 columns):
  #   Column  Non-Null Count   Dtype
 ---  ------  --------------   -----
  0   url           525433 non-null  object
  1   result  525433 non-null  int64
```

```
 dtypes: int64(1), object(1)
 memory usage: 8.0+ MB

urldata.isnull().sum()
url       0
 result  0
 dtype: int64
No missing values in any column.

!pip install tld

Looking in indexes: https://pypi.org/simple,
https://us-python.pkg.dev/colab-wheels/public/simple/
 Collecting tld
   Downloading tld-0.13-py2.py3-none-any.whl (263 kB)

     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
     ━━━━ 263.8/263.8 kB 8.7 MB/s eta 0:00:00
#Importing dependencies
 from urllib.parse import urlparse
 from tld import get_tld
 import os.path
#Length of URL
 urldata['url_length'] = urldata['url'].apply(lambda i: len(str(i)))
#Hostname Length
 urldata['hostname_length'] = urldata['url'].apply(lambda i: len(urlparse(i).netloc))
#Path Length
 urldata['path_length'] = urldata['url'].apply(lambda i: len(urlparse(i).path))
#First Directory Length
 def fd_length(url):
          urlpath= urlparse(url).path
          try:
          return len(urlpath.split('/')[1])
          except:
          return 0

 urldata['fd_length'] = urldata['url'].apply(lambda i: fd_length(i))
#Length of Top Level Domain
 urldata['tld'] = urldata['url'].apply(lambda i: get_tld(i,fail_silently=True))
 def tld_length(tld):
```

```
        try:
        return len(tld)
        except:
        return -1

urldata['tld_length'] = urldata['tld'].apply(lambda i: tld_length(i))
urldata.head()
                  url  result  url_length  hostname_length  \
0       https://www.google.com       0          22               14
1       https://www.youtube.com      0          23               15
2   https://www.facebook.com         0          24               16
3       https://www.baidu.com        0          21               13
4  https://www.wikipedia.org         0          25               17

   path_length  fd_length  tld  tld_length
0          0          0  com           3
1            0        0  com           3
2          0          0  com           3
3          0          0  com           3
4          0          0  org           3
urldata = urldata.drop("tld",1)
<ipython-input-18-4fbc22699343>:1: FutureWarning: In a future version of pandas all
arguments of DataFrame.drop except for the argument 'labels' will be keyword-only.
  urldata = urldata.drop("tld",1)
urldata.head()
                  url  result  url_length  hostname_length  \
0       https://www.google.com       0          22               14
1       https://www.youtube.com      0          23               15
2   https://www.facebook.com         0          24               16
3       https://www.baidu.com        0          21               13
4  https://www.wikipedia.org         0          25               17

   path_length  fd_length  tld_length
0          0          0           3
1          0          0           3
2          0          0           3
3          0          0           3
4          0          0           3
```

```
urldata['count-'] = urldata['url'].apply(lambda i: i.count('-'))
urldata['count@'] = urldata['url'].apply(lambda i: i.count('@'))
urldata['count?'] = urldata['url'].apply(lambda i: i.count('?'))
urldata['count%'] = urldata['url'].apply(lambda i: i.count('%'))
urldata['count.'] = urldata['url'].apply(lambda i: i.count('.'))
urldata['count='] = urldata['url'].apply(lambda i: i.count('='))
urldata['count-http'] = urldata['url'].apply(lambda i : i.count('http'))
urldata['count-https'] = urldata['url'].apply(lambda i : i.count('https'))
urldata['count-www'] = urldata['url'].apply(lambda i: i.count('www'))
def digit_count(url):
        digits = 0
        for i in url:
        if i.isnumeric():
                digits = digits + 1
        return digits
 urldata['count-digits']= urldata['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
 urldata['count-letters']= urldata['url'].apply(lambda i: letter_count(i))
def no_of_dir(url):
        urldir = urlparse(url).path
        return urldir.count('/')
 urldata['count_dir'] = urldata['url'].apply(lambda i: no_of_dir(i))
urldata.head()
```

	url	result	url_length	hostname_length	\
0	https://www.google.com	0	22	14	
1	https://www.youtube.com	0	23	15	
2	https://www.facebook.com	0	24	16	
3	https://www.baidu.com	0	21	13	
4	https://www.wikipedia.org	0	25	17	

	path_length	fd_length	tld_length	count-	count@	count?	count%	count.	\
0	0	0	3	0	0	0	0	2	
1	0	0	3	0	0	0	0	2	

```
2       0       0       3       0       0       0       0       2
3       0       0       3       0       0       0       0       2
4       0       0       3       0       0       0       0       2

   count=  count-http  count-https  count-www  count-digits  count-letters  \
0       0          1            1          1             0             17
1       0          1            1          1             0             18
2       0          1            1          1             0             19
3       0          1            1          1             0             16
4       0          1            1          1             0             20

   count_dir
0          0
1          0
2          0
3          0
4          0
```

```
import re
#Use of IP or not in domain
 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:
        # print match.group()
        return -1
        else:
        # print 'No matching pattern found'
        return 1
 urldata['use_of_ip'] = urldata['url'].apply(lambda i: having_ip_address(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\.g
d|'
                'tr\.im|link\.zip\.net',
                url)
        if match:
      return -1
        else:
        return 1
urldata['short_url'] = urldata['url'].apply(lambda i: shortening_service(i))
Data after extracting Binary Features
urldata.head()
                url  result  url_length  hostname_length  \
0       https://www.google.com       0       22              14
1       https://www.youtube.com      0       23              15
2   https://www.facebook.com         0       24              16
3       https://www.baidu.com        0       21              13
4  https://www.wikipedia.org         0       25              17

   path_length  fd_length  tld_length  count-  count@  count?  ...  count.  \
0        0         0         3         0       0          0  ...       2
1        0         0         3         0       0          0  ...       2
2        0         0         3         0       0          0  ...       2
3        0         0         3         0       0          0  ...       2
4        0         0         3         0       0          0  ...     2

   count=  count-http  count-https  count-www  count-digits  count-letters  \
0       0         1          1          1          0             17
1       0         1          1          1          0               18
```

```
2       0       1       1       1       0       19
3       0       1       1       1       0       16
4       0       1       1       1       0       20

   count_dir  use_of_ip  short_url
0       0       1       1
1       0       1       1
2       0       1       1
3       0       1       1
4       0       1       1

[5 rows x 21 columns]
from sklearn.ensemble import RandomForestClassifier
 from sklearn.model_selection import train_test_split

 from sklearn.metrics import confusion_matrix,classification_report,accuracy_score

 from sklearn.tree import DecisionTreeClassifier

 from sklearn.linear_model import LogisticRegression


#Predictor Variables
 x = urldata[['hostname_length',
          'path_length', 'fd_length', 'tld_length', 'count-', 'count@', 'count?',
          'count%', 'count.', 'count=', 'count-http','count-https', 'count-www', 'count-digits',
          'count-letters', 'count_dir', 'use_of_ip']]

 #Target Variable
 y = urldata['result']
#Splitting the data into Training and Testing
 x_train, x_test, y_train, y_test = train_test_split(x, y, train_size=0.2, random_state=42)
import numpy as np
 import matplotlib.pyplot as plt
 plt.style.use('fivethirtyeight')
 import pandas as pd
 from sklearn.preprocessing import MinMaxScaler
 from keras.models import Sequential
 from keras.layers import Dense, LSTM, Dropout, GRU, Bidirectional
```

```
from keras.optimizers import SGD
import math
from sklearn.metrics import mean_squared_error
import tensorflow as tf
x_train.shape
(105086, 17)
x_train.shape[1],1
(17, 1)
from keras import backend as K

def recall_m(y_true, y_pred):
        true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
        possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
        recall = true_positives / (possible_positives + K.epsilon())
        return recall

def precision_m(y_true, y_pred):
        true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
        predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
        precision = true_positives / (predicted_positives + K.epsilon())
        return precision

def f1_m(y_true, y_pred):
        precision = precision_m(y_true, y_pred)
        recall = recall_m(y_true, y_pred)
        return 2*((precision*recall)/(precision+recall+K.epsilon()))
regressor = Sequential()
# First GRU layer with Dropout regularisation
regressor.add(GRU(units=50, return_sequences=True, input_shape=(x_train.shape[1],1)))
regressor.add(Dropout(0.2))
# Second GRU layer
regressor.add(GRU(units=50, return_sequences=True))
regressor.add(Dropout(0.2))
# Third GRU layer
regressor.add(GRU(units=50, return_sequences=True))
regressor.add(Dropout(0.2))
#fourth layer
regressor.add(GRU(units=50, return_sequences=True))
regressor.add(Dropout(0.2))
```

```
 # Fifth GRU layer
 regressor.add(GRU(units=50))
 regressor.add(Dropout(0.2))
 # The output layer
 regressor.add(Dense(units=1))

 # Compiling the GRU
 regressor.compile(optimizer='adam',loss='mean_squared_error',
metrics=['acc',f1_m,precision_m, recall_m])


regressor.fit(x_train,y_train,epochs=10,batch_size=32)
Epoch 1/10
 3284/3284 [==============================] - 213s 61ms/step - loss: 0.0222 - acc:
0.9781 - f1_m: 0.9586 - precision_m: 0.9797 - recall_m: 0.9462
 Epoch 2/10
 3284/3284 [==============================] - 203s 62ms/step - loss: 0.0085 - acc:
0.9930 - f1_m: 0.9892 - precision_m: 0.9954 - recall_m: 0.9842
 Epoch 3/10
 3284/3284 [==============================] - 200s 61ms/step - loss: 0.0080 - acc:
0.9934 - f1_m: 0.9897 - precision_m: 0.9956 - recall_m: 0.9849
 Epoch 4/10
 3284/3284 [==============================] - 201s 61ms/step - loss: 0.0080 - acc:
0.9932 - f1_m: 0.9896 - precision_m: 0.9952 - recall_m: 0.9851
 Epoch 5/10
 3284/3284 [==============================] - 202s 62ms/step - loss: 0.0077 - acc:
0.9935 - f1_m: 0.9900 - precision_m: 0.9957 - recall_m: 0.9854
 Epoch 6/10
 3284/3284 [==============================] - 201s 61ms/step - loss: 0.0077 - acc:
0.9936 - f1_m: 0.9899 - precision_m: 0.9956 - recall_m: 0.9852
 Epoch 7/10
 3284/3284 [==============================] - 201s 61ms/step - loss: 0.0077 - acc:
0.9935 - f1_m: 0.9901 - precision_m: 0.9954 - recall_m: 0.9858
 Epoch 8/10
 3284/3284 [==============================] - 201s 61ms/step - loss: 0.0075 - acc:
0.9937 - f1_m: 0.9903 - precision_m: 0.9955 - recall_m: 0.9860
 Epoch 9/10
 3284/3284 [==============================] - 200s 61ms/step - loss: 0.0074 - acc:
0.9937 - f1_m: 0.9902 - precision_m: 0.9957 - recall_m: 0.9857
```

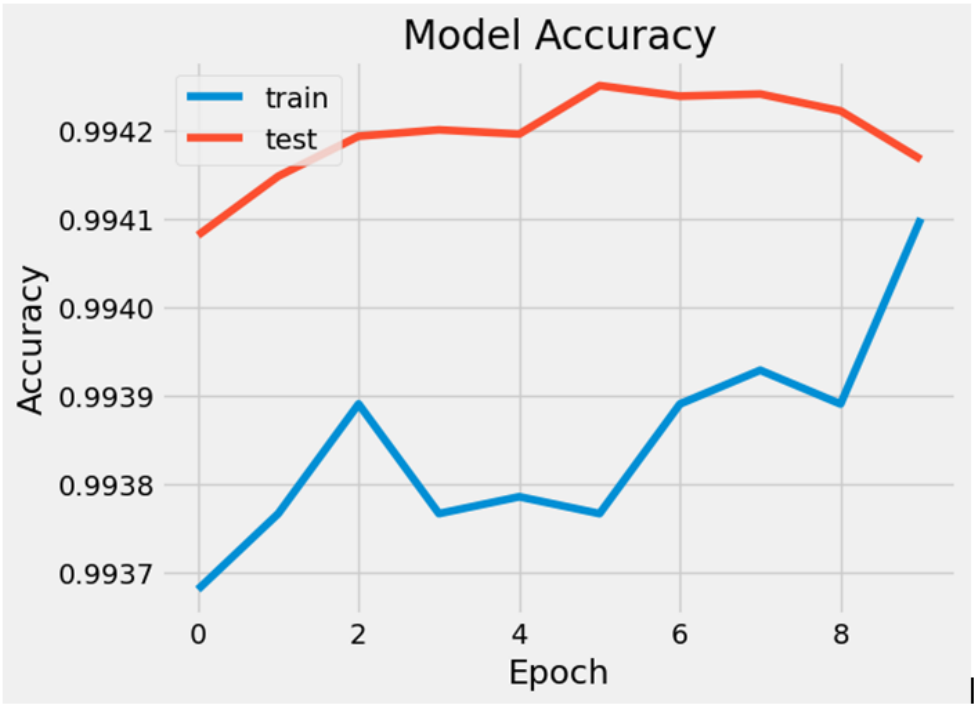
```
 Epoch 10/10
 3284/3284 [==============================] - 200s 61ms/step - loss: 0.0075 - acc:
0.9936 - f1_m: 0.9900 - precision_m: 0.9954 - recall_m: 0.9855
<keras.callbacks.History at 0x7faffc342830>
# Calculate accuracy
 accuracy = regressor.evaluate(x_test, y_test)[1]
13136/13136 [==============================] - 188s 14ms/step - loss: 0.0058 -
acc: 0.9941 - f1_m: 0.9908 - precision_m: 0.9957 - recall_m: 0.9869
print(accuracy)
0.9940786957740784
regressor.save('/content/drive/MyDrive/updated/urldata.csv (2)')
WARNING:absl:Found untraced functions such as _update_step_xla, gru_cell_layer_call_fn,
gru_cell_layer_call_and_return_conditional_losses, gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses while saving (showing 5 of 11). These
functions will not be directly callable after loading.
from keras import models as M
 reconstructed_model =M.load_model("/content/drive/MyDrive/updated/urldata.csv (2)",
custom_objects={'f1_m': f1_m, 'precision_m': precision_m, 'recall_m': recall_m})
tf.keras.utils.plot_model(
        reconstructed_model,
        to_file='model.png',
        show_shapes=False,
        show_dtype=False,
        show_layer_names=True,
        rankdir='TB',
        expand_nested=False,
        dpi=96,
        layer_range=None,
        show_layer_activations=False,
        show_trainable=False
 )
import matplotlib.pyplot as plt

 # train the model and save the history
 history = regressor.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=10,
batch_size=32)

 # plot the accuracy curve
 plt.plot(history.history['acc'])
```

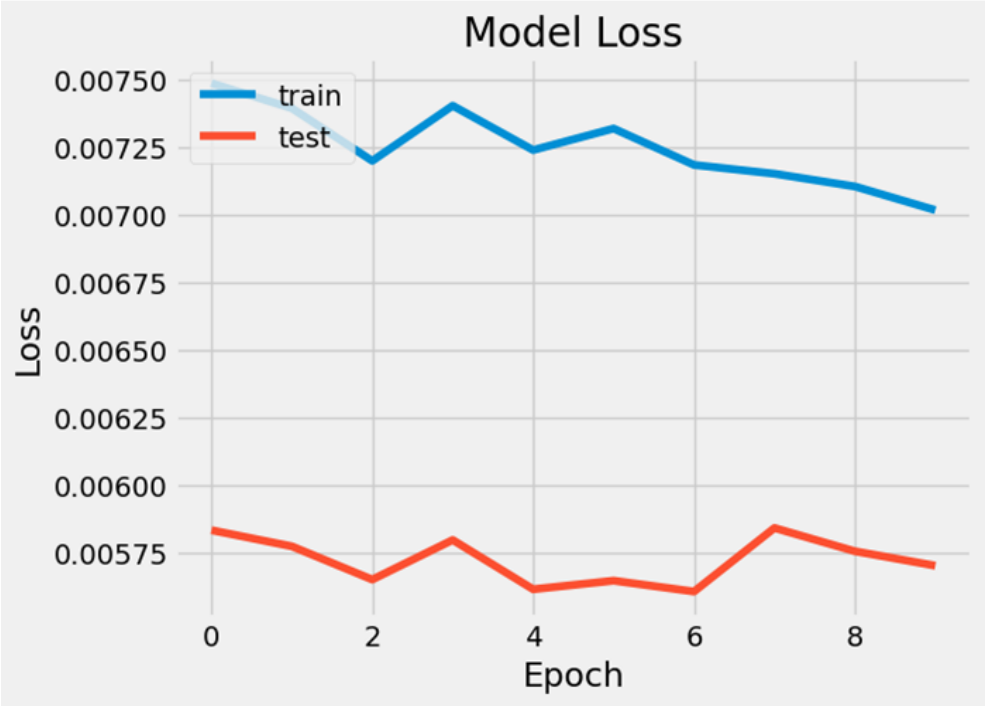
```
plt.plot(history.history['val_acc'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

# plot the loss curve
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
Epoch 1/10
3284/3284 [==============================] - 381s 116ms/step - loss: 0.0075 -
acc: 0.9937 - f1_m: 0.9900 - precision_m: 0.9957 - recall_m: 0.9854 - val_loss: 0.0058 -
val_acc: 0.9941 - val_f1_m: 0.9908 - val_precision_m: 0.9960 - val_recall_m: 0.9866
Epoch 2/10
3284/3284 [==============================] - 405s 123ms/step - loss: 0.0074 -
acc: 0.9938 - f1_m: 0.9903 - precision_m: 0.9956 - recall_m: 0.9860 - val_loss: 0.0058 -
val_acc: 0.9941 - val_f1_m: 0.9909 - val_precision_m: 0.9959 - val_recall_m: 0.9869
Epoch 3/10
3284/3284 [==============================] - 403s 123ms/step - loss: 0.0072 -
acc: 0.9939 - f1_m: 0.9904 - precision_m: 0.9956 - recall_m: 0.9862 - val_loss: 0.0056 -
val_acc: 0.9942 - val_f1_m: 0.9910 - val_precision_m: 0.9959 - val_recall_m: 0.9870
Epoch 4/10
3284/3284 [==============================] - 403s 123ms/step - loss: 0.0074 -
acc: 0.9938 - f1_m: 0.9904 - precision_m: 0.9956 - recall_m: 0.9862 - val_loss: 0.0058 -
val_acc: 0.9942 - val_f1_m: 0.9910 - val_precision_m: 0.9960 - val_recall_m: 0.9869
Epoch 5/10
3284/3284 [==============================] - 405s 123ms/step - loss: 0.0072 -
acc: 0.9938 - f1_m: 0.9904 - precision_m: 0.9956 - recall_m: 0.9861 - val_loss: 0.0056 -
val_acc: 0.9942 - val_f1_m: 0.9910 - val_precision_m: 0.9959 - val_recall_m: 0.9870
Epoch 6/10
3284/3284 [==============================] - 403s 123ms/step - loss: 0.0073 -
acc: 0.9938 - f1_m: 0.9903 - precision_m: 0.9957 - recall_m: 0.9860 - val_loss: 0.0056 -
val_acc: 0.9942 - val_f1_m: 0.9911 - val_precision_m: 0.9959 - val_recall_m: 0.9871
```

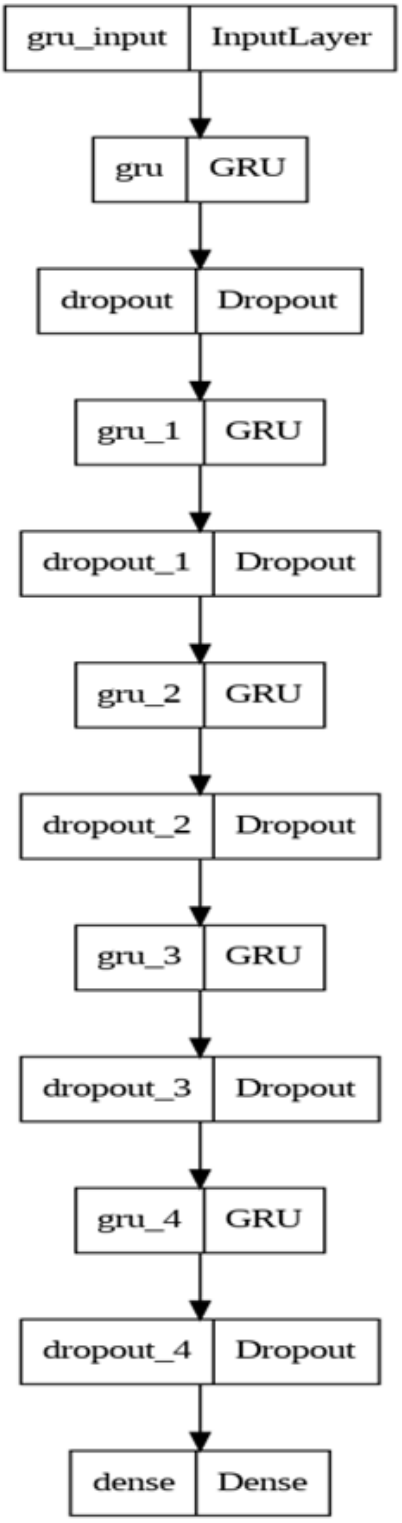
```
 Epoch 7/10
 3284/3284 [==============================] - 403s 123ms/step - loss: 0.0072 -
acc: 0.9939 - f1_m: 0.9903 - precision_m: 0.9957 - recall_m: 0.9859 - val_loss: 0.0056 -
val_acc: 0.9942 - val_f1_m: 0.9910 - val_precision_m: 0.9958 - val_recall_m: 0.9873
 Epoch 8/10
 3284/3284 [==============================] - 403s 123ms/step - loss: 0.0072 -
acc: 0.9939 - f1_m: 0.9907 - precision_m: 0.9957 - recall_m: 0.9866 - val_loss: 0.0058 -
val_acc: 0.9942 - val_f1_m: 0.9911 - val_precision_m: 0.9959 - val_recall_m: 0.9871
 Epoch 9/10
 3284/3284 [==============================] - 404s 123ms/step - loss: 0.0071 -
acc: 0.9939 - f1_m: 0.9906 - precision_m: 0.9955 - recall_m: 0.9866 - val_loss: 0.0058 -
val_acc: 0.9942 - val_f1_m: 0.9910 - val_precision_m: 0.9958 - val_recall_m: 0.9872
 Epoch 10/10
 3284/3284 [==============================] - 402s 122ms/step - loss: 0.0070 -
acc: 0.9941 - f1_m: 0.9907 - precision_m: 0.9957 - recall_m: 0.9868 - val_loss: 0.0057 -
val_acc: 0.9942 - val_f1_m: 0.9909 - val_precision_m: 0.9954 - val_recall_m: 0.9874
reconstructed_model.evaluate(
        x=x_test,
        y=y_test,
        batch_size=32,
        verbose="auto",
        sample_weight=None,
        steps=None,
        callbacks=None,
        max_queue_size=10,
        workers=1,
        use_multiprocessing=False,
        return_dict=False
 )
13136/13136 [==============================] - 191s 14ms/step - loss: 0.0058 -
acc: 0.9941 - f1_m: 0.9908 - precision_m: 0.9957 - recall_m: 0.9869
[0.0057611060328781605,
  0.9940786957740784,
  0.9908038377761841,
  0.9956569075584412,
  0.9869295358657837]
reconstructed_model.fit(x_test,y_test)
13136/13136 [==============================] - 825s 62ms/step - loss: 0.0071 -
acc: 0.9941 - f1_m: 0.9909 - precision_m: 0.9958 - recall_m: 0.9869
```

```
<keras.callbacks.History at 0x7fafedf716c0>
url=input("url  :")
 length_url=len(url)
 hostname_length=len(urlparse(url).netloc)
 path_length=len(urlparse(url).path)
 #print(path_length)
 fd_len=fd_length(url)
 tld_len= tld_length(url)
 minus=url.count('-')
 #print(minus)
 at=url.count('@')
 qm=url.count('?')
 mod=url.count('%')
 dot=url.count('.')
 eq=url.count('=')
 htp=url.count('http')
 htps=url.count('https')
 ww=url.count('www')
 numdigit= digit_count(url)
 #print(numdigit)
 numletters=letter_count(url)
 numdir= no_of_dir(url)
 ipuse= having_ip_address(url)
 shorturl=  shortening_service(url)


 urlfeat = [hostname_length,
           path_length, fd_len, tld_len, minus, at, qm,
           mod, dot, eq, htp,htps, ww, numdigit,
           numletters, numdir, ipuse]
 print(urlfeat)


 urlfeat=[urlfeat]


predict=reconstructed_model.predict(urlfeat)
 p=predict[0]
```

```
 q=p[0]
 print(q)
if(q>0.9):
   print("malicious")
 else:
   print("benign")
```


Model Accuracy
train
test
0.9942
0.9941
0.9940
0.9939
0.9938
0.9937
Accuracy
0
2
4
6
8
Epoch


Model Loss
train
test
0.00750
0.00725
0.00700
0.00675
0.00650
0.00625
0.00600
0.00575
Loss
0
2
4
6
8
Epoch


gru_input InputLayer
gru GRU
dropout Dropout
gru_1 GRU
dropout_1 Dropout
gru_2 GRU
dropout_2 Dropout
gru_3 GRU
dropout_3 Dropout
gru_4 GRU
dropout_4 Dropout
dense Dense

Manifest.json

```
{
 "name": "phish",
 "version": "2.0.0",
 "description": "A URL analysis tool for detecting phishing attempts",
 "manifest_version": 2,
 "author": "nvn",
 "background": {
   "scripts": ["script.js"],
   "persistent": true
 },
 "permissions": [
   "storage",
   "activeTab",
   "tabs",
   "notifications",
   "webRequestBlocking",
   "http://*/*",
   "https://*/*"
 ],
 "content_security_policy": "script-src 'self' 'unsafe-eval'; object-src 'self'",
 "content_scripts": [
         {
    "matches": ["<all_urls>"],
    "js": ["script1.js", "script.js"]
         }
 ],
 "icons": {
   "48": "/images/Capture9.PNG"
 },
 "browser_action": {
   "default_popup": "index.html",
   "default_icon": {
     "48": "/images/Capture9.PNG"
         }
 }
```

```
}
```

Index.html

```
<html>

<head>
  <title>WARNING</title>
</head>
<style>
  body {
    background-image: url(bgm.jpeg);
    background-color: rgba(244, 235, 235, 0.812);
    width: 250px;
    height: 400px;
    text-align: center;
  }
  h2 {
    color: rgb(82, 75, 75);
    text-align: center;
  }
  button {
    background-color: rgb(177, 181, 178);
    border-radius: 4px;
  }
  .text-bottom {
    display: flex;
    justify-content: center;
    align-items: center;
    bottom:0px;
}

</style>

<body id="body">
  <strong>
    <h2>PHISHING DETECTION</h2>
  </strong><br>
  <div class="des" id="url"></div>
```

```
  <br>
  <div id="cb" style="background-color: rgb(158, 167, 167);">
    <input type="checkbox" id="block" name="block" />
    <label for="Block"><strong>Block</strong></label><br />
  </div>
  <a href="https://nafizeahamed.pythonanywhere.com/report1" target="_blank"><div
class=text-bottom >
    <button id="buttonOne" class="text-bottom">REPORT</button>
  </div>
</a>
  <div id="p" style="width: 250px; height: 25px; background-color: lightgray"></div>
  <div id="q" style="width: 250px; height: 25px; background-color: lightgray"></div>
  <div id="r" style="width: 250px; height: 25px; background-color: lightgray"></div>
  <div id="s" style="width: 250px; height: 25px; background-color: lightgray"></div>
  <div id="t" style="width: 250px; height: 25px; background-color: lightgray"></div>
  <div id="u" style="width: 250px; height: 25px; background-color: lightgray"></div>
  <div id="v" style="width: 250px; height: 25px; background-color: lightgray"></div>
  <div id="w" style="width: 250px; height: 25px; background-color: lightgray"></div>
  <div id="m" style="width: 250px; height: 25px; background-color: lightgray"></div>
  <div id="n" style="width: 250px; height: 25px; background-color: lightgray"></div>
  <div id="o" style="width: 250px; height: 25px; background-color: lightgray"></div>

  <script src="script1.js"></script>
</body>

</html>
```

Popup.html

```
<div style="background-color: #e16f6f; color: #d41515; padding: 10px; border-radius: 5px;
box-shadow: 0px 0px 5px #ccc;">
    This is a popup notification!
    This Website may be unsafe to use....
</div>
```

Redirect.html

```
<html>
```

```
<head>
<meta http-equiv=Content-Type content="text/html; charset=windows-1252">
<meta name=Generator content="Microsoft Word 15 (filtered)">
<style>
<!--
/* Font Definitions */
 @font-face
        {font-family:"Cambria Math";
        panose-1:2 4 5 3 5 4 6 3 2 4;}
@font-face
        {font-family:Calibri;
        panose-1:2 15 5 2 2 2 4 3 2 4;}
@font-face
        {font-family:Cambria;
        panose-1:2 4 5 3 5 4 6 3 2 4;}
 /* Style Definitions */
 p.MsoNormal, li.MsoNormal, div.MsoNormal
        {margin-top:4.5pt;
        margin-right:12.75pt;
        margin-bottom:0cm;
        margin-left:15.6pt;
        text-align:center;
        line-height:150%;
        font-size:11.0pt;
        font-family:"Calibri",sans-serif;}
h1
        {mso-style-link:"Heading 1 Char";
        margin-top:24.0pt;
        margin-right:12.75pt;
        margin-bottom:0cm;
        margin-left:15.6pt;
        text-align:center;
        line-height:150%;
        page-break-after:avoid;
        font-size:14.0pt;
        font-family:"Cambria",serif;
        color:#365F91;}
span.Heading1Char
        {mso-style-name:"Heading 1 Char";
```

```
        mso-style-link:"Heading 1";
        font-family:"Cambria",serif;
        color:#365F91;
        font-weight:bold;}
.MsoChpDefault
        {font-family:"Calibri",sans-serif;}
.MsoPapDefault
        {margin-top:4.5pt;
        margin-right:12.75pt;
        margin-bottom:0cm;
        margin-left:15.6pt;
        margin-bottom:.0001pt;
        text-align:center;
        line-height:150%;}
@page WordSection1
        {size:612.0pt 792.0pt;
        margin:72.0pt 72.0pt 72.0pt 72.0pt;}
div.WordSection1
        {page:WordSection1;}-->
</style>

</head>

<body lang=EN-IN style='word-wrap:break-word'>

<div class=WordSection1>

<h1><span style='color:red'>This page is blocked</span></h1>

<p class=MsoNormal>This page has been blocked by our security system due to the
possibility of phishing. Phishing is a fraudulent practice of sending emails or
creating websites that appear to be from reputable companies in order to induce
individuals to reveal personal information, such as passwords and credit card
numbers . Phishing can lead to identity theft, financial loss, and other
serious consequences. If you believe this page has been blocked in error,
please contact our customer support team at support@company.com. Thank you for
your cooperation and understanding.</p>

</div>
```

```
</body>

</html>
```

Script.js

```
"use strict";
function block() {

  chrome.storage.local.get("myCheckboxState", function (result) {
    var storedData = result.myCheckboxState;
    // Do something with the data
    //alert(storedData);
    if (storedData === true) {

      chrome.tabs.update({ url: "redirect.html" });
    }

    else {

      chrome.windows.create({
        url: "popup.html",
        type: "popup",
        width: 300,
        height: 200,
      });


    }

  });
}
function isFilePath(url) {
  var x = url.slice(0, 4);
  if (x === "http") {
    return false;
```

```
  } else {
    return true;
  }
}

let cloud = "https://nafizeahamed.pythonanywhere.com/report1";

let firstexe = false;

chrome.runtime.onInstalled.addListener(function () {
  chrome.tabs.onCreated.addListener(function (tab) {
    firstexe = true;
    chrome.tabs.query({ active: true, currentWindow: true }, function (tabs) {
      let currentTabUrl = "";
      currentTabUrl = tabs[0].url;
      const message = currentTabUrl;

      if (cloud != message) {
        if (isFilePath(message) === false) {

          fetch("http://nafizeahamed.pythonanywhere.com/", {
            method: "POST",
            mode: "no-cors",
            credentials: "include",
            headers: {
              "X-CSRFToken": "123456789asdfghjkl",

              "Content-Type": "application/json",
            },
            body: JSON.stringify({ url: message }),
          })
            .then((response) => response.json())
            .then((response) => {
              const x = JSON.stringify(response);
              const k1 = JSON.stringify({ result: "[1]" });

              if (x === k1) {
                currentTabUrl = currentTabUrl + " IS MALICIOUS!!";
```

```
            alert(currentTabUrl + " ⚠️");
            block(currentTabUrl);
          } else {

            currentTabUrl = currentTabUrl + " IS LEGITIMATE";
          }
        })
        .catch((error) => {
        });
    }
  }
 });
});
if (firstexe === false) {
 chrome.tabs.onUpdated.addListener(function (tabId, changeInfo, tab) {
  if (changeInfo.status === "complete") {
   let currentTabUrl = "";
   currentTabUrl = tab.url;
   const message = currentTabUrl;

   if (cloud != message) {
    if (isFilePath(message) === false) {
     fetch("http://nafizeahamed.pythonanywhere.com/", {
      method: "POST",
      mode: "no-cors",
      credentials: "include",
      headers: {
       "X-CSRFToken": "123456789asdfghjkl",

       "Content-Type": "application/json",
      },
      body: JSON.stringify({ url: message }),
     })
      .then((response) => response.json())
      .then((response) => {
       const x = JSON.stringify(response);
       const k1 = JSON.stringify({ result: "[1]" });
```

```
            if (x === k1) {
              currentTabUrl = currentTabUrl + " IS MALICIOUS!!";

              block(currentTabUrl);

              alert(currentTabUrl + " ⚠️ ");
            } else {
              currentTabUrl = currentTabUrl + " IS LEGITIMATE";
            }
          })
          .catch((error) => {
          });
      }
    }
  }
 });
}
```

Script1.js

```
"use strict";

function saveCheckboxState(checkboxState) {
 chrome.storage.local.set({ myCheckboxState: checkboxState });
}

// Function to load the checkbox state
function loadCheckboxState(callback) {
 chrome.storage.local.get("myCheckboxState", function (result) {
  callback(result.myCheckboxState);
 });
}

var checkbox = document.getElementById("block");
checkbox.addEventListener("change", function () {
 saveCheckboxState(checkbox.checked);
});

loadCheckboxState(function (state) {
```

```
  checkbox.checked = state;
});

function parsing(url) {
  var parser = new URL(url);
  const p = document.createElement("p");
  p.innerHTML = "protocol  :" + parser.protocol;
  document.getElementById("p").appendChild(p);

  const q = document.createElement("q");
  q.innerHTML = "host  :" + parser.host;
  document.getElementById("q").appendChild(q);

  const r = document.createElement("r");
  r.innerHTML = "port  :" + parser.port + "\n\n";
  document.getElementById("r").appendChild(r);

  const s = document.createElement("s");
  s.innerHTML = "hostname :" + parser.hostname;
  document.getElementById("s").appendChild(s);

  const t = document.createElement("t");
  t.innerHTML = "search element  :" + parser.search;
  document.getElementById("t").appendChild(t);

  const u = document.createElement("t");
  u.innerHTML = "pathname  :" + parser.pathname;
  document.getElementById("u").appendChild(u);

  const v = document.createElement("v");
  v.innerHTML = "pathname  :" + parser.hash;
  document.getElementById("v").appendChild(v);

  const w = document.createElement("w");
  w.innerHTML = "url  :" + parser.href;
  document.getElementById("w").appendChild(w);

  let y = "";
  fetch(`https://cloudflare-dns.com/dns-query?name=${parser.hostname}&type=A`, {
```

```
    headers: {
      accept: "application/dns-json",
    },
  })
    .then((response) => response.json())
    .then((data) => {
      const ipAddress = data.Answer[0].data;
      y = ipAddress;
      console.log(ipAddress);
      const m = document.createElement("m");
      m.innerHTML = "ip  :" + y;
      document.getElementById("m").appendChild(m);

      fetch(`https://ipapi.co/${ipAddress}/json/`)
        .then((response) => response.json())
        .then((data) => {
          const n = document.createElement("n");
          n.innerHTML = "city  :" + data.city;
          document.getElementById("n").appendChild(n);

          const o = document.createElement("o");
          o.innerHTML = "country  :" + data.country;
          document.getElementById("o").appendChild(o);
        });
    });
}

chrome.tabs.onCreated.addListener(function (tab) {
  chrome.tabs.query({ active: true, currentWindow: true }, function (tabs) {
    let currentTabUrl = "";
    currentTabUrl = tabs[0].url;
    const message = currentTabUrl;
    parsing(currentTabUrl);
    alert("parsing");
    fetch("https://nafizeahamed.pythonanywhere.com/", {
      method: "POST",
      mode: "no-cors",
      credentials: "include",
      headers: {
```

```
    "X-CSRFToken": "123456789asdfghjkl",

    "Content-Type": "application/json",
  },
  body: JSON.stringify({ url: message }),
})
  .then((response) => response.json())
  .then((response) => {
    const x = JSON.stringify(response);
    const k1 = JSON.stringify({ result: "[1]" });

    if (x === k1) {
      const ele = document.getElementById("body");
      const custom_styles = {
        color: "red",
      };

      Object.assign(ele.style, custom_styles);
      image.src = "Capture13.PNG";
      image.width = "75";
      image.height = "75";
      document.body.appendChild(image);
      x = currentTabUrl;
      currentTabUrl = currentTabUrl + " IS MALICIOUS!!";
      document.getElementById("url").textContent = currentTabUrl;
    } else {
      const ele = document.getElementById("body");
      const custom_styles = {
        color: "green",
      };
      Object.assign(ele.style, custom_styles);
      image.src = "Capture12.PNG";
      image.width = "75";
      image.height = "75";
      document.body.appendChild(image);
      currentTabUrl = currentTabUrl + " IS LEGITIMATE";
      document.getElementById("url").textContent = currentTabUrl;
    }
  })
```

```
      .catch((error) => {
        alert(error);
      });
  });
});
if (typeof browser === "undefined") {
  var browser = chrome;
}
const image = document.createElement("img");
chrome.tabs.query({ active: true, currentWindow: true }, function (tabs) {
  let currentTabUrl = "";
  currentTabUrl = tabs[0].url;
  const message = currentTabUrl;
  parsing(currentTabUrl);
  fetch("https://nafizeahamed.pythonanywhere.com/", {
    method: "POST",
    mode: "no-cors",
    credentials: "include",
    headers: {
      "X-CSRFToken": "123456789asdfghjkl",

      "Content-Type": "application/json",
    },
    body: JSON.stringify({ url: message }),
  })
    .then((response) => response.json())
    .then((response) => {
      const x = JSON.stringify(response);
      const k1 = JSON.stringify({ result: "[1]" });

      if (x === k1) {
        const ele = document.getElementById("body");
        const custom_styles = {
          color: "red",
        };

        Object.assign(ele.style, custom_styles);
        image.src = "Capture13.PNG";
        image.width = "75";
```

```
      image.height = "75";
      document.body.appendChild(image);

      currentTabUrl = currentTabUrl + " IS MALICIOUS!!";
      document.getElementById("url").textContent = currentTabUrl;

    }
     else {
      const ele = document.getElementById("body");
      const custom_styles = {
        color: "green",
      };
      Object.assign(ele.style, custom_styles);
      image.src = "Capture12.PNG";
      image.width = "75";
      image.height = "75";
      document.body.appendChild(image);
      currentTabUrl = currentTabUrl + " IS LEGITIMATE";
      document.getElementById("url").textContent = currentTabUrl;
    }
  })
  .catch((error) => {
    alert(error);
  });
});
```

#Predictive System

```
import pandas as pd
import pickle

# Load the saved model
with open('/content/RandomForestModel_saved', 'rb') as f:
    model = pickle.load(f)

# Get input URL from user
url = input("Enter the url:")
```

```
# Create a dictionary with the new URL features
new_data = {
    'ip_in_URL': [having_ip_address(url)],
    'random_words': [count_random_words(url)],
    'avg_len_random_words':[avg_len_random(url)],
    'http_in_URL':[url.count('http')],
    'semicolon_in_URL': [semicolon_in_URL(url)],
    'AND_in_URL':[AND_in_URL(url)],
    'URL_length': [URL_length(url)],
    'alphabets_in_URL':[letter_count(url)],
    'uppercase_letters_inURL': [uppercase_count(url)],
    'lowercase_letters_inURL':[lowercase_count(url)],
    'uppercase_letters_ratio_inURL':[uppercase_ratio(url)],
    'lowercase_letters_ratio_inURL': [lowercase_ratio(url)],
    'Special_char_in_URL': [spl_char_count(url)],
    'numbers_ratio_inURL': [number_ratio(url)],
    'alphabet_ratio_inURL': [alphabet_ratio(url)],
    'fd_length':[fd_length(url)],
    'fd_length2':[fd_length(url)],
    'tld_length':[ tld_length(get_tld(url,fail_silently=True))],
    'tld_length2':[ tld_length(get_tld(url,fail_silently=True))],
    'has_server': [has_server_in_string(url)],
    'has_login': [has_login_in_string(url)],
    'has_client': [has_client_in_string(url)],
    'has_admin': [has_admin_in_string(url)],
    'count_dir': [no_of_dir(url)],
    'count_dir2': [no_of_dir(url)],
    'domain_length': [domain_len(url)],
    'dots_in_domain': [dots_in_domain(url)],
    'short_url':[shortening_service(url)],
    'short_url2':[shortening_service(url)],
    'short_url3':[shortening_service(url)],
    'number_subdom': [number_subdomain(url)],
    'hyphens_in_domain': [hyphens_in_domain(url)],

}

# Create a DataFrame from the new_data dictionary
```

```
new_data_df = pd.DataFrame(new_data)


# Use the trained model to make predictions
prediction = model.predict(new_data_df)

print(prediction)
```

Django

views.py

```
from django.shortcuts import render
from django.http import HttpResponse
import pickle
#import numpy as np
from django.http import JsonResponse
from django.views.decorators.csrf import csrf_exempt
from .models import UrlDataset
import json
import datetime
import urllib.parse
import dns.resolver
import requests
from bs4 import BeautifulSoup
import ssl
import socket
# Create your views here.
import re
from urllib.parse import urlparse
from tld import get_tld
import gzip

def having_ip_address(urls):
    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}', urls)  # Ipv6
    if match:
        return 1
    else:
        return 0
def count_random_words(urls):
    li1 = re.findall("[A-Za-z]+",urls)
    return len(li1)

def avg_len_random(urls):
    sum = 0
    li2 = re.findall("[A-Za-z]+",urls)
    for u in li2:
        sum = sum + len(u)
    return (sum/len(li2))
def semicolon_in_URL(urls):
  count = urls.count(';')
  return count
def AND_in_URL(urls):
  count = urls.count('&')
  return count
def URL_length(urls):
  return len(str(urls))
def uppercase_count(url):
    letters1 = 0
    for i in url:
        if i>='A' and i<='Z':
            letters1 = letters1 + 1
    return letters1
def lowercase_ratio(url):
    url_len = len(str(url))
    letters2 = 0
```

```
    for i in url:
        if i>='a' and i<='z':
            letters2 = letters2 + 1
    return (letters2/url_len)
def spl_char_count(url):
    url_len = len(str(url))
    chars = 0
    for i in url:
        if i=='!' or i=='@' or i=='#' or i=='%' or i=='^' or i=='&' or i=='*' or i=='(' or i==')':
            chars = chars + 1
    return chars
def number_ratio(url):
    url_len = len(str(url))
    nums = 0
    for i in url:
        if i>='0' and i<='9':
            nums = nums + 1
    return (nums/url_len)
def alphabet_ratio(url):
    url_len = len(str(url))
    letters = 0
    for i in url:
        if i.isalpha():
            letters = letters + 1
    return (letters/url_len)
def has_server_in_string(url):
    if 'server' in url.lower():
        return 1
    else:
        return 0
def has_login_in_string(url):
    if 'login' in url.lower():
        return 1
    else:
        return 0
def has_client_in_string(url):
    if 'client' in url.lower():
        return 1
    else:
```

```
        return 0
def has_admin_in_string(url):
    if 'admin' in url.lower():
        return 1
    else:
        return 0
def no_of_dir(url):
    urldir = urlparse(url).path
    return urldir.count('/')
def domain_len(url):
    domain = urlparse(str(url)).netloc
    return len(domain)
def dots_in_domain(url):
    count1 = url.count('.')
    return count1
def number_subdomain(url):
    count = 0
    domain = urlparse(url).netloc
    dom = '.'.join(domain.split('.')[:-2])
    for i in dom:
        if i=='.':
            count = count+1
    return (count-1)
def hyphens_in_domain(url):
    count1 = url.count('-')
    return count1
def letter_count(url):
    letters = 0
    for i in url:
        if i.isalpha():
            letters = letters + 1
    return letters
def fd_length(url):
    urlpath= urlparse(url).path
    try:
        return len(urlpath.split('/')[1])
    except:
        return 0
def lowercase_count(url):
```

```
    letters2 = 0
    for i in url:
        if i>='a' and i<='z':
            letters2 = letters2 + 1
    return letters2
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\.g
d|'
                    'tr\.im|link\.zip\.net',
                    url)
    if match:
        return -1
    else:
        return 1
def tld_length(tld):
    try:
        return len(tld)
    except:
        return -1
def uppercase_ratio(url):
    url_len = len(str(url))
    letters1 = 0
    for i in url:
        if i>='A' and i<='Z':
            letters1 = letters1 + 1
    return (letters1/url_len)
def check_ssl_certificate(domain):
```

```
  try:
    context = ssl.create_default_context()
    with socket.create_connection((domain, 443)) as sock:
        with context.wrap_socket(sock, server_hostname=domain) as sslsock:
            certificate = sslsock.getpeercert()

    if certificate:
        print("The domain has an SSL/TLS certificate.")
        return True
    else:
        print("The domain does not have an SSL/TLS certificate.")
        return False
  except Exception as e:
      return False

def contains_form_tag(url):
    try:
    # Find all <form> tags
        response = requests.get(url)
        soup = BeautifulSoup(response.text, 'html.parser')
        forms = soup.find_all('form')
        inputs = soup.find_all('input')
        temp=0
        for inp in  inputs :
            if(inp.get('type')=='password'):
                temp=1
        if(temp==1):
            return True
        else:
            return False


    except Exception as e:
        print("Error in form_tag ",e)
        return None

def verify_domain(domain):
    try:
        response = requests.get(f"https://{domain}")
```

```
        if response.status_code == 200:
            return True
        else:
            return False
    except requests.exceptions.RequestException as e:
        return False

def is_dns(domain):
    try:
        answers = dns.resolver.resolve(domain, 'A')
    except dns.exception.DNSException as e:
        print(f"Error resolving domain {domain}: {e}")
        return False

    if answers:
        return True
    else:
        return False

@csrf_exempt
def home(request):
    if request.method == "POST" :
        data = json.loads(request.body.decode('utf-8'))
        url = data['url']
        url = url.split('@')[-1]
        print(url)
        parsed_url = urllib.parse.urlparse(url)
        parsed_url = parsed_url.netloc.split('www.')[-1]
        if(is_dns(parsed_url)==True) :
            s = "The url is legitimate"
            prediction = '[0]'
            print(s+" DNS records and Has SSL Certificate!!")
            return JsonResponse(data={'result': str(prediction)})

        else :
            classifier =
pickle.load(open(r"/home/nafizeahamed/django_cloud/Django_Project/Detecting_PhishingU
rls/logisticModel_savedV2", 'rb'))
```

```
        vectorizer1 =
pickle.load(open(r"/home/nafizeahamed/django_cloud/Django_Project/Detecting_PhishingU
rls/vectorizer_savedv2", 'rb'))
        df1 = np.array(url)
        vect_url = vectorizer1.transform(df1.ravel())
        X_pred = vect_url

        prediction = classifier.predict(X_pred)
        print(prediction)
        if (prediction == 1):
            s = "The url is malicious"
            print(s+" No dns records,contains form tag,No ssl certificate,ML prediction")

        else:
            s = "The url is legitimate"
            print(s+" No dns records,contains form tag,No ssl certificate,ML prediction")
        return JsonResponse(data={'result': str(prediction)})
    #print(HttpResponse(json.dumps({'result': 'i love you'}),
content_type='application/json'))
  return render(request,"home.html")
def result(request):
    url = request.GET['url']
    url = url.split('@')[-1]
    print(url)
    parsed_url = urllib.parse.urlparse(url)
    parsed_url = parsed_url.netloc.split('www.')[-1]

    #gauth = GoogleAuth()
    #gauth.LocalWebserverAuth()
    #drive = GoogleDrive(gauth)

    #drive =
pickle.load(open(r"/home/nafizeahamed/django_cloud/Django_Project/Detecting_PhishingU
rls/drive_saved", 'rb'))
    #drive = google_drive.GoogleDrive(gauth)
    #file_list = drive.ListFile({'q': "'root' in parents and trashed=false"}).GetList()
    #for file in file_list:
    #    print('Title: %s, ID: %s' % (file['title'], file['id']))
```

```
        #file_id = '1SvGsHIhB30XVdj4YRmCFYOuJqC5w6wB1' # Replace with the ID of
your pickle file
        #file = drive.CreateFile({'id': file_id})
        #file.GetContentFile('model.pkl')


        with
gzip.open(r"/home/nafizeahamed/django_cloud/Django_Project/Detecting_PhishingUrls/mo
del.pkl.gz", 'rb') as f:
            model = pickle.load(f)


            #vectorizer1 =
pickle.load(open(r"/home/nafizeahamed/django_cloud/Django_Project/Detecting_PhishingU
rls/vectorizer_savedv2", 'rb'))
        new_data = {
        'ip_in_URL': [having_ip_address(url)],
        'random_words': [count_random_words(url)],
        'avg_len_random_words':[avg_len_random(url)],
        'http_in_URL':[url.count('http')],
        'semicolon_in_URL': [semicolon_in_URL(url)],
        'AND_in_URL':[AND_in_URL(url)],
        'URL_length': [URL_length(url)],
        'alphabets_in_URL':[letter_count(url)],
        'uppercase_letters_inURL': [uppercase_count(url)],
        'lowercase_letters_inURL':[lowercase_count(url)],
        'uppercase_letters_ratio_inURL':[uppercase_ratio(url)],
        'lowercase_letters_ratio_inURL': [lowercase_ratio(url)],
        'Special_char_in_URL': [spl_char_count(url)],
        'numbers_ratio_inURL': [number_ratio(url)],
        'alphabet_ratio_inURL': [alphabet_ratio(url)],
        'fd_length':[fd_length(url)],
        'fd_length2':[fd_length(url)],
        'tld_length':[ tld_length(get_tld(url,fail_silently=True))],
        'tld_length2':[ tld_length(get_tld(url,fail_silently=True))],
        'has_server': [has_server_in_string(url)],
        'has_login': [has_login_in_string(url)],
        'has_client': [has_client_in_string(url)],
        'has_admin': [has_admin_in_string(url)],
```

```
        'count_dir': [no_of_dir(url)],
        'count_dir2': [no_of_dir(url)],
        'domain_length': [domain_len(url)],
        'dots_in_domain': [dots_in_domain(url)],
        'short_url':[shortening_service(url)],
        'short_url2':[shortening_service(url)],
        'short_url3':[shortening_service(url)],
        'number_subdom': [number_subdomain(url)],
        'hyphens_in_domain': [hyphens_in_domain(url)],
         }

# Create a DataFrame from the new_data dictionary
        new_data_df = pd.DataFrame(new_data)


# Use the trained model to make predictions
        prediction = model.predict(new_data_df)

           #prediction = classifier.predict(X_pred)
        print(prediction)
        if (prediction == 1):
              s = "The url is malicious"
              print(s)

        else:
              s = "The url is legitimate"
              print(s)

        return render(request, "result.html",{"URL":url,"OUTPUT":s})
```

report1.html

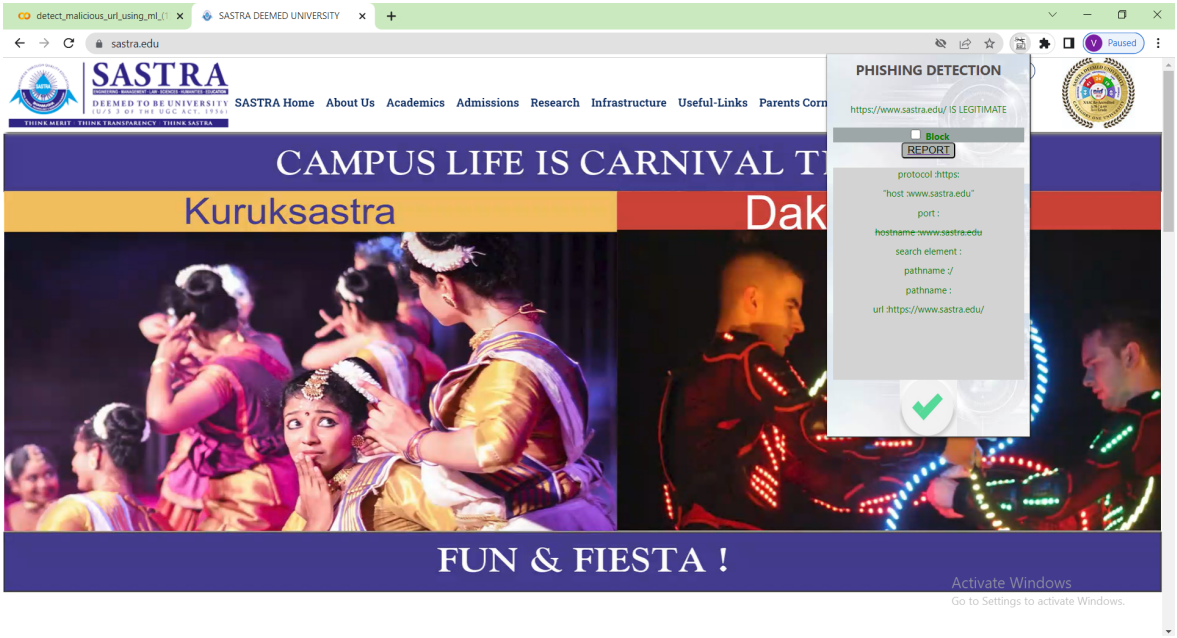
```
<!DOCTYPE html>
<html>
<head>
   <meta charset="UTF-8">
   <title>Report</title>
</head>
<style>
    body {background-repeat: no-repeat;
```

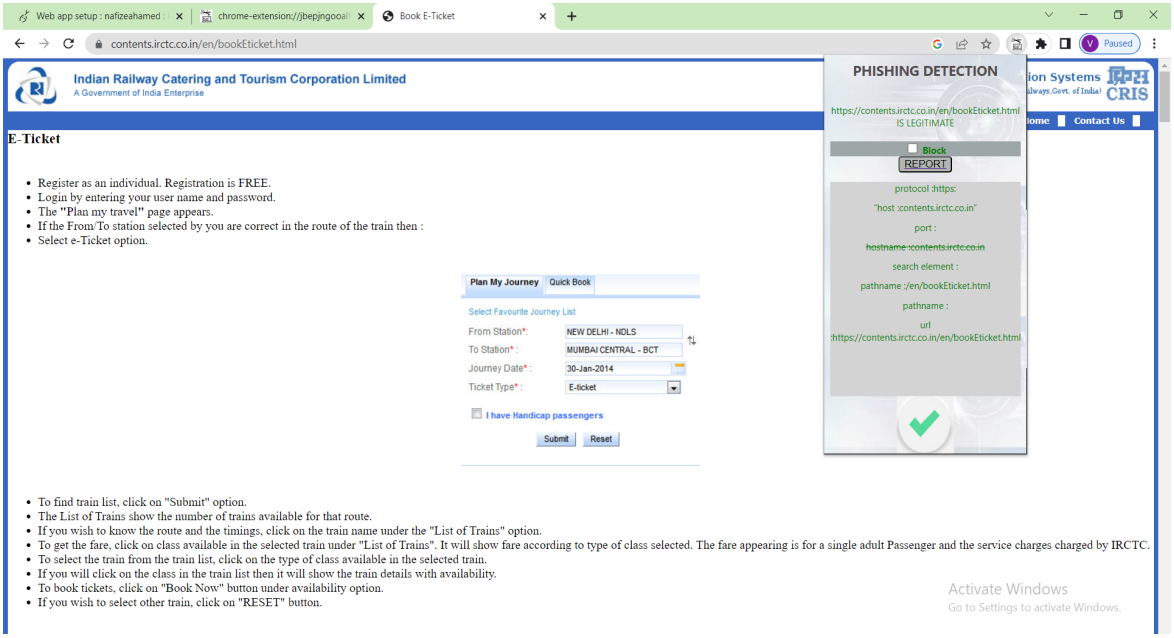
```
    background-size:cover;
  }
    h2{
       color:rgb(114, 8, 8);
    }
    form{
       text-align: center;
       font-size:larger;
    }
    fieldset{
  border: 4px solid rgb(15, 15, 15);
  float: center;
}
</style>
<body background="bgm.JPEG">
    <form action="" method="POST">
       <!--<fieldset style="width:1480px">-->
       <legend><U><h2>FEEDBACK REPORT</h2></U></legend>
       <b>Username <input type="text" name="name" placeholder="Enter the
name"></b><br>
       <br>
                   <b>URL Link<input type="text" name="url" placeholder="Enter the
URL"></b><br>
                   <br>
       <label for="url_type"><b>Type:</b></label>
          <select id="url_type" name="url_type">
             <option hidden disabled selected value>select</option>
             <option value="Phishing">Phishing</option>
             <option value="Legitimate">Legitimate</option>
          </select>
       <input type="submit" value="Submit"><br>
       </form>
       <!--</fieldset>-->
</body>
</html>
```

CHAPTER 4

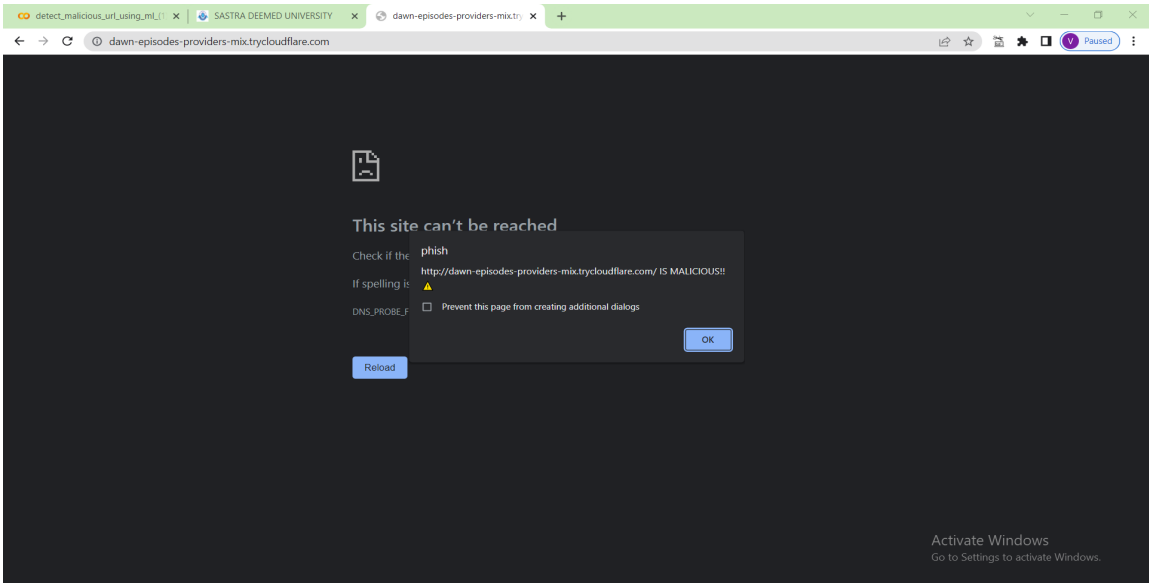
OUTPUT SNAPSHOTS

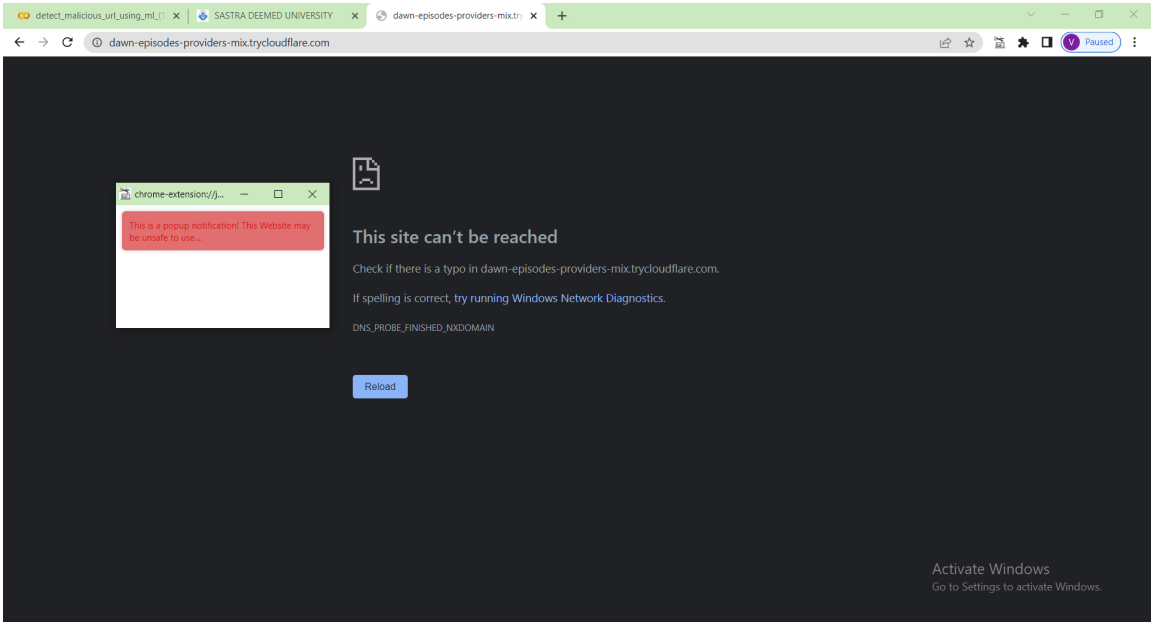
Legitimate URL

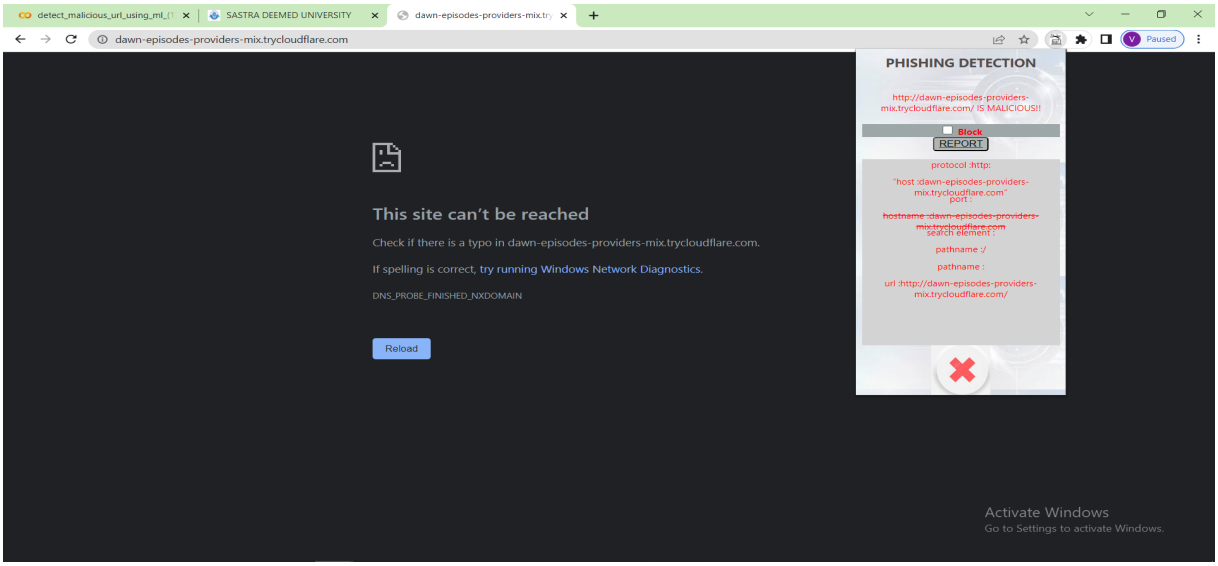


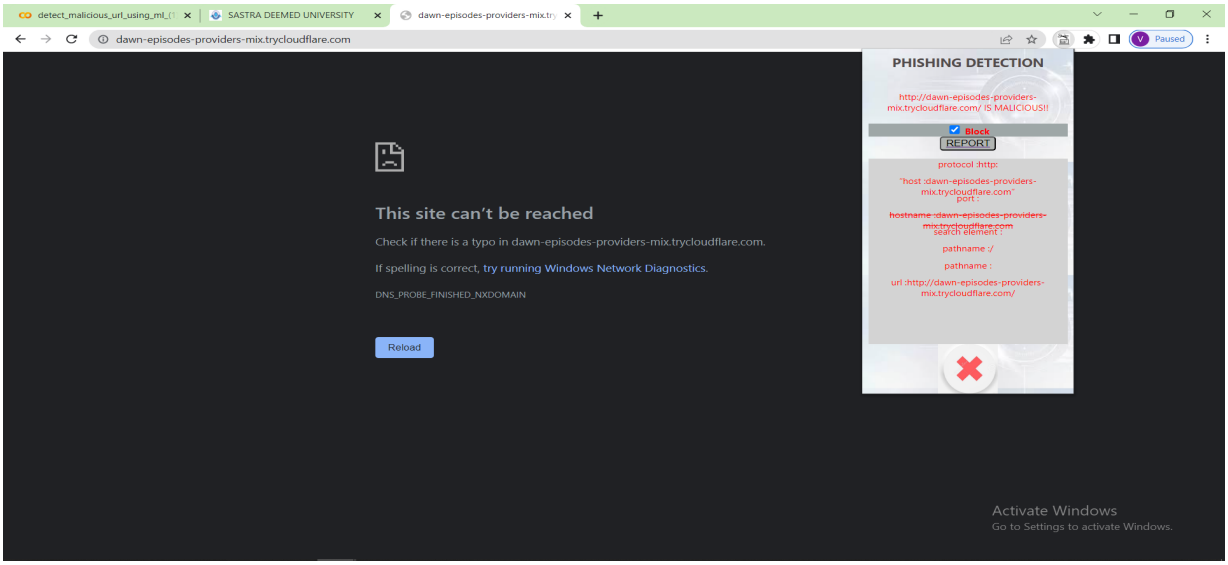


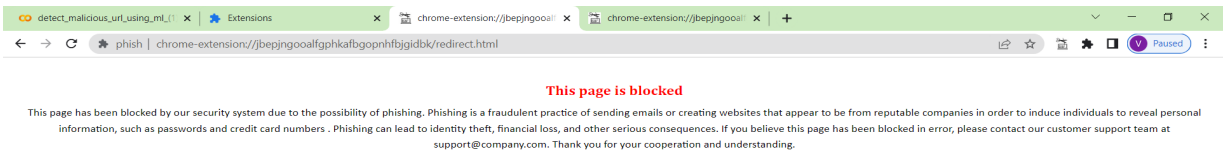
Phishing URL



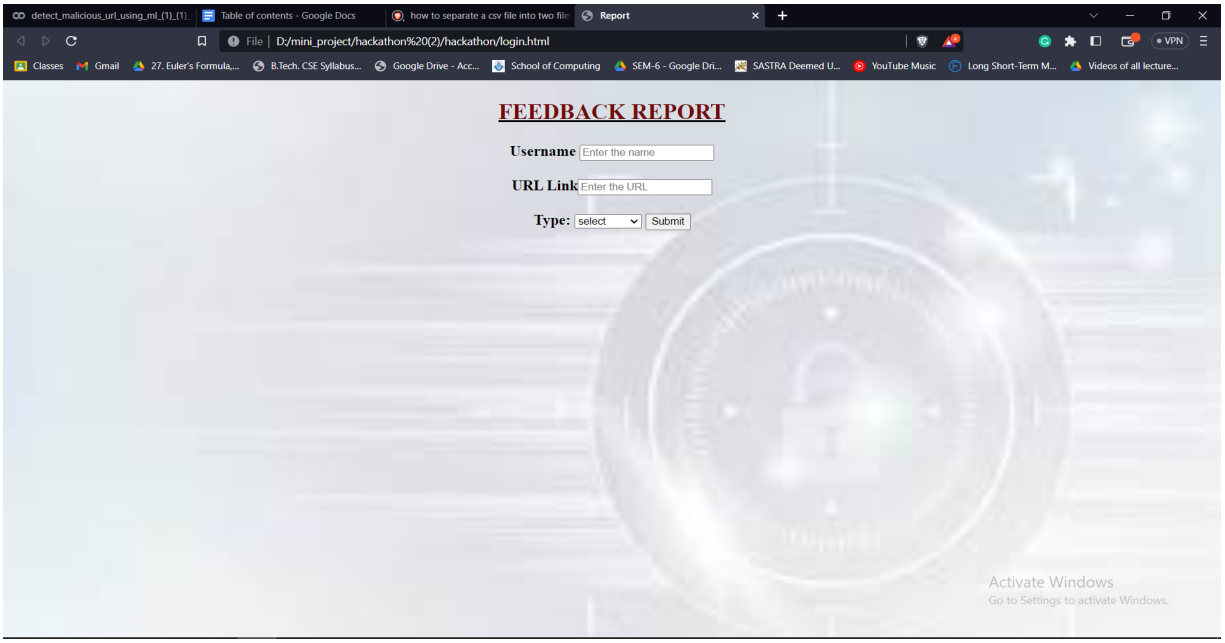



detect_malicious_url_using_ml_(1
SASTRA DEEMED UNIVERSITY
dawn-episodes-providers-mix.try
dawn-episodes-providers-mix.trycloudflare.com
Paused
This site can't be reached
Check if there is a typo in dawn-episodes-providers-mix.trycloudflare.com.
If spelling is correct, try running Windows Network Diagnostics.
DNS_PROBE_FINISHED_NXDOMAIN
Reload
PHISHING DETECTION
http://dawn-episodes-providers-mix.trycloudflare.com/ IS MALICIOUS!!
Block
REPORT
protocol :http:
"host :dawn-episodes-providers-mix.trycloudflare.com"
port :
hostname :dawn-episodes-providers-mix.trycloudflare.com
search element :
pathname :/
pathname :
url :http://dawn-episodes-providers-mix.trycloudflare.com/
Activate Windows
Go to Settings to activate Windows.

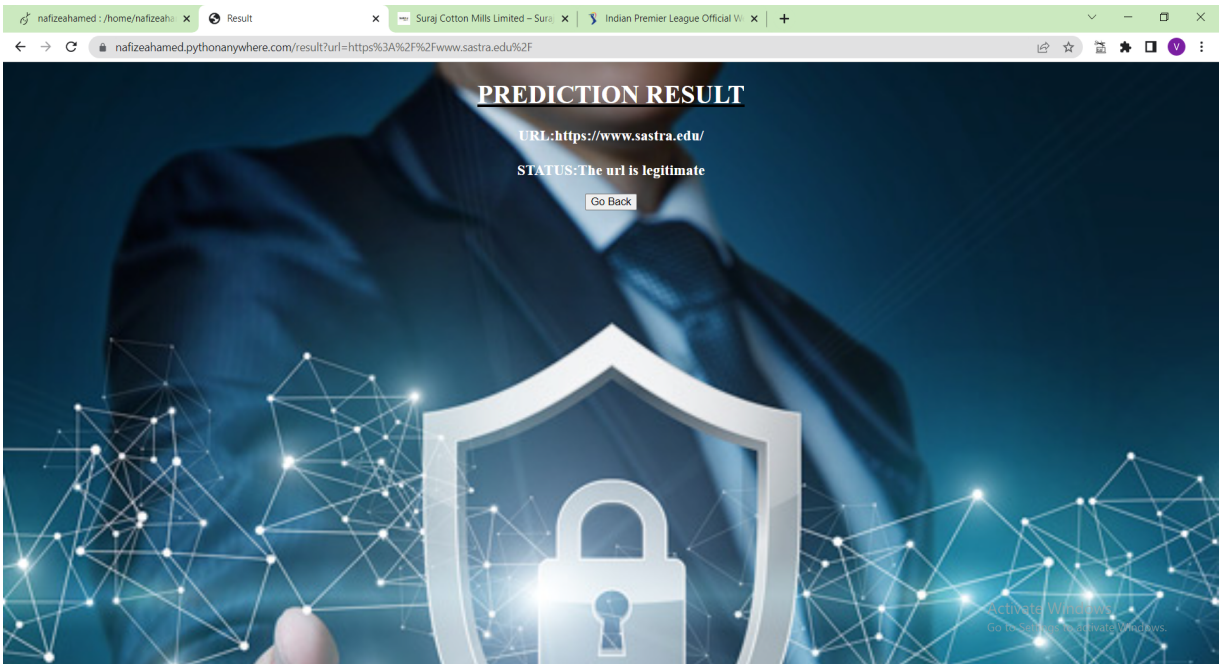

detect_malicious_url_using_ml_(1
SASTRA DEEMED UNIVERSITY
dawn-episodes-providers-mix.try
dawn-episodes-providers-mix.trycloudflare.com
Paused
This site can't be reached
Check if there is a typo in dawn-episodes-providers-mix.trycloudflare.com.
If spelling is correct, try running Windows Network Diagnostics.
DNS_PROBE_FINISHED_NXDOMAIN
Reload
PHISHING DETECTION
http://dawn-episodes-providers-mix.trycloudflare.com/ IS MALICIOUS!!
Block
REPORT
protocol :http:
"host :dawn-episodes-providers-mix.trycloudflare.com"
port :
hostname :dawn-episodes-providers-mix.trycloudflare.com
search element :
pathname :/
pathname :
url :http://dawn-episodes-providers-mix.trycloudflare.com/
Activate Windows
Go to Settings to activate Windows.


detect_malicious_url_using_ml_(1
Extensions
chrome-extension://jbepjngooalf
chrome-extension://jbepjngooalf
phish | chrome-extension://jbepjngooalfgphkafbgopnhfbjgidbk/redirect.html
Paused
This page is blocked
This page has been blocked by our security system due to the possibility of phishing. Phishing is a fraudulent practice of sending emails or creating websites that appear to be from reputable companies in order to induce individuals to reveal personal information, such as passwords and credit card numbers . Phishing can lead to identity theft, financial loss, and other serious consequences. If you believe this page has been blocked in error, please contact our customer support team at support@company.com. Thank you for your cooperation and understanding.

FEEDBACK REPORT



PHISHING DETECTION



CHAPTER 5

CONCLUSION AND FUTURE WORK

There are many deep-learning and ML-based solutions for phishing attacks that have been proposed in recent years, but results have not been verified in real-time browsing environments. In this project, we proposed a framework for phishing detection in a live browsing environment.

The Novel features are:

1) Deploying the whole system to a cloud platform

2) The web page can be blocked if the user wishes.

3) The feedback data from users are updated data with accuracy and sensitivity.

4) Analyzing the URL based on DNS records, SSL/tls certificates,etc.

5) The feature extraction process in the ML or DL model.

In the Future, users can download the plugin through the Chrome Web Store. In addition, we plan to implement our framework as a plug-in for other browsers.

CHAPTER 6

REFERENCES:

- https://developer.chrome.com/docs/extensions/
- URL 2016 | Datasets | Research | Canadian Institute for Cybersecurity | UNB. Accessed: Oct. 20, 2021. www.unb.ca. [Online]. Available: https://www.unb.ca/cic/datasets/url-2016.html
- PhishTank > See All Suspected Phish Submissions. Accessed: Oct. 20, 2021. www.phishtank.com.[Online]. Available: https://www. phishtank.com/phish_archive.php
- M. Sánchez-Paniagua, E. F. Fernández, E. Alegre, W. Al-Nabki and V. González-Castro, "Phishing URL Detection: A Real-Case Scenario Through Login URLs," in IEEE Access, vol. 10, pp. 42949-42960, 2022, doi: 10.1109/ACCESS.2022.3168681.
- P. Yang, G. Zhao and P. Zeng, "Phishing Website Detection Based on Multidimensional Features Driven by Deep Learning," in IEEE Access, vol. 7, pp. 15196-15209, 2019, doi: 10.1109/ACCESS.2019.2892066.