## 1. PROPOSED APPROACH

Our objective is to create an android application which checks the legitimacy of the URL entered by the user. The proposed approach for it is mainly divided into 4 stages which is illustrated in Figure 1. The first stage checks whether Domain name is an IP or not. The second stage checks whether the URL has already been verified before and in the third stage, various features have been extracted from the URL and the last stage is classifying whether the website is legitimate or a malicious one and updating the database accordingly.

**Stage 1:**

If IP address has been used, then the system will warn the user that it might be a phishing/malicious site. Here only Warning is issued because IP address can be used by some legitimate sites also. If IP has not been used as Domain name, then system moves to stage 2.

**Stage 2:**

The system maintains its own database which contains all the URL of the websites that has been already checked by system and reported under the malicious or legitimate category. Whenever user clicks for a website, to minimize the wastage of time, before starting the analysis, we first determine through the database if the particular URL has already been checked, if yes, it informs the user about the legitimacy of the website accordingly. If the URL does not exist in the database, then system enters stage 3 for further analysis.

**Stage 3:**

In this stage, the application sends the URL to a server. It extracts a set of features that will be used to decide the legitimacy of the website. There are many feature groups which have been considered by the system to form the feature set as described in section 3.1. These groups constitute of related features which have been taken by studying various proposed solutions and their effectiveness.

### 1.1 Feature extraction

A web page has many components which includes HTML, Javascript, images and Unified Resource Locator. The web pages running on mobile devices can further communicate with different apks in the user's device using different tools. Our focus is on extracting mobile relevant features from the URLs collected in the data collection process because we believe that they are notable indicators for the legitimacy of the website. We have considered 34 features as illustrated in Table 1.

- **Benign Probability and Malicious Probability:** The probability of a URL to be benign or malicious is calculated using Naive Bayes classifier and added as a feature. It is a collection of classification algorithms based on Bayes theorem. It is a simple and fast algorithm and works well even when there is a small amount of training data. It gives an advantage of using two algorithms at the same time, Naïve Bayes being the supporting algorithm.

Table 1. Features in our dataset

| Feature Group | Features | Total |
|---|---|---|
| Mobile Specific | Number of sms, tel, mms, apk API calls | 4 |
| Javascript | Presence of JS, noscript, internal JS and external Js<br>Number of Js, noscript, internal JS and external JS | 8 |
| HTML | Presence of images, links, iframes and redirects<br>Number of images, links, iframes and redirects | 8 |
| URL | Length of URL, Number of forward slashes, question marks, dots, hyphens, underscores, equal signs, ampersand, semicolon and digits | 10 |
| Site popularity | World-wide traffic rank, country traffic rank | 2 |
| - | Benign and malicious probability | 2 |

Feature analysis is performed to get important indicators for the model and then various classification algorithms has been applied and evaluated which will classify the URL into either a legitimate or a malicious site. After the detection, the system proceeds to the final stage.

**Stage 4:**

In this stage, the detected website URL will be added to the database handled by the server for future use so that if the same URL is being requested by the user again, it can be detected at an earlier stage. Now, the result is sent to the application which displays it to the user.

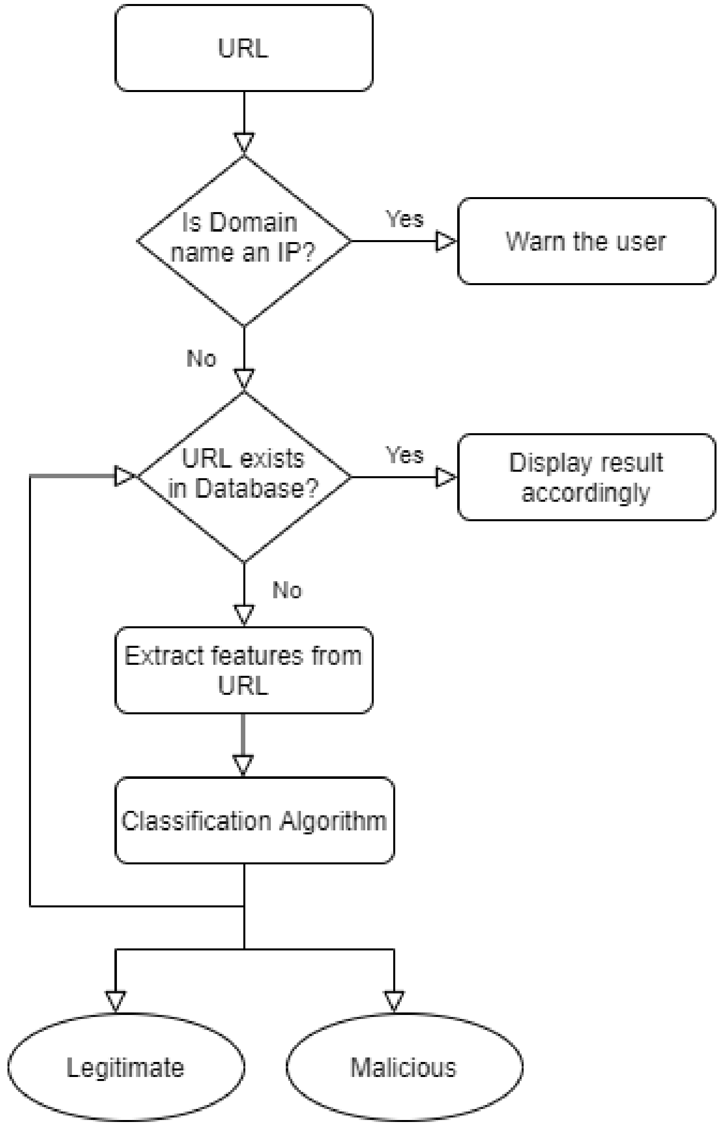



Fig 1. Flowchart of Proposed approach

## 2. DATA FLOW DIAGRAM

## 5.1 Level 0 DFD

Figure 2 shows the data LEVEL 0 DFD for Malicious web page detection system implemented in android using machine learning.

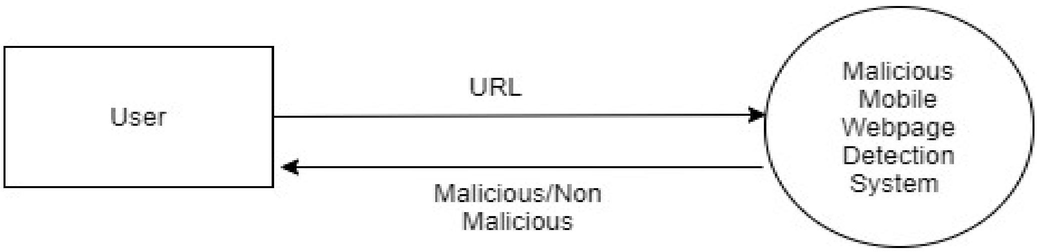



Fig 2. Level 0 DFD for Malicious mobile web page detection system.

## 5.2 Level 1 DFD

Figure 3 represents the different processes that are incorporated in the project

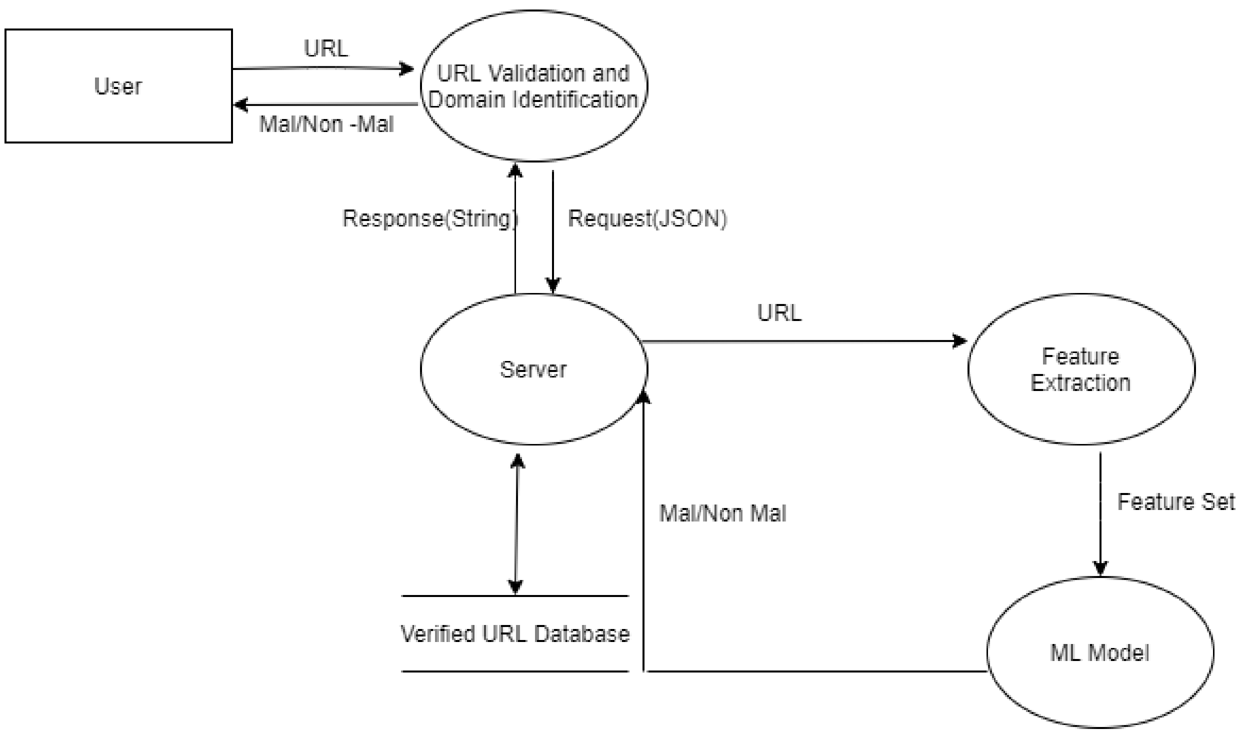



Fig 3. Level 1 DFD for malicious mobile web page detection system

## 5.3 Level 2 DFD

**i) Level 2 DFD for Feature extraction Process:** The figure 4 represnts all the different feature group that are considered for this project

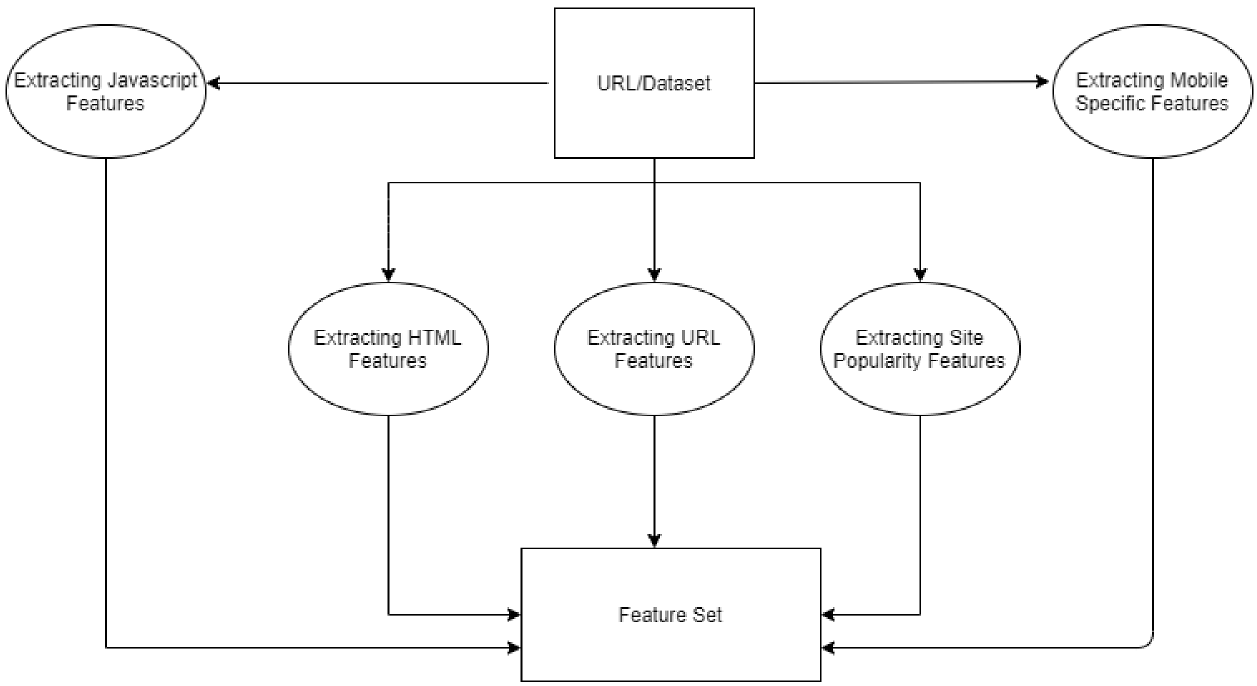



Fig 4. Level 2 DFD for feature extraction process

**ii) Level 2 DFD for ML Model:** This represents all the steps taken to create a ML model for the backend of this project.

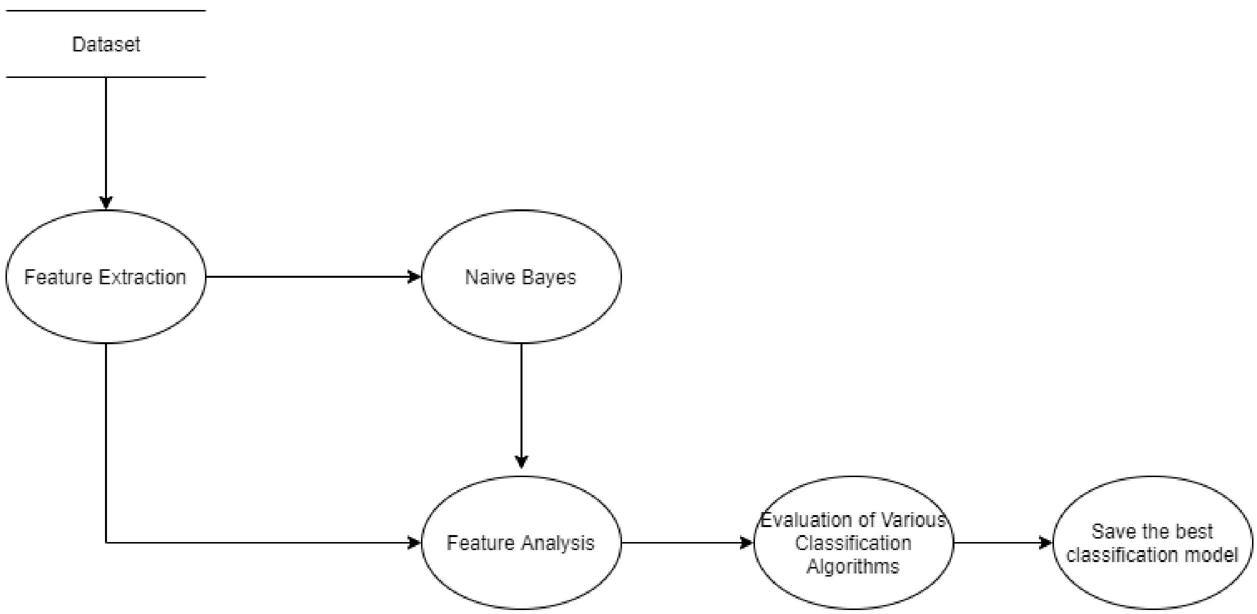



Fig 5. Level 2 DFD for ML model

## 3. IMPLEMENTATION

The architecture of our detection of mobile malicious web page system is as shown in Fig 6. We create an end point for the user to interact with our system using his/her mobile device. This system works for any android device which has an active internet connection and has an Android version greater than or equal to 6.0 (Marshmallow). The implementation of the processes that been used to create our entire system is as follows.

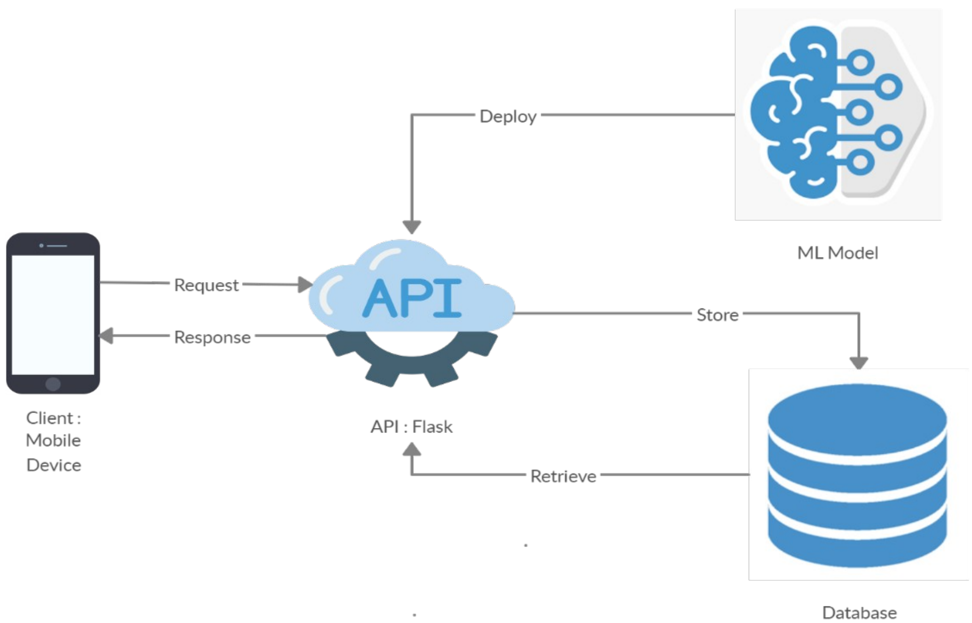



Fig 6. Architecture of the Malicious Mobile Webpage Detection Application

### 3.1 Data collection and Extracting Features

Our data collection process includes gathering of labelled benign and malicious mobile specific web pages. We first, understand and identify what are mobile specific web pages. Mobile specific web pages include the web pages which have different URL in mobile and desktop browser. We analyze the URLs of such pages and identify the significant characteristics of them [Table 2]. We have manually collected 697 benign URLs by obtaining popular websites from Alexa [14] from an Android mobile browser. For the malicious URLs, we have collected 330 mobile specific malicious URLs from OpenPhish [15] using algorithm based on the characteristics shown in Table 2. This results in a dataset of size 1027.

We have extracted all the relevant features in section 4.1 from the following feature groups – mobile specific features, HTML features, URL features and site popularity features. We have ten applied Naïve Bayes to the features set and calculated the benign and malicious probability of each feature tuple and have added them as a feature. All this has been done in Spyder IDE in Windows 10 Operating System.

Table 2. Mobile Web page Indicators

| | |
|---|---|
| Top Level Domain | .mobi |
| Sub domain | m.,mobile.,touch.,3g.,sp.,s.,mini.,mobileweb.,t. |
| URL Path Prefix | /mobile, /mobileweb, /m, /mobi, /?m=1, /mobil, /m_home |

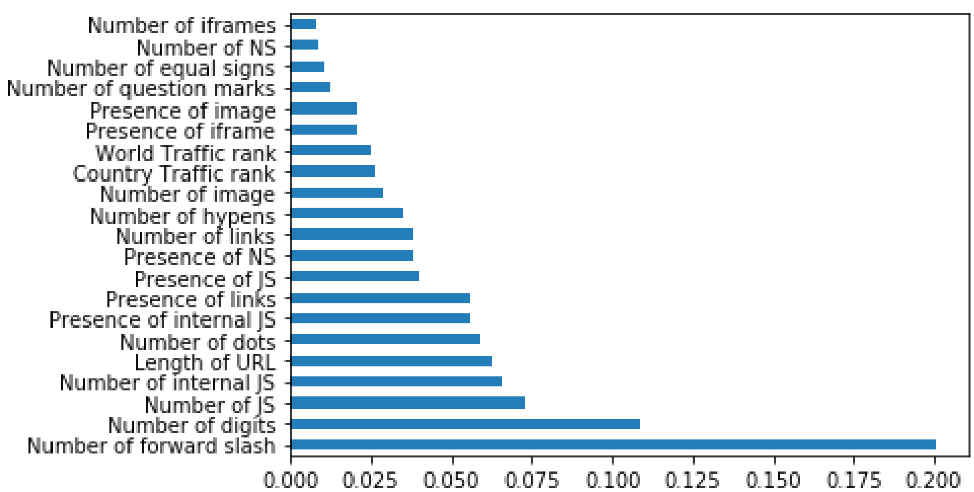



Fig 7. Representation of Feature Importance

### 3.2 Flask API

Now that we have the model ready, we save the model using pickle library. We create an API (endpoint) for the saved model by using Flask micro web framework. It has been implemented in python and returns a string stating Malicious or Non malicious as response. A database containing already verified URLs is stored in a file. When the server starts, it reads the contents of the file and converts it into a dictionary. The URL when received from a client in the form of JSON object, is first checked through the dictionary and a faster response is given when the URL is found in it. If the URL doesn't exists in it, the legitimacy of it is found through our model and is updated in the dictionary and file so that the result of it can be retained throughout the server session and even after the server restarts respectively.

### 3.3 Android Application

An android application acts as a client in which the user can enter the URL he/she wants to check. The android device first checks if the entered text is a URL or not. If it is a URL, it checks if the URL's domain is a name or an IP address. It displays a warning if it is an IP address. If not, the device sends the URL to the server (created by Flask) through JSON object, the legitimacy of the webpage is determined and a string is returned as response. The response is obtained by the android device which displays the result to the user. The android application is implemented in android studio and emulator having Android 6.0 with API 23 has been used to test and run the application.

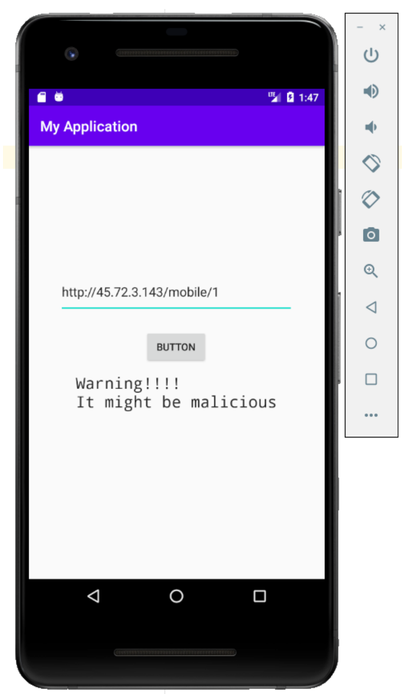
## 4. RESULTS AND OBSERVATIONS

The prepared feature set is split into training and test set. Various classification algorithms such as Logistic regression, K nearest neighbors, support vector machine, decision tree and random forest classification have been applied and checked against the test set and their respective true positive rate(TPR), true negative rate(TNR), accuracy, precision and f1 score values are calculated and a comparison is drawn between them (Table 3). By analyzing the below table, we have determined that Random Forest Classification works best for our model. An endpoint (API) has been made for our model which communicates with Android device to display results to the user. An emulator with Android 6 and API 23 has been used and few results of different URLs are shown in the below figure (Fig 2).

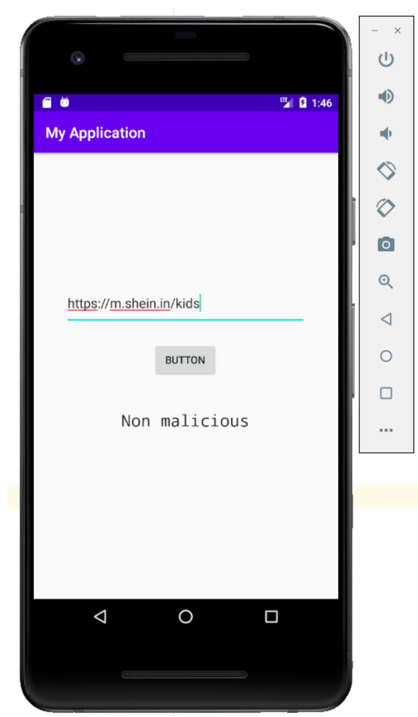
Table 3. Comparison between various classification algorithms

| Algorithm | TPR | TNR | Accuracy | Precision | F1 score |
|---|---|---|---|---|---|
| Logistic Regression | 0.7368 | 0.625 | 0.7282 | 0.7368 | 0.8334 |

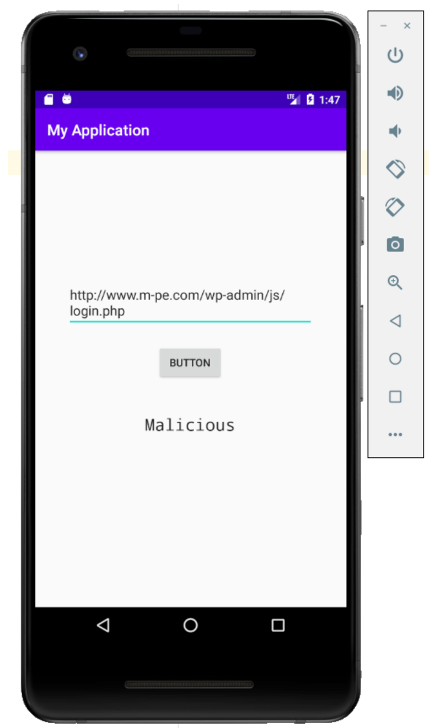
| | | | | | |
|---|---|---|---|---|---|
| K Nearest Neighbors | 0.9857 | 0.8788 | 0.9515 | 0.9857 | 0.9650 |
| Support Vector Machine | 0.7087 | nan | 0.7087 | 0.7087 | 0.8295 |
| Decision Tree | 0.9863 | 0.9667 | 0.9806 | 0.9863 | 0.9863 |
| Random Forest | 1.0 | 0.9677 | 0.9903 | 1 | 0.9931 |




(a)




(b)




(c)

Fig 8. Screenshots of emulator's results for different types of URLs

## 5. SOFTWARE AND HARDWARE REQUIREMENTS

- Android Studio 3.6
- SDK
- Emulator (AVD) with Android 6 and API 23
- Spyder IDE
- Python (version 2.7.13+)

  Python libraries required:

  Flask
  pickle
  json
  BeautifulSoup
  urllib
  request
  numpy
  pandas
  math
  sklearn.model_selection
  sklearn.linear_model
  sklearn.metrics
  sklearn.neighbors
  sklearn.svm
  sklearn.tree
  sklearn.ensemble
- Operating system

  The project has been built and tested on Windows 10 (64 Bit O S)

## 6. REFERENCES

[1] Yousif, H., Al-saedi, K. H., & Al-Hassani, M. D. (2019). Mobile Phishing Websites Detection and Prevention Using Data Mining Techniques. *International Journal of Interactive Mobile Technologies*, *13*(10).

[2] Joshi, R. (2015). Interactive phishing filter. Available at: https://scholarworks.sjsu.edu/cgi/viewcontent.cgi?article=1426&context=etd_projects (Last accessed on 26 May 2020)

[3] Shcherbakova, T., Vergelis, M., & Demidova, N. (2015). Spam and phishing in Q2 2015. *Quarterly Spam Reports*. Available at: https://securelist.com/spam-and-phishing-in-q2-of-2015/71759/ (Last accessed on 26 May 2020)

[4] Crane C. (2019). 20 Phishing Statistics to Keep You from Getting Hooked in 2019. Available at https://www.thesslstore.com/blog/20-phishing-statistics-to-keep-you-from-getting-hooked-in-2019/ (Last accessed on 26 May 2020)

[5] Arghire I. (2019). Phishers Serve Fake Login Pages via Google Translate. Available at https://www.securityweek.com/phishers-serve-fake-login-pages-google-translate (Last accessed on 26 May 2020)

[6] Crane C. (2020). Coronavirus Scams: Phishing Websites & Emails Target Unsuspecting Users. Available at https://www.thesslstore.com/blog/coronavirus-scams-phishing-websites-emails-target-unsuspecting-users/ (Last accessed on 26 May 2020)

[7] S. O'Dea. (2020). Number of smartphones sold to end users worldwide from 2007 to 2020*(in million units)* Available at: https://www.statista.com/statistics/263437/global-smartphone-sales-to-end-userssince-2007/ (last accessed: 26 May 2020).

[8] Li, L., Berki, E., Helenius, M., & Ovaska, S. (2014). Towards a contingency approach with whitelist-and blacklist-based anti-phishing applications: what do usability tests indicate?. *Behaviour & Information Technology*, *33*(11), 1136-1147.

[9] Bian, R. M. (2013). Alice in battlefield: an evaluation of the effectiveness of various UI phishing warnings.

[10] Gastellier-Prevost, S., Granadillo, G. G., & Laurent, M. (2011, February). A dual approach to detect pharming attacks at the client-side. In *2011 4th IFIP International Conference on New Technologies, Mobility and Security* (pp. 1-5). IEEE.

[11] Ferolin, R. J., & Kang, C. U. (2012). Phishing attack detection, classification and proactive prevention using fuzzy logic and data mining algorithm.

[12] Moghimi, M., & Varjani, A. Y. (2016). New rule-based phishing detection method. *Expert systems with applications*, *53*, 231-242.

[13] Amrutkar, C., Kim, Y. S., & Traynor, P. (2016). Detecting mobile malicious webpages in real time. *IEEE Transactions on Mobile Computing*, *16*(8), 2184-2197.

[14] Alexa, the web information company. Available at: http://www.alexa.com/topsites (Last accessed on 24 May 2020)

[15] Open Phish. Available at: https://openphish.com/phishing_feeds.html (Last accessed on 24 May 2020)

[16] Y. Zhang, J. I. Hong, and L. F. Cranor, "Cantina: A content-based approach to detecting phishing web sites," in Proc. 16th Int. Conf. World Wide Web, 2007, pp. 639–648

## APPENDIXES:

## A. COMPLETE CONTRIBUTORY SOURCE CODE:

### 1. Python code to create the ML model

```
import pandas as pd

from math import sqrt

from math import pi

from math import exp

import numpy as np

import requests

from bs4 import BeautifulSoup

import urllib,bs4

#algorithm to extract mobile specific urls
```

```
y=set()

r1=set()

f1=open('feed.txt','r') #feed.txt contains malicious sites from OpenPhish

y=f1.readlines()

for z in y:

    res=False

    url=z.strip()

    x1=url.split('/')

    x2=list()

    for i in x1:

        x3=i.split('.')

        for j in x3:

            x2.append(j)

    for i in x2:

            if (i=='mobi' or i=='m' or i=='mobile' or i=="touch" or i=='3g' or i=='sp' or i=='s' or
i=='mini' or i=='mobileweb' or i=='t' or i=='?m=1'or i=='mobil' or i=='m_home' ):

            res=True

    if(res==True):

        r1.add(url)

num_mal=len(r1)

print("Number of mobile specific malicious sites : ", num_mal)
```

```
features=list()

#extractig features from malicious sites

for URL in r1:

    try :

        r = requests.get(URL)

    except :

        continue

    temp_list=list()

    soup = BeautifulSoup(r.content, 'html5lib')

    #print(soup)

    p_js=0

    c_js=0

    for row in soup.findAll('script'):

        c_js=c_js+1

    if c_js != 0 :

        p_js=1

    p_ns=0

    c_ns=0

    for row in soup.findAll('noscript'):

        c_ns=c_ns+1

    if c_ns != 0 :
```

```
        p_ns=1

    p_ejs=0

    p_ijs=0

    c_ejs=0

    for row in soup.findAll('script',attrs='src'):

        c_ejs=c_ejs+1

    c_ijs=c_js-c_ejs

    if c_ejs != 0 :

        p_ejs=1

    if c_ijs != 0 :

        p_ijs=1

    p_img=0

    c_img=0

    for row in soup.findAll('img'):

        c_img=c_img+1

    if c_img != 0 :

        p_img=1

    p_iframe=0

    c_iframe=0

    for row in soup.findAll('iframe'):

        c_iframe=c_iframe+1
```

```
if c_iframe != 0 :
    p_iframe=1
p_rdirect=0
c_rdirect=0
for row in soup.findAll('meta', attrs={'http-equiv':'Refresh' }):
    c_rdirect=c_rdirect+1
if c_rdirect != 0 :
    p_rdirect=1


p_lnks=0
temp=list()
for row in soup.findAll('a'):
    temp.append(row.get('href'))
c_lnks=len(temp)
if c_lnks != 0 :
    p_lnks=1
temp_list.append(p_js)
temp_list.append(p_ns)
temp_list.append(p_ejs)
temp_list.append(p_ijs)
temp_list.append(p_img)
```

```
temp_list.append(p_iframe)

temp_list.append(p_rdirect)

temp_list.append(p_lnks)

temp_list.append(c_js)

temp_list.append(c_ns)

temp_list.append(c_ejs)

temp_list.append(c_ijs)

temp_list.append(c_img)

temp_list.append(c_iframe)

temp_list.append(c_rdirect)

temp_list.append(c_lnks)

c_sms=0

c_tel=0

c_apk=0

c_mms=0

for s in temp:

    if(isinstance(s,str)):

        if(s.startswith('sms')):

            c_sms=c_sms+1

        if(s.startswith('tel')):

            c_tel=c_tel+1
```

```
        if(s.endswith('apk')):

            c_apk=c_apk+1

        if(s.startswith('mms')):

            c_mms=c_mms+1

    temp_list.append(c_sms)

    temp_list.append(c_tel)

    temp_list.append(c_apk)

    temp_list.append(c_mms)

    url_len= len(URL)

    #print(url_len)

    temp_list.append(url_len)

    num_fslash=0

    num_qm=0

    num_dots=0

    num_hypen=0

    num_uscore=0

    num_eqls=0

    num_amp=0

    num_smcolon=0

    num_digi=0

    for i in URL:
```

```
        if i== '/' :

            num_fslash=num_fslash+1

        if i=='?' :

            num_qm=num_qm+1

        if i=='.' :

            num_dots=num_dots+1

        if i=='-' :

            num_hypen=num_hypen+1

        if i=='_' :

            num_uscore=num_uscore+1

        if i=='=' :

            num_eqls=num_eqls+1

        if i=='&' :

            num_amp=num_amp+1

        if i==';' :

            num_smcolon=num_smcolon+1

        if i=='0' or i=='1' or i=='2' or i=='3' or i=='4' or i=='5' or i=='6' or i=='7' or i=='8' or i=='9' :

            num_digi=num_digi+1

    temp_list.append(num_fslash)

    temp_list.append(num_qm)

    temp_list.append(num_dots)
```

```
    temp_list.append(num_hypen)

    temp_list.append(num_uscore)

    temp_list.append(num_eqls)

    temp_list.append(num_amp)

    temp_list.append(num_smcolon)

    temp_list.append(num_digi)




        try:     traff_rnk=int(bs4.BeautifulSoup(urllib.request.urlopen("http://data.alexa.com/data?
cli=10&dat=s&url=" + URL).read(), "xml").find("REACH")["RANK"])

    except:

        traff_rnk=0

    temp_list.append(traff_rnk)

    try:

cntry_traff_rnk=int(bs4.BeautifulSoup(urllib.request.urlopen("http://data.alexa.com/data?
cli=10&dat=s&url=" + URL).read(), "xml").find("COUNTRY")["RANK"])

    except:

        cntry_traff_rnk=0

    temp_list.append(cntry_traff_rnk)

    temp_list.append(1)

    features.append(temp_list)
```

```
#extracting features from legitimate sites

f2=open('legitimate_sites.txt','r')

urls=f2.readlines()

r2=set()

for URL in urls:

    URL=URL.strip()

    r2.add(URL)

for URL in r2:

    try :

        r = requests.get(URL)

    except :

        continue

    temp_list=list()

    soup = BeautifulSoup(r.content, 'html5lib')

    #print(soup)

    p_js=0

    c_js=0

    for row in soup.findAll('script'):

        c_js=c_js+1

    if c_js != 0 :

        p_js=1
```

```
p_ns=0

c_ns=0

for row in soup.findAll('noscript'):

    c_ns=c_ns+1

if c_ns != 0 :

    p_ns=1

p_ejs=0

p_ijs=0

c_ejs=0

for row in soup.findAll('script',attrs='src'):

    c_ejs=c_ejs+1

c_ijs=c_js-c_ejs

if c_ejs != 0 :

    p_ejs=1

if c_ijs != 0 :

    p_ijs=1

p_img=0

c_img=0

for row in soup.findAll('img'):

    c_img=c_img+1

if c_img != 0 :
```

```
        p_img=1

    p_iframe=0

    c_iframe=0

    for row in soup.findAll('iframe'):

        c_iframe=c_iframe+1

    if c_iframe != 0 :

        p_iframe=1

    p_rdirect=0

    c_rdirect=0

    for row in soup.findAll('meta', attrs={'http-equiv':'Refresh' }):

        c_rdirect=c_rdirect+1

    if c_rdirect != 0 :

        p_rdirect=1

    p_lnks=0

    temp=list()

    for row in soup.findAll('a'):

        temp.append(row.get('href'))

    c_lnks=len(temp)

    if c_lnks != 0 :

        p_lnks=1

    temp_list.append(p_js)
```

```
temp_list.append(p_ns)

temp_list.append(p_ejs)

temp_list.append(p_ijs)

temp_list.append(p_img)

temp_list.append(p_iframe)

temp_list.append(p_rdirect)

temp_list.append(p_lnks)

temp_list.append(c_js)

temp_list.append(c_ns)

temp_list.append(c_ejs)

temp_list.append(c_ijs)

temp_list.append(c_img)

temp_list.append(c_iframe)

temp_list.append(c_rdirect)

temp_list.append(c_lnks)

c_sms=0

c_tel=0

c_apk=0

c_mms=0

for s in temp:

    if(isinstance(s,str)):
```

```
        if(s.startswith('sms')):
            c_sms=c_sms+1
        if(s.startswith('tel')):
            c_tel=c_tel+1
        if(s.endswith('apk')):
            c_apk=c_apk+1
        if(s.startswith('mms')):
            c_mms=c_mms+1
    temp_list.append(c_sms)
    temp_list.append(c_tel)
    temp_list.append(c_apk)
    temp_list.append(c_mms)
    url_len= len(URL)
    #print(url_len)
    temp_list.append(url_len)
    num_fslash=0
    num_qm=0
    num_dots=0
    num_hypen=0
    num_uscore=0
    num_eqls=0
```

```
num_amp=0

num_smcolon=0

num_digi=0

for i in URL:

    if i== '/' :

        num_fslash=num_fslash+1

    if i=='?' :

        num_qm=num_qm+1

    if i=='.' :

        num_dots=num_dots+1

    if i=='-' :

        num_hypen=num_hypen+1

    if i=='_' :

        num_uscore=num_uscore+1

    if i=='=' :

        num_eqls=num_eqls+1

    if i=='&' :

        num_amp=num_amp+1

    if i==';' :

        num_smcolon=num_smcolon+1

    if i=='0' or i=='1' or i=='2' or i=='3' or i=='4' or i=='5' or i=='6' or i=='7' or i=='8' or i=='9' :
```

```
            num_digi=num_digi+1
    temp_list.append(num_fslash)
    temp_list.append(num_qm)
    temp_list.append(num_dots)
    temp_list.append(num_hypen)
    temp_list.append(num_uscore)
    temp_list.append(num_eqls)
    temp_list.append(num_amp)
    temp_list.append(num_smcolon)
    temp_list.append(num_digi)
     try:           traff_rnk=int(bs4.BeautifulSoup(urllib.request.urlopen("http://data.alexa.com/data?
cli=10&dat=s&url=" + URL).read(), "xml").find("REACH")["RANK"])
    except:
        traff_rnk=0
    temp_list.append(traff_rnk)
      try:        cntry_traff_rnk=int(bs4.BeautifulSoup(urllib.request.urlopen("http://data.alexa.com/
data?cli=10&dat=s&url=" + URL).read(), "xml").find("COUNTRY")["RANK"])
    except:
        cntry_traff_rnk=0
    temp_list.append(cntry_traff_rnk)
    temp_list.append(0)
    features.append(temp_list)
```

df=pd.DataFrame(features, columns = ['Presence of JS' ,'Presence of NS' ,'Presence of external JS' ,'Presence of internal JS' ,'Presence of image' ,'Presence of iframe','Presence of redirects' ,'Presence of links' ,'Number of JS' ,'Number of NS' ,'Number of external JS' ,'Number of internal JS' ,'Number of image' ,'Nuber of iframes' ,'Number of redirects' ,'Number of links' ,'Number of sms API call','Number of tel API call','Number of apk API call','Number of mms API call','Length of URL','Number of forward slash','Number of question marks', 'Number of dots', 'Number of hypens' , 'Number of underscore', 'Number of equal signs', 'Number of ampersand' , 'Number of semi-colon','Number of digits','World Traffic rank','Country Traffic rank','Output'])

```
#Implemeting Naive Bayes from scratch to add as a feature

# Splitting the dataset based on class value

def split(dset):

    split = dict()

    for i in range(len(dset)):

        feature_vector = dset[i]

        val_cls = feature_vector[-1]

        if (val_cls not in split):

            split[val_cls] = list()

        split[val_cls].append(feature_vector)

    return split

def nums_mean(nums):

    return sum(nums)/float(len(nums))

def nums_stdev(nums):

    nums_avg = nums_mean(nums)
```

```
        vari = sum([(x-nums_avg)**2 for x in nums]) / float(len(nums)-1)

        return sqrt(vari)

def smrze(dset):

        smrize = [(nums_mean(column), nums_stdev(column), len(column)) for column in
zip(*dset)]

        del(smrize[-1])

        return smrize

def summarize_by_class(dset):

        separated = split(dset)

        smries = dict()

        for val_cls, rows in separated.items():

                smries[val_cls] = smrze(rows)

        return smries

def cal_prob(x, mn, std):

   expo = exp(-((x-mn)**2 / (2 * std**2 )))

   try:

      return (1 / (sqrt(2 * pi) * std)) * expo

   except:

      return 1

def cal_cls_prob(smries, row):

        tr = sum([smries[label][0][2] for label in smries])
```

```
        probs = dict()

        for cls_val, cls_smries in smries.items():

                probs[cls_val] = smries[cls_val][0][2]/float(tr)

                for i in range(len(cls_smries)):

                        mn, std, _ = cls_smries[i]

                        probs[cls_value] *= cal_prob(row[i], mn, std)

        return probs

arr=df.to_numpy()

#Calculating class probabilities

temp0=list()

temp1=list()

smries = summarize_by_class(arr)

for i in range(len(df)) :

   probability = cal_cls_prob(smries, arr[i])

   temp0.append(probability[0])

   temp1.append(probability[1])

#inserting the calculated class probability

da_frm=pd.DataFrame(arr)

da_frm.insert(32, 'benign_probability', temp0)

da_frm.insert(33, 'malicious_probability', temp1)
```

da_frm.columns= ['Presence of JS' ,'Presence of NS' ,'Presence of external JS' ,'Presence of internal JS' ,'Presence of image' ,'Presence of iframe','Presence of redirects' ,'Presence of links' ,'Number of JS' ,'Number of NS' ,'Number of external JS' ,'Number of internal JS' ,'Number of image' ,'Number of iframes' ,'Number of redirects' ,'Number of links' ,'Number of sms API call','Number of tel API call','Number of apk API call','Number of mms API call','Length of URL','Number of forward slash','Number of question marks', 'Number of dots', 'Number of hypens' , 'Number of underscore', 'Number of equal signs', 'Number of ampersand' , 'Number of semi-colon','Number of digits','World Traffic rank','Country Traffic rank','benign_probability','malicious_probability','Output']

#Feature analysis process

am = da_frm.iloc[:,:-1]

ar = da_frm.iloc[:,-1]

from sklearn.ensemble import ExtraTreesClassifier

import matplotlib.pyplot as plt

mdl = ExtraTreesClassifier()

mdl.fit(am,ar)

features_imp = pd.Series(mdl.feature_importances_, index=am.columns)

features_imp.nlargest(21).plot(kind='barh')

plt.show()

#dropping insignificant features

final_df=df1.drop(['malicious_probability','benign_probability','Presence of external JS' ,'Number of external JS','Number of mms API call', 'Number of semi-colon','Number of redirects','Number of sms API call','Presence of redirects','Number of apk API call','Number of tel API call','Number of underscore','Number of ampersand'],axis=1)

final_df.columns

```
#Final feature matrix and response vector

final_X=final_df.iloc[:,:-1].values

final_y=final_df.iloc[:,-1].values

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(final_X,final_y, test_size = 0.1, random_state=0)

#Applying various Classification Algorithms

from sklearn.linear_model import LogisticRegression

classifierLR = LogisticRegression(random_state = 0)

classifierLR.fit(X_train,y_train)

y_predLR=classifierLR.predict(X_test)

from sklearn.metrics import confusion_matrix

cmLR = confusion_matrix(y_test,y_predLR)

from sklearn import metrics

accuracyLR=metrics.accuracy_score(y_test,y_predLR)

precisionLR=cmLR[0][0]/(cmLR[0][0]+cmLR[1][0])

recallLR=cmLR[0][0]/(cmLR[0][0]+cmLR[0][1])

f1_scoreLR=(2*precisionLR*recallLR)/(precisionLR+recallLR)

TPLR = cmLR[0][0]

FPLR = cmLR[0][1]

FNLR = cmLR[1][0]
```

```
TNLR = cmLR[1][1]

TPRLR=TPLR/(TPLR+FNLR)

TNRLR=TNLR/(TNLR+FPLR)

print("TPR for Logistic Regression algorithm : ",TPRLR)

print("TNR for Logistic Regression algorithm : ",TNRLR)

print("Accuracy of Logistic Regression algorithm : ",accuracyLR)

print("Precision for Logistic Regression algorithm : ",precisionLR)

print("f1 score for Logistic Regression algorithm : ",f1_scoreLR)

from sklearn.neighbors import KNeighborsClassifier

classifierKNN = KNeighborsClassifier(n_neighbors = 9, metric = 'minkowski' , p=2)

classifierKNN.fit(X_train,y_train)

y_predKNN=classifierKNN.predict(X_test)

cmKNN = confusion_matrix(y_test,y_predKNN)

accuracyKNN=metrics.accuracy_score(y_test,y_predKNN)

precisionKNN=cmKNN[0][0]/(cmKNN[0][0]+cmKNN[1][0])

recallKNN=cmKNN[0][0]/(cmKNN[0][0]+cmKNN[0][1])

f1_scoreKNN=(2*precisionKNN*recallKNN)/(precisionKNN+recallKNN)

TPKNN = cmKNN[0][0]

FPKNN = cmKNN[0][1]

FNKNN = cmKNN[1][0]

TNKNN = cmKNN[1][1]
```

```
TPRKNN=TPKNN/(TPKNN+FNKNN)

TNRKNN=TNKNN/(TNKNN+FPKNN)

print("TPR for K Nearest Neighbors algorithm : ",TPRKNN)

print("TNR for K Nearest Neighbors algorithm : ",TNRKNN)

print("Accuracy of K Nearest Neighbors algorithm : ",accuracyKNN)

print("Precision for K Nearest Neighbor algorithm : ",precisionKNN)

print("f1 score for K Nearest Neighbor algorithm : ",f1_scoreKNN)

from sklearn.svm import SVC

classifierSVM = SVC(kernel='rbf',random_state=0)

classifierSVM.fit(X_train,y_train)

y_predSVM=classifierSVM.predict(X_test)

#Calculating the accuracy, precision and recall for Support Vector Machine by checking it against the test set

cmSVM = confusion_matrix(y_test,y_predSVM)

accuracySVM=metrics.accuracy_score(y_test,y_predSVM)

precisionSVM=cmSVM[0][0]/(cmSVM[0][0]+cmSVM[1][0])

recallSVM=cmSVM[0][0]/(cmSVM[0][0]+cmSVM[0][1])

f1_scoreSVM=(2*precisionSVM*recallSVM)/(precisionSVM+recallSVM)

TPSVM = cmSVM[0][0]

FPSVM = cmSVM[0][1]

FNSVM = cmSVM[1][0]
```

```
TNSVM = cmSVM[1][1]

TPRSVM=TPSVM/(TPSVM+FNSVM)

TNRSVM=TNSVM/(TNSVM+FPSVM)

print("TPR for Support Vector Machine algorithm : ",TPRSVM)

print("TNR for Support Vector Machine algorithm : ",TNRSVM)

print("Accuracy of Support Vector Machine algorithm : ",accuracySVM)

print("Precision for Support Vector Machine algorithm : ",precisionSVM)

print("f1 score for Support Vector Machine algorithm : ",f1_scoreSVM)

from sklearn.tree import DecisionTreeClassifier

classifierDT = DecisionTreeClassifier(criterion='entropy',random_state=0)

classifierDT.fit(X_train,y_train)

y_predDT=classifierDT.predict(X_test)

cmDT = confusion_matrix(y_test,y_predDT)

accuracyDT=metrics.accuracy_score(y_test,y_predDT)

precisionDT=cmDT[0][0]/(cmDT[0][0]+cmDT[1][0])

recallDT=cmDT[0][0]/(cmDT[0][0]+cmDT[0][1])

f1_scoreDT=(2*precisionDT*recallDT)/(precisionDT+recallDT)

TPDT = cmDT[0][0]

FPDT = cmDT[0][1]

FNDT = cmDT[1][0]

TNDT = cmDT[1][1]
```

```
TPRDT=TPDT/(TPDT+FNDT)

TNRDT=TNDT/(TNDT+FPDT)

print("TPR for Decision Tree algorithm : ",TPRDT)

print("TNR for Decision Tree algorithm : ",TNRDT)

print("Accuracy of Decision Tree algorithm : ",accuracyDT)

print("Precision for Decision Tree algorithm: ",precisionDT)

print("f1 score for Decision Tree algorithm: ",f1_scoreDT)

from sklearn.ensemble import RandomForestClassifier

classifierRFC = RandomForestClassifier(n_estimators = 15,criterion='entropy',random_state=0)

classifierRFC.fit(X_train,y_train)

y_predRFC=classifierRFC.predict(X_test)

cmRFC = confusion_matrix(y_test,y_predRFC)

accuracyRFC=metrics.accuracy_score(y_test,y_predRFC)

precisionRFC=cmRFC[0][0]/(cmRFC[0][0]+cmRFC[1][0])

recallRFC=cmRFC[0][0]/(cmRFC[0][0]+cmRFC[0][1])

f1_scoreRFC=(2*precisionRFC*recallRFC)/(precisionRFC+recallRFC)

TPRFC = cmRFC[0][0]

FPRFC = cmRFC[0][1]

FNRFC = cmRFC[1][0]

TNRFC = cmRFC[1][1]

TPRRFC=TPRFC/(TPRFC+FNRFC)
```

```
TNRRFC=TNRFC/(TNRFC+FPRFC)

print("TPR for Random Forest Classification algorithm : ",TPRRFC)

print("TNR for Random Forest Classification algorithm : ",TNRRFC)

print("Accuracy of Random Forest Classification algorithm : ",accuracyRFC)

print("Precision for Random Forest Classification algorithm : ",precisionRFC)

print("f1 score for Random Forest Classification algorithm : ",f1_scoreRFC)

#saving the RFC model

import pickle

pickle.dump(classifierRFC,open('saved_model.pkl','wb'),protocol=2)
```

**2. Python code to retrieve the result from saved model when URL is passed**

```
import requests

from bs4 import BeautifulSoup

import urllib,bs4

import urllib.request

import numpy as np

import pickle

def retrive_pred(URL):

    try :

        r = requests.get(URL)

    except :

        return "Error"
```

```
temp_list=list()

soup = BeautifulSoup(r.content, 'html.parser')

p_js=0

c_js=0

for row in soup.findAll('script'):

    c_js=c_js+1

if c_js != 0 :

    p_js=1

p_ns=0

c_ns=0

for row in soup.findAll('noscript'):

    c_ns=c_ns+1

if c_ns != 0 :

    p_ns=1

p_ijs=0

c_ejs=0

for row in soup.findAll('script',attrs='src'):

    c_ejs=c_ejs+1

c_ijs=c_js-c_ejs

if c_ijs != 0 :

    p_ijs=1
```

```
p_img=0

c_img=0

for row in soup.findAll('img'):

    c_img=c_img+1

if c_img != 0 :

    p_img=1

p_iframe=0

c_iframe=0

for row in soup.findAll('iframe'):

    c_iframe=c_iframe+1

if c_iframe != 0 :

    p_iframe=1

p_lnks=0

temp=list()

for row in soup.findAll('a'):

    temp.append(row.get('href'))

c_lnks=len(temp)

if c_lnks != 0 :

    p_lnks=1

temp_list.append(p_js)

temp_list.append(p_ns)
```

```
temp_list.append(p_ijs)

temp_list.append(p_img)

temp_list.append(p_iframe)

temp_list.append(p_lnks)

temp_list.append(c_js)

temp_list.append(c_ns)

temp_list.append(c_ijs)

temp_list.append(c_img)

temp_list.append(c_iframe)

temp_list.append(c_lnks)

url_len= len(URL)

temp_list.append(url_len)

num_fslash=0

num_qm=0

num_dots=0

num_hypen=0

num_eqls=0

num_digi=0

for i in URL:

    if i== '/' :

        num_fslash=num_fslash+1
```

```
        if i=='?' :
            num_qm=num_qm+1
        if i=='.' :
            num_dots=num_dots+1
        if i=='-' :
            num_hypen=num_hypen+1
        if i=='=' :
            num_eqls=num_eqls+1
        if i=='0' or i=='1' or i=='2' or i=='3' or i=='4' or i=='5' or i=='6' or i=='7' or i=='8' or i=='9' :
            num_digi=num_digi+1
    temp_list.append(num_fslash)
    temp_list.append(num_qm)
    temp_list.append(num_dots)
    temp_list.append(num_hypen)
    temp_list.append(num_eqls)
    temp_list.append(num_digi)
            try:    traff_rnk=int(bs4.BeautifulSoup(urllib.request.urlopen("http://data.alexa.com/data?
cli=10&dat=s&url=" + URL).read(), "xml").find("REACH")["RANK"])
    except Exception as err:
        print(err)
        traff_rnk=0
```

```
    temp_list.append(traff_rnk)

    try:  cntry_traff_rnk=int(bs4.BeautifulSoup(urllib.request.urlopen("http://data.alexa.com/data?cli=10&dat=s&url=" + URL).read(), "xml").find("COUNTRY")["RANK"])

    except Exception as err:

        print(err)

        cntry_traff_rnk=0

    temp_list.append(cntry_traff_rnk)

    print(temp_list)

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

    r=np.array2string(model.predict([temp_list]))

    if r=='[0]':

        return "Non malicious"

    else:

        return "Malicious"
```

**3. Python code to create the server**

```
from flask import Flask, request, redirect, url_for, flash, jsonify
from model import *
import numpy as np
import pandas as pd
import pickle
import json
sd_db = dict()
with open("sdb.txt") as f: #sdb.txt contains the URLs which are already verified
    for line in f:
        (key, val) = line.split()
```

```
        sd_db[key] = int(val)
app = Flask(__name__)
@app.route('/',methods=['GET'])
def make_pred():
    return "Home url"
@app.route('/predict',methods=['GET','POST'])
def api_pred():
    data=request.get_json()
    s=data['url']
    if s in sd_db:
        if sd_db[s] == 1:
            return "Malicious"
        else:
            return "Non malicious"
    output=retrive_pred(s)
    f= open("sdb.txt","a+")
    if output == "Malicious":
        sd_db[s]=1
        s=s+" 1\n"
        f.write(s)
    elif output == "Non malicious":
        sd_db[s]=0
        s=s+" 0\n"
        f.write(s)
    return output
if __name__ == '__main__':
    app.run(debug=True)
```

**4. Android code to interact with user and call server when required.**

**MainActivity.java**

```
package com.example.myapplication;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import com.android.volley.AuthFailureError;
import com.android.volley.NetworkResponse;
import com.android.volley.Request;
import com.android.volley.RequestQueue;
import com.android.volley.Response;
import com.android.volley.RetryPolicy;
import com.android.volley.VolleyError;
import com.android.volley.VolleyLog;
import com.android.volley.toolbox.StringRequest;
import com.android.volley.toolbox.Volley;
import org.json.JSONException;
import org.json.JSONObject;
import java.io.UnsupportedEncodingException;
import java.net.URL;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        final TextView t= (TextView)findViewById(R.id.t1);
```

```
final EditText e=(EditText)findViewById(R.id.e1);
Button b1= (Button)findViewById(R.id.b1);

b1.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View v) {
      String s=e.getText().toString();
      t.setText("");
      Boolean b=true;
      try {
         new URL(s).toURI(); //validate the URL
      }
      catch (Exception e) {
         t.setText("Not a valid URL");
         b = false;
      }
      if(b==true)
      {
         String[] arr=s.split(":");
         String tem=arr[1];
         String temp=tem.substring(2);
         String []temp1=temp.split("/");
         temp=temp1[0];
         String[] arr1=temp.split("\\.");
         Boolean b1=true;
         //checking if the domain name is IP address
         for(int i=0;i<arr1.length;i++)
         {
            if(arr1[i].matches("\\d+(?:\\.\\d+)?"))
               continue;
            else
```

```
                    {
                        b1=false;
                        break;
                    }
                }
                if(b1==true)
                {
                    t.append("Warning!!!!\nIt might be malicious\n");
                }
                if(b1==false)
                {
                    //Sending data to the server
                    sendDataToServer(s,t);
                }
            }
        }
    });
}
private void sendDataToServer(final String s, final TextView t) {
    try {
        RequestQueue requestQueue = Volley.newRequestQueue(this);
        String URL = "http://10.0.2.2:5000/predict";
        JSONObject jsonBody = new JSONObject();
        jsonBody.put("url", s);
        final String mRequestBody = jsonBody.toString();

        StringRequest stringRequest = new StringRequest(Request.Method.POST, URL, new
Response.Listener<String>() {
            @Override
            public void onResponse(String response) {
                //t.append("it reacheddd\n");
```

```
                t.append(response.toString());
                Log.i("LOG_VOLLEY", response);
            }
        }, new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) {
                Log.e("LOG_VOLLEY", error.toString());
            }
        }) {
            @Override
            public String getBodyContentType() {
                return "application/json; charset=utf-8";
            }

            @Override
            public byte[] getBody() throws AuthFailureError {
                try {
                    return mRequestBody == null ? null : mRequestBody.getBytes("utf-8");
                } catch (UnsupportedEncodingException uee) {
                    VolleyLog.wtf("Unsupported Encoding while trying to get the bytes of %s using
%s", mRequestBody, "utf-8");
                    return null;
                }
            }

            @Override
            protected Response<String> parseNetworkResponse(NetworkResponse response) {
                String statusCode = String.valueOf(response.statusCode);
                //Handling logic
                return super.parseNetworkResponse(response);
            }
```

```
            };
            stringRequest.setRetryPolicy(new RetryPolicy() {
                @Override
                public int getCurrentTimeout() {
                    return 50000;
                }

                @Override
                public int getCurrentRetryCount() {
                    return 50000;
                }

                @Override
                public void retry(VolleyError error) throws VolleyError {

                }
            });
            requestQueue.add(stringRequest);
        } catch (JSONException e) {
            e.printStackTrace();
        }
    }
}
```