A Project report on

DETECTION OF PHISHING URL's USING MACHINE LEARNING

A Dissertation submitted to JNTU Hyderabad in partial fulfillment of the academic requirements for the award of the Degree.

Bachelor of Technology

in

Computer Science and Engineering

Submitted by

G. Keerthi 19H51A0572

K.V. Sri Divya 19H51A05D6

L. Bhargavi 19H51A05D8

Under the esteemed guidance of

Mr.B.Sivaiah

(Associate Professor, Dept of CSE)

Department of Computer Science and Engineering

CMR COLLEGE OF ENGINEERING AND TECHNOLOGY

(An Autonomous Institution under UGC & JNTUH, Approved by AICTE, Permanently Affiliated to JNTUH, Accredited by NBA.)

KANDLAKOYA, MEDCHAL ROAD, HYDERABAD - 501401.

2019- 2023

CMR COLLEGE OF ENGINEERING & TECHNOLOGY

KANDLAKOYA, MEDCHAL ROAD, HYDERABAD – 501401

DEPARTMENT OF COMPTUER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the Mini Project-II report entitled "**DETECTION OF PHISHING URL's USING MACHINE LEARNING**" being submitted by **G. Keerthi, K. V. Sri Divya, L. Bhargavi** (19H51A0572, 19H51A05D6, 19H51A05D8) in partial fulfillment for the award of **Bachelor of Technology** in **Computer Science and Engineering** is a record of bonafide work carried out her/his under the guidance and supervision.

The results embodies in this project report have not been submitted to any other University or Institute for the award of any degree.

Mr. B. Sivaiah
Associate Professor
Dept. of CSE

Dr. S Siva Skandha
Associate Professor and HOD
Dept. of CSE

Acknowledgement

With great pleasure I want to take this opportunity to express my heartfelt gratitude to all the people who helped in making this project work a grand success.

We are grateful to **Mr. B. Sivaiah,** Associate Professor, Dept of Computer Science and Engineering for his valuable technical suggestions and guidance during the execution of this project work.

We would like to thank **Dr. S.Siva Skandha,** Head of the Department of Computer Science and Engineering, CMR College of Engineering and Technology, who is the major driving forces to complete my project work successfully.

We are very grateful to **Dr. Vijaya Kumar Koppula**, Dean-Academic, CMR College of Engineering and Technology, for his constant support and motivation in carrying out the project work successfully.

We are highly indebted to **Dr. V A Narayana,** Principal, CMR College of Engineering and Technology, for giving permission to carry out this project in a successful and fruitful way.

We would like to thank the Teaching & Non- teaching staff of Department of Computer Science and Engineering for their co-operation

Finally we express our sincere thanks to **Mr. Ch. Gopal Reddy**, Secretary, CMR Group of Institutions, for his continuous care. We sincerely acknowledge and thank all those who gave support directly and indirectly in completion of this project work.

G. Keerthi
19H51A0572

K.V. Sri Divya
19H51A05D6

L. Bhargavi
19H51A05D8

TABLE OF CONTENTS

ABSTRACT

In recent years, with the increasing use of mobile devices, there is a growing trend to move almost all real-world operations to the cyber world. Although this makes easy our daily lives, it also brings many security breaches due to the anonymous structure of the Internet. Phishing is a form of cyber Crime where an attacker imitates a real person / institution by promoting them as an official person or entity through e-mail or other communication mediums. In this type of cyber attack, the attacker sends malicious links or attachments through phishing e-mails that can perform various functions, including capturing the login credentials or account information of the victim. These e-mails harm victims because of money loss and identity theft.we proposed a machine learning-based phishing detection system by using different algorithms to analyze the URLs, and datasets.

1. INTRODUCTION

In recent years, we have witnessed a significant increase in cybersecurity attacks such as ransomware, phishing, injection of malware, etc. on different websites all over the world. As a result of this, various financial institutions, e-commerce companies, and individuals incurred huge financial losses.In such type of scenario containing a cyber security attack is a major challenge for cyber security professionals as different types of new attacks are coming day by day. Typically a victim receives a message that appears to have been sent by a known contact or organization. The message contains malicious software targeting the user's computer or has links to direct victims to malicious websites in order to trick them into divulging personal and financial information, such as passwords, account IDs or credit card details.

1.1 PROBLEM STATEMENT:

A phishing website is a common social engineering method that mimics trustful uniform resource locators (URLs) and webpages. Phishing is a form of fraud in which the attacker tries to learn sensitive information such as login credentials or account information by sending as a reputable entity or person in email or other communication channels. Both phishing and benign URLs of websites are gathered to form a dataset and from them required URL and website content-based features are extracted. The problem is derived after making a thorough observation and study about the method of classification of phishing websites that makes use of machine learning techniques. We must design a system that should allow us to:

- Accurately and efficiently classify the websites into legitimate or phishing.
- Time consumed for detection should be less and should be cost effective.

1.3 OBJECTIVES:

The project's objectives are as follows:

- To study various automatic phishing detection methods.
- To identify the appropriate machine learning techniques and define a solution using the selected method.
- To select an appropriate dataset for the problem statement.
- To apply appropriate algorithms to achieve the solution to phishing attacks.

2. BACKGROUND WORK

2.1 Detection Using Heuristic Approach:

Srinivasa Rao and Syed taqi has introduced a desktop application system [1] called Phish shield to detect the phishing webpages. It concentrates on URL and website content of phishing page. The input to this system is URL and it outputs the status of URL as either suspicious or trusted website. Here heuristic approach is employed which studies the structure of content and URL of phishing websites to extract the features of the phishing sites. It then designs the model to detect phishing site based on the extracted features. Website identity, title content, copy right content, zero links in the body of the HTML are the heuristic approaches to detect phishing. Phish shield is able to detect zero hour phishing attack. The tool used to develop phish shield application include NetBeans 8.0.2 IDE,JAVA compiler, JSoup api and firebug. Even if the content can be replaced with images, this application is also able to detect phishing sites.

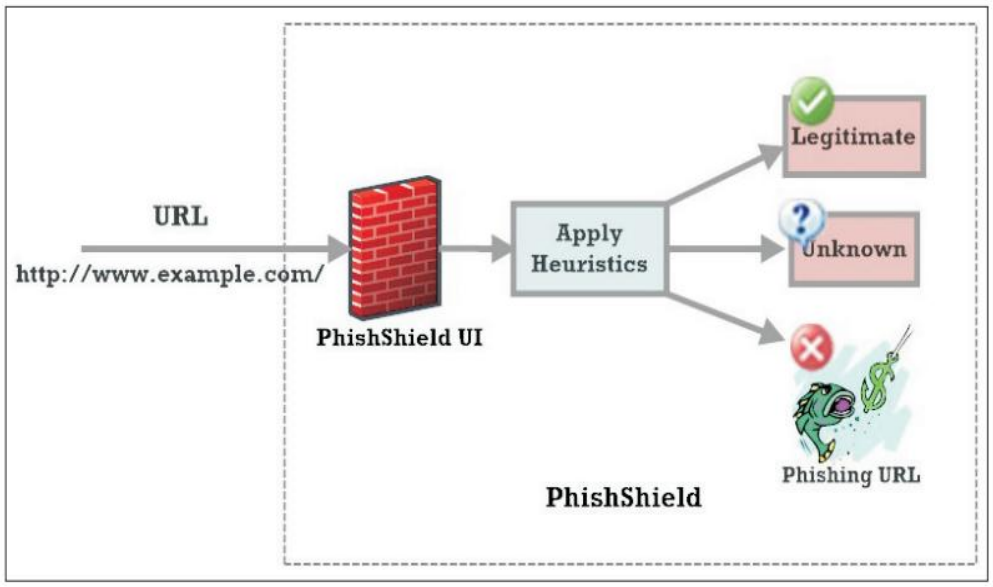


Fig: 2.1.1 Architecture of Heuristic Approach

2.2 Case Based Reasoning Technique:

Hassan and Abdelfettah introduced Case Based Reasoning(CBR) Phishing Detection System[3].The core part of the system is CBR methodology. To predict a solution for a problem, it uses historical data namely experiences or cases. The main feature of this system is that it works similar to human thinking such that it can solve problems from past experiences as similar problem have similar solution. It consist of

four phases; Retrieve, Reuse, Revise and Retain. This technique is designed to be updated with approved phishing attack experiences. There are two types of experiences such as offline and online experiences. It can detect new phishing attack even if the dataset is small. So the system is highly dynamic and adaptive.

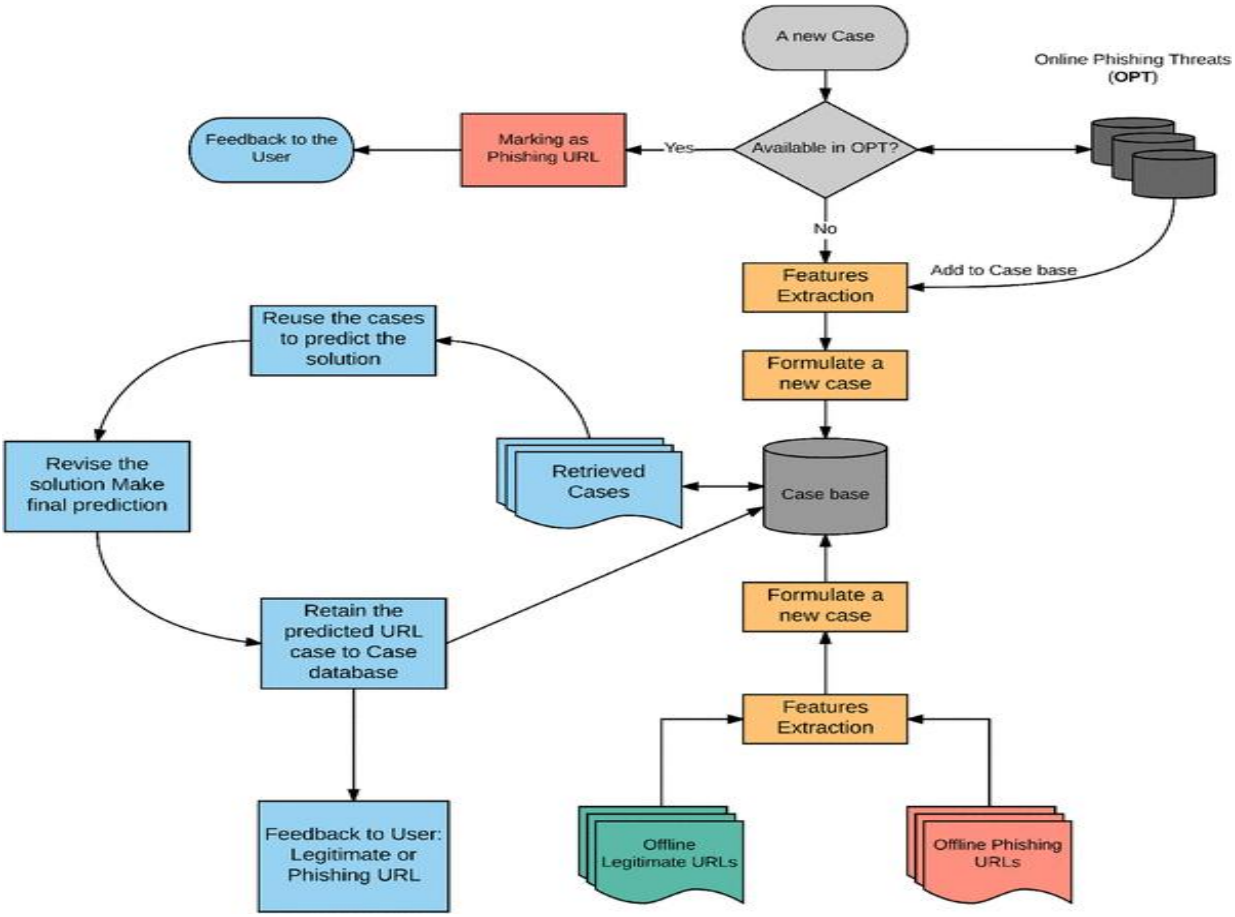
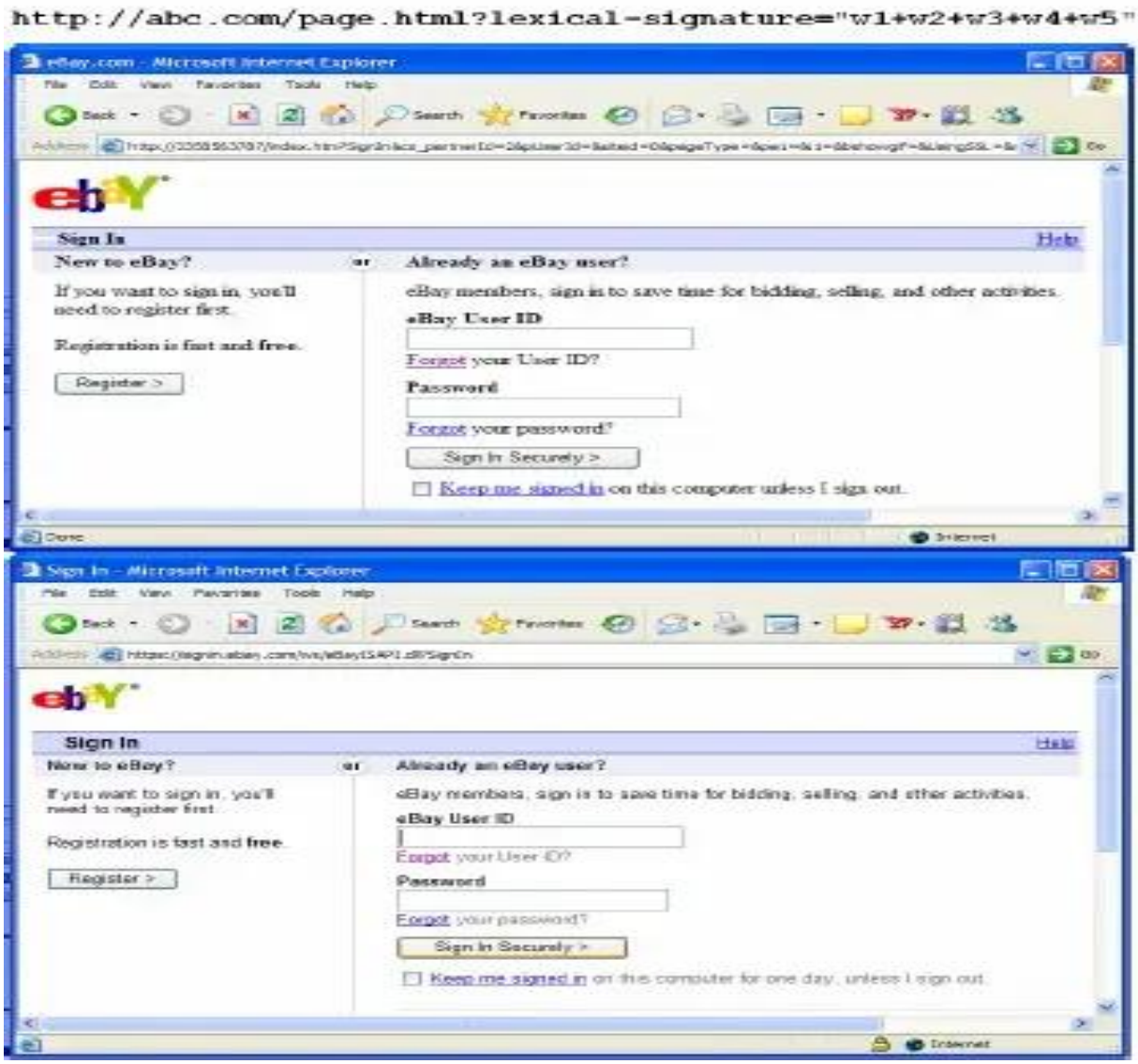


Fig: 2.2.1: Case based reasoning Phisihing Detection system.

2.3 Content Based Approach:

Yue Zhang and Jason proposed content based approach for detecting phishing website[2]. Cantina make use of TFIDF information retrieval algorithm. It is used for comparing and classifying documents as well as retrieving documents from a large corpus. Robust hyperlink is an application of TF-IDF and the basic idea is to provide independent descriptions of networked resources that is URL. Cantina works as follows. When we are given webpage, it calculates the TF-IDF scores of the webpage based on each term. It takes five terms with highest TFIDF weights and generate a lexical signature. In this case, we use Google as the search engine to feed this lexical signature. If the domain name of N top search result matches with the domain name of the given webpage, Then it considers website to be a legitimate

one, otherwise it is considered as the phishing website. Cantina is implemented as the Microsoft Internet Explorer extension.



2.4 Machine Learning Approach:

Sirageldin, B. B. Baharudin, and L. T. Jung, proposed a framework for detecting malicious webpage using Artificial Neural Network[5]. The algorithm is based on two group of features namely URL lexical and page content features. Feature extractor, Learning and model content features. Fea- ture extractor, learning and model selector and Detector are the three components of the model. Webpage features are extracted by feature extractor, Here we collects the dataset. Learning algorithm with best result is selected in learning and model selector to build the model. Final classification model generated from the learning and model selector component by using detector.

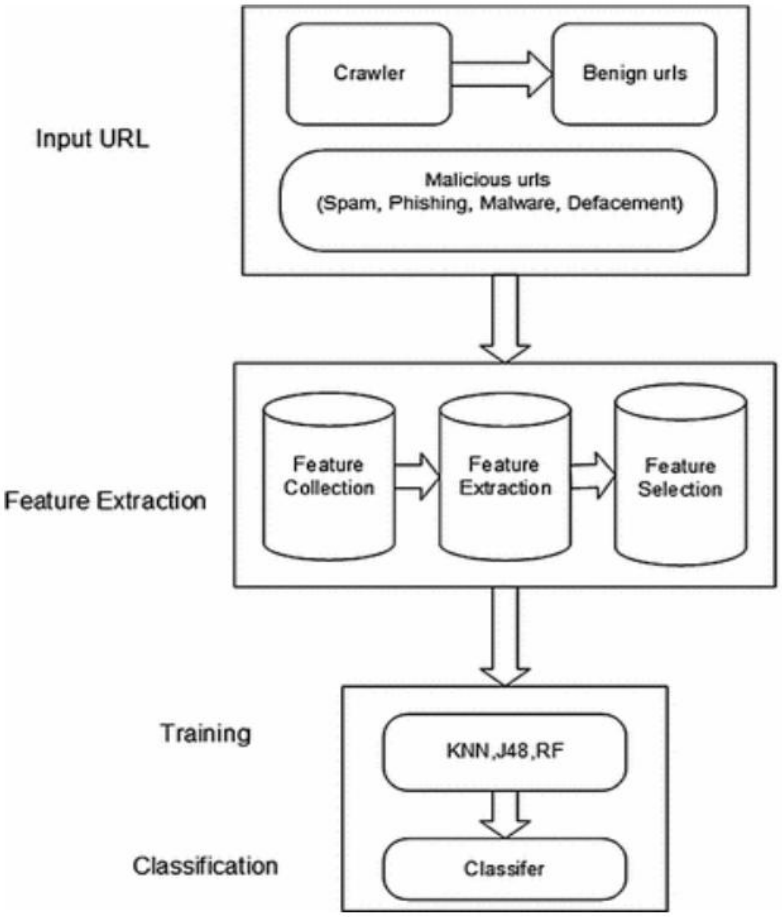


Fig: 2.4.1: Block Diagram of Proposed system.

3. PROPOSED SYSTEM

3.1 Project Flow:

As we know machine learning algorithms only support numeric inputs so we will create lexical numeric features from input URLs. So the input to machine learning algorithms will be the numeric lexical features rather than actual raw URLs.

So, we will be using three well-known machine learning ensemble classifiers namely Random Forest, Light GBM, and XGBoost.

Later, we will also compare their performance and plot average feature importance plot to understand which features are important in predicting malicious URLs.

3.2 Dataset description:

In this Project, we will be using a Malicious URLs dataset of 6,51,191 URLs, out of which 4,28,103 benign or safe URLs, 96,457 defacement URLs, 94,111 phishing URLs, and 32,520 malware URLs.
Now, let's discuss different types of URLs in our dataset i.e., Benign, Malware, Phishing, and Defacement URLs.

- **Benign URLs:** These are safe to browse URLs. Some of the examples of benign URLs are as follows:
    - mp3raid.com/music/krizz_kaliko.html
    - infinitysw.com
    - google.co.in
    - myspace.com
- **Malware URLs:** These type of URLs inject malware into the victim's system once he/she visit such URLs. Some of the examples of malware URLs are as follows:
    - proplast.co.nz
    - http://103.112.226.142:36308/Mozi.m
    - microencapsulation.readmyweather.com
    - xo3fhvm5lcvzy92q.download
- **Defacement URLs:** Defacement URLs are generally created by hackers with the intention of breaking into a web server and replacing the hosted website with one of their own, using techniques such as code injection, cross-site scripting, etc. Common targets of defacement URLs are religious websites, government websites, bank websites, and corporate websites. Some of the examples of defacement URLs are as follows:

- http://www.vnic.co/khach-hang.html
- http://www.raci.it/component/user/reset.html
- http://www.approvi.com.br/ck.htm
- http://www.juventudelirica.com.br/index.html

- **Phishing URLs:** By creating phishing URLs, hackers try to steal sensitive personal or financial information such as login credentials, credit card numbers, internet banking details, etc. Some of the examples of phishing URLs are shown below:
  - roverslands.net
  - corporacionrossenditotours.com
  - http://drive-google-com.fanalav.com/6a7ec96d6a
  - Citiprepaid-salarysea-at.tk

Next, we will plot the word cloud of different types of URLs.

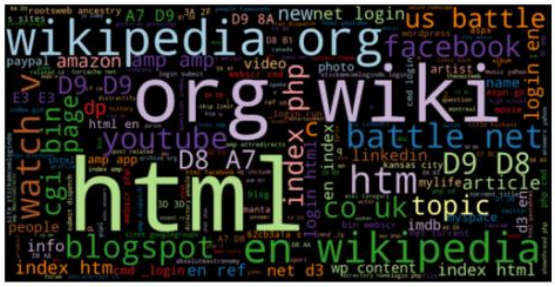
3.2.1 Wordcloud of URLs

The word cloud helps in understanding the pattern of words/tokens in particular target labels.

It is one of the most appealing techniques of natural language processing for understanding the pattern of word distribution.
As we can see in the below figure word cloud of benign URLs is pretty obvious having frequent tokens such as *html, com, org, wiki* etc. Phishing URLs have frequent tokens as tools, *ietf, www, index, battle, net* whereas *html, org, html* are higher frequency tokens as these URLs try to mimick original URLs for deceiving the users.

The word cloud of malware URLs has higher frequency tokens of *exe, E7, BB, MOZI*. These tokens are also obvious as malware URLs try to install trojans in the form of executable files over the users' system once the user visits those URLs.
The defacement URLs' intention is to modify the original website's code and this is the reason that tokens in its word cloud are more common development terms such as *index, php, itemid, https, option,* etc.

Benign URL



Phishing URL

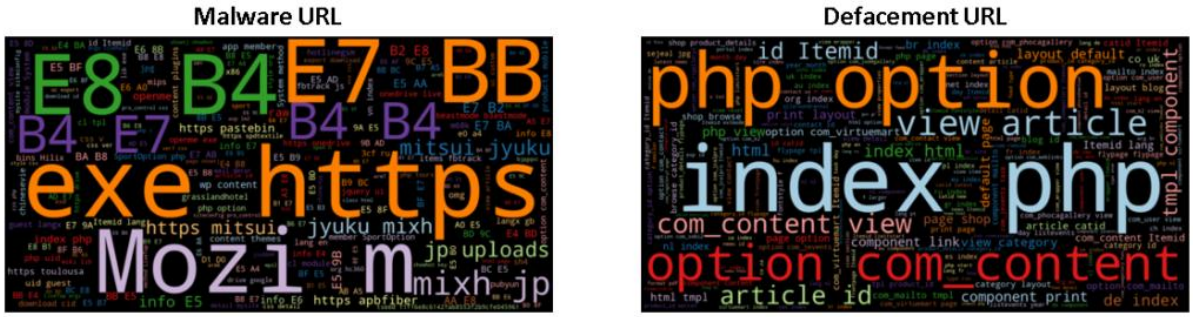


Fig: 3.2.1.1: Word clouds of Benign, Phishing, Malware, and Defacement URLs

So, from the above word clouds, most of the tokens distribution in different types of URLs is

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

now crystal clear.

3.2.2 Importing Libraries

In this step, we will import all the necessary python libraries which will be used in this project.

```
from lightgbm import LGBMClassifier
import os
import seaborn as sns
from wordcloud import WordCloud
```

Next, we will load the dataset and will check sample records in the dataset to get an understanding of the data.

3.3 Loading dataset:

In this step, we will import the dataset using the *pandas* library and check the sample entries in the dataset.

```
df=pd.read_csv('malicious_phish.csv')
print(df.shape)
df.head()
```

```
(651191, 2)
```

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

So from the above output, we can observe that the dataset has 6,51,191 records with two columns *url* containing the raw URLs and *type* which is the target variable.

Next, we will move towards the feature engineering part in which we will create lexical features from raw URLs.

3.4 Feature engineering:

In this step, we will extract the following lexical features from raw URLs, as these features will be used as the input features for training the machine learning model. The following features are created as follows:

- **having_ip_address:** Generally cyber attackers use an IP address in place of the domain name to hide the identity of the website. this feature will check whether the URL has IP address or not.
- **abnormal_url:** This feature can be extracted from the WHOIS database. For a legitimate website, identity is typically part of its URL.
- **google_index:** In this feature, we check whether the URL is indexed in google search console or not.
- **Count. :** The phishing or malware websites generally use more than two sub-domains in the URL. Each domain is separated by dot (.). If any URL contains more than three dots(.), then it increases the probability of a malicious site.
- **Count-www:** Generally most of the safe websites have one www in its URL. This feature helps in detecting malicious websites if the URL has no or more than one www in its URL.
- **count@:** The presence of the "@" symbol in the URL ignores everything previous to it.
- **Count_dir:** The presence of multiple directories in the URL generally indicates suspicious websites.
- **Count_embed_domain:** The number of the embedded domains can be helpful in detecting malicious URLs. It can be done by checking the occurrence of "//" in the URL.

- **Suspicious words in URL:** Malicious URLs generally contain suspicious words in the URL such as PayPal, login, sign in, bank, account, update, bonus, service, ebayisapi, token, etc. We have found the presence of such frequently occurring suspicious words in the URL as a binary variable i.e., whether such words present in the URL or not.
- **Short_url:** This feature is created to identify whether the URL uses URL shortening services like bit. \ly, goo.gl, go2l.ink, etc.
- **Count_https:** Generally malicious URLs do not use HTTPS protocols as it generally requires user credentials and ensures that the website is safe for transactions. So, the presence or absence of HTTPS protocol in the URL is an important feature.
- **Count_http:** Most of the time, phishing or malicious websites have more than one HTTP in their URL whereas safe sites have only one HTTP.
- **Count%:** As we know URLs cannot contain spaces. URL encoding normally replaces spaces with symbol (*%*). Safe sites generally contain less number of spaces whereas malicious websites generally contain more spaces in their URL hence more number of %.
- **Count?:** The presence of symbol (*?*) in URL denotes a query string that contains the data to be passed to the server. More number of ? in URL definitely indicates suspicious URL.
- **Count-:** Phishers or cybercriminals generally add dashes(-) in prefix or suffix of the brand name so that it looks genuine URL. For example. *www.flipkart-india.com*.
- **Count=:** Presence of equal to (=) in URL indicates passing of variable values from one form page t another. It is considered as riskier in URL as anyone can change the values to modify the page.
- **url_length:** Attackers generally use long URLs to hide the domain name. We found the average length of a safe URL is 74.
- **hostname_length**: The length of the hostname is also an important feature for detecting malicious URLs.
- **First directory length:** This feature helps in determining the length of the first directory in the URL. So looking for the first '/' and counting the length of the URL up to this point helps in finding the first directory length of the URL. For accessing directory level information we need to install python library TLD. You can check this link for installing TLD.
- **Length of top-level domains:** A top-level domain (TLD) is one of the domains at the highest level in the hierarchical Domain Name System of the Internet. For example, in the domain name www.example.com, the top-level domain is com. So, the length of TLD is also important in identifying malicious

URLs. As most of the URLs have .com extension. TLDs in the range from 2 to 3 generally indicate safe URLs.

- **Count_digits:** The presence of digits in URL generally indicate suspicious URLs. Safe URLs generally do not have digits so counting the number of digits in URL is an important feature for detecting malicious URLs.
- **Count_letters:** The number of letters in the URL also plays a significant role in identifying malicious URLs. As attackers try to increase the length of the URL to hide the domain name and this is generally done by increasing the number of letters and digits in the URL.

The code for creating above mentioned features is shared below.

```
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 0
df['use_of_ip'] = df['url'].apply(lambda i: having_ip_address(i))
from urllib.parse import urlparse
def abnormal_url(url):
  hostname = urlparse(url).hostname
  hostname = str(hostname)
  match = re.search(hostname, url)
  if match:
    # print match.group()
    return 1
  else:
    # print 'No matching pattern found'
    return 0
df['abnormal_url'] = df['url'].apply(lambda i: abnormal_url(i))
#pip install googlesearch-python
from googlesearch import search
def google_index(url):
  site = search(url, 5)
  return 1 if site else 0
df['google_index'] = df['url'].apply(lambda i: google_index(i))
def count_dot(url):
```

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

  return url.count('@')
df['count@'] = df['url'].apply(lambda i: count_atrate(i))
def no_of_dir(url):
  urldir = urlparse(url).path
  return urldir.count('/')
df['count_dir'] = df['url'].apply(lambda i: no_of_dir(i))
def no_of_embed(url):
  urldir = urlparse(url).path
  return urldir.count('//')
df['count_embed_domian'] = df['url'].apply(lambda i: no_of_embed(i))
def shortening_service(url):
  match                                                                    =
re.search('bit\.ly|goo\.gl|shorte\.st|go2l\.ink|x\.co|ow\.ly|t\.co|tinyurl|tr\.im|is\.gd|cli\.gs|'

'yfrog\.com|migre\.me|ff\.im|tiny\.cc|url4\.eu|twit\.ac|su\.pr|twurl\.nl|snipurl\.com|'

'short\.to|BudURL\.com|ping\.fm|post\.ly|Just\.as|bkite\.com|snipr\.com|fic\.kr|loopt\.us|'

'doiop\.com|short\.ie|kl\.am|wp\.me|rubyurl\.com|om\.ly|to\.ly|bit\.do|t\.co|lnkd\.in|'

'db\.tt|qr\.ae|adf\.ly|goo\.gl|bitly\.com|cur\.lv|tinyurl\.com|ow\.ly|bit\.ly|ity\.im|'

'q\.gs|is\.gd|po\.st|bc\.vc|twitthis\.com|u\.to|j\.mp|buzurl\.com|cutt\.us|u\.bb|yourls\.org|'

'x\.co|prettylinkpro\.com|scrnch\.me|filoops\.info|vzturl\.com|qr\.net|1url\.com|tweez\.me
|v\.gd|'
                  'tr\.im|link\.zip\.net',
                  url)
  if match:
     return 1
  else:
     return 0

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

```
  return url.count('%')
df['count%'] = df['url'].apply(lambda i : count_per(i))
def count_ques(url):
  return url.count('?')
df['count?'] = df['url'].apply(lambda i: count_ques(i))
def count_hyphen(url):
  return url.count('-')
df['count-'] = df['url'].apply(lambda i: count_hyphen(i))
def count_equal(url):
  return url.count('=')
df['count='] = df['url'].apply(lambda i: count_equal(i))
def url_length(url):
  return len(str(url))
#Length of URL
df['url_length'] = df['url'].apply(lambda i: url_length(i))
#Hostname Length
def hostname_length(url):
  return len(urlparse(url).netloc)
df['hostname_length'] = df['url'].apply(lambda i: hostname_length(i))
df.head()
def suspicious_words(url):
  match                                                                    =
re.search('PayPal|login|signin|bank|account|update|free|lucky|service|bonus|ebayisapi|w
ebscr',
                  url)
  if match:
     return 1
  else:
     return 0
df['sus_url'] = df['url'].apply(lambda i: suspicious_words(i))
def digit_count(url):
  digits = 0
  for i in url:
     if i.isnumeric():
        digits = digits + 1
  return digits
df['count-digits']= df['url'].apply(lambda i: digit_count(i))
def letter_count(url):
  letters = 0
  for i in url:
     if i.isalpha():
        letters = letters + 1
  return letters
df['count-letters']= df['url'].apply(lambda i: letter_count(i))
# pip install tld
from urllib.parse import urlparse
from tld import get_tld
import os.path
#First Directory Length
def fd_length(url):
```

```
  urlpath= urlparse(url).path
  try:
     return len(urlpath.split('/')[1])
  except:
     return 0
df['fd_length'] = df['url'].apply(lambda i: fd_length(i))
#Length of Top Level Domain
df['tld'] = df['url'].apply(lambda i: get_tld(i,fail_silently=True))
def tld_length(tld):
  try:
     return len(tld)
  except:
     return -1
df['tld_length'] = df['tld'].apply(lambda i: tld_length(i))
```

So, now after creating the above 22 features, the dataset looks like the below.

	use_of_ip	abnormal_url	count.	count-www	count@	count_dir	count_embed_domian	short_url	count-https	count-http	...	count?	count-	count=	url_length
0	0	0	2	0	0	0	0	0	0	0	...	0	1	0	16
1	0	0	2	0	0	2	0	0	0	0	...	0	0	0	35
2	0	0	2	0	0	3	0	0	0	0	...	0	0	0	31
3	0	1	3	1	0	1	0	0	0	1	...	1	1	4	88
4	0	1	2	0	0	1	0	0	0	1	...	1	1	3	235

Fig: 3.4.1: Dataset

Now, in the next step, we drop the irrelevant columns i.e., URL,google_index, and tld. The reason for dropping the URL column is that we have already extracted relevant features from it that can be used as input in machine learning algorithms.
The tld column is dropped because it is the indirect textual column as for finding the length of the top-level domain we have created tld column.

The google_index feature denotes if the URL is indexed in google search console or not. In this dataset, all the URLs are google indexed and have a value of 1.

3.5 Exploratory Data Analysis (EDA):

In this step, we will check the distribution of different features for all four classes of URLs.

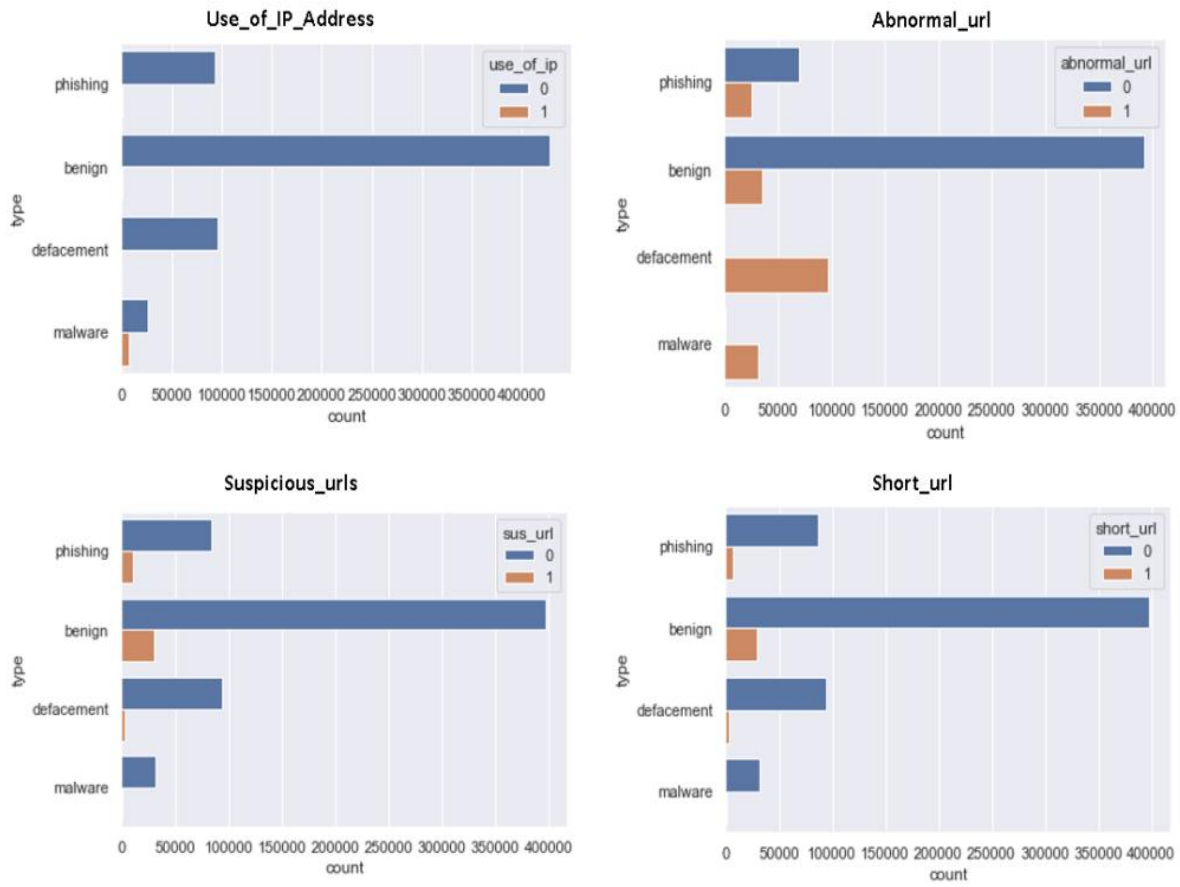


Fig: 3.5.1: Different features of 4 classes of URL's

As we can observe from the above distribution of *use_ip_address* feature, only malware URLs have IP addresses. In the case of *abnormal_url*, defacement URLs have higher distribution.

From the distribution of suspicious_urls, it is clear that benign URLs have highest distribution while phishing URLs have a second highest distribution. As suspicious URLs consist of transaction and payment-related keywords and generally genuine banking or payment-related URLs consist of such keywords that's why benign URLs have the highest distribution.

As per the short_url distribution, we can observe that benign URLs have the highest short URLs as we know that generally, we use URL shortening services for easily sharing long-length URLs.

3.6 Label Encoding:

After that, the most important step is to label and encode the target variable (type) so that it can be converted into numerical categories 0,1,2, and 3. As machine learning algorithms only understand numeric target variable.

```
from sklearn.preprocessing import LabelEncoder
lb_make = LabelEncoder()
df["type_code"] = lb_make.fit_transform(df["type"])
```

3.7 Segregating Feature and Target variables:

So, in the next step, we have created a predictor and target variable. Here predictor variables are the independent variables i.e., features of URL, and target variable type.

```
#Predictor Variables
# filtering out google_index as it has only 1 value
X = df[['use_of_ip','abnormal_url', 'count.', 'count-www', 'count@',
    'count_dir', 'count_embed_domian', 'short_url', 'count-https',
    'count-http', 'count%', 'count?', 'count-', 'count=', 'url_length',
    'hostname_length', 'sus_url', 'fd_length', 'tld_length', 'count-digits',
    'count-letters']]
#Target Variable
y = df['type_code']
```

3.8 Training & Test Split:

The next step is to split the dataset into train and test sets. We have split the dataset into 80:20 ratio i.e., 80% of the data was used to train the machine learning models, and the rest 20% was used to test the model.

As we know we have an imbalanced dataset. The reason for this is around 66% of the data has benign URLs, 5% malware, 14% phishing, and 15% defacement URLs. So after randomly splitting the dataset into train and test, it may happen that the distribution of different categories got disturbed which will highly affect the performance of the machine learning model. So to maintain the same proportion of the target variable *stratification* is needed.

This stratify parameter makes a split so that the proportion of values in the sample produced will be the same as the proportion of values provided to the parameter stratify.

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

So, now we are ready for the most awaited part which is Model Building !!!

4.DESIGNING

4.1 Model building:

Implementation Steps are given below:

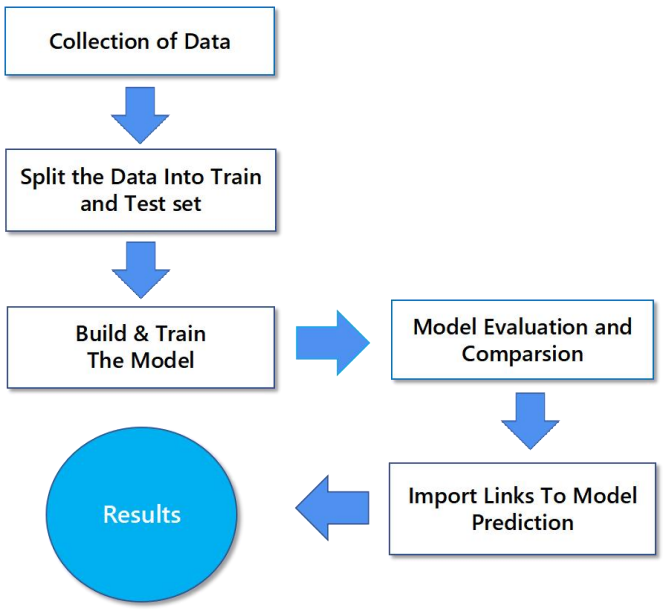


Fig: 4.1.1: Implementation Steps.

In this step, we will build three tree-based ensemble machine learning models i.e., Light GBM, XGBoost, and Random Forest.

4.2Machine learning models:

- **Random Forest:**
  Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of **ensemble learning,** which is a process of *combining multiple classifiers to solve a complex problem and to improve the performance of the model.*
  As the name suggests, ***"Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset."*** Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

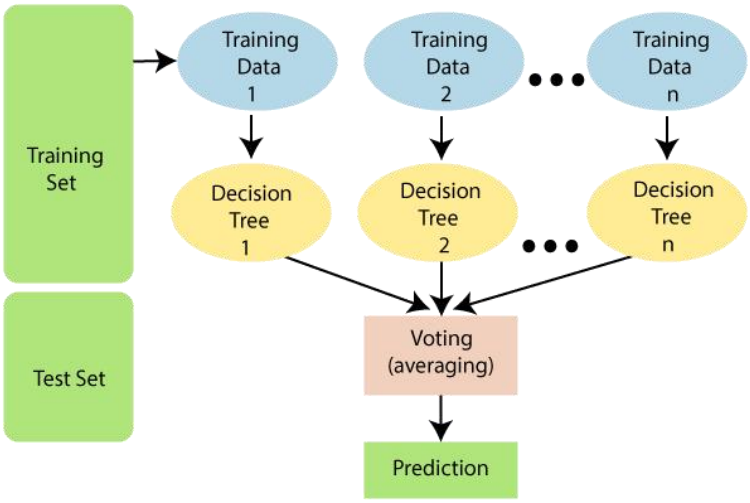


Fig: 4.2.1: Random Forest Work Flow

- **Light GBM:**
  LightGBM is a gradient boosting framework based on decision trees to increases the efficiency of the model and reduces memory usage.
  It uses two novel techniques: **Gradient-based One Side Sampling** and **Exclusive Feature Bundling (EFB)** which fulfills the limitations of histogram-based algorithm that is primarily used in all GBDT (Gradient Boosting Decision Tree) frameworks. The two techniques of GOSS and EFB described below form the characteristics of LightGBM Algorithm. They comprise together to make the model work efficiently and provide it a cutting edge over other GBDT frameworks.

- **XGBoost:**
  Gradient boosted decision trees are implemented by the XGBoost library of Python, intended for speed and execution, which is the most important aspect of ML (machine learning).XgBoost (Extreme Gradient Boosting) library of Python was introduced at the University of Washington by scholars. It is a module of Python written in C++, which helps ML model algorithms by the training for Gradient Boosting.
  Gradient boosting: This is an AI method utilized in classification and regression assignments, among others. It gives an expectation model as a troupe of feeble forecast models, commonly called decision trees.

The code for building machine learning models is shared below.

```
# Random Forest Model
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=100,max_features='sqrt')
rf.fit(X_train,y_train)
y_pred_rf = rf.predict(X_test)
print(classification_report(y_test,y_pred_rf,target_names=['benign',
'defacement','phishing','malware']))
score = metrics.accuracy_score(y_test, y_pred_rf)
print("accuracy:   %0.3f" % score)
#XGboost
xgb_c = xgb.XGBClassifier(n_estimators= 100)
xgb_c.fit(X_train,y_train)
y_pred_x = xgb_c.predict(X_test)
print(classification_report(y_test,y_pred_x,target_names=['benign',
'defacement','phishing','malware']))
score = metrics.accuracy_score(y_test, y_pred_x)
print("accuracy:   %0.3f" % score)
# Light GBM Classifier
lgb = LGBMClassifier(objective='multiclass',boosting_type= 'gbdt',n_jobs = 5,
        silent = True, random_state=5)
LGB_C = lgb.fit(X_train, y_train)
y_pred_lgb = LGB_C.predict(X_test)
print(classification_report(y_test,y_pred_lgb,target_names=['benign',
'defacement','phishing','malware']))
score = metrics.accuracy_score(y_test, y_pred_lgb)
print("accuracy:   %0.3f" % score)
```

5. RESULT

5.1 Model evaluation & comparison

After fitting the model, as shown above, we have made predictions on the test set. The performance of Light GBM, XGBoost, and Random Forest are shown below.

RANDOM FOREST:

	precision	recall	f1-score	support
benign	0.97	0.98	0.98	85621
defacement	0.98	0.99	0.99	19292
phishing	0.98	0.94	0.96	6504
malware	0.91	0.86	0.88	18822
accuracy			0.97	130239
macro avg	0.96	0.95	0.95	130239
weighted avg	0.97	0.97	0.97	130239
accuracy:	0.966			

XGBOOST:

	precision	recall	f1-score	support
benign	0.97	0.99	0.98	85621
defacement	0.97	0.99	0.98	19292
phishing	0.98	0.92	0.95	6504
malware	0.91	0.83	0.87	18822
accuracy			0.96	130239
macro avg	0.96	0.93	0.94	130239
weighted avg	0.96	0.96	0.96	130239
accuracy:	0.962			

LIGHT GBM:

	precision	recall	f1-score	support
benign	0.97	0.99	0.98	85621
defacement	0.96	0.99	0.98	19292
phishing	0.97	0.91	0.94	6504
malware	0.9	0.83	0.86	18822
accuracy			0.96	130239
macro avg	0.95	0.93	0.94	130239
weighted avg	0.96	0.96	0.96	130239
accuracy:	0.959			

Fig: 5.1.1: Result Analysis

From the above result, it is evident that **Random Forest** shows the best performance in terms of test accuracy as it attains the highest accuracy of **96.6%** with a higher detection rate for benign, defacement, phishing, and malware.

So based on the above performance, we have selected Random Forest as our main model for detecting malicious URLs and in the next step, we will also plot the feature importance plot.

5.2 Feature Importance

After selecting our model i.e., Random Forest, next, we will be checking highly contributing features. The code for plotting feature importance plot.

```
feat_importances = pd.Series(rf.feature_importances_,
index=X_train.columns)
feat_importances.sort_values().plot(kind="barh",figsize=(10, 6))
```

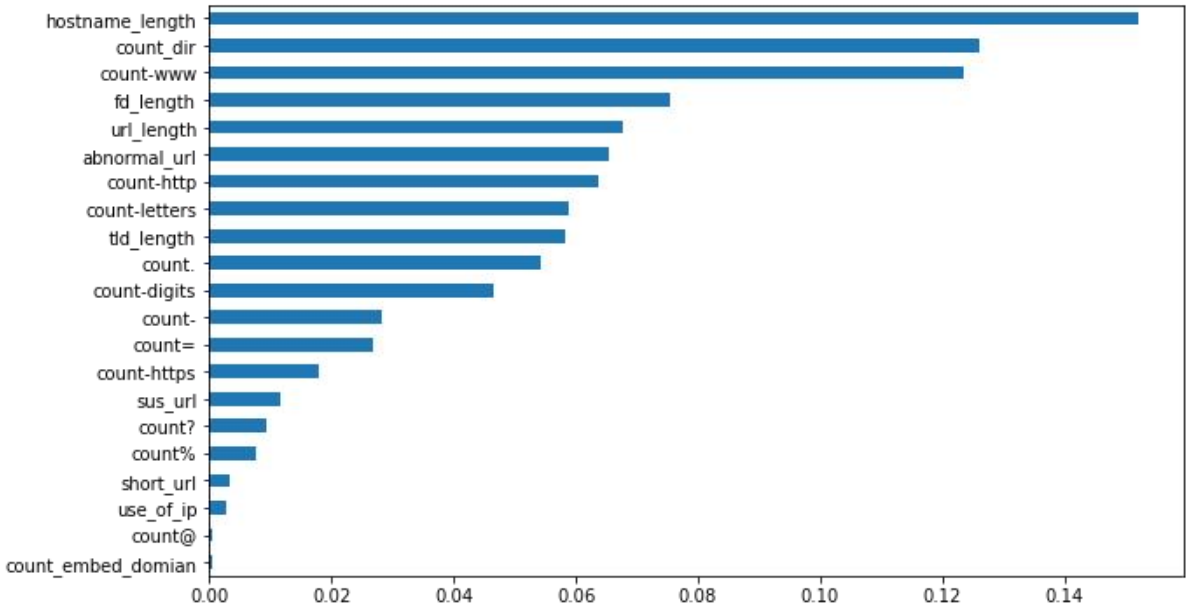


Fig: 5.2.1: Features Plot

From the above plot, we can observe that the top 5 features for detecting malicious URLs are ***hostname_length***, ***count_dir***, ***count-www***, ***fd_length***, and ***url_length***.

5.3 Model prediction:

In this final step, we will predict malicious URLs using our best-performed model i.e., Random Forest.

The code for predicting raw URLs using our saved model is given below:

```
def main(url):

  status = []

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

```
    status.append(no_of_embed(url))

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

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

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

    status.append(tld_length(tld))

    return status
# predict function
def get_prediction_from_url(test_url):
    features_test = main(test_url)
    # Due to updates to scikit-learn, we now need a 2D array as a parameter to the predict
function.
    features_test = np.array(features_test).reshape((1, -1))
    pred = lgb.predict(features_test)
    if int(pred[0]) == 0:
```

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

    res="DEFACEMENT"
    return res
  elif int(pred[0]) == 2.0:
    res="PHISHING"
    return res

  elif int(pred[0]) == 3.0:

    res="MALWARE"
    return res
# predicting sample raw URLs
urls = ['titaniumcorporate.co.za','en.wikipedia.org/wiki/North_Dakota']
for url in urls:
   print(get_prediction_from_url(url))
```

MALWARE

SAFE

6. CONCLUSION

6.1 PROJECT CONCLUSION:

We have demonstrated a machine learning approach to detect Malicious URLs. We have created 22 lexical features from raw URLs and trained three machine learning models **XG Boost**, **Light GBM**, and **Random forest**. Further, we have compared the performance of the 3 machine learning models and found that **Random forest** outperformed others by attaining the highest accuracy of **96.6%**. By plotting the feature importance of Random forest we found that ***hostname_length***, ***count_dir***, ***count-www***, ***fd_length***, and ***url_length*** are the top 5 features for detecting the malicious URLs. At last, we have coded the prediction function for classifying any raw URL using our saved model i.e., Random Forest.

6.2 FUTURE WORK:

A new subsystem can be conducted for shorter URLs, which contain only a domain/subdomain. To detect this type of web pages, some active data about the web page is needed such as the number of visitors in the last week, when the domain name is received, etc. If a web page is detected as phishing this page can be added to the local blacklist of the network and can be forbidden for the further requests.

7. REFERNCES

- https://towardsdatascience.com/phishing-domain-detection-with-ml-5be9c99293e5
- https://www.researchgate.net/publication/344952543_Machine_learning_based_phishing_detection_from_URLs
- https://www.ijert.org/malicious-website-detection-using-machine-learning
- https://www.researchgate.net/publication/328541785_Phishing_Website_Detection_using_Machine_Learning_Algorithms
- https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0258361
- https://ieeexplore.ieee.org/abstract/document/8769571