Whaling Guard

Phishing Detection Using Machine Learning

CONTENTS

- Introduction
- Abstract
- Literature Survey
- Problem Statement
- Objectives
- Scope
- Methodology
- System Requirements
- System Implementation
- Results and Discussion
- Conclusion
- References

Introduction

- Phishing attack is a simplest way to obtain sensitive information from innocent users.
- Cyber security persons are now looking for trustworthy and steady detection techniques for phishing websites.
- Phishing detection is a crucial aspect of cybersecurity aimed at identifying and preventing phishing attacks.
- The objective of this paper is to extract different features from a large dataset containing URLs and to analyse the accuracy levels for different machine learning algorithms and implementing the best among them.

Abstract

- In this paper, we have done preprocessing on 2 datasets containing URLs and they are merged to form a new output dataset. Feature extraction is done on this new dataset and have extracted 74 features.
- After comparing 15 different machine learning models with the extracted features, we have found that Logistic Regression(LR) model provides the highest accuracy.
- The trained LR model is then integrated into our Web Application for predicting whether the input URL from the user interface is phishing or non-phishing.

Literature Survey

Year	Title	Author	Methodology	Conclusion/Results
2020	PhishHaven—An Efficient Real-Time AI Phishing URLs Detection System	Maria Sameen; Kyunghyun Han; Seong Oun Hwang	Design a PhishHaven which detects and classifies a URL using three subcomponents. First subcomponent, URL Hit The second subcomponent is Features Extractor. The third subcomponent is Modelics,	In this new paradigm executes ensemble-based machine learning models in parallel using multi-threading technique, and results in real-time detection by significant speed-up in the classification process.
2021	Sufficiency of Ensemble Machine Learning Methods for Phishing Websites Detection	Yi Wei; Yuji Sekiya	• Phishing instances are usually derived from PhishTank • Other legitimate instances are from Alexa, DMOZ, and Common Crawl. • Features used in phishing detection are usually extracted from URLs (protocol, domain, path, parameters)	This feature selection framework achieves a remarkable 87.6% reduction in feature quantity with suffering from only a 0.1% deterioration in detecting accuracy, making it possible for up-date training and real-time detecting in a production environment.
2021	Eth-PSD: A Machine Learning-Based Phishing Scam Detection Approach in Ethereum	Arkan Hammoodi Hasan Kabla; Mohammed Anbar; Selvakumar Manickam; Shankar Karupayah	Detect phishing scam-related transactions using a novel machine learning-based approach. Eth-PSD tackles some of the limitations in the existing works, such as the use of imbalanced datasets, complex feature engineering, and lower detection accuracy.	Proposed Eth-PSD to detect the phishing scam in Ethereum. Started with derived requirements based on the limitations of related works and other effective IDSs from previous related works.

Year	Title	Author	Methodology	Conclusion/Results
2019	OFS-NN: An Effective Phishing Websites Detection Model Based on Optimal Feature Selection and Neural Network	Erzhou Zhu; Yuyang Chen; Chengcheng Ye; Xuejun Li; Feng Liu	In the proposed OFS-NN, a new index, feature validity value (FVV), is first introduced to evaluate the impact of sensitive features on the phishing websites detection. Then, based on the new FVV index, an algorithm is designed to select the optimal features from the phishing websites.	This algorithm could properly deal with problems of big number of phishing sensitive features and the continuous changes of features. Consequently, it can mitigate the over-fitting problem of the neural network classifier.
2020	Comparison of Classification Algorithms for Detection of Phishing Websites	Paulius Vaitkevicius	Compare classic supervised machine learning algorithms on all publicly available phishing datasets with predefined features and to distinguish the best performing algorithm for solving the problem of phishing websites detection, regardless of a specific dataset design.	The comparison results are presented in this paper, showing ensembles and neural networks outperforming other classical algorithms.
2020	Phishing Detection using Random Forest, SVM and Neural Network with Backpropagation	Sindhu, Sunil Parameshwar Patil,Arya Sreevalsan, Faiz Rahman	The paper explains the improved Random Forest classification method, SVM classification algorithm and Neural Network with backpropagation classification methods which have been implemented with accuracies of 97.369%, 97.451% and 97.259% respectively.	This paper explains the existing machine learning methods that are used to detect phishing websites.
2014	Intelligent rule-based phishing websites classification	Rami M. Mohammad,Fadi Thabtah,Lee McLusky	The authors shed light on the important features that distinguish phishing websites from legitimate ones and assess how good rule-based data mining classification techniques are in predicting phishing websites and which classification technique is proven to be more reliable.	These features extracted automatically without any intervention from the users using computerised developed tools.

Year	Title	Author	Methodology	Conclusion/Results
2016	Phishing sites detection based on Url Correlation	Ying Xue, Yang Li,Yuangang Yao, Xianghui Zhao, Jianyi Liu,Ru Zhang	proposed Vulnerable Sites List and a new feature which is named URL Correlation. URL Correlation is based on the similarity of URLs with the List above that we created.	a large improvement of accuracy is observed by comparing methods which use our new feature with the others which use the normal one.
2022	Characteristics of Understanding URLs and Domain Names Features: The Detection of Phishing Websites With Machine Learning Methods	Ilker Kara; Murathan Ok; Ahmet Ozaday	The proposed method simplifies the process of feature extraction, and reduces processing overhead while going beyond analyzing on HTML, DOM, and URL based features by considering URLs, and domain names.	A minimum loss in data conversion, selecting the appropriate machine learning technique, and consistency of definitions in the data set.
2018	Detecting phishing websites using machine learning technique	Ashit Kumar Dutta, Zhihan Lv	The proposed framework employs RNN—LSTM to identify the properties Pm and Pl in an order to declare an URL as malicious or legitimate.	The proposed method (LURL) is developed in Python 3.0 with the support of Sci—Kit Learn and NUMPY packages. Also, the existing URL detectors are constructed for evaluating the performance of LURL. LURL has produced an average of 97.4% and 96.8% for Phishtank and Crawler datasets respectively.
2022	Web Phishing Detection Using Machine Learning	N Kumaran, Purandhar Sri Sai, Lokesh Manikanta	• Data Collection • Data Pre-Processing • Feature Extraction • Evaluation Model	Machine learning methods were imported using the Scikit-learn library. Each classification is performed using a training set, and the performance of the classifiers is evaluated using a testing set. The accuracy score of classifiers was calculated to assess their performance.

Year	Title	Author	Methodology	Conclusion/Results
2021	Detection of Phishing Websites using Machine Learning	Atharva Deshpande, Omkar Pedamkar, Nachiket Chaudhary	Collect unstructured data of URLs from Phishtank website, Kaggle website and Alexa website, etc. Train the three unique classifiers and analyse their presentation based on exactness two classifiers utilized are Decision Tree and Random Forest algorithm.	Scikit-learn tool has been used to import Machine learning algorithms. Each classifier is trained using training set and testing set is used to evaluate performance of classifiers. Performance of classifiers has been evaluated by calculating classifiers accuracy score. improve the accuracy of our models with better feature extraction.
2018	A New Method for Detection of Phishing Websites: URL Detection	Shraddha Parekh, Dhwanil Parikh , Srushti Kotak , Prof. Smita Sankhe	Random forest algorithm is implemented using Rstudio. The parsed dataset undergoes heuristic classification where the dataset is spilt into 70% and 30%. The 70% data is considered for training and 30% for testing.	In this paper, a different methodology has been proposed to detect phishing websites by using random forests as the classification algorithm with the help of Rstudio.
2018	Detection of URL based Phishing attacks Using Machine Learning	Ms. Sophiya Shikalgar , Dr. S. D. Sawarkar , Mrs. Swati Narwane	Hybrid Algorithm Approach is a mixture of different classifiers working together which gives good prediction rate and improves the accuracy of the system.	This system provides us with 85.5 % of accuracy for XG Boost Classifier, 86.3% accuracy for SVM Classifier, 80.2 % accuracy for NaÃ¯ve Bayes Classifier and finally 85.6 percentage of accuracy when using Stacking Classifier.
2010	Large-Scale Automatic Classification of Phishing Pages	Collin Whittaker,Brian Ryner,Maria Nasif	We describe the design and performance characteristics of a scalable machine learning classifier we developed to detect phishing websites. We use this classifier to maintain Googles phishing blacklist automatically.	Despite the noise in the training data, our classifier learns a robust model for identifying phishing pages which correctly classifies more than 90% of phishing pages several weeks after training concludes.

Problem Statement

- Phishing has a list of negative effects on a Business, including loss of money, loss of intellectual property, damage to reputation, and disruption of operational activities.
- According to the FBI, phishing incidents nearly doubled in frequency, from 114,702 incidents in 2019, to 241,324 incidents in 2020.
- Therefore, we suggest a phishing detection model based on machine learning that compares the features of the target websites mainly the URLs

Objectives

- To extract features that can produce effective accuracy in model evaluation
- To analyze the accuracy level for different machine learning algorithms and implementing the best among them
- To design and Implement a Web Application to search and detect whether it is phishing or not.
- To store URLs, which are detected as either phishing or non-phishing

Scope

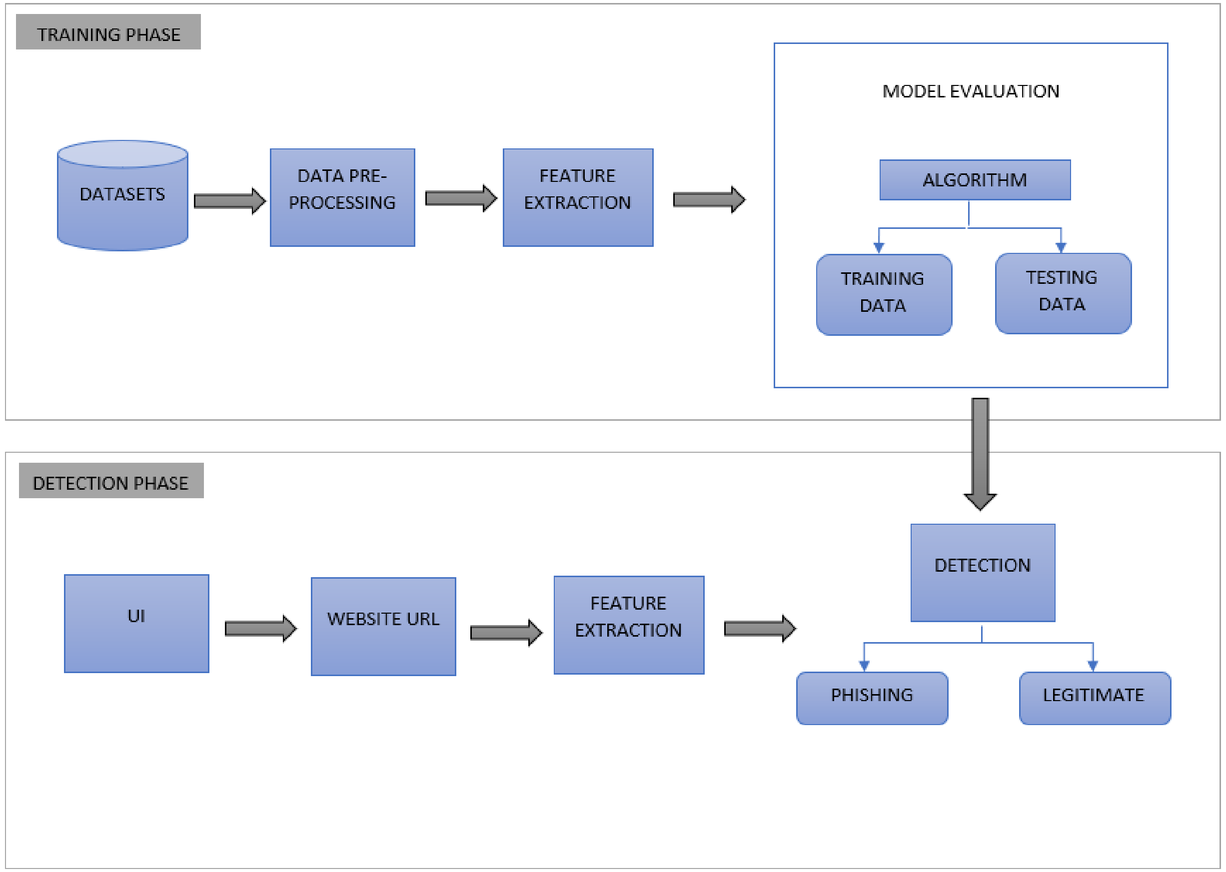
- Algorithm will analyze various blacklisted and legitimate URL features to accurately detect the phishing websites including zero-hour phishing websites.
- A total of 5,45,895 samples are used as dataset.
- And 74 different features are extracted from this large dataset for evaluating the model

Methodology

System Architecture Design :

2 PHASES :

1. Training Phase
2. Detection Phase



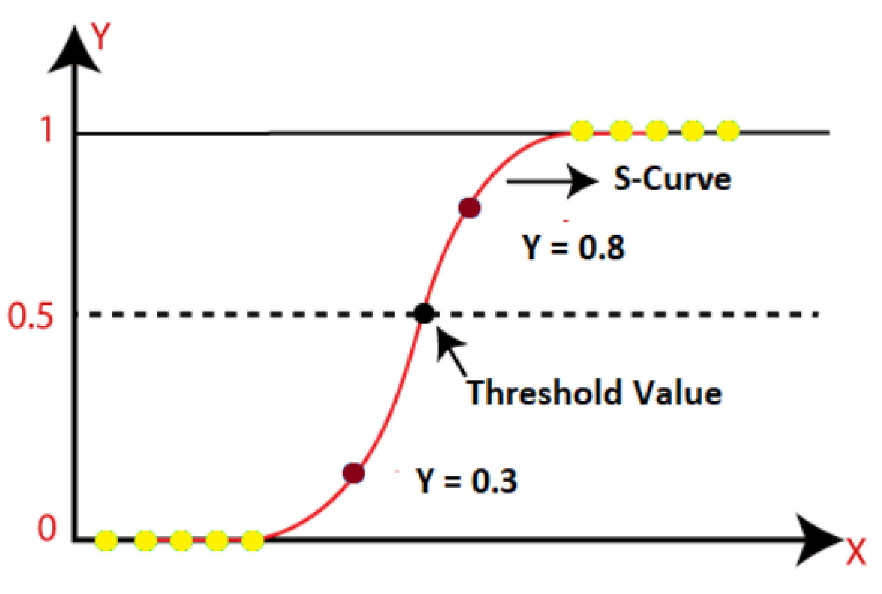
1. **TRAINING PHASE**
   - Datasets:
     - Dataset-I : from phishtank.com, contains 96,020 data, columns- domain & label
     - Dataset –II : from Kaggle.com, contains 450,176 data, columns – Unnamed, urls, label & result
   - Data pre-processing:
     - Data Cleaning
     - Data Reduction
     - Data Integration

	domain	label
0	nobell.it/70ffb52d079109dca5664cce6f317373782/...	1.0
1	www.dghjdgf.com/paypal.co.uk/cycgi-bin/webscrc...	1.0
2	serviciosbys.com/paypal.cgi.bin.get-into.herf....	1.0
3	mail.printakid.com/www.online.americanexpress....	1.0
4	thewhiskeydregs.com/wp-content/themes/widescre...	1.0

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

- Feature Extraction:
    - Lexical Features
    - Numerical Features
- Model Evaluation:
    - Logistic Regression Algorithm

2. **DETECTION PHASE**
    - User Interface:
        - React Web Application
        - Nodejs API server
    - Website URL: Input URL
    - Detection:
        - Phishing
        - Non-Phishing



System Requirements

I. Software Requirements

- **Programming Languages:** Python, JavaScript
- **Integrated Development Environment (IDE):** VisualStudioCode, Google-Colab
- **Libraries:** numpy, panda, scikit-learn, Matplotlib/Seaborn, pycaret, pickle, React
- **Frameworks:** Express.js
- **Web Browsers:** Google Chrome, Mozilla Firefox, Microsoft Edge
- **Package Managers:** pip, npm
- **Version Control:** Git, Github

II. Hardware Requirements

- **Processor:** Intel i5 and above
- **RAM:** 8gb and above

Implementation

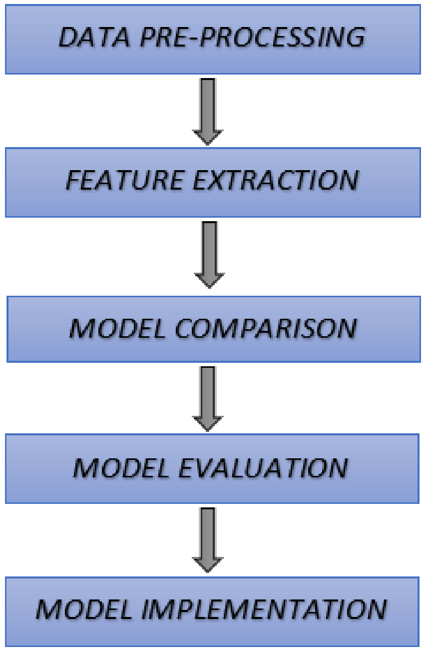
System Implementation is divided into 5:

1. Data Pre-processing
2. Feature Extraction
3. Model Comparison
4. Model Evaluation
5. Model Implementation

1. Data Pre-processing

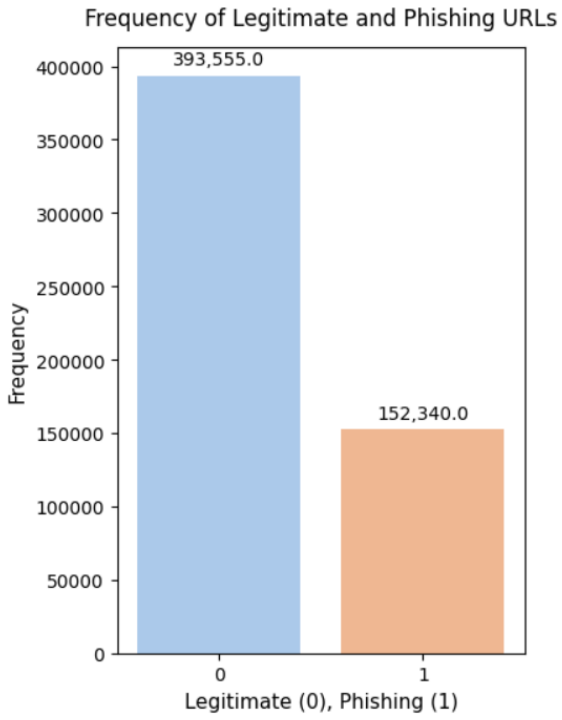
Inputs: Dataset I and Dataset II

- Data Cleaning
  - Dropping Null Values
- Data Reduction
  - Dropping Unwanted Columns



- Data Integration
    - Changing Data Types
    - Changing Column names
    - Merging the 2 Datasets
    - Removing Duplicates

Output: New Dataset after pre-processing




```python
import pandas as pd

df=pd.read_csv("../data/initial_urls.csv")
df2=pd.read_csv("../data/additional_urls.csv")

def initial_read(df): ...

df.dropna(inplace=True)
df2.drop(columns = ['Unnamed: 0', 'label'], inplace=True)

df['label']=df['label'].astype(int)

df.rename(columns={"domain": "url", "label": "phishing"},inplace=True)
df2.rename(columns={"result": "phishing"},inplace=True)

df['url']= 'https://' + df['url'].astype(str)

df_final = pd.concat([df,df2])
df_final .drop_duplicates(inplace=True)

df_final.to_csv('../data/final_urls.csv', index=False)
print(df.columns)
print(df2.columns)
```

2. Feature Extraction

1. Lexical Features
2. Numerical Features

Input: Dataset generated after pre-processing

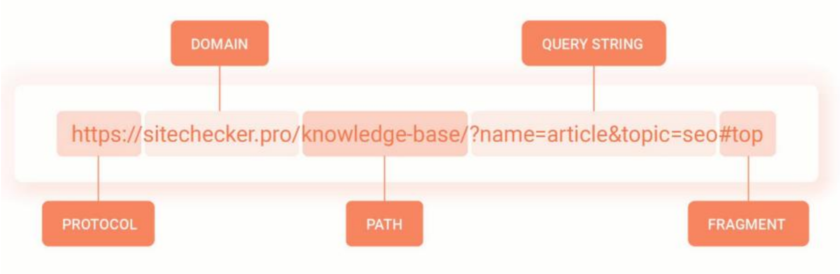
i. Lexical Features

- Involve analyzing the textual components and patterns within a URL.
- These features capture characteristics related to the structure, keywords, and other textual elements of a URL.

Slno	Lexcial Features	Description
1	**getEntropy**	Measuring the entropy of URL strings
2	**hasLogin**	Check if the URL contains specific keyword ”login”
3	**Redirection**	Check for the presence of redirection in URL string
4	**lenClassify**	Check if the length of the URL is greater than or equal 54 characters
5	**haveAtSign**	Checks for the presence of ’@’ symbol in the URL
6	**getDepth**	Calculate the number of subpages in the given URL
7	**tinyURL**	Check if the URL is URL shortened
8	**isDomainIp**	Check if there is IP address instead of hostname
9	**prefixSufix**	Check the presence of ’-’ in the domain of URL

ii. Numerical Features

- Features that represent quantitative or continuous values.
- These features can take on a wide range of numeric values



- ➢ URL text features are basically classified into - protocol, domain, path, query, fragment.
- ➢ The length of this each feature (excluding protocol) and the count of the different special characters in that specific feature are extracted
- ➢ The special characters like ‘.’ ‘-’ ‘/’ ‘?’ ‘=’ ‘@’ ‘&’ ‘!’ ‘ ’ ‘~’ ‘,’ ‘+’ ‘*’ ‘#’ ‘$’ ‘%’ .
- ➢ Number of features = 65

Finally, a total of 74 features are extracted.

Output: New Dataset with extracted Features

Numerical Features

url_length	qty_asterisk_url	qty_and_path	qty_questionmark_query	qty_dot_fragment
qty_dot_url	qty_hashtag_url	qty_exclamation_path	qty_equal_query	qty_hyphen_fragment
qty_hyphen_url	qty_dollar_url	qty_space_path	qty_at_query	qty_slash_fragment
qty_slash_url	qty_percent_url	qty_tilde_path	qty_and_query	qty_questionmark_fragment
qty_questionmark_url	domain_length	qty_comma_path	qty_exclamation_query	qty_equal_fragment
qty_equal_url	qty_dot_domain	qty_plus_path	qty_space_query	qty_and_fragment
qty_at_url_	qty_hyphen_domain	qty_asterisk_path	qty_tilde_query	qty_exclamation_fragment
qty_and_url	path_length	qty_dollar_path	qty_comma_query	qty_space_fragment
qty_exclamation_url	qty_dot_path	qty_percent_path	qty_plus_query	qty_comma_fragment
qty_space_url	qty_hyphen_path	query_length	qty_asterisk_query	qty_asterisk_fragment
qty_tilde_url	qty_slash_path	qty_dot_query	qty_dollar_query	qty_hashtag_fragment
qty_comma_url	qty_equal_path	qty_hyphen_query	qty_percent_query	qty_dollar_fragment
qty_plus_url	qty_at_path	qty_slash_query	fragment_length	qty_percent_fragment

a) Lexical Feature Extraction

```python
def getDepth(url):
    s = urlparse(url).path.split('/')
    depth = 0
    for j in range(len(s)):
        if len(s[j]) != 0:
            depth = depth+1
    return depth

def tinyURL(url):
    match=re.search(shortening_services,url)
    if match:
        return 1
    else:
        return 0

def isDomainIp(domain):
    domain = domain.split(':')
    pattern = r'^(?:[0-9]{1,3}\.){3}[0-9]{1,3}$|^ \
                (?:[a-f0-9]{1,4}:){7}[a-f0-9]{1,4}$'
    match = re.match(pattern, domain[0])
    if match is not None:
        return 1
    else:
        return 0

def prefixSuffix(domain):
    if '-' in domain:
        return 1
    else:
        return 0
```

```python
def getEntropy(url):
    url = url.lower()
    probs = [url.count(c) / len(url) for c in set(url)]
    entropy = -sum([p * log(p) / log(2.0) for p in probs])
    return entropy

def hasLogin(url):
    return int('login' in url.lower())

def redirection(url):
    pos = url.rfind('//')
    if pos > 6:
        if pos > 7:
            return 1
        else:
            return 0
    else:
        return 0

def lenClassify(url):
    if len(url) < 54:
        length = 0
    else:
        length = 1
    return length

def haveAtSign(url):
    if "@" in url:
        at = 1
    else:
        at = 0
    return at
```

b) Numerical Feature Extraction

```python
needed_cols = ['url', 'domain', 'path', 'query', 'fragment']
for col in needed_cols:
    df[f'{col}_length']=df[col].str.len()
    df[f'qty_dot_{col}'] = df[[col]].applymap(lambda x: str.count(x, '.'))
    df[f'qty_hyphen_{col}'] = df[[col]].applymap(lambda x: str.count(x, '-'))
    df[f'qty_slash_{col}'] = df[[col]].applymap(lambda x: str.count(x, '/'))
    df[f'qty_questionmark_{col}'] = df[[col]].applymap(lambda x: str.count(x, '?'))
    df[f'qty_equal_{col}'] = df[[col]].applymap(lambda x: str.count(x, '='))
    df[f'qty_at_{col}'] = df[[col]].applymap(lambda x: str.count(x, '@'))
    df[f'qty_and_{col}'] = df[[col]].applymap(lambda x: str.count(x, '&'))
    df[f'qty_exclamation_{col}'] = df[[col]].applymap(lambda x: str.count(x, '!'))
    df[f'qty_space_{col}'] = df[[col]].applymap(lambda x: str.count(x, ' '))
    df[f'qty_tilde_{col}'] = df[[col]].applymap(lambda x: str.count(x, '~'))
    df[f'qty_comma_{col}'] = df[[col]].applymap(lambda x: str.count(x, ','))
    df[f'qty_plus_{col}'] = df[[col]].applymap(lambda x: str.count(x, '+'))
    df[f'qty_asterisk_{col}'] = df[[col]].applymap(lambda x: str.count(x, '*'))
    df[f'qty_hashtag_{col}'] = df[[col]].applymap(lambda x: str.count(x, '#'))
    df[f'qty_dollar_{col}'] = df[[col]].applymap(lambda x: str.count(x, '$'))
    df[f'qty_percent_{col}'] = df[[col]].applymap(lambda x: str.count(x, '%'))

col_in_question = ['qty_slash_domain', 'qty_questionmark_domain','qty_equal_domain', 'qty_at_domain', 'qty_and_domain',
'qty_exclamation_domain', 'qty_space_domain', 'qty_tilde_domain','qty_comma_domain', 'qty_plus_domain',
'qty_asterisk_domain','qty_hashtag_domain', 'qty_dollar_domain', 'qty_percent_domain', 'qty_questionmark_path',
'qty_hashtag_path', 'qty_hashtag_query', 'qty_at_fragment','qty_tilde_fragment', 'qty_plus_fragment']

df.drop(columns = col_in_question, inplace=True)
```

3. Model Comparison

Pycaret

- PyCaret, machine learning library in Python is used to compare 15 different machine learning models.
- **compare_models()** function of pycaret intiates the comparison and returns a table of model performance metrics sorted by a specified evaluation metric.

Result: Logistic Regression shows highest Accuracy

```
import pandas as pd
from pycaret.classification import *
import time

start_time = time.time()

df = pd.read_csv('/content/drive/MyDrive/ML-Phishing Detection Project/data/url_features.csv')
data = df.drop(columns=['url','protocol','domain','path','query','fragment'])

s = setup(data, target = 'phishing', session_id = 123)
best = compare_models()

print("\n--- pycaret check completed in %s seconds ---" % (time.time() - start_time))
```

4. Model Evaluation

Evaluating the LR model includes the following steps:

- Splitting the dataset into training and testing sets
- Fitting the logistic regression model to the training data
- Use the trained model to predict test dataset.
- Calculating evaluation metrics(accuracy, precision, recall, F1-score from confusion matrix)

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import time
start_time = time.time()
import warnings
warnings.filterwarnings("ignore")

df = pd.read_csv("../data/url_features.csv")

x = df.drop(columns=['url','protocol','domain','path','query','fragment','phishing'])
y = df['phishing']
X_train, X_test, y_train, y_test = train_test_split(x,y,random_state = 42, stratify = y )

clf = LogisticRegression(penalty="l2",C=10,max_iter=1000, random_state=42)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

acc = accuracy_score(y_test, y_pred)*100
train_accuracy = clf.score(X_train, y_train)*100
test_accuracy = clf.score(X_test, y_test)*100

print(f"LogisticRegression Accuracy: {acc}")
print("Training accuracy:", train_accuracy)
print("Test accuracy:", test_accuracy)

print("\n--- Model Evaluation ended in %s seconds ---" % (time.time() - start_time))
```

5. Model Implementation

This include the following processes:

- Saving the trained model in a serialized format
- Integrating the model into the target Web Application
- Get the input URL from the user
- Extract the 74 different features from the input URL
- Loading the serialized model into the implementation environment
- Utilize the loaded model to generate predictions on the extracted features.
- Output Delivery : The prediction result is then passed to the API

a) api.js

```
app.post('/api/url', (req, res) => {
    const data = req.body.url;
    try {
        parsedUrl = new URL(data);
        const dataStr = JSON.stringify(data);
        const pythonProcess = spawn('python', ['./predictor.py']);
        pythonProcess.stdin.write(dataStr);
        pythonProcess.stdin.end();
        let outputData = '';
        pythonProcess.stdout.on('data', (data) => {
            outputData += data;
        });
        pythonProcess.stderr.on('data', (data) => {
            res.status(400).json(error)
            console.error(`stderr: ${data}`);
        });
        pythonProcess.on('error', (error) => {
            res.status(400).json(error)
            console.error(`Python process error: ${error}`);
        });
        pythonProcess.on('close', (code) => {
        if (code !== 0) {
            res.status(505).json(`Python process exited with code ${code}`)
            console.error(`Python process exited with code ${code}`);
        } else {
            const result = JSON.parse(outputData);
            res.status(200).json(result.phishing_value)
        }
        });
    } catch (err) {
        res.status(400).json(err)
    }
})
```

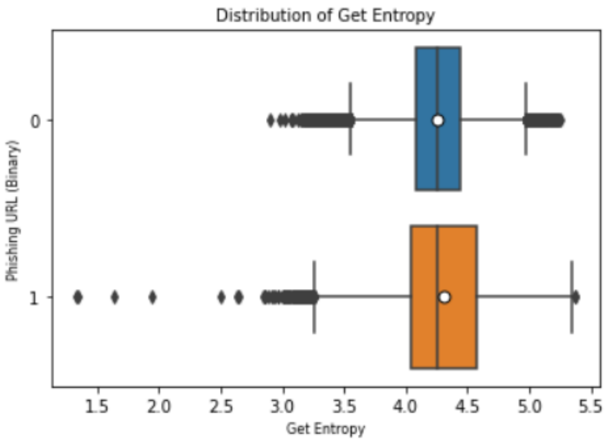
b) predictor.py

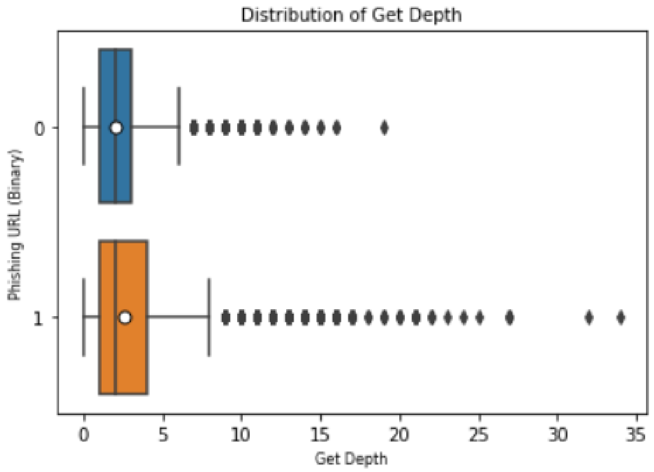
```
with open('./model/model.pkl', 'rb') as file: #LOAD MODEL
    model = pickle.load(file)
with open(filename, 'w', newline='') as file:
    df = pd.read_csv(filename,names=['url'])
    url=[]
    data = json.loads(sys.stdin.read()) # RECEIVE INPUT
    url.append(data)
    df["url"]=url
    urls = [url for url in df['url']]
    df['protocol'],df['domain'],df['path'],df['query'],df['fragment'] = \
                        zip(*[urllib.parse.urlsplit(x) for x in urls])
    shortening_services = r"bit\.ly|goo\.gl|shorte\.st|go2l\.ink|x\.co|ow\ \···
    # FEATURE EXTRACTION
    def getEntropy(url):···
    def hasLogin(url):···
    def redirection(url):···
    def lenClassify(url):···
    def haveAtSign(url):···
    def getDepth(url):···
    def tinyURL(url):···
    def isDomainIp(domain):···
    def prefixSuffix(domain):···
    def get_features(df):···
    get_features(df)
    df.to_csv("./dataTest/test_features.csv",index=False)
    file.close()

x_test = df.drop(columns=['url','protocol','domain','path','query','fragment'])
y_pred = model.predict(x_test) #PREDICT
result = {'phishing_value': str(y_pred[0])}
sys.stdout.write(json.dumps(result)) #SEND OUTPUT
```

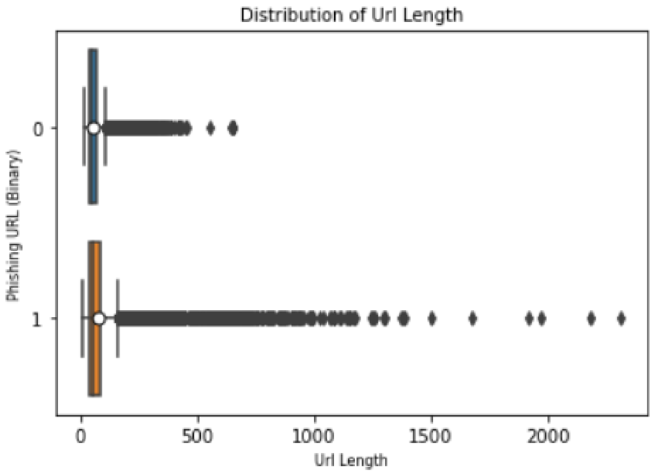
Results and Discussion

A. Feature Analysis

<u>Boxplot Graph:</u> It is employed to explore and understand the distribution of feature values during the analysis of the dataset and provides insights including measures of central tendency, dispersion, mean and identification of outliers. Visualization of some features are shown below:







Feature Importance:

- Based on the Logistic Regression Model, it has assigned feature importance score to each extracted features.
- It helps to provide insights into which features have the most influence on the model's predictions

Features	Importance
has_Login	17.50104731596883
qty_slash_url	17.081183381852934
qty_slash_query	17.07447575442481
qty_slash_path	15.593042042519762
is_Domain_Ip	13.390253867250552
fragment_length	8.262890358688013
domain_length	8.190691646606577
query_length	8.152779676465919
path_length	8.102829702824081
url_length	8.08924680042928
qty_questionmark_url	7.782155913174315
redirection	4.103847946602051
qty_questionmark_query	2.6621056912247427
qty_hashtag_url	1.7503123607980398
qty_dot_domain	1.5527849024752542
qty_space_url	1.3967219137540452
qty_at_query	1.2732887325157511
qty_equal_query	1.2387965290096226
get_Depth	1.1292673252545378
qty_space_path	1.0647616851107968
qty_equal_path	1.0638010784853535
qty_plus_url	0.9771358588903926
qty_exclamation_url	0.9299441385669271
qty_equal_fragment	0.8144508208267502
prefix_Suffix	0.7672738941547081
qty_exclamation_path	0.7656261964945975
qty_at_path	0.760984663437558
qty_equal_url	0.7548966518428326
qty_space_fragment	0.7387934507115947
qty_dot_path	0.6893527568072485
len_Classify	0.6189908642550525
get_Entropy	0.586740456967459
qty_at_url	0.5710510032327958
qty_dot_query	0.5508604180692102
have_At_Sign	0.5241119783831402
qty_plus_path	0.5155919642094619
qty_dot_fragment	0.4975024682541953
qty_hashtag_fragment	0.4884676449055037
qty_percent_fragment	0.4721693681148116
qty_plus_query	0.4615438946798377
qty_hyphen_query	0.40983814725188505
qty_percent_query	0.397959391147 6135
qty_comma_query	0.3802826938822419
qty_and_query	0.3322483356612761
qty_hyphen_domain	0.294109294011 4936
qty_tilde_query	0.29176359438556876
qty_hyphen_path	0.28695563631840615
qty_dollar_url	0.26438820981016137
qty_and_fragment	0.25891134594098597
qty_tilde_path	0.24272259917237
qty_and_path	0.234560047166241
qty_asterisk_query	0.21841953339249173
qty_space_query	0.20668753332159234
qty_hyphen_url	0.20604667522719167
qty_hyphen_fragment	0.19663781433112265
qty_comma_path	0.19436416188649505
qty_comma_url	0.194131394506 8137
qty_slash_fragment	0.19151390408523206
qty_dot_url	0.18493074066094836
qty_asterisk_path	0.17178166383517102
qty_and_url	0.16603644856400335
qty_dollar_fragment	0.16440943018532014
qty_exclamation_query	0.12563001068773397
qty_questionmark_fragment	0.12140466028934585
qty_dollar_path	0.1167456675467784
tiny_URL	0.11124906947800393
qty_percent_path	0.10913155953709368
qty_asterisk_fragment	0.08942613083734235
qty_tilde_url	0.04904099521305312
qty_asterisk_url	0.04278826128011399
qty_exclamation_fragment	0.03868793138408396
qty_percent_url	0.030244689775533563
qty_dollar_query	0.016766887922030665
qty_comma_fragment	0.0082128625104 4417

B. Pycaret Analysis

Comparison results obtained using PyCaret, guide in selecting the most suitable machine learning model(s) for this project, taking into account performance metrics, statistical significance, feature importance, and other relevant factors.Below Fig shows the comparison result.

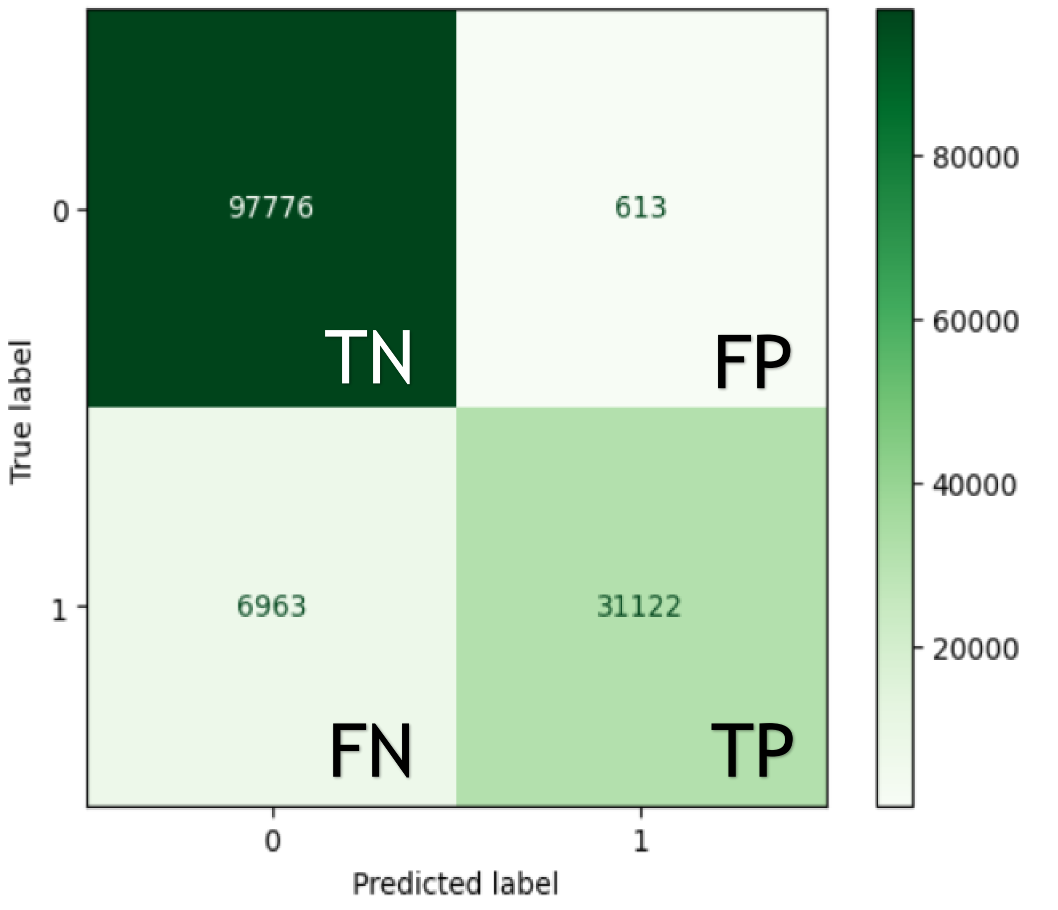
	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
lr	Logistic Regression	0.9445	0.9619	0.8166	0.9816	0.8915	0.8547	0.8612	74.4470
et	Extra Trees Classifier	0.9402	0.9677	0.8584	0.9221	0.8891	0.8483	0.8493	125.1360
rf	Random Forest Classifier	0.9400	0.9716	0.8526	0.9266	0.8880	0.8472	0.8485	114.7950
xgboost	Extreme Gradient Boosting	0.9318	0.9677	0.8028	0.9446	0.8679	0.8223	0.8273	156.8630
qda	Quadratic Discriminant Analysis	0.9313	0.9444	0.7752	0.9738	0.8630	0.8179	0.8276	5.9550
lda	Linear Discriminant Analysis	0.9284	0.9571	0.7442	0.9987	0.8529	0.8069	0.8221	9.6110
ridge	Ridge Classifier	0.9279	0.0000	0.7425	0.9988	0.8518	0.8055	0.8210	1.3870
svm	SVM - Linear Kernel	0.9271	0.0000	0.8320	0.9215	0.8679	0.8182	0.8261	16.7550
lightgbm	Light Gradient Boosting Machine	0.9226	0.9608	0.7715	0.9403	0.8475	0.7963	0.8033	9.2440
dt	Decision Tree Classifier	0.9204	0.8977	0.8463	0.8655	0.8558	0.8009	0.8010	7.8590
knn	K Neighbors Classifier	0.9187	0.9402	0.7856	0.9106	0.8435	0.7890	0.7929	264.9220
gbc	Gradient Boosting Classifier	0.9092	0.9379	0.7161	0.9454	0.8149	0.7563	0.7691	118.1240
ada	Ada Boost Classifier	0.9076	0.9321	0.7240	0.9294	0.8139	0.7537	0.7641	30.9280
nb	Naive Bayes	0.7918	0.8363	0.2822	0.9089	0.4306	0.3439	0.4324	2.9420
dummy	Dummy Classifier	0.7209	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000	1.3600

C. Model Analysis

The confusion matrix provides a more detailed understanding of the model's performance beyond simple accuracy.

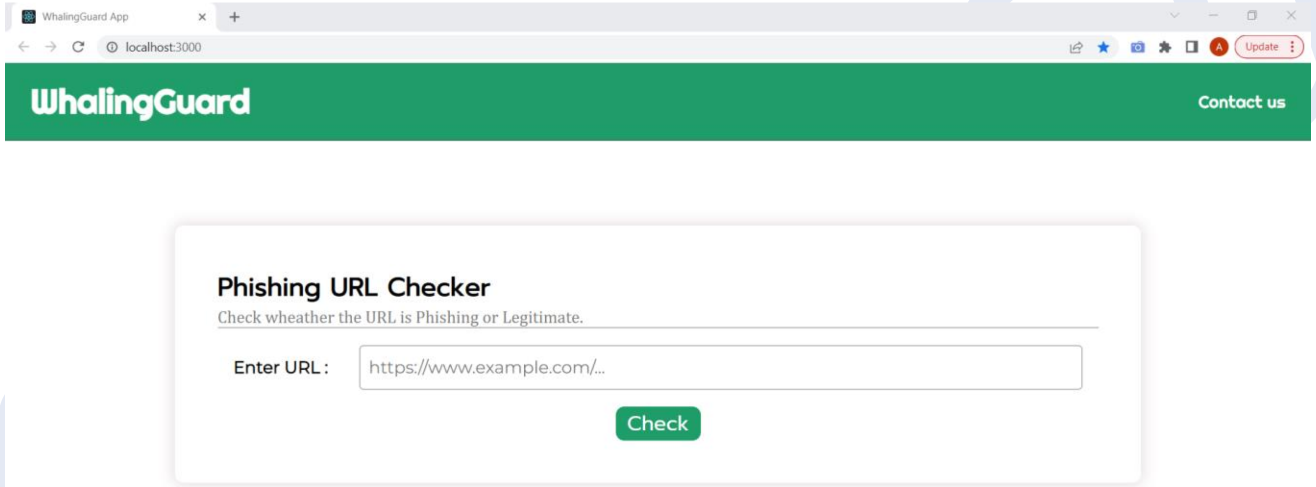
Evaluation Metrics:

```
Accuracy:  94.4487594706684
Misclassification Rate:  5.5512405293315945
Recall:  81.71721150059078
Specificity:  99.37696287186576
Precision:  98.06837876161967
```

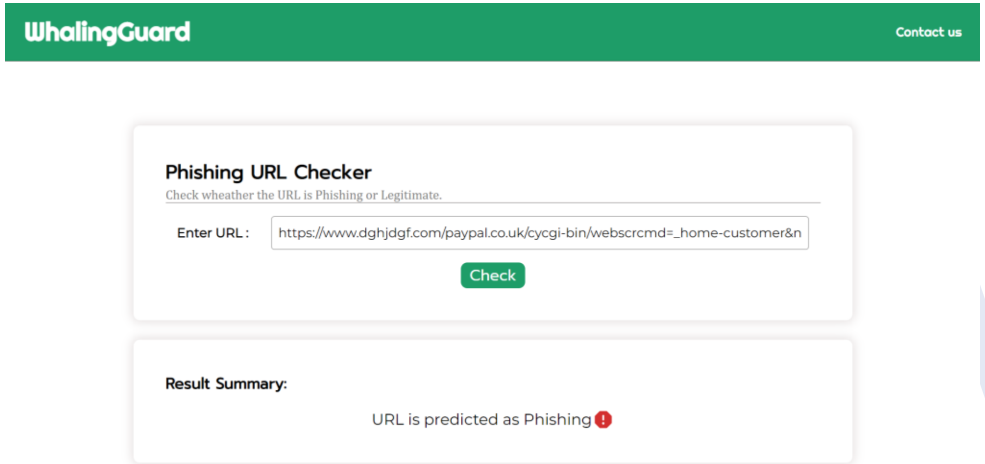


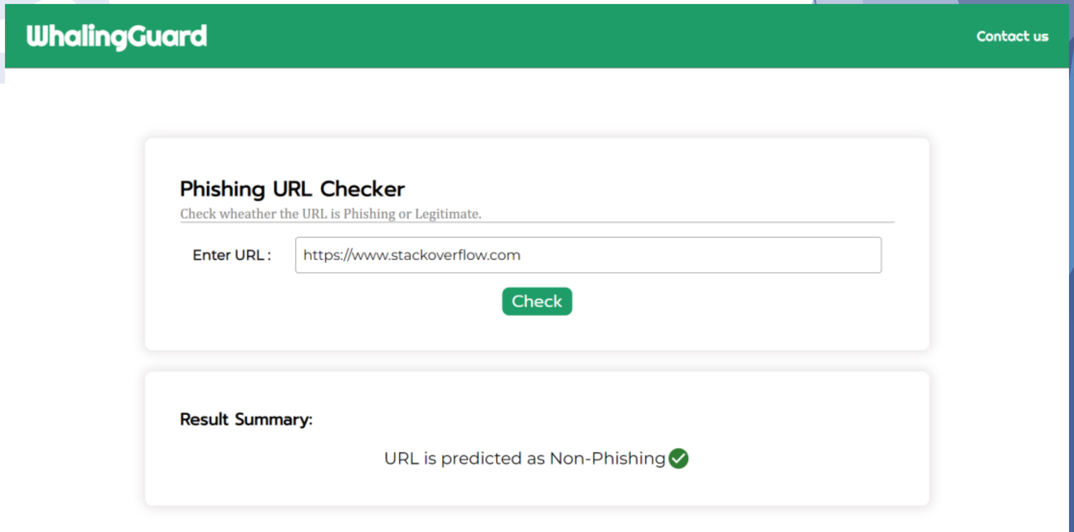
D. Web Application

We have developed a web application where the users can input URLs to check if it is phishing or non-phishing. UI of the application is developed using Reactjs and the API setup for the application is implemented using Nodejs.



The URL entered by the user is received by the server and it is predicted by the evaluated LR model. Based, on the prediction, the summary of the result is shown as output in the user-interface. The following figures shows the 2 different output produced





Conclusion

In particular, phishing has become more common and has begun to raise significant issues. There needs to design phishing detection method to affectively detect if the website is phishing or non-phishing. Considering the significance of phishing detection, in this study, we extracted 74 different features from the website URL. After comparing different machine learning algorithms, we found that Logistic Regression gives higher accuracy. So, we used Logistic Regression model for phishing prediction. In future we are planning to design a framework for identifying and preventing phishing attacks that are delivered through email messages.

References

- [1] ”PhishHaven—An Efficient Real-Time AI Phishing URLs Detection System” (Maria Sameen, Kyunghyun Han, Seong Oun Hwang [2020])
- [2] ”Sufficiency of Ensemble Machine Learning Methods for Phishing Websites Detection” (Yi Wei, Yuji Sekiya. [2021])
- [3] ”Eth-PSD: A Machine Learning-Based Phishing Scam Detection Approach in Ethereum” (Arkan Hammoodi Hasan Kabla, Mohammed Anbar, Selvakumar Manickam, Shankar Karupayah . [2021])
- [4] ”OFS-NN: An Effective Phishing Websites Detection Model Based on Optimal Feature Selection and Neural Network” (Erzhou Zhu, Yuyang Chen, Chengcheng Ye, Xuejun Li, Feng Liu. [2019])
- [5] ”Comparison of Classification Algorithms for Detection of Phishing Websites” (Paulius Vaitkevicius. [2020])
- [6] ”Phishing Detection using Random Forest, SVM and Neural Network with Backpropagation” (Sindhu, Sunil Parameshwar Patil, Arya Sreevalsan, Faiz Rahman [2020])
- [7] “Intelligent rule-based phishing websites classification” (Rami M. Mohammad, Fadi Thabtah, Lee McLusky. [2014])
- [8] ”Phishing sites detection based on Url Correlation” (Ying Xue, Yang Li,Yuangang Yao, Xianghui Zhao, Jianyi Liu,Ru Zhang. [2016])

- [9] ”Characteristics of Understanding URLs and Domain Names Features: The Detection of Phishing Websites With Machine Learning Methods” (Ilker Kara, Murathan Ok, Ahmet Ozaday . [2022])
- [10] Detecting phishing websites using machine learning technique” (Ashit Kumar Dutta, Zhihan Lv. [2018])
- [11] ”Web Phishing Detection Using Machine Learning ” (N Kumaran, Purandhar Sri Sai, Lokesh Manikanta. [2022])
- [12] ”Detection of Phishing Websites using Machine Learning” (Atharva Deshpande, Omkar Pedamkar, Nachiket Chaudhary . [2021])
- [13] New Method for Detection of Phishing Websites: URL Detection” (Shraddha Parekh, Dhwanil Parikh , Srushti Kotak , Prof. Smita Sankhe. [2018])
- [14] ”Detection of URL based Phishing attacks Using Machine Learning” (Ms. Sophiya Shikalgar , Dr. S. D. Sawarkar , Mrs. Swati Narwane. [2018])
- [15] ”Large-Scale Automatic Classification of Phishing Pages ” (Collin Whittaker, Brian Ryner, Maria Nasif. [2010])

Thank You