Malicious URL detection using machine learning techniques based on lexical features

Report submitted to the SASTRA Deemed to be University
as the requirement for the course

INT300 - MINI PROJECT

Submitted by

Repala Srinivas Yaswanth
Reg.No:123015128, Information Technology
Annam Venkata sai
Reg.No:123003283, Computer Science and Engineering
Chevuru Venkata Rajesh
Reg.No:123003284, Computer Science and Engineering

January 2022

SCHOOL OF COMPUTING
THANJAVUR, TAMIL NADU, INDIA - 613401

SCHOOL OF COMPUTING
THANJAVUR – 613 401

Bonafide Certificate

This is to certify that the report titled **"Malicious URL Detection Using Machine Learning Techniques Based On Lexical Features"** submitted as a requirement for the course, **INT300: MINI PROJECT** for B.Tech is a Bonafide record of the work done by **Mr. Repala Srinivas Yaswanth(Reg.No:123015128, Information Technology), Mr. Annam Venkata sai (Reg.No:123003283, Computer Science and Engineering), Mr. Chevuru Venkata Rajesh (Reg.No:123003284, Computer Science and Engineering)** academic year 2020-21, in the School of Computing, under my supervision.

Signature of the Project Supervisor	: [illegible]
Name with Affiliation	: Dr. Priya S, AP-3, SOC
Date	: 29-06-2022

Mini Project *Viva voce* held on 29-06-2022

Examiner 1 [illegible] **Examiner 2** [illegible]

Acknowledgements

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

We would like to thank our honourable Vice-Chancellor **Dr. S. Vaidhiyasubramaniam** and **Dr. S. Swaminathan**, Dean, Planning & Development, for the encouragement and strategic support throughout our college life. We extend our sincere thanks to **Dr. R. Chadramouli**, Registrar, SASTRA Deemed to be University for providing us the opportunity to pursue this project.

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

Our guide **Dr. Priya S**, Assistant Professor III, was the driving force behind this whole idea from the start. Her deep insight in the field and invaluable suggestions helped us in making progress throughout our project work. We also thank the project review panel members **Mr.Srinivasan B** and **Dr.AnushiaDevi R** for their valuable comments and insights which made this project better.

We would like to extend our gratitude to all the teaching and non-teaching faculties of the School of Computing for being a strong pillar of support.

List of Figures

Abbreviations

The student must take utmost care in the use of technical abbreviations. For example, "KM" stands for "Kelvin Mega" and not kilometre (which should be abbreviated as km) and "gms" stands for "gram meter second" and "grams" (which should be abbreviated as g). In addition, abbreviations pertinent to any specific discipline should be listed in alphabetical order as shown below.

AI	Artificial Intelligence
BMI	Brain Machine Interface
CED	Cardiovascular Electronic Device
CNT	Carbon Nano Tube
EPS	Extracellular Polymeric
Substances ICMS	Intracortical Microstimulation
NMES	Neuromuscular Electrical
Stimulation NMJ	Neuromuscular Junction
PNI	Peripheral Nerve Interfaces
QS	Quorum Sensing

Notations

The student must explain the meaning of special symbols and notations used in the thesis. Define English symbols, Greek symbols, and Miscellaneous symbols separately. Some examples are presented below.

English Symbols (in alphabetical order)

k_d	Microbial decay coefficient
K_s	Substrate concentration when growth rate is half of
maximum bp	base pair
ml	Mililiter
cm	Centimeter
K	Kelvin
Hz	Hertz

Greek Symbols (in alphabetical order)

α	Rate
$\sigma(x)$	the standard deviation of x
$\sum$	Summation
Ω	Ohm
∞	Infinity

Miscellaneous Symbols (in alphabetical order)

□x□	absolute value of x

Abstract

Cyber criminals' most advanced cyber-attack approach is to create and propagate malicious domain names or malicious URLs via email, messaging, pop ups, and other means. Malicious URLs are web pages that are designed to deliver malware, viruses, and other harmful software to internet users. The main goal of the attack is to steal the victim’s personal information, user credentials, or install malware on their computer. As a result, a system must be developed to recognize bad URLs and prevent attacks. Researchers propose a variety of strategies, but machine learning-based detection outperforms them all. This study proposes a light-weighted technique that simply considers the URL’s lexical properties. As a result, the Random Forest classifier performs better than the other classifiers in terms of accuracy.

.

Table of Contents

CHAPTER-1

BASE PAPER SUMMARY

Base paper Title : Lexical features based malicious Uniform Resource Locators (URL) detection using ML Techniques.

Name of the Journal	**:** ELSEVIER
Publisher name	**:** ScienceDirect
Publication year	2021
Index	**:** Scopus

Base paper Introduction:

MachineLearning is advancing quickly across many industries, and by putting these algorithms into practice, we may even attempt to foresee the possibility of PC damage or the theft of private data. To identify URLs that are harmful to our PC, we can use ML methods like SVM,RF,LR. Using the publicly accessible Benchmark dataset, we implemented a total of 5 MachineLearning (ML) methods in this case. The project's primary goal was to better accurately forecast whether a URL is benign or malicious. In order to evaluate models, accuracy, recall score, and F1-Score are used.

1.1 SPECIFICATION OF SOFTWARE

Python: Python is considered as a high-level programming language that allows us for simple implementation of a few certain ML methods.

Python Libraries used

- **Numpy:** An open-source library called Numpy which is used in Python to manipulate arrays and perform advanced operations on multi-size arrays.
- **Pandas:** Pandas is an open-source Python module that reads data in the form of a data frame and allows for the evaluation and manipulation of that data.
- **MatplotLib**: It is used for mathematical visualization of all the attributes and provides us with various visualizations in the interface like visualizing bar graphs, pie charts etc.
- **Sklearn:** It is a machine learning library module that comes with different types of built-in algorithms, including LR, SVM, RF, and soon.
- **Seaborn:** It is used for visualizing information libraries based completely on Matplotlib Libraries.
- **Urllib Package:** It is a "urllib" MODULE that gathers different modules for communicating with URLs, including "urllib.parse" for parsing the URLs.

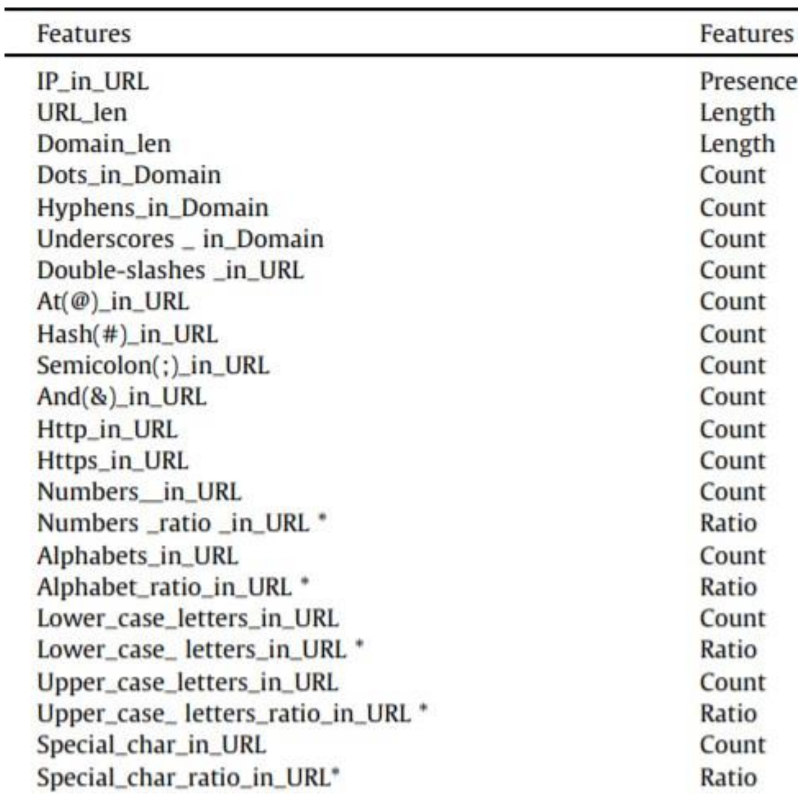
○ **PROPOSED WORK IN BASE PAPER**

Work is divided into 4 Different phases:

- Extracting the features
- Reducing the features
- Model Training
- Model Testing

Extracting the features:

In **Feature Extraction,** we extracted all the lexical features like presence of IP in a URL, characters count, URL length and many more as given in the base paper.For extracting these features from the URLs we required some dependency libraries like URL parse and re (Regular Expressions).



Features	Features
IP_in_URL	Presence
URL_len	Length
Domain_len	Length
Dots_in_Domain	Count
Hyphens_in_Domain	Count
Underscores _ in_Domain	Count
Double-slashes _in_URL	Count
At(@)_in_URL	Count
Hash(#)_in_URL	Count
Semicolon(;)_in_URL	Count
And(&)_in_URL	Count
Http_in_URL	Count
Https_in_URL	Count
Numbers__in_URL	Count
Numbers _ratio _in_URL *	Ratio
Alphabets_in_URL	Count
Alphabet_ratio_in_URL *	Ratio
Lower_case_letters_in_URL	Count
Lower_case_ letters_in_URL *	Ratio
Upper_case_letters_in_URL	Count
Upper_case_ letters_ratio_in_URL *	Ratio
Special_char_in_URL	Count
Special_char_ratio_in_URL*	Ratio

Reducing the features:

The next phase is **Feature Reduction,** here we are finding the relationship between target feature and other features using correlation analysis.Based on this correlation analysis, we are removing the unnecessary features from the dataset.Initially the dataset contains only 3 columns (URL, label and result). After performing the Feature Extraction and Feature Reduction steps, we generated a new dataset which contains all the lexical features as given in the base paper.

The Initial dataset size is (450176,3).The Size of the dataset after Feature Extraction is (450176,26).The Size of the dataset after Feature Reduction is (450176,22)

For the Feature Reduction phase we have used the Heatmap to show the correlation between lexical features and target features.With the analysis of Heatmap, we removed the below features which are not correlated with our target feature.

Removed Features are Upper_case_letters_ratio,Underscores,Hash(#),Special_char

Balanced Data Versus Imbalanced Data:

A dataset that has more no of observations in one class and less no of observations in another class is known as imbalanced data. Biasing will result from biasing the machine learning models during training. There are two types of targets in our Benchmark dataset: benign and malicious. 76.40% of the URLs in our sample are benign, while 23.20% are malicious. This dataset is unbalanced. Therefore, a sampling approach is employed to avoid this issue. Sampling involves two techniques. Undersampling and Oversampling

Over Sampling: It is the process of increasing the attributes in lower class till reaching the higher class.

Under Sampling: It is the process of decreasing the attributes in the higher class till reaching the lower class.

To overcome this problem we utilized a library called Oversampling library, Using the SMOTE Oversampling method we balanced the dataset. Dataset size- (55350, 19)

Training and Testing the Model with the Data:

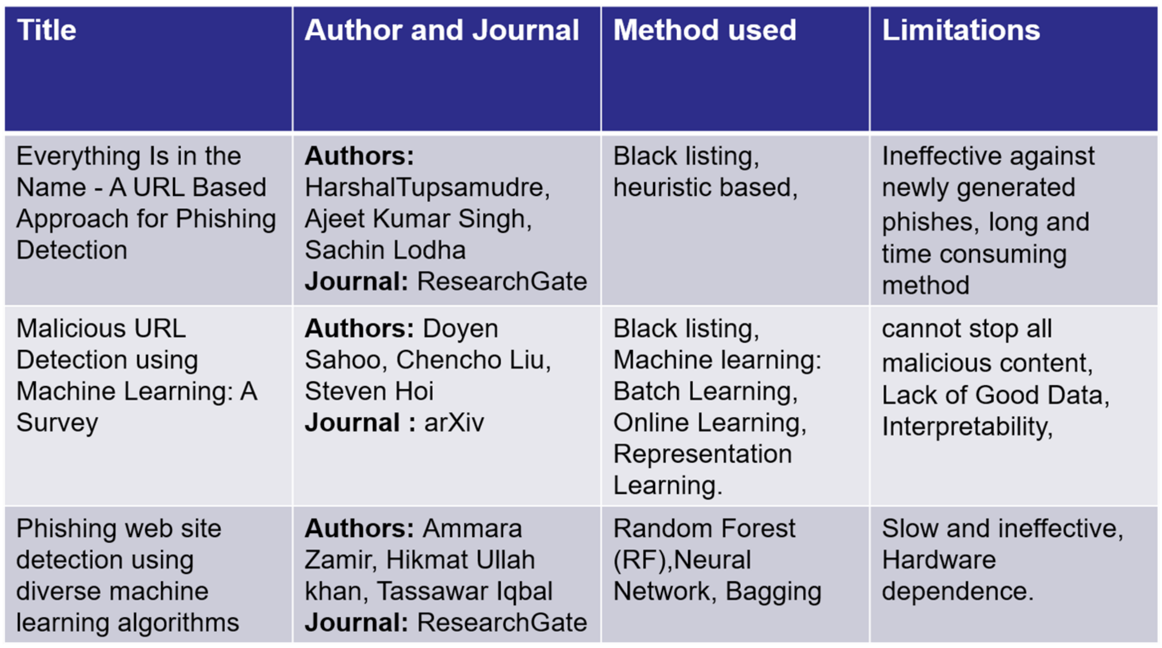
Dataset is stored in a CSV file and fed in to the classifier for training and testing with train and test split as 70:30.Using sklearn python library, we used following classifiers:

1. Support vector Classification.
2. Logistic Regression.
3. K-Nearest Neighbor.
4. Naïve Bayes.
5. Random Forest.

Evaluating the Performance Metrics:

Performance measures like **Precision,Recall,Accuracy and F1-Score**, we are interested in finding out which might be the well organized model to detect whether a URL is benign or malicious.

CHAPTER-2
BASE PAPER MERITS AND DE-MERITS



Title	Author and Journal	Method used	Limitations
Everything Is in the Name - A URL Based Approach for Phishing Detection	**Authors:** HarshalTupsamudre, Ajeet Kumar Singh, Sachin Lodha **Journal:** ResearchGate	Black listing, heuristic based,	Ineffective against newly generated phishes, long and time consuming method
Malicious URL Detection using Machine Learning: A Survey	**Authors:** Doyen Sahoo, Chencho Liu, Steven Hoi **Journal :** arXiv	Black listing, Machine learning: Batch Learning, Online Learning, Representation Learning.	cannot stop all malicious content, Lack of Good Data, Interpretability,
Phishing web site detection using diverse machine learning algorithms	**Authors:** Ammara Zamir, Hikmat Ullah khan, Tassawar Iqbal **Journal:** ResearchGate	Random Forest (RF),Neural Network, Bagging	Slow and ineffective, Hardware dependence.

Blacklist-based detection,Heuristic rules-based detection and Machine Learning/deep learning based detection are few present methods used for malicious URL detection..

Blacklist based technique :
This technique keeps track of a sizable list of dangerous URLs that have been detected either manually or automatically. This method's main benefits are minimal false positive rates and quick detection. One of the limitations is that it must be updated frequently and promptly if it is to identify freshly created dangerous URLs. Additionally, cybercriminals never carry out additional assaults using the existing URLs.

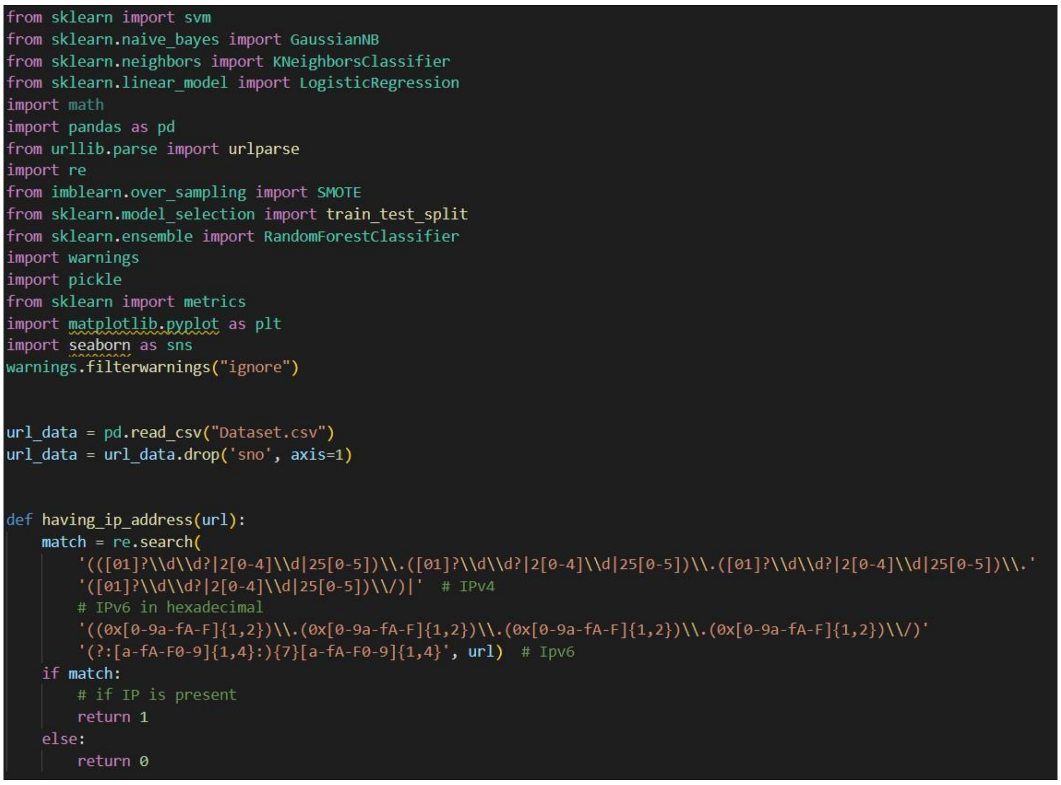
Heuristic rule-based technique :
Researchers developed a novel, heuristic rule-based technique to address these problems. Based on the features that are taken from the datasets, this technique develops rules. Feature extraction typically involves machine learning or deep learning methods. The fact that we have to set the threshold value and give the rules weights are drawbacks of this approach. It is really challenging to adjust the threshold value for each dataset.

Machine learning technique : - Technique which we are using in our project.
This method uses the URLs to extract the features. Features namely IP presence,URL length, count of characters, etc are taken from the datasets then trained for the specific model.

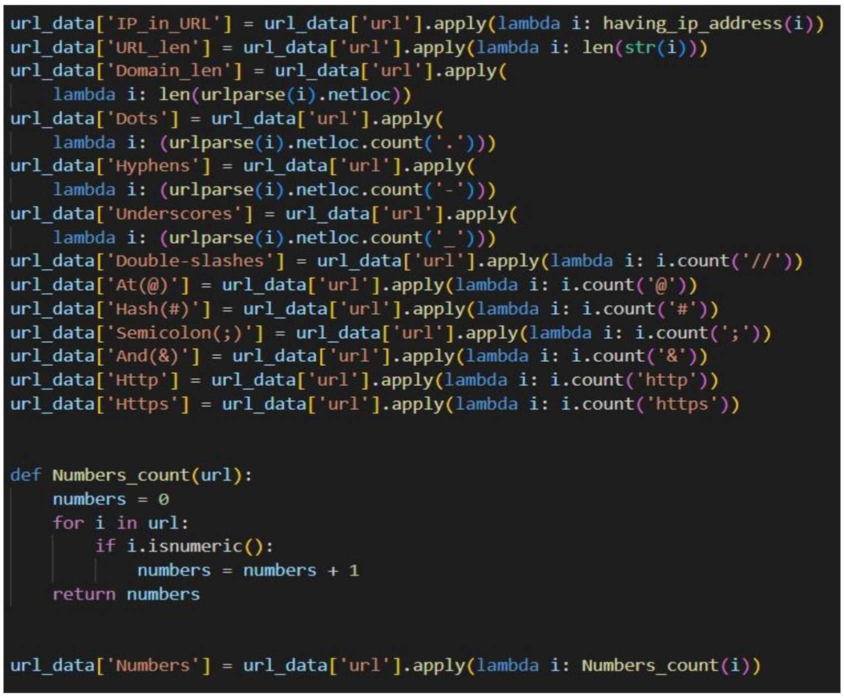
For testing the model, we used the model which is trained with the training data. As we know that, machine learning algorithms instantly provide weight to the features and can anticipate freshly created dangerous URLs.

This approach requires less storage and takes less time to execute when compared to previous approaches. This is the main advantage of the model . As opposed to blacklist-based techniques, it requires less storage because we are not storing a large list. Execution time decreases because we are just obtaining a smaller number of characteristics.

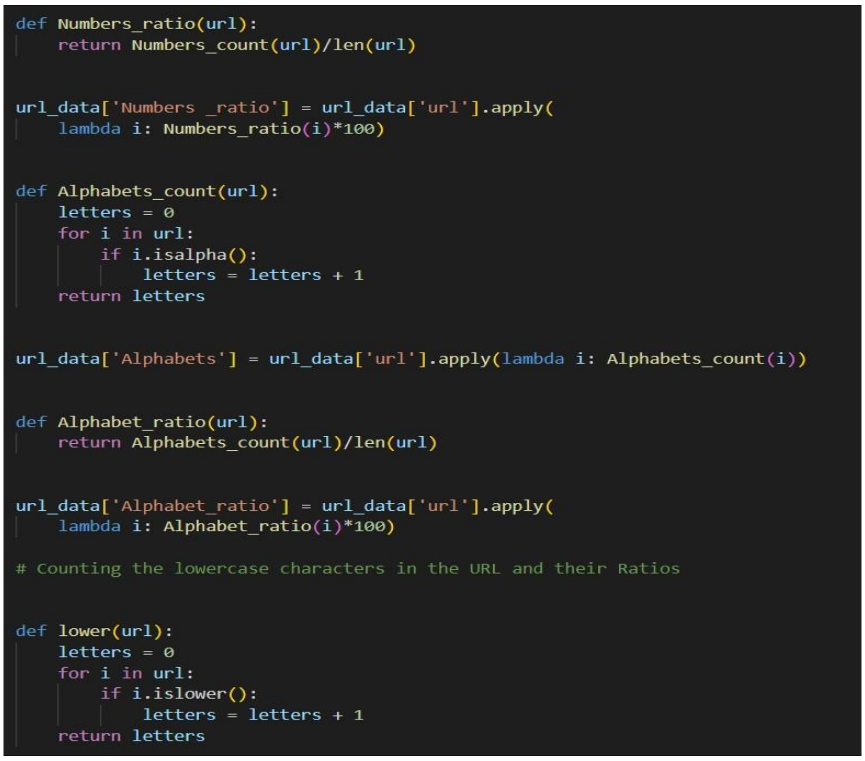
CHAPTER 3
SOURCE CODE



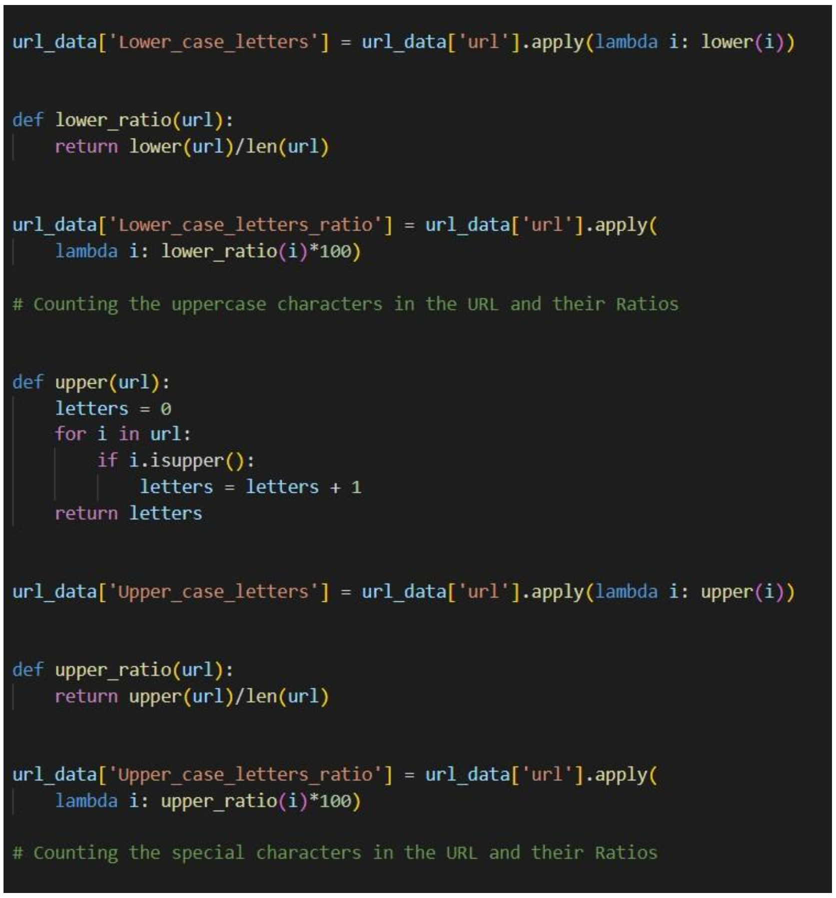
```python
from sklearn import svm
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
import math
import pandas as pd
from urllib.parse import urlparse
import re
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
import warnings
import pickle
from sklearn import metrics
import matplotlib.pyplot as plt
import seaborn as sns
warnings.filterwarnings("ignore")


url_data = pd.read_csv("Dataset.csv")
url_data = url_data.drop('sno', axis=1)


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
        # IPv6 in hexadecimal
        '((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})\\/)'
        '(?:[a-fA-F0-9]{1,4}:){7}[a-fA-F0-9]{1,4}', url)  # Ipv6
    if match:
        # if IP is present
        return 1
    else:
        return 0
```



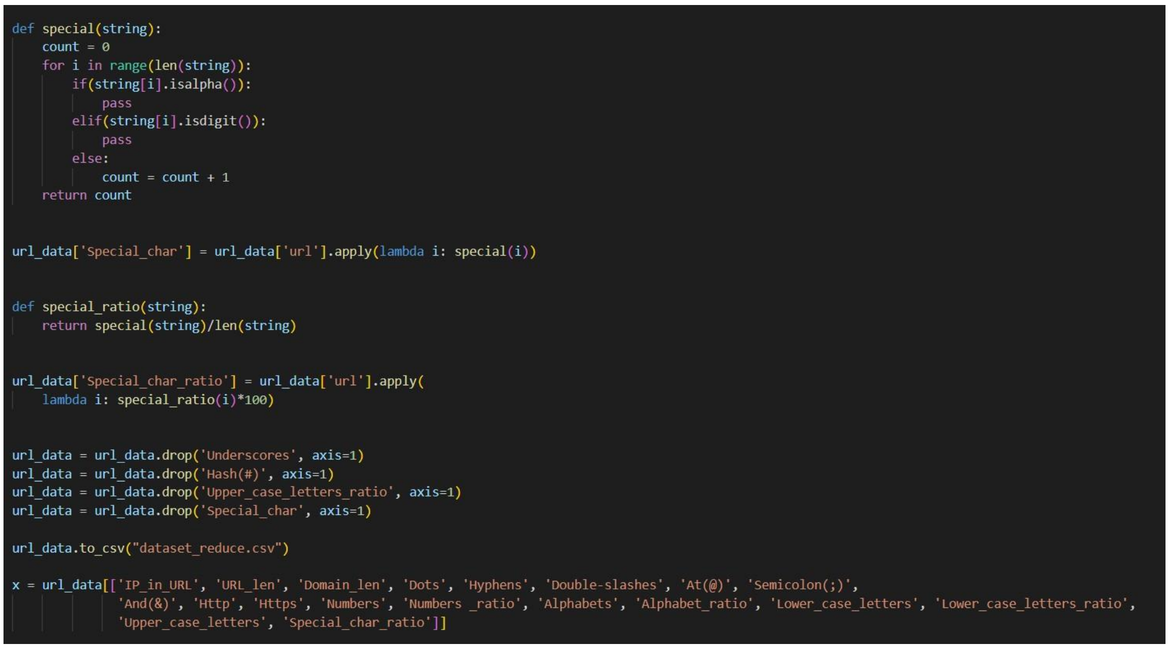
```python
url_data['IP_in_URL'] = url_data['url'].apply(lambda i: having_ip_address(i))
url_data['URL_len'] = url_data['url'].apply(lambda i: len(str(i)))
url_data['Domain_len'] = url_data['url'].apply(
    lambda i: len(urlparse(i).netloc))
url_data['Dots'] = url_data['url'].apply(
    lambda i: (urlparse(i).netloc.count('.')))
url_data['Hyphens'] = url_data['url'].apply(
    lambda i: (urlparse(i).netloc.count('-')))
url_data['Underscores'] = url_data['url'].apply(
    lambda i: (urlparse(i).netloc.count('_')))
url_data['Double-slashes'] = url_data['url'].apply(lambda i: i.count('//'))
url_data['At(@)'] = url_data['url'].apply(lambda i: i.count('@'))
url_data['Hash(#)'] = url_data['url'].apply(lambda i: i.count('#'))
url_data['Semicolon(;)'] = url_data['url'].apply(lambda i: i.count(';'))
url_data['And(&)'] = url_data['url'].apply(lambda i: i.count('&'))
url_data['Http'] = url_data['url'].apply(lambda i: i.count('http'))
url_data['Https'] = url_data['url'].apply(lambda i: i.count('https'))


def Numbers_count(url):
    numbers = 0
    for i in url:
        if i.isnumeric():
            numbers = numbers + 1
    return numbers


url_data['Numbers'] = url_data['url'].apply(lambda i: Numbers_count(i))
```



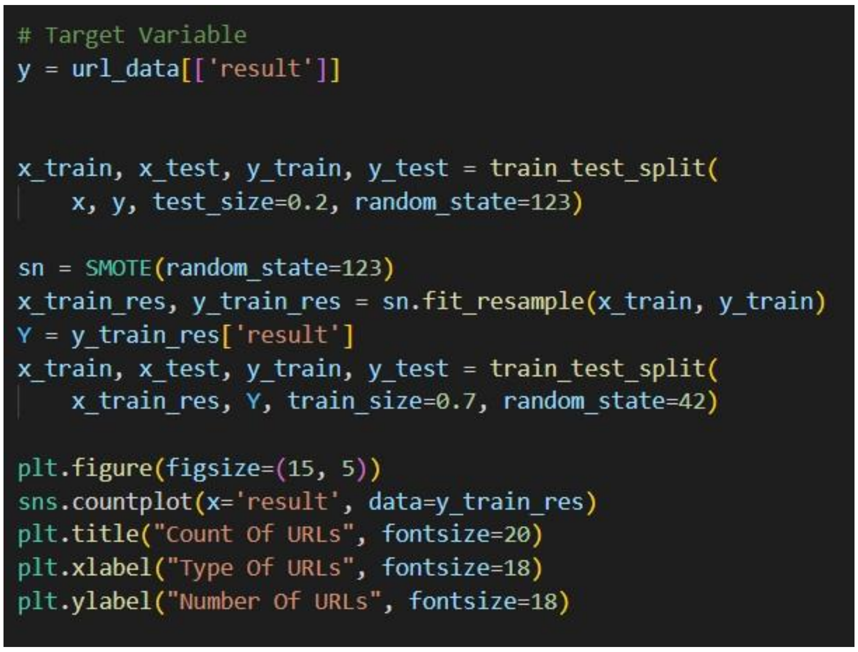
```python
def Numbers_ratio(url):
    return Numbers_count(url)/len(url)


url_data['Numbers _ratio'] = url_data['url'].apply(
    lambda i: Numbers_ratio(i)*100)


def Alphabets_count(url):
    letters = 0
    for i in url:
        if i.isalpha():
            letters = letters + 1
    return letters


url_data['Alphabets'] = url_data['url'].apply(lambda i: Alphabets_count(i))


def Alphabet_ratio(url):
    return Alphabets_count(url)/len(url)


url_data['Alphabet_ratio'] = url_data['url'].apply(
    lambda i: Alphabet_ratio(i)*100)

# Counting the lowercase characters in the URL and their Ratios


def lower(url):
    letters = 0
    for i in url:
        if i.islower():
            letters = letters + 1
    return letters
```



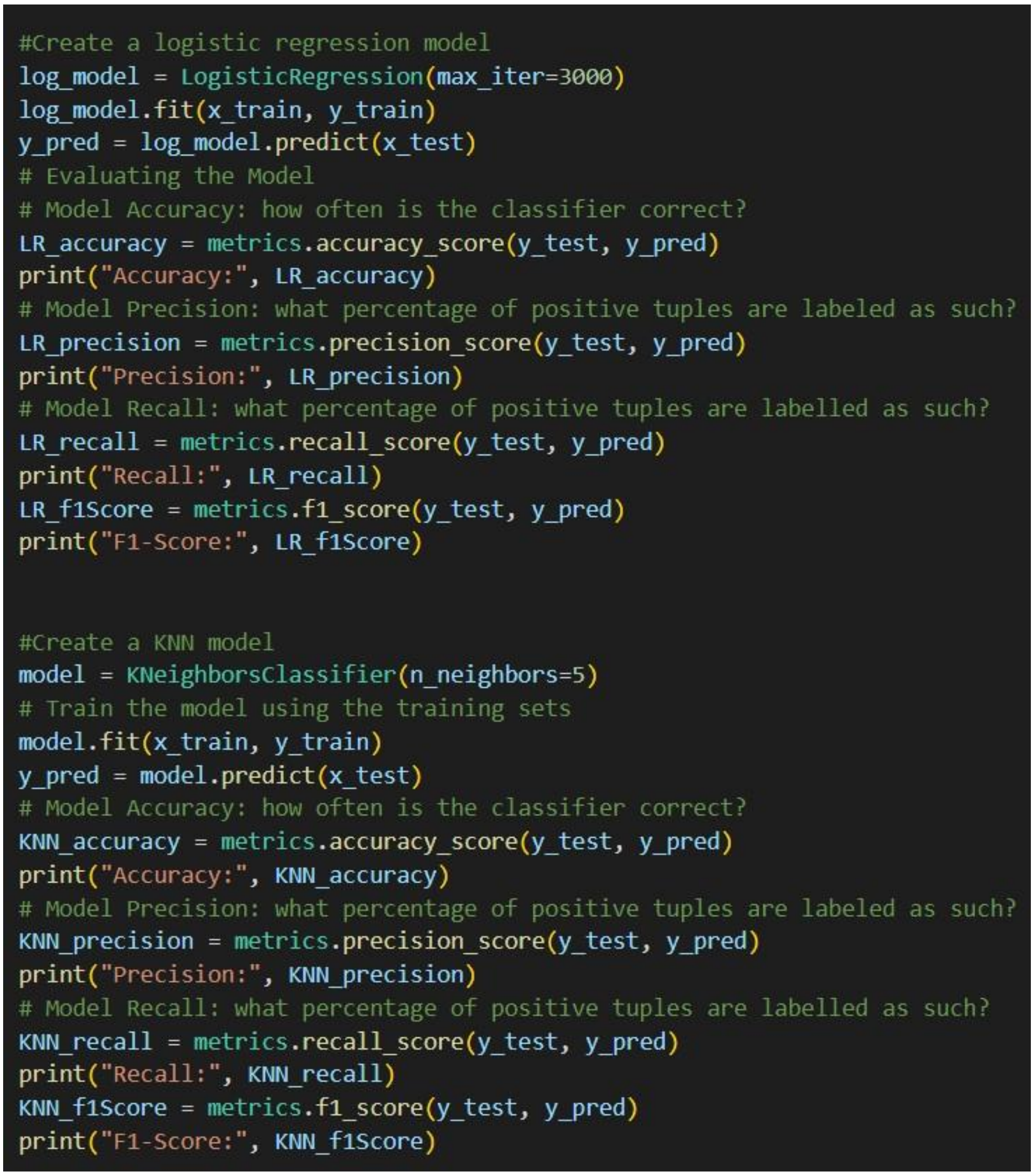
```python
url_data['Lower_case_letters'] = url_data['url'].apply(lambda i: lower(i))


def lower_ratio(url):
    return lower(url)/len(url)


url_data['Lower_case_letters_ratio'] = url_data['url'].apply(
    lambda i: lower_ratio(i)*100)

# Counting the uppercase characters in the URL and their Ratios


def upper(url):
    letters = 0
    for i in url:
        if i.isupper():
            letters = letters + 1
    return letters


url_data['Upper_case_letters'] = url_data['url'].apply(lambda i: upper(i))


def upper_ratio(url):
    return upper(url)/len(url)


url_data['Upper_case_letters_ratio'] = url_data['url'].apply(
    lambda i: upper_ratio(i)*100)

# Counting the special characters in the URL and their Ratios
```



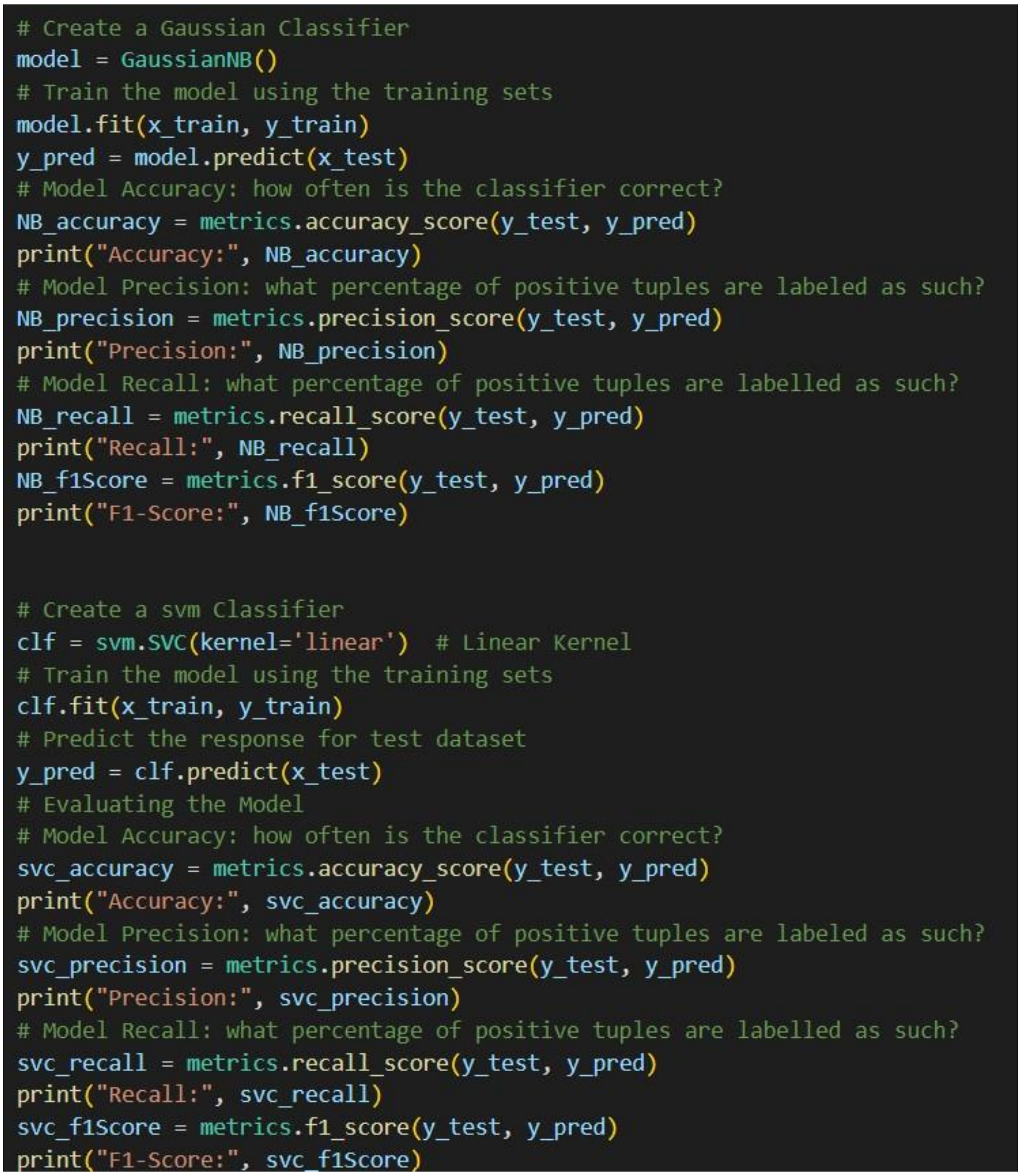
```python
def special(string):
    count = 0
    for i in range(len(string)):
        if(string[i].isalpha()):
            pass
        elif(string[i].isdigit()):
            pass
        else:
            count = count + 1
    return count


url_data['Special_char'] = url_data['url'].apply(lambda i: special(i))


def special_ratio(string):
    return special(string)/len(string)


url_data['Special_char_ratio'] = url_data['url'].apply(
    lambda i: special_ratio(i)*100)


url_data = url_data.drop('Underscores', axis=1)
url_data = url_data.drop('Hash(#)', axis=1)
url_data = url_data.drop('Upper_case_letters_ratio', axis=1)
url_data = url_data.drop('Special_char', axis=1)

url_data.to_csv("dataset_reduce.csv")

x = url_data[['IP_in_URL', 'URL_len', 'Domain_len', 'Dots', 'Hyphens', 'Double-slashes', 'At(@)', 'Semicolon(;)',
              'And(&)', 'Http', 'Https', 'Numbers', 'Numbers _ratio', 'Alphabets', 'Alphabet_ratio', 'Lower_case_letters', 'Lower_case_letters_ratio',
              'Upper_case_letters', 'Special_char_ratio']]
```



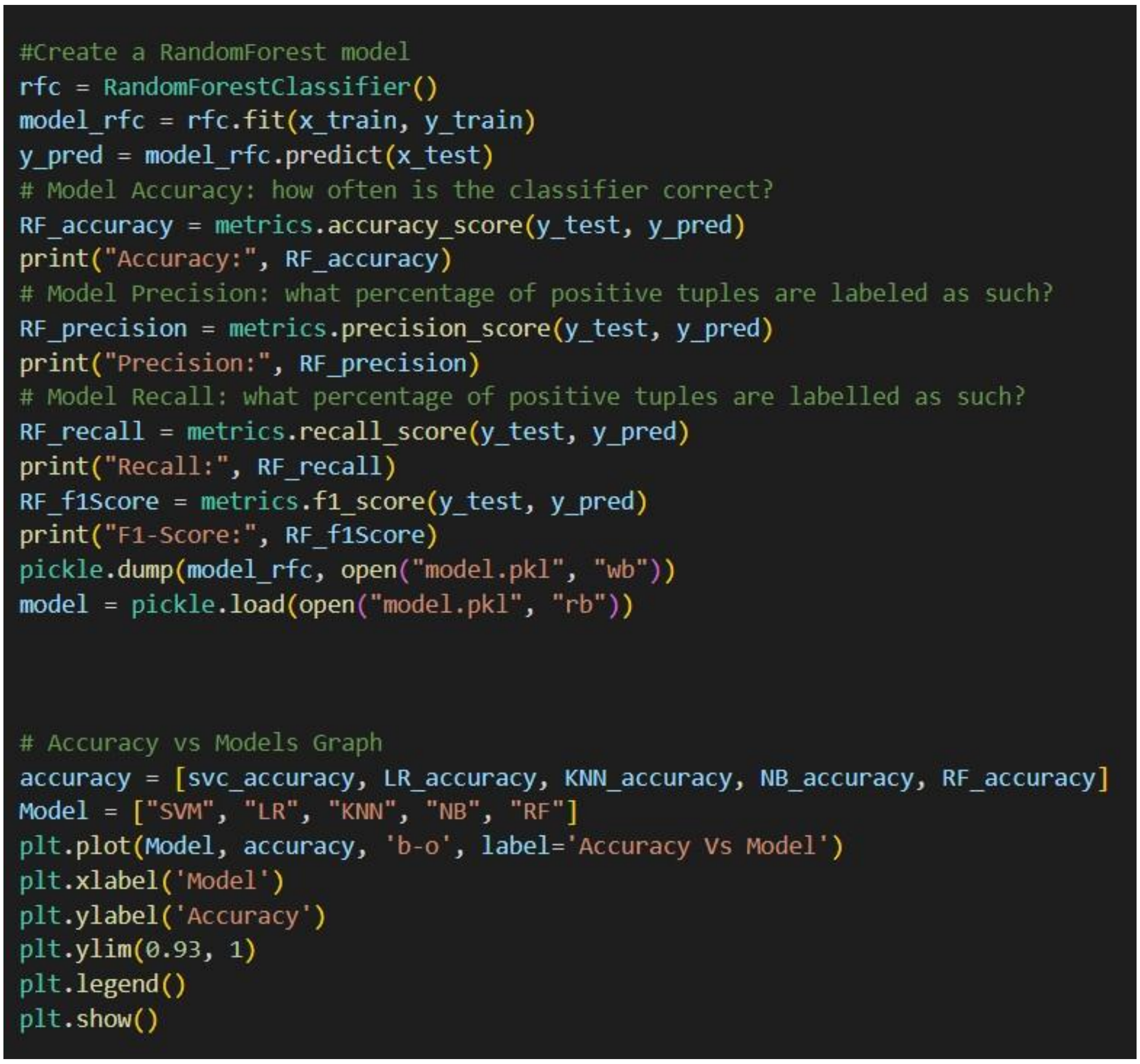
```python
# Target Variable
y = url_data[['result']]


x_train, x_test, y_train, y_test = train_test_split(
    x, y, test_size=0.2, random_state=123)

sn = SMOTE(random_state=123)
x_train_res, y_train_res = sn.fit_resample(x_train, y_train)
Y = y_train_res['result']
x_train, x_test, y_train, y_test = train_test_split(
    x_train_res, Y, train_size=0.7, random_state=42)

plt.figure(figsize=(15, 5))
sns.countplot(x='result', data=y_train_res)
plt.title("Count Of URLs", fontsize=20)
plt.xlabel("Type Of URLs", fontsize=18)
plt.ylabel("Number Of URLs", fontsize=18)
```

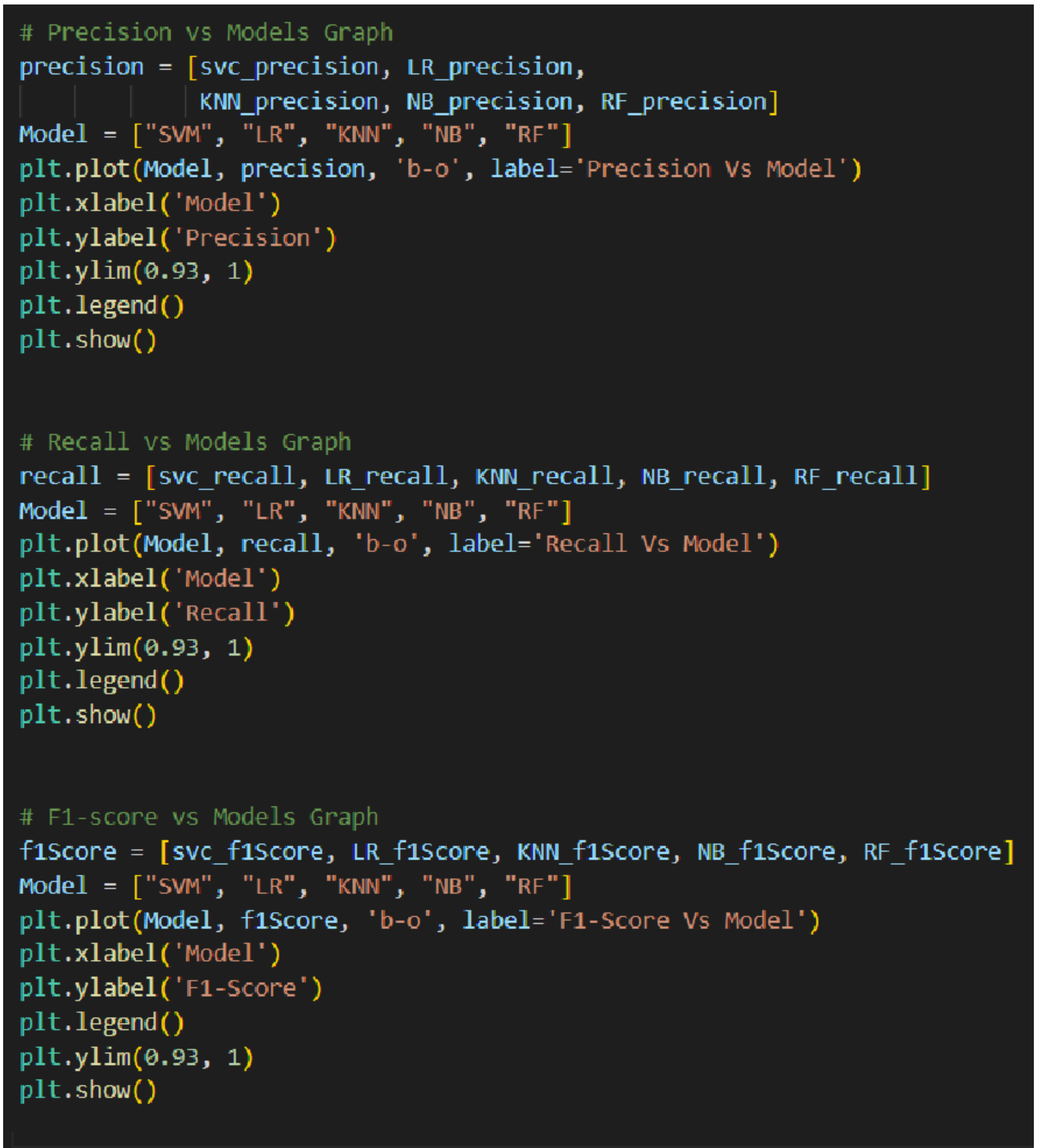


```
#Create a logistic regression model
log_model = LogisticRegression(max_iter=3000)
log_model.fit(x_train, y_train)
y_pred = log_model.predict(x_test)
# Evaluating the Model
# Model Accuracy: how often is the classifier correct?
LR_accuracy = metrics.accuracy_score(y_test, y_pred)
print("Accuracy:", LR_accuracy)
# Model Precision: what percentage of positive tuples are labeled as such?
LR_precision = metrics.precision_score(y_test, y_pred)
print("Precision:", LR_precision)
# Model Recall: what percentage of positive tuples are labelled as such?
LR_recall = metrics.recall_score(y_test, y_pred)
print("Recall:", LR_recall)
LR_f1Score = metrics.f1_score(y_test, y_pred)
print("F1-Score:", LR_f1Score)


#Create a KNN model
model = KNeighborsClassifier(n_neighbors=5)
# Train the model using the training sets
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
# Model Accuracy: how often is the classifier correct?
KNN_accuracy = metrics.accuracy_score(y_test, y_pred)
print("Accuracy:", KNN_accuracy)
# Model Precision: what percentage of positive tuples are labeled as such?
KNN_precision = metrics.precision_score(y_test, y_pred)
print("Precision:", KNN_precision)
# Model Recall: what percentage of positive tuples are labelled as such?
KNN_recall = metrics.recall_score(y_test, y_pred)
print("Recall:", KNN_recall)
KNN_f1Score = metrics.f1_score(y_test, y_pred)
print("F1-Score:", KNN_f1Score)
```

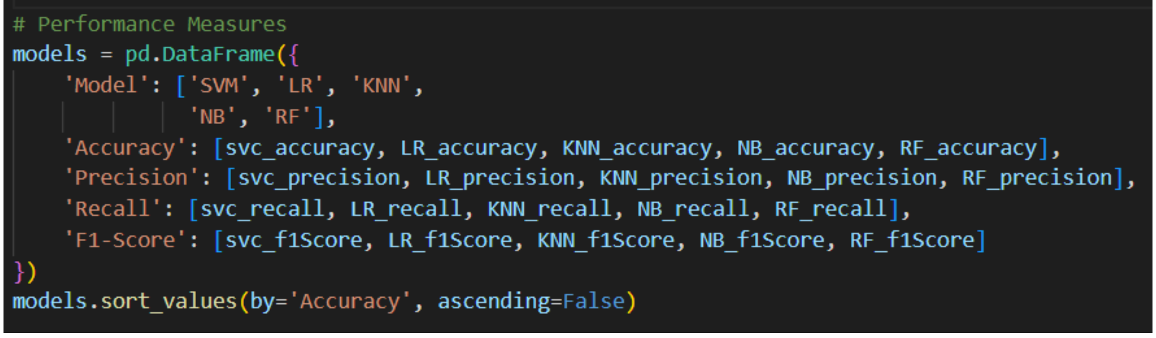


```python
# Create a Gaussian Classifier
model = GaussianNB()
# Train the model using the training sets
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
# Model Accuracy: how often is the classifier correct?
NB_accuracy = metrics.accuracy_score(y_test, y_pred)
print("Accuracy:", NB_accuracy)
# Model Precision: what percentage of positive tuples are labeled as such?
NB_precision = metrics.precision_score(y_test, y_pred)
print("Precision:", NB_precision)
# Model Recall: what percentage of positive tuples are labelled as such?
NB_recall = metrics.recall_score(y_test, y_pred)
print("Recall:", NB_recall)
NB_f1Score = metrics.f1_score(y_test, y_pred)
print("F1-Score:", NB_f1Score)


# Create a svm Classifier
clf = svm.SVC(kernel='linear')  # Linear Kernel
# Train the model using the training sets
clf.fit(x_train, y_train)
# Predict the response for test dataset
y_pred = clf.predict(x_test)
# Evaluating the Model
# Model Accuracy: how often is the classifier correct?
svc_accuracy = metrics.accuracy_score(y_test, y_pred)
print("Accuracy:", svc_accuracy)
# Model Precision: what percentage of positive tuples are labeled as such?
svc_precision = metrics.precision_score(y_test, y_pred)
print("Precision:", svc_precision)
# Model Recall: what percentage of positive tuples are labelled as such?
svc_recall = metrics.recall_score(y_test, y_pred)
print("Recall:", svc_recall)
svc_f1Score = metrics.f1_score(y_test, y_pred)
print("F1-Score:", svc_f1Score)
```


```
#Create a RandomForest model
rfc = RandomForestClassifier()
model_rfc = rfc.fit(x_train, y_train)
y_pred = model_rfc.predict(x_test)
# Model Accuracy: how often is the classifier correct?
RF_accuracy = metrics.accuracy_score(y_test, y_pred)
print("Accuracy:", RF_accuracy)
# Model Precision: what percentage of positive tuples are labeled as such?
RF_precision = metrics.precision_score(y_test, y_pred)
print("Precision:", RF_precision)
# Model Recall: what percentage of positive tuples are labelled as such?
RF_recall = metrics.recall_score(y_test, y_pred)
print("Recall:", RF_recall)
RF_f1Score = metrics.f1_score(y_test, y_pred)
print("F1-Score:", RF_f1Score)
pickle.dump(model_rfc, open("model.pkl", "wb"))
model = pickle.load(open("model.pkl", "rb"))



# Accuracy vs Models Graph
accuracy = [svc_accuracy, LR_accuracy, KNN_accuracy, NB_accuracy, RF_accuracy]
Model = ["SVM", "LR", "KNN", "NB", "RF"]
plt.plot(Model, accuracy, 'b-o', label='Accuracy Vs Model')
plt.xlabel('Model')
plt.ylabel('Accuracy')
plt.ylim(0.93, 1)
plt.legend()
plt.show()
```

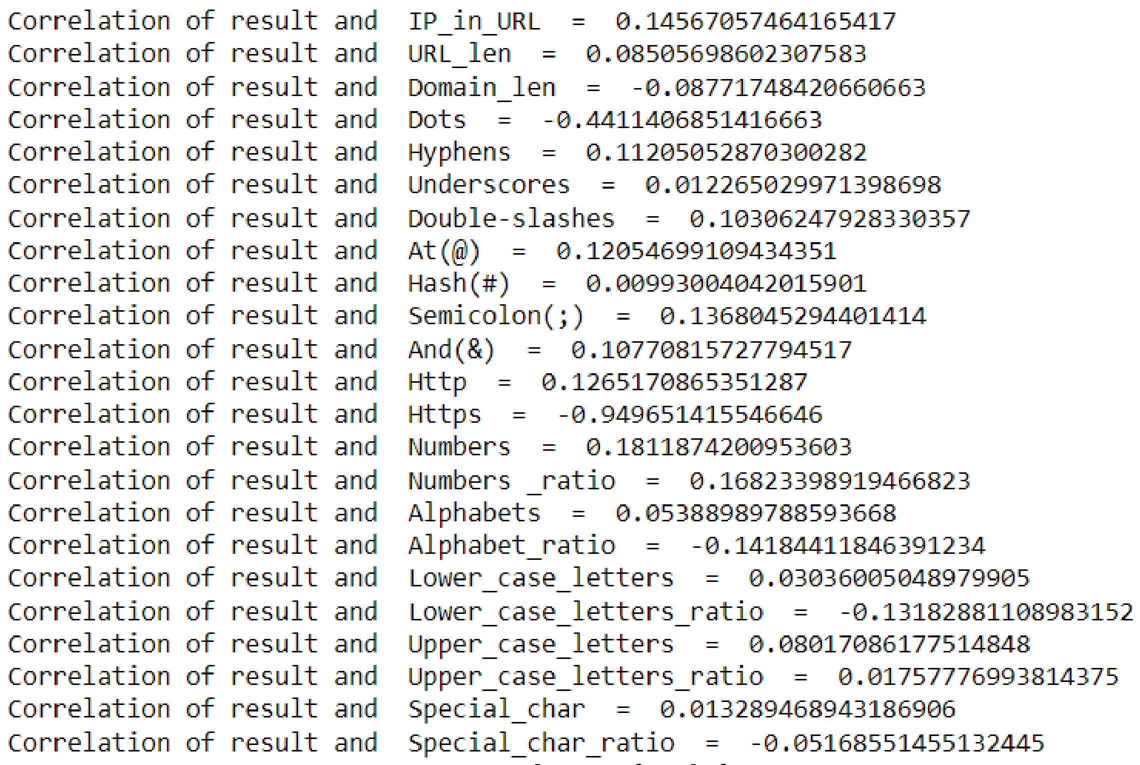


```python
# Precision vs Models Graph
precision = [svc_precision, LR_precision,
             KNN_precision, NB_precision, RF_precision]
Model = ["SVM", "LR", "KNN", "NB", "RF"]
plt.plot(Model, precision, 'b-o', label='Precision Vs Model')
plt.xlabel('Model')
plt.ylabel('Precision')
plt.ylim(0.93, 1)
plt.legend()
plt.show()


# Recall vs Models Graph
recall = [svc_recall, LR_recall, KNN_recall, NB_recall, RF_recall]
Model = ["SVM", "LR", "KNN", "NB", "RF"]
plt.plot(Model, recall, 'b-o', label='Recall Vs Model')
plt.xlabel('Model')
plt.ylabel('Recall')
plt.ylim(0.93, 1)
plt.legend()
plt.show()


# F1-score vs Models Graph
f1Score = [svc_f1Score, LR_f1Score, KNN_f1Score, NB_f1Score, RF_f1Score]
Model = ["SVM", "LR", "KNN", "NB", "RF"]
plt.plot(Model, f1Score, 'b-o', label='F1-Score Vs Model')
plt.xlabel('Model')
plt.ylabel('F1-Score')
plt.legend()
plt.ylim(0.93, 1)
plt.show()
```



```python
# Performance Measures
models = pd.DataFrame({
    'Model': ['SVM', 'LR', 'KNN',
              'NB', 'RF'],
    'Accuracy': [svc_accuracy, LR_accuracy, KNN_accuracy, NB_accuracy, RF_accuracy],
    'Precision': [svc_precision, LR_precision, KNN_precision, NB_precision, RF_precision],
    'Recall': [svc_recall, LR_recall, KNN_recall, NB_recall, RF_recall],
    'F1-Score': [svc_f1Score, LR_f1Score, KNN_f1Score, NB_f1Score, RF_f1Score]
})
models.sort_values(by='Accuracy', ascending=False)
```

CHAPTER-4
SNAPSHOTS OF RESULTS



```
Correlation of result and  IP_in_URL  =  0.14567057464165417
Correlation of result and  URL_len  =  0.08505698602307583
Correlation of result and  Domain_len  =  -0.08771748420660663
Correlation of result and  Dots  =  -0.4411406851416663
Correlation of result and  Hyphens  =  0.11205052870300282
Correlation of result and  Underscores  =  0.012265029971398698
Correlation of result and  Double-slashes  =  0.10306247928330357
Correlation of result and  At(@)  =  0.12054699109434351
Correlation of result and  Hash(#)  =  0.00993004042015901
Correlation of result and  Semicolon(;)  =  0.1368045294401414
Correlation of result and  And(&)  =  0.10770815727794517
Correlation of result and  Http  =  0.1265170865351287
Correlation of result and  Https  =  -0.949651415546646
Correlation of result and  Numbers  =  0.1811874200953603
Correlation of result and  Numbers _ratio  =  0.16823398919466823
Correlation of result and  Alphabets  =  0.05388989788593668
Correlation of result and  Alphabet_ratio  =  -0.14184411846391234
Correlation of result and  Lower_case_letters  =  0.030360050489799 05
Correlation of result and  Lower_case_letters_ratio  =  -0.13182881108983152
Correlation of result and  Upper_case_letters  =  0.08017086177514848
Correlation of result and  Upper_case_letters_ratio  =  0.017577769938143 75
Correlation of result and  Special_char  =  0.013289468943186906
Correlation of result and  Special_char_ratio  =  -0.05168551455132445
```

Fig. 2.1. Correlation of each feature

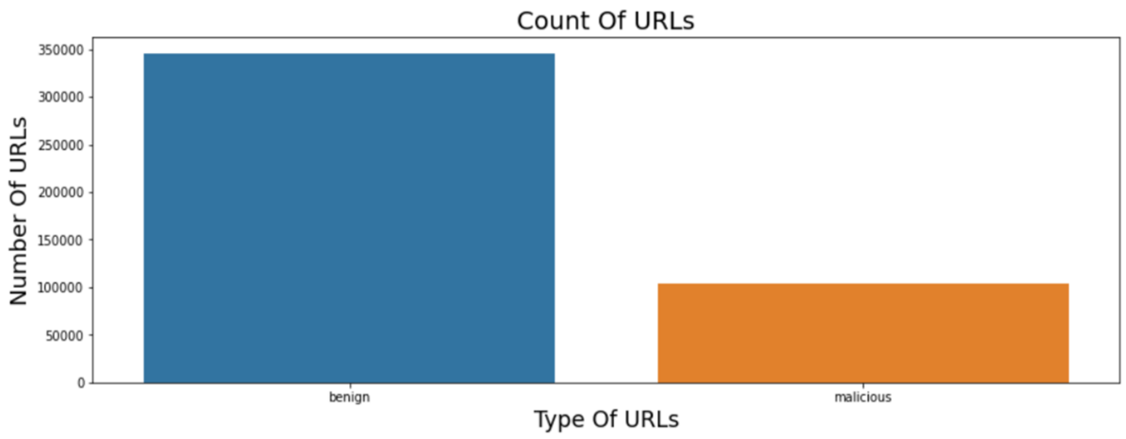


Fig. 2.2. Im-Balance problem

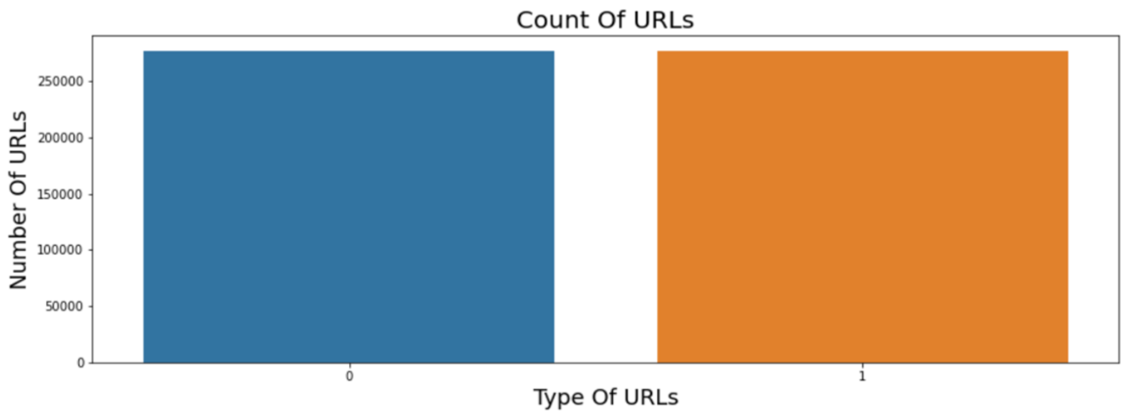


Fig. 2.3. Dataset balanced using SMOTE

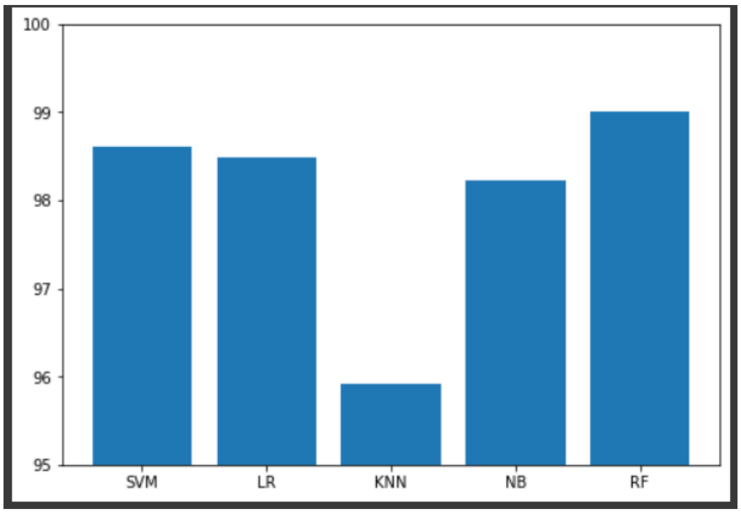


Fig. 2.4. Graph showing accuracy of the classifiers

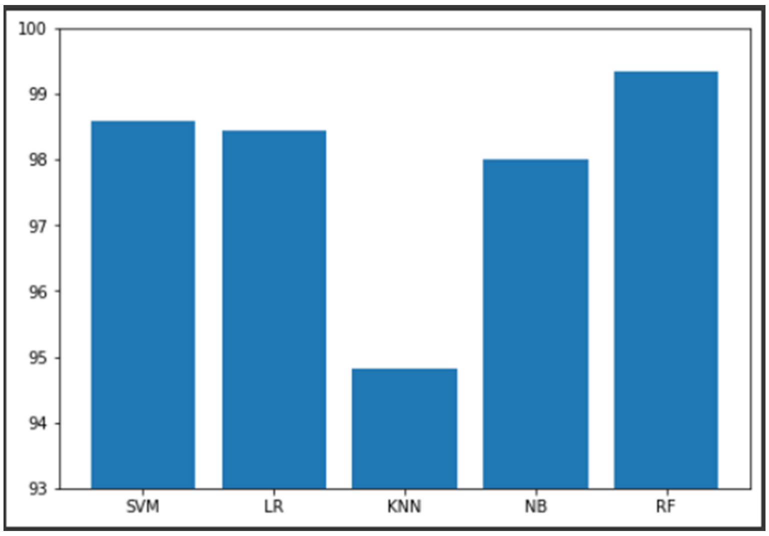


Fig. 2.5. Graph showing accuracy versus models

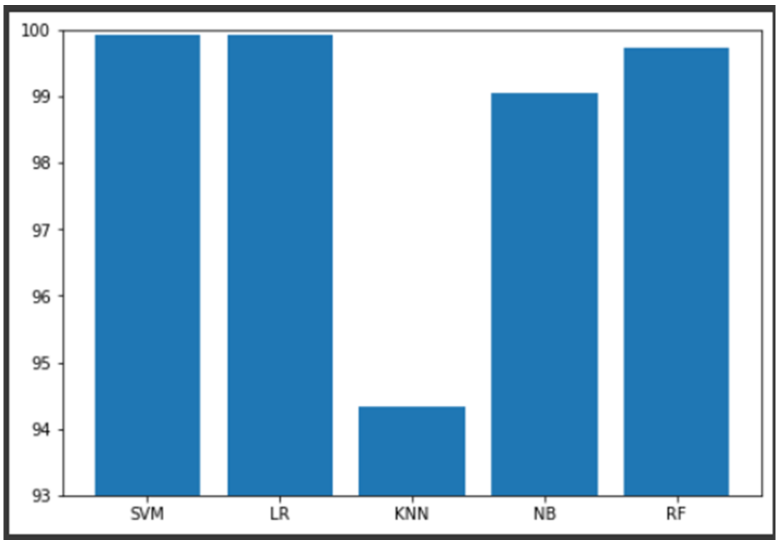


Fig. 2.6. Graph showing precision versus models

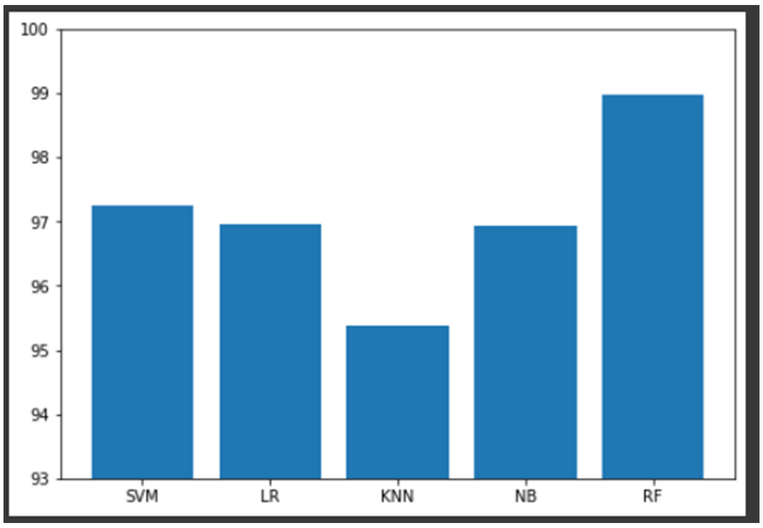


Fig. 2.7. Graph showing recall versus models

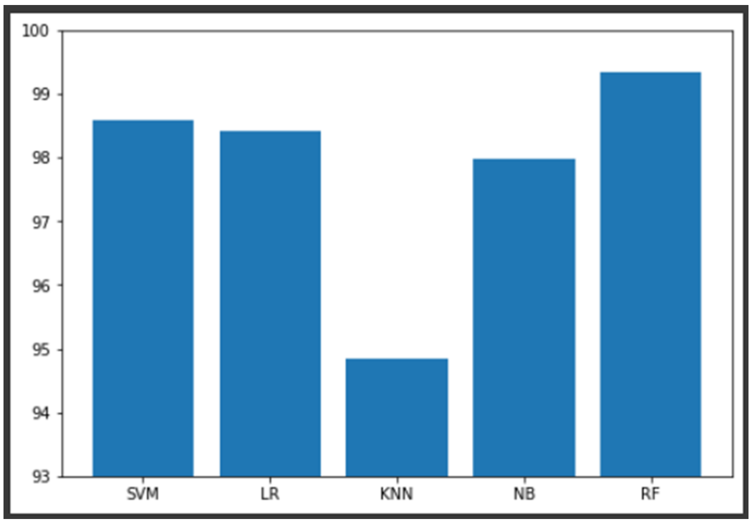
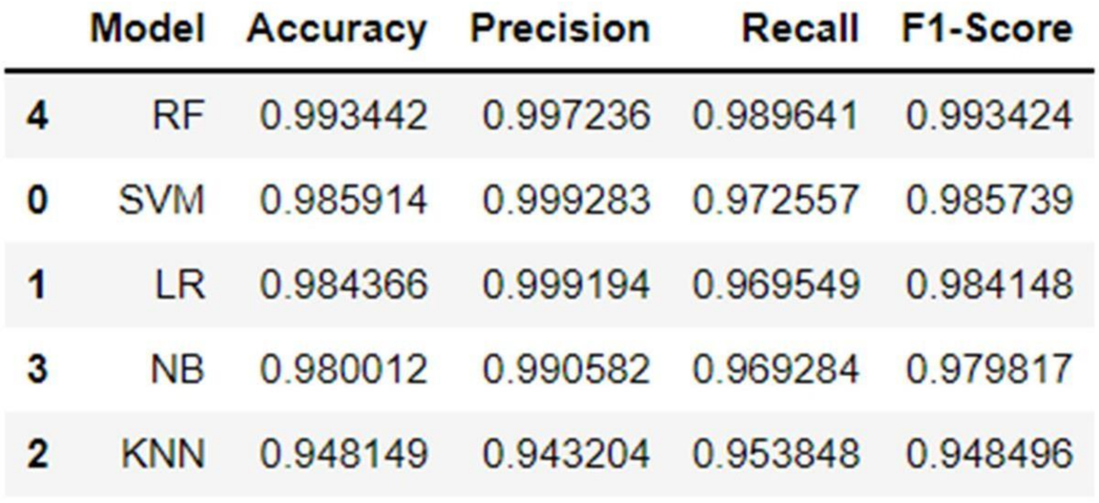
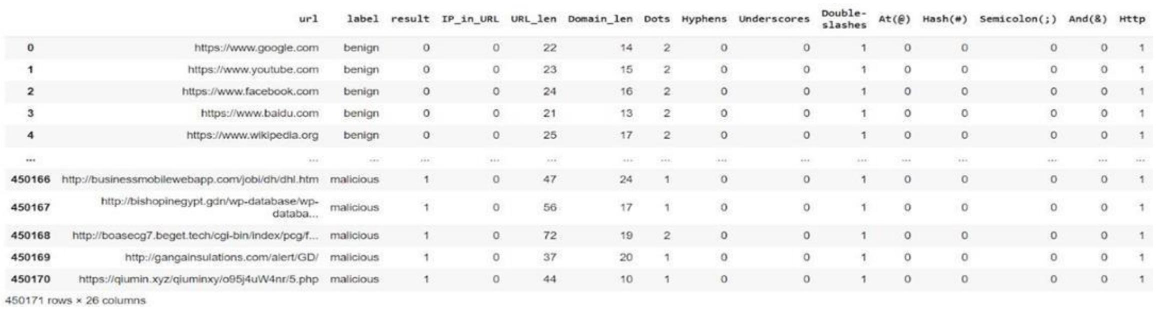


Fig. 2.8. Graph showing f1 Score versus models



	Model	Accuracy	Precision	Recall	F1-Score
4	RF	0.993442	0.997236	0.989641	0.993424
0	SVM	0.985914	0.999283	0.972557	0.985739
1	LR	0.984366	0.999194	0.969549	0.984148
3	NB	0.980012	0.990582	0.969284	0.979817
2	KNN	0.948149	0.943204	0.953848	0.948496

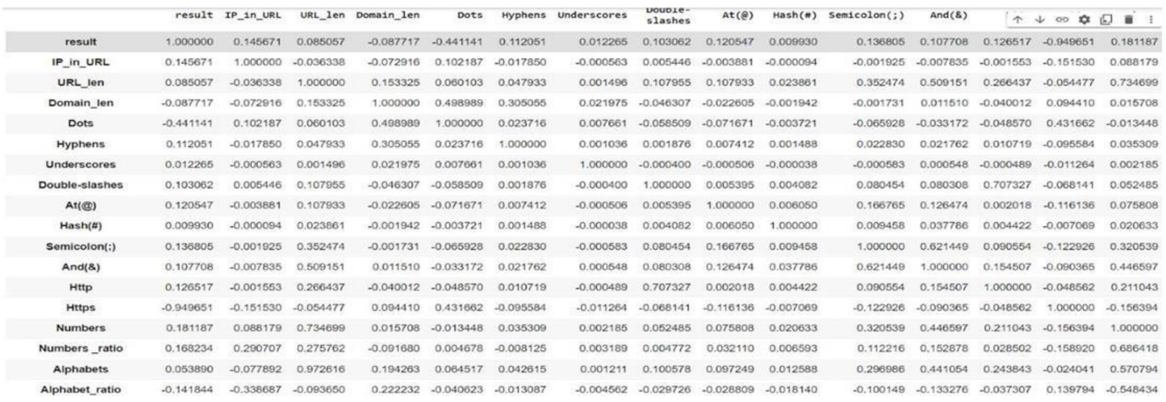
Fig. 2.9. Table showing performance metrics of various models



	url	label	result	IP_in_URL	URL_len	Domain_len	Dots	Hyphens	Underscores	Double-slashes	At(@)	Hash(#)	Semicolon(;)	And(&)	Http
0	https://www.google.com	benign	0	0	22	14	2	0	0	1	0	0	0	0	1
1	https://www.youtube.com	benign	0	0	23	15	2	0	0	1	0	0	0	0	1
2	https://www.facebook.com	benign	0	0	24	16	2	0	0	1	0	0	0	0	1
3	https://www.baidu.com	benign	0	0	21	13	2	0	0	1	0	0	0	0	1
4	https://www.wikipedia.org	benign	0	0	25	17	2	0	0	1	0	0	0	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
450166	http://businessmobilewebapp.com/jobi/dh/dhl.htm	malicious	1	0	47	24	1	0	0	1	0	0	0	0	1
450167	http://bishopinegypt.gdn/wp-database/wp-databa...	malicious	1	0	56	17	1	0	0	1	0	0	0	0	1
450168	http://boasecg7.beget.tech/cgi-bin/index/pcg/f...	malicious	1	0	72	19	2	0	0	1	0	0	0	0	1
450169	http://gangainsulations.com/alert/GD/	malicious	1	0	37	20	1	0	0	1	0	0	0	0	1
450170	https://qiumin.xyz/qiuminxy/o95j4uW4nr/5.php	malicious	1	0	44	10	1	0	0	1	0	0	0	0	1

450171 rows × 26 columns

Fig. 2.10. Dataset



	result	IP_in_URL	URL_len	Domain_len	Dots	Hyphens	Underscores	Double-slashes	At(@)	Hash(#)	Semicolon(;)	And(&)			
result	1.000000	0.145671	0.085057	-0.087717	-0.441141	0.112051	0.012265	0.103062	0.120547	0.009930	0.136805	0.107708	0.126517	-0.949651	0.181187
IP_in_URL	0.145671	1.000000	-0.036338	-0.072916	0.102187	-0.017850	-0.000563	0.005446	-0.003881	-0.000094	-0.001925	-0.007835	-0.001553	-0.151530	0.088179
URL_len	0.085057	-0.036338	1.000000	0.153325	0.060103	0.047933	0.001496	0.107955	0.107933	0.023861	0.352474	0.509151	0.266437	-0.054477	0.734699
Domain_len	-0.087717	-0.072916	0.153325	1.000000	0.498989	0.305055	0.021975	-0.046307	-0.022605	-0.001942	-0.001731	0.011510	-0.040012	0.094410	0.015708
Dots	-0.441141	0.102187	0.060103	0.498989	1.000000	0.023716	0.007661	-0.058509	-0.071671	-0.003721	-0.065928	-0.033172	-0.048570	0.431662	-0.013448
Hyphens	0.112051	-0.017850	0.047933	0.305055	0.023716	1.000000	0.001036	0.001876	0.007412	0.001488	0.022830	0.021762	0.010719	-0.095584	0.035309
Underscores	0.012265	-0.000563	0.001496	0.021975	0.007661	0.001036	1.000000	-0.000400	-0.000506	-0.000038	-0.000583	0.000548	-0.000489	-0.011264	0.002185
Double-slashes	0.103062	0.005446	0.107955	-0.046307	-0.058509	0.001876	-0.000400	1.000000	0.005395	0.004082	0.080454	0.080308	0.707327	-0.068141	0.052485
At(@)	0.120547	-0.003881	0.107933	-0.022605	-0.071671	0.007412	-0.000506	0.005395	1.000000	0.006050	0.166765	0.126474	0.002018	-0.116136	0.075808
Hash(#)	0.009930	-0.000094	0.023861	-0.001942	-0.003721	0.001488	-0.000038	0.004082	0.006050	1.000000	0.009458	0.037786	0.004422	-0.007069	0.020633
Semicolon(;)	0.136805	-0.001925	0.352474	-0.001731	-0.065928	0.022830	-0.000583	0.080454	0.166765	0.009458	1.000000	0.621449	0.090554	-0.122926	0.320539
And(&)	0.107708	-0.007835	0.509151	0.011510	-0.033172	0.021762	0.000548	0.080308	0.126474	0.037786	0.621449	1.000000	0.154507	-0.090365	0.446597
Http	0.126517	-0.001553	0.266437	-0.040012	-0.048570	0.010719	-0.000489	0.707327	0.002018	0.004422	0.090554	0.154507	1.000000	-0.048562	0.211043
Https	-0.949651	-0.151530	-0.054477	0.094410	0.431662	-0.095584	-0.011264	-0.068141	-0.116136	-0.007069	-0.122926	-0.090365	-0.048562	1.000000	-0.156394
Numbers	0.181187	0.088179	0.734699	0.015708	-0.013448	0.035309	0.002185	0.052485	0.075808	0.020633	0.320539	0.446597	0.211043	-0.156394	1.000000
Numbers _ratio	0.168234	0.290707	0.275762	-0.091680	0.004678	-0.008125	0.003189	0.004772	0.032110	0.006593	0.112216	0.152878	0.028502	-0.158920	0.686418
Alphabets	0.053890	-0.077892	0.972616	0.194263	0.064517	0.042615	0.001211	0.100578	0.097249	0.012588	0.296986	0.441054	0.243843	-0.024041	0.570794
Alphabet_ratio	-0.141844	-0.338687	-0.093650	0.222232	-0.040623	-0.013087	-0.004562	-0.029726	-0.028809	-0.018140	-0.100149	-0.133276	-0.037307	0.139794	-0.548434

Fig. 2.11. Correlation Matrix

CHAPTER 5

CONCLUSION AND FUTURE PLANS

With fewer information derived simply from the URL, this current project presents a new way for discovering fraudulent URLs. Count, Length, Ratio, and Presence are the varieties of features extracted. In this research, MachineLearning models are used to determine whether a URL is harmful or benign. This approach use less storage and takes minimal time to execute when compared to previous approaches.It also identifies problems with the current body of work and recent research in the field.For identifying dangerous (url)Uniform Resource Locators, we have executed many standard ML algorithms like NB,LR,RF,KNN,SVC.We have evaluated each algorithm's Precision,Accuracy,F1-score,Recall to decide which algorithm is the most fruitful at determining if a URL is malicious or benign.According to the results, the Random Forest classifier performed better than the remaining classifiers.SVC requires more time to execute than variety of features.

We can utilize this completed model to decide whether the given URL is good(benign) or bad(malicious). So, we came up with a (GUI) for Users to utilize the Flask. At the moment it is working for everyone and we want to add the security to the model we created. This study will be expanded further with more recent feature sets and reduction methods.

CHAPTER 6
REFERENCES

[1] Doyen Sahoo, Chencho Liu, Steven Hoi, Malicious URL detection using machine learning: a survey, arXiv, 1(1) (2019).

[2] HarshalTupsamudre, Ajeet Kumar Singh, Sachin Lodha, Everything is in the name – a URL based approach for phishing detection, in: International Symposium on Cyber Security Cryptography and Machine Learning.Lecture Notes in Computer Science, vol. 11527, Springer, Cham, 2019, pp. 231–248.

[3] D.P.J. Patil, Feature-based malicious URL and attack type detection using multiclass classification, ISC Int. J. Inform. Security 10 (2) (2018) 141–162.

[4] B. Djaballah, B. Ghalem, A new approach for detection and analysis of phishing in social network: a case of twitter, in: Seventh International Conference on Social Networks Analysis, Management and Security (SNAM), 2020, pp. 1–8,

[5] Ammara Zamir, Hikmat Ullah Khan, Tassawar Iqbal, Nazish Yousaf, Farah Aslam, Almas Anjum, Maryam Hamdani, Phishing web site detection using diverse machine learning algorithms.

[6] Hafiz Mohammd Junaid Khan, Quamar Niyaz, Vijay K. Devabhaktuni, Site Guo, Umair Shaikh, Identifying generic features for malicious URL detection system, in: IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON), 2019, pp. 0347–0352.

[7] Katari Patgiri, Sharma Kumar, Empirical study on malicious URL detection using machine learning, in: International Conference on Distributed Computing and Internet Technology. Lecture Notes in Computer Science, vol.11319, Springer, Cham.

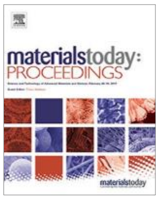
[8] Bc. Andreaturiokova, Detecting malicious URLs, Master thesis, Masaryk University, 2019.

Materials Today: Proceedings xxx (xxxx) xxx

Contents lists available at ScienceDirect

Materials Today: Proceedings

journal homepage: www.elsevier.com/locate/matpr



Lexical features based malicious URL detection using machine learning techniques

Saleem Raja A.[a,⁎], Vinodini R.[b,⁎], Kavitha A.[c,⁎]

[a] Information Technology Department, University of Technology and Applied Sciences-Shinas, Sultanate of Oman, Oman
[b] Department of Computer Science, Mother Theresa Women's University, Kodaikanal, Tamil Nadu, India
[c] Department of Electronics and Communication Engineering, M.Kumarasamy College of Engineeringy, Karur, Tamil Nadu, India

ARTICLE INFO

Article history:
Received 19 March 2021
Accepted 4 April 2021
Available online xxxx

Keywords:
Malicious URL detection
Machine learning
Feature extraction
Feature reduction

ABSTRACT

Most sophisticated cyber-attack technique used by the cyber criminals is creating and spreading malicious domain names or malicious URLs through email, messages, popups etc. Malicious URL are the web pages targeted towards the internet user to spread the malware, virus, warms etc once the user visited. Main intension of the attack is to steal the victim information, user credentials or install the malware in the victim's system. So, it is necessary to adapt the system which should detect the malicious URLs and prevent from the attack. Researchers suggest numerous methods but machine learning based detection method performs better then methods. This paper presents the light weighted method which includes only lexical features of the URL. The result shows the Random Forest classifier performs better than the other classifiers in terms of accuracy.

1. Introduction

Increasing online web services makes the people life simpler than ever before. People are depending on online web services for their daily activities. This creates space for the cyber criminals to attack mass victims through specially designed, short living, malicious URL or web page. These malicious URLs are propagated through email, messages, twitter, facebook, popups, website ads etc. These URLs or websites contains some short of malicious files such as virus (ransomware) or malwares or keylogger etc. Once the victims click the malicious link, the malicious files are downloaded to the victim's system and opens the doors for the attack and data theft. Recent security reports states that more than 40,000 malicious URL are created every day, produces losses of $17,700 per minutes. More that 80% of the system around the world were compromised in 2020 [1,2]. Most of the (60%) of malicious URLs are spread through mail. So it becomes serious threat to the internet users. Rigid mitigation method is required to address the issue.

Common techniques used formalicious URL detection are blocklist based detection, heuristic rules-based detection and machine learning / deep learning-based detection [3]. Blacklist based technique uses huge list of URL which are already identified as malicious by programmatically or manually. Even though blacklist method gives low false positive and faster detection, poses some issues such as the database of blacklisted URL requires frequent and timely updates otherwise it will fail to detect the newly generated malicious URLs. Most of the cyber criminals never uses the existing malicious URLs for further attacks. To overcome these issues, researchers use heuristic rule-based detection technique which generalized rules are used for malicious URL detection. The rules the generated based on the features extracted from the datasets. In fact, feature extraction is the based for machine learning / deep learning technique. Problems in this technique are assigning weightage to the rules and fixing threshold value for each rule [4]. Threshold value may change based on datasets. On the other hand, machine learning technique extract the features from the dataset and train the model. The trained model is used for testing. Machine learning algorithms automatically assign the weightage of selected features and identify the malicious URL including newly generated malicious URLs.

Rest of the paper is organized as follows. Section 2 presents the existing research works in the research domain. Proposed system is presented in section 3. Section 4 presents the experiment results. Section 5 presents the conclusion and future work.

⁎ Corresponding authors.
E-mail addresses: asaleemrajasec@gmail.com (A. Saleem Raja), avinodinimca@gmail.com (R. Vinodini), kavivenkat99@gmail.com (A. Kavitha).

https://doi.org/10.1016/j.matpr.2021.04.041
2214-7853/© 2021 Elsevier Ltd. All rights reserved.
Selection and peer-review under responsibility of the scientific committee of the 12th National Conference on Recent Advancements in Biomedical Engineering.

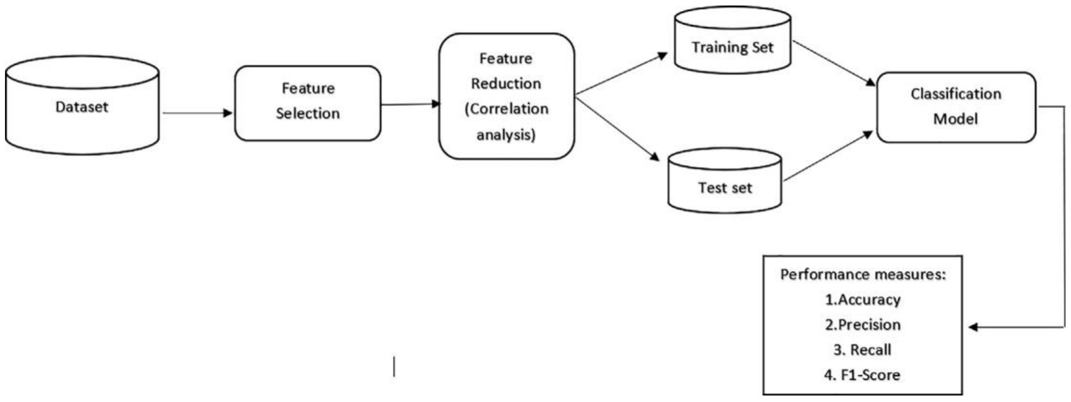


Fig. 1. Phases of proposed method.

Table 1
UNB dataset.

Category of URL	No. of URLs
Benign	35,378
Phishing	9965
Malware	11,566
Spam	11,942

Table 3
Confusion matrix.

	Classified Malicious	Classified Benign
Malicious URLs	True Positive (TP)	False Negative (FN)
benign URLs	False Positive (FP)	True Negative (TN)

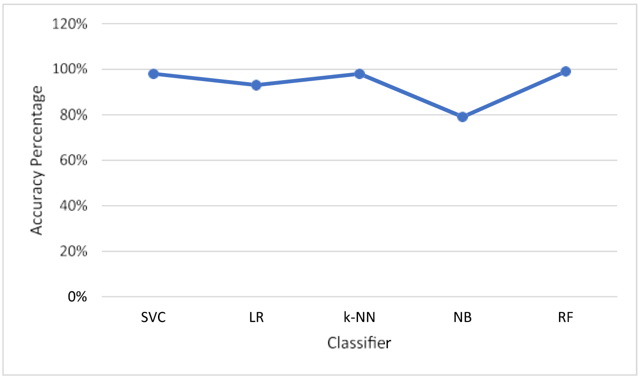


Fig. 2. Accuracy of different classifiers.

2. Related work

Table 2
Lexical features.

Features	Features Category
IP_in_URL	Presence
URL_len	Length
Domain_len	Length
Dots_in_Domain	Count
Hyphens_in_Domain	Count
Underscores _ in_Domain	Count
Double-slashes _in_URL	Count
At(@)_in_URL	Count
Hash(#)_in_URL	Count
Semicolon(;)_in_URL	Count
And(&)_in_URL	Count
Http_in_URL	Count
Https_in_URL	Count
Numbers__in_URL	Count
Numbers _ratio _in_URL *	Ratio
Alphabets_in_URL	Count
Alphabet_ratio_in_URL *	Ratio
Lower_case_letters_in_URL	Count
Lower_case_ letters_in_URL *	Ratio
Upper_case_letters_in_URL	Count
Upper_case_ letters_ratio_in_URL *	Ratio
Special_char_in_URL	Count
Special_char_ratio_in_URL*	Ratio
English_words_in_URL*	Count
Random_words_in_URL*	Count
Avg_english_word_len_in_URL*	Length
Avg_random_words_in_URL*	Length

Continuous efforts of researchers, various new methods have been introduced to improve the detection accuracy and faster detection. Doyen et al. [3] presented a survey on malicious URL detection using machine learning. The paper highlights the pros and cons of different techniques in malicious URL detection in literature. Features used in machine learning techniques are grouped into four categories such as lexical features, host-based features, content-based features and other features (popularity features).

Dhamaraj et al. [5] proposed a multiclass classification method to detect the malicious URL. The paper considers 63 lexical features, 34 content related features and 18 hos-based features. Two algorithms were used for experiments such as SVM and multiclass CW learning. Data are collected from PhishTank, alexa and jwSpamSpy which includes 49,935 URLs (26041 benign URL, 8976 Phishing URL, 11,297 malware, 3621 spam URLs. 98.44% average prediction accuracy was achieved in the test.

Djaballah et al. [6] presented an approach to detect malicious URL in social network such as twitter. The paper includes 25 URL features (grouped under four categories 1. Lexical features 2. Webpage features 3. Host-based features and 4. Popularity features) and user account related features of twitter. Datasets includes UCI phishing datasets and 10,000 twitter accounts. Three machine learning algorithms (Logistic Regression, SVM and Random forest) were used for the experiments and it gives 90.28%, 93.43% and 95.51% accuracy respectively.

Ammara et al. [7] presented a method for phish website detection which uses different feature selection techniques and different machine learning techniques with PCA such as RF, NN, bagging, SVN, NB and KNN were used for experiments. The techniques are

Table 4
Result of classifiers.

Classifier	Ratio	Accuracy	Precision	Recall	F1-score	Execution time in sec
SVC	70: 30	0.98	0.99	0.98	0.98	251.66
LR	70: 30	0.93	0.94	0.93	0.94	2.696
k-NN (N = 5)	70: 30	0.98	0.98	0.98	0.98	2.233
NB (Gaussian)	70: 30	0.79	0.84	0.78	0.77	0.612
RF (100 trees)	70: 30	0.99	0.99	0.99	1.00	4.709

grouped as Stacking1 (RF + NN + Bagging) and Stacking2 (kNN + RF + Bagging) to improve the accuracy of classification. Stacking 1 produces highest accuracy of 97.4% than the other techniques.

Hafiz et al. [8] presented a way for selecting optimal features from dataset to detect malicious URL. UNB ang Kaggle datasets were used for feature extraction and testing. Nearly 41 word-based features, 36 count based features and 29 other features were used. KNN, SVM, LR, AdaBoost, Gradient Boost, ExtraTrees, RF and Voting classifiers algorithms were used for experiments. Feature's fitness and dependency with before target was tested before selecting the feature. The result shows 99% accuracy in UNB data-set and 94% accuracy in Kaggle dataset.

Ripon et al. [9] presents the experimental study on mostly used machine learning classifier such as SVM and Random forest. Lexical and host-based features are extracted from dataset and tested with ratio of 60:40, 70:30, and 80:20. And result confirms Random forest out performed than the SVM classifier. SVM accuracy result were more fluctuating than the Random forest.

Most of the existing work focus on the improvement of accuracy by selecting different feature sets. Host based, Content and Popularity based features incurs additional processing time and resources to extract the desired features [10]. Most of the works uses training and test ratio as 70% and 30% or 75% and 25% respectively.

3. Proposed method

The proposed method consists of following phases 1. Feature Extraction 2. Feature reduction 3. Train the Model 4. Testing as shown in the Fig. 1.

Feature extraction is the primary phase for machine learning techniques. UNB dataset 2016 is used for feature extraction and

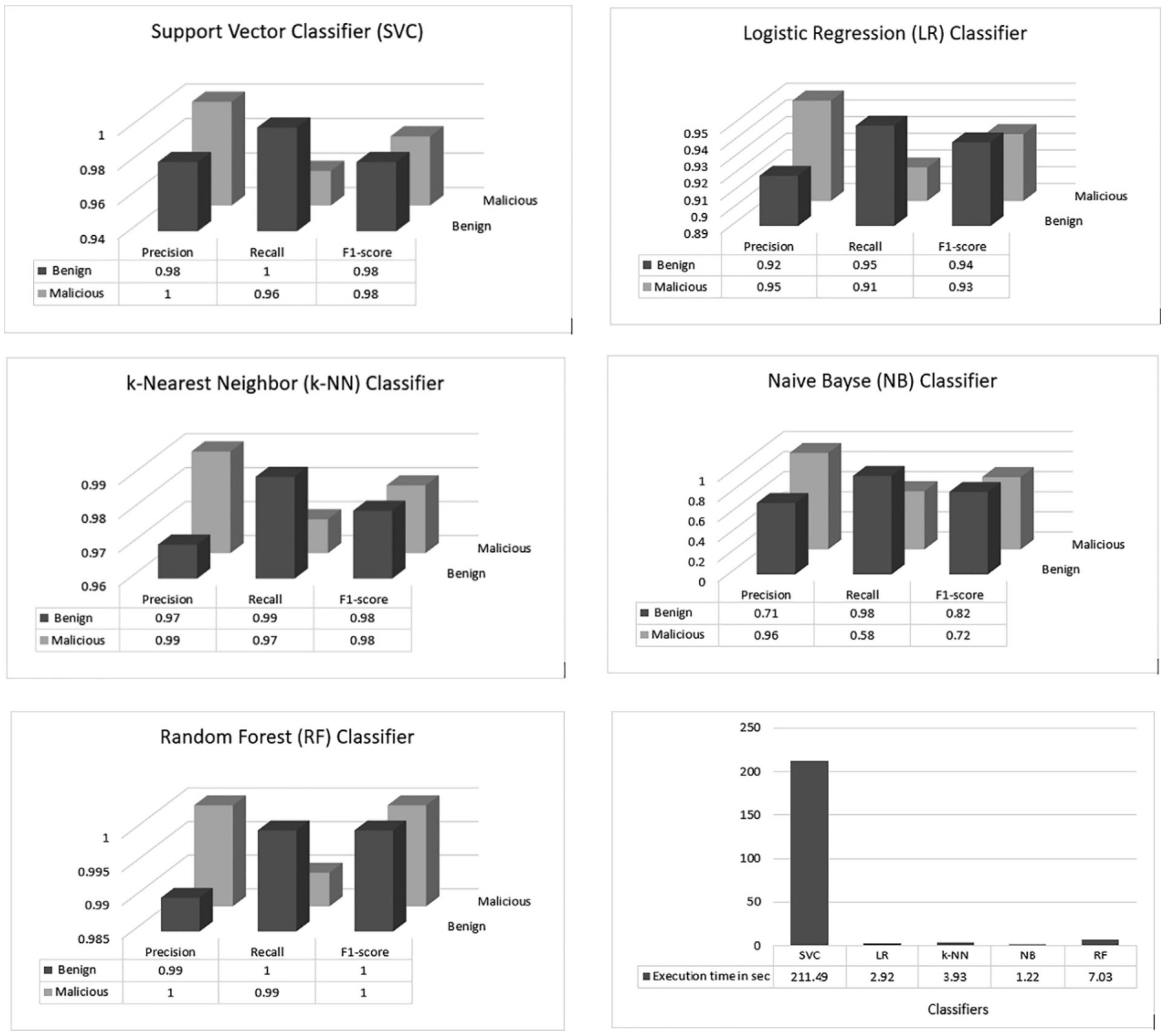


Fig. 3. Result of the performance metrics.

experiments with different machine learning algorithms. The UNB dataset includes benign, phishing, malware, spam url dataset as shown in Table 1.

Minimum set of optimal features which reduces the execution time and storage consumption. Initially identified 18 most common features and 9 newly identified features (*) in the dataset as shown in Table 2.

All extract features are not always suitable for classification. So it requires feature reduction method to identify the fitness of the features. Correlation analysis help to find the relationship between target feature and other features which in turn help to reduce the feature. Features are removed if there is no significant correction is present. Nearly 20 features are selected after correction analysis and 7 features are removed from features set (Underscores _ in_Domain, Double-slashes _in_URL, At(@)_in_URL, Hash(#) _in_URL, Https_in_URL, Numbers__in_URL, Special_char_ratio_in_URL).

4. Experiment results

Testing is conducted in the system using windows 10 operating and with 1.80 GHz intel core i7 processor and 16 GB RAM. Python with scikit learn package is used for programming to test different classifiers accuracy. The accuracy is computed based on the equation (1) and Table 3. Accuracy is defined as the percentage of correct decisions among all testing samples.

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (1)$$

Other performance metrics evaluated with the proposed system are precision, recall and F1-score using formulae 2, 3 and 4 respectively.

$$precision = \frac{TP}{TP + FP} \quad (2)$$

$$Recall = \frac{TP}{TP + FN} \quad (3)$$

$$F1\_Score = \frac{2xprecisionxrecall}{precision + recall} \quad (4)$$

Data of selected 20 features stored as a CSV file and fed in to the classifiers for training and testing with ratio of 70:30. Using scikit-learn python library, the classifiers (Support Vector Classification (SVC), Logistic Regression (LR), K-Nearest Neighbors (k-NN), Naïve Bayes (NB), Random Forest (RF)) are trained and tested. Accuracy of each classifier is tested, and results are presented in Table 4 and Fig. 2.

Performance metrics of different classifiers in presented in Table 4. The result shows the RF is outperformed than the other classifiers. But considering execution time and accuracy then k-NN gives better result than other classifiers as shown in Fig. 3.

5. Conclusion & future work

This paper presents the new method for malicious URL detection with fewer number of features extracted only from URL. That reduces execution time and storage requirements. It also highlights the recent research work in the domain and issues in the existing work. Result shows that random forest classifier is outperformed than the other classifiers. But considering execution time and accuracy then k-NN gives better result than other classifiers. In future, the research work will be extended with newer feature set and reduction techniques.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] https://www.csoonline.com/article/3153707/top-cybersecurity-facts-figures-and-statistics.html
[2] https://cyber-edge.com/wp-content/uploads/2020/03/CyberEdge-2020-CDR-Report-v1.0.pdf
[3] Doyen Sahoo, Chencho Liu, Steven Hoi, Malicious URL detection using machine learning: a survey, arXiv, 1(1) (2019).
[4] HarshalTupsamudre, Ajeet Kumar Singh, Sachin Lodha, Everything is in the name – a URL based approach for phishing detection, in: International Symposium on Cyber Security Cryptography and Machine Learning. Lecture Notes in Computer Science, vol. 11527, Springer, Cham, 2019, pp. 231–248.
[5] D.P.J. Patil, Feature-based malicious URL and attack type detection using multi-class classification, ISC Int. J. Inform. Security 10 (2) (2018) 141–162.
[6] B. Djaballah, B. Ghalem, A new approach for detection and analysis of phishing in social network: a case of twitter, in: Seventh International Conference on Social Networks Analysis, Management and Security (SNAM), 2020, pp. 1–8, https://doi.org/10.1109/SNAMS52053.2020.9336572.
[7] Ammara Zamir, Hikmat Ullah Khan, Tassawar Iqbal, Nazish Yousaf, Farah Aslam, Almas Anjum, Maryam Hamdani, Phishing web site detection using diverse machine learning algorithms.
[8] Hafiz Mohammd Junaid Khan, Quamar Niyaz, Vijay K. Devabhaktuni, Site Guo, Umair Shaikh, Identifying generic features for malicious URL detection system, in: IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON), 2019, pp. 0347–0352. doi: 10.1109/UEMCON47517.2019.8992930.
[9] Katari Patgiri, Sharma Kumar, Empirical study on malicious URL detection using machine learning, in: International Conference on Distributed Computing and Internet Technology. Lecture Notes in Computer Science, vol. 11319, Springer, Cham. https://doi.org/10.1007/978-3-030-05366-6_31.
[10] Bc. Andreaturiokova, Detecting malicious URLs, Master thesis, Masaryk University, 2019.