MOST EFFECTIVE URL VECTORS AND METHODS FOR FINDING MALICIOUS URLS: A RESEARCH

A PROJECT REPORT

Submitted by

SIDDHANT TIWARI [Reg No. RA1811030010066]

SYED ABBAS HAIDER RIZVI [Reg No. RA1811030010082]

Under the guidance of

Dr. K. KALAISELVI

Assistant professor, Department of Networking and Communications

In Partial Fulfillment of the Requirements for the degree of,

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

with specialization in CYBER SECURITY

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
Deemed to be University u/s 3 of UGC Act, 1956

DEPARTMENT OF NETWORKING AND COMMUNICATIONS

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR-603203

MAY 2022

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY KATTANKULATHUR– 603203

BONAFIDE CERTIFICATE

Certified that this B.Tech project report titled “**MOST EFFECTIVE URL VECTORS AND METHODS FOR FINDING MALICIOUS URLS: A RESEARCH”** is the bonafide work of **Mr. SIDDHANT TIWARI [Reg No.: RA1811030010066] and Mr. SYED ABBAS HAIDER RIZVI [Reg No.: RA1811030010082]** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion for this or any other candidate.

SIGNATURE	**SIGNATURE**
DR. K. KALAISELVI	DR. ANNAPURANI PANAIYAPPAN .K
SUPERVISOR Assistant Professor Networking and Communications	**HEAD OF THE DEPARTMENT** Networking and Communications

Signature of the Internal Examiner	Signature of the External Examiner

Department of Computer Science and Engineering

SRM Institute of Science & Technology

Own Work Declaration Form

Degree/Course :B.Tech in Computer Science and Engineering with Specialization in Cyber Security

Student Name : SIDDHANT TIWARI

Registration Number : RA1811030010066

Student Name : SYED ABBAS HAIDER RIZVI

Registration Number : RA1811030010082

Title of Work : MOST EFFECTIVE URL VECTORS AND METHODS FOR FINDING MALICIOUS URLS: A RESEARCH

We hereby certify that this assessment compiles with the University's Rules and Regulations relating to Academic misconduct and plagiarism, as listed in the University Website, Regulations, and the Education Committee guidelines.

We confirm that all the work contained in this assessment is our own except where indicated, and that we have met the following conditions:

- Clearly references/listed all sources as appropriate
- Referenced and put in inverted commas all quoted text (from books, web, etc.)
- Given the sources of all pictures, data etc. that are not my own
- Not made any use of the report(s) or essay(s) of any other student(s) either past or present
- Acknowledged inappropriate places any help that I have received from others (e.g. fellow students, technicians, statisticians, external sources)
- Compiled with any other plagiarism criteria specified in the Course hand book/University website

I understand that any false claim for this work will be penalized in accordance with the University policies and regulations.

DECLARATION:

I am aware of and understand the University's policy on Academic misconduct and plagiarism and I certify that this assessment is my / our own work, except where indicated by referring, and that I have followed the good academic practices noted above.

RA1811030010066 04/05/2022	RA1811030010082 04/05/2022

If you are working in a group, please write your registration numbers and sign with the date for every student in your group.

ACKNOWLEDGEMENT

We express our humble gratitude to **Dr. C. Muthamizhchelvan**, Vice-Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support.

We extend our sincere thanks to Dean-CET, SRM Institute of Science and Technology, **Dr. T.V.Gopal**, for his invaluable support.

We wish to thank **Dr. Revathi Venkataraman**, Professor & Chairperson, School of Computing, SRM Institute of Science and Technology, for her support throughout the project work.

We are incredibly grateful to our Head of the Department**, Dr. Annapurani Panaiyappan K.,** Professor, Department of Networking and Communications, SRM Institute of Science and Technology, for her suggestions and encouragement at all the stages of the project work.

We want to convey our thanks to our Panel Head**, Dr. K. Kalaiselvi,** Assistant Professor, Department of Networking and Communications and program coordinators **Dr. M.B. Mukesh Krishnan,** Associate Professor, Department of Networking and Communications, SRM Institute of Science and Technology, for their inputs during the project reviews and support.

We register our immeasurable thanks to our Faculty Advisor**, Dr. M. Uma,** Associate Professor, Department of Computational Intelligence, SRM Institute of Science and Technology, for leading and helping us to complete our course.

Our inexpressible respect and thanks to my guide, **Dr. K. Kalaiselvi**, Assistant Professor, Department of Networking and Communications, SRM Institute of Science and Technology, for providing me with an opportunity to pursue my project under her mentorship. She provided me with the freedom and support to explore the research topics of my interest. Her passion for solving problems and making a difference in the world has always been inspiring.

We sincerely thank the Networking and Communications Department staff and students, SRM Institute of Science and Technology, for their help during our project. Finally, we would like to thank parents, family members, and friends for their unconditional love, constant support, and encouragement.

SYED ABBAS HAIDER RIZVI

SIDDHANT TIWARI

ABSTRACT

This paper consists of our research on machine learning models that would help us detect malicious URLs. There are a variety of models available, but we have taken CNNs and Basic ML models along with URL vectors and features, because using RNNs or CNN LSTMs is not feasible for 1D data. The main highlights of our thesis have been that the accuracy measures of the two mains algorithms have been really close but there are discrepancies in the confusion matrix itself. Although these differences arise because of the time bindings, we propose a lightweight voting system for the most accurate system which works the best. Our research has also led us to find the most important URL vector which we came across while testing different databases.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

Sr. No.	Symbol	Pg. No.
1.	CNN - Convoluted Neural Networks	10
2.	RNN - Recurrent Neural Networks	10
3.	ANN - Artificial Neural Networks	10
4.	NLP - Natural Language Processing	12
5.	LDA – Linear Discriminant Analysis	13
6.	PCA – Principal Component Analysis	13
7.	URL - Uniform Resource Locator	13

CHAPTER 1

INTRODUCTION

Malicious URLs have been deemed the major factor in online security threats. Attackers can gain backdoor access to sensitive data by just sending one well disguised URL to an innocent person. Malicious URLs are the weapon of choice in Cyber Attacks [1].A survey shows that 75% of ransomware infected companies were using up to date protection systems.[2]

Blacklisting methods are fast but they are again outdated in the era of ever evolving internet.In our previous research project we analyzed various models to detect Malicious URLs and compared their accuracies. Extending the project and its implementation we faced time boundings in the traditional CNN model but we did gain a stellar accuracy with this character embedded CNN model.

Moving on we thought of finding of the URL features that are important or rather more relevant to the malicious or benign prediction, following through on this we proposed creating a voting algorithm that utilized all the other algorithms that we used before and also performed the same procedures by not including some URL features to find if they are more relevant that the others or not.

Also, by finding which features are more important than the others users can manually see if the URL is malicious or not and have a pretty good idea of the analysis.

1.1 MOTIVATION

Our motivations were derived from the following needs:-

- 75% of the world's organizations witnessed phishing attacks in 2020 only.
- 96% of phishing attacks arrive by emails containing a malicious URL.

These attacks further cause a lot of damage to society. Some attacks may install spyware, ransomware etc. India saw a huge incline in ransomware-based attacks in 2020 and was the 6th most affected country in the world.

Trend Micro Incorporated announced that it blocked 40.9 billion email threats, malicious files and malicious URLs for its customers worldwide in the first half of 2021.

The present detection technique of blacklisting in a database fails at detecting new variations of malicious URLs. The shortened versions of the URLs (bit.ly) are not checked because they do not reveal the destination.

We plan to develop a voting model that can overcome these problems and effectively start detecting new and varied versions of URL attacks. We will extract in-depth lexical features from URL strings namely – lexical features, content-based and host-based features.

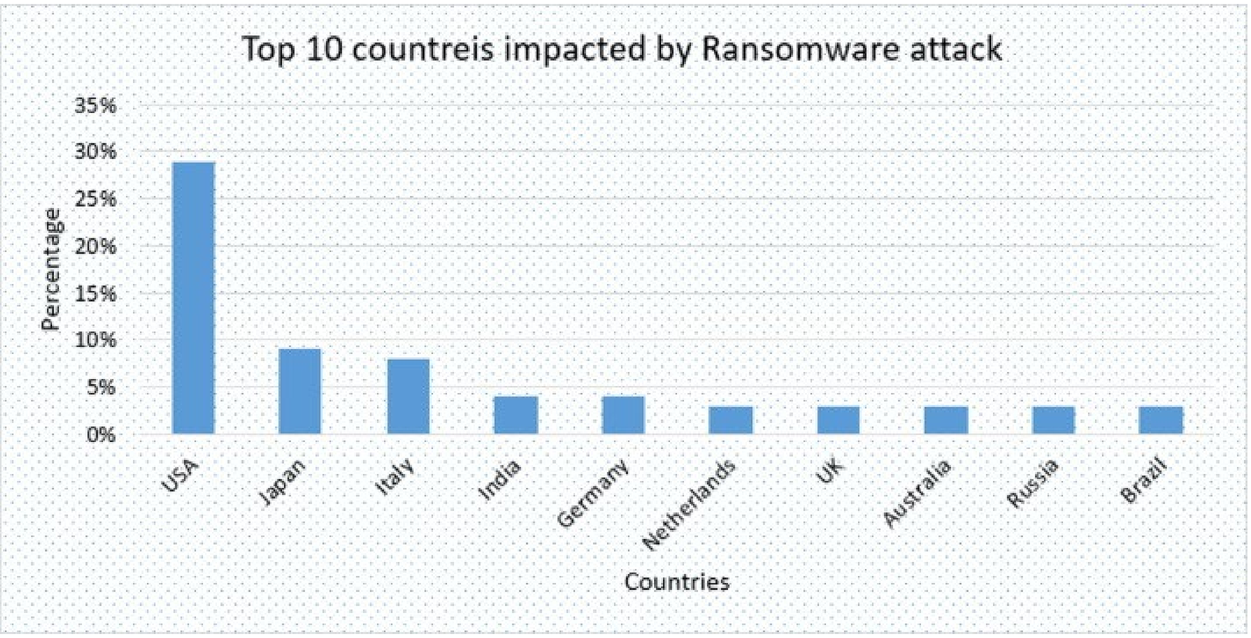


Figure 1 Top 10 countries affected by Ransomware Attacks

1.2: OBJECTIVES

The objectives of this project are –

1) Case study on ML models and other efficient methods to resolve the problem of URL phishing.
2) Testing the ML models with variety of datasets.
3) Comparing URL vectors and features against each other and discovering the important ones.

1.3: SCOPE OF PROJECT

- The project can detect and inform the user for the malicious URL but cannot restrict navigation to the website. Betterment of the existing systems could give a faster analysis.
- Also, the project focuses on the various ML algorithms available and suggests methods for both faster and accurate detections.

1.4: EXISTING SYSTEMS

The technological advancements in the 21st century have led to a great need for online safety. And in the post covid era, where almost everyone has most of their life shifted to the virtual ways, whether its online transactions or even their work, a single URL could greatly jeopardize a big company's resources.

The classification of URLs has utilized a variety of algorithms since the URL phishing was discovered.

Here we describe other methods that have been utilized in the past which we have not used in our project.

- Blacklisting is a traditional but outdated method which employs pattern matching techniques and is used by web browsers/plugins.(eg. McAfee Site Advisor freemium add-on).[3] Other methods viz honeypots, web crawling etc. utilize analytics to scan sites but again both of these fail when a URL outside their database is faced.

- URL features like lexical, headers and other data were used by Liang Bin [4].

- Garera et al.[5] also employed the same lexical features to counter phishing like hostname length, hidden host domains, page ranks and IP addresses, domain tables.

- Complex machine learning techniques like SVM, Naive Bayes etc. have been used in tandem with word embedding by Crisan et al. [6] where data processing is simpler and the traditional process of feature selection is innovatively used.

- Singhal et al.[7] used ML along with the drift detection concept to compare data between feature vectors of the training dataset and another recently gathered one which would help curb the bypassing of the system.

- Even deep learning models like CNN, CNN LSTM, RNN, and simple LSTMs have been drafted for the same purpose. Das et al.[8] gives a comparative analysis of these. On a careful analysis of his work its evident that all these models understand features differently and so fusing models is a good approach.

CHAPTER 2

LITERATURE SURVEY

Some papers contain a deep analysis of all the techniques that can be applied to URL classifications. They craftily develop the idea of URLs being exploited and then go on to list the different methods of classification along with a short summary and utilization analysis of the same. The data visualization is then done along with basic comparisons. Feature extraction techniques like LDA and PCA are used to extract features and then along with NLP the model is trained. Various machine learning models are then compared.

An advantage is that the various ML models are utilized which helps us get an idea of accuracies for the different models.

A disadvantage is that the common techniques of feature extraction are used like LDA and PCA where we hope specific URL features may give us better accuracy. In one of the papers, we use machine learning models along with the URL vectors. Also, we get a good insight of what Machine learning techniques can be utilized and then goes on to list the various URL features that can be attributed. But then the problem comes when the focus is shifted to a single model and various models aren't tested. The deep learning model is trained on URL features and then is used for classification. An advantage is that URL features like host-based features, lexical features are used which help in a better method of feature extraction. he URL features were found to be a better fit than the traditional blacklisting and other methods. There are a variety of models available, but we chose CNNs and Basic ML models, as well as URL vectors and features, because utilising RNNs or CNN LSTMs for 1D data is not practical. The accuracy measures of the two primary algorithms were extremely similar, however there were discrepancies in the confusion matrix itself, according to our thesis. We have analyzed different research papers and we have listed out the methodologies, positives and drawbacks of each.

Sr. No.	Title of The Project	Methodology	Author Name	Advantages	Drawbacks
1	Malicious URL Detection: A Comparative Study	Feature extraction techniques like LDA and PCA are used to extract features and then along with NLP the model is trained. Various machine learning models are then compared.	Shantanu Maheshwari, Janet B, Joshua Arul Kumar R	The various ML models are utilized which helps us get an idea of accuracies for the different models.	The common techniques of feature extraction are used like LDA and PCA where we hope specific URL features may give us better accuracy.
2	Malicious URL Detection based on Machine Learning	The deep learning model is trained on URL features and then is used for classification.	Cho Do Xuan, Hoa Dinh Nguyen, Tisenko Victor Nikolaevich	URL features like host-based features, lexical features are used which help in a better method of feature extraction.	Only a single model of machine learning is employed.
3	Malicious URL Detection using Deep Learning	Deep Learning with Character Level Embedding is used for malicious URL detection.	Vinayakumar R, Sriram S, Soman KP, and Mamoun Alazab, Senior	Deep Learning and Neural Networks are used.	URL features like host-based, lexical features and suspicious keywords are not

			Fellow, IEEE		involved.
4	Malicious URL Detection Based on Associative Classification	Works on rule generators and classifier builders.	Sandra Kumi, Chase-Ho lim, Sang-Gon Lee	Newer models involving rule generators are used. CBA-RG and CBA-CB are put to use.	These algorithms are a little complex plus accuracies are not up to the mark

Table 1 Literature Survey

CHAPTER 3

UNIFIED MODELLING LANGUAGE DIAGRAMS

3.1 USE CASE DIAGRAM

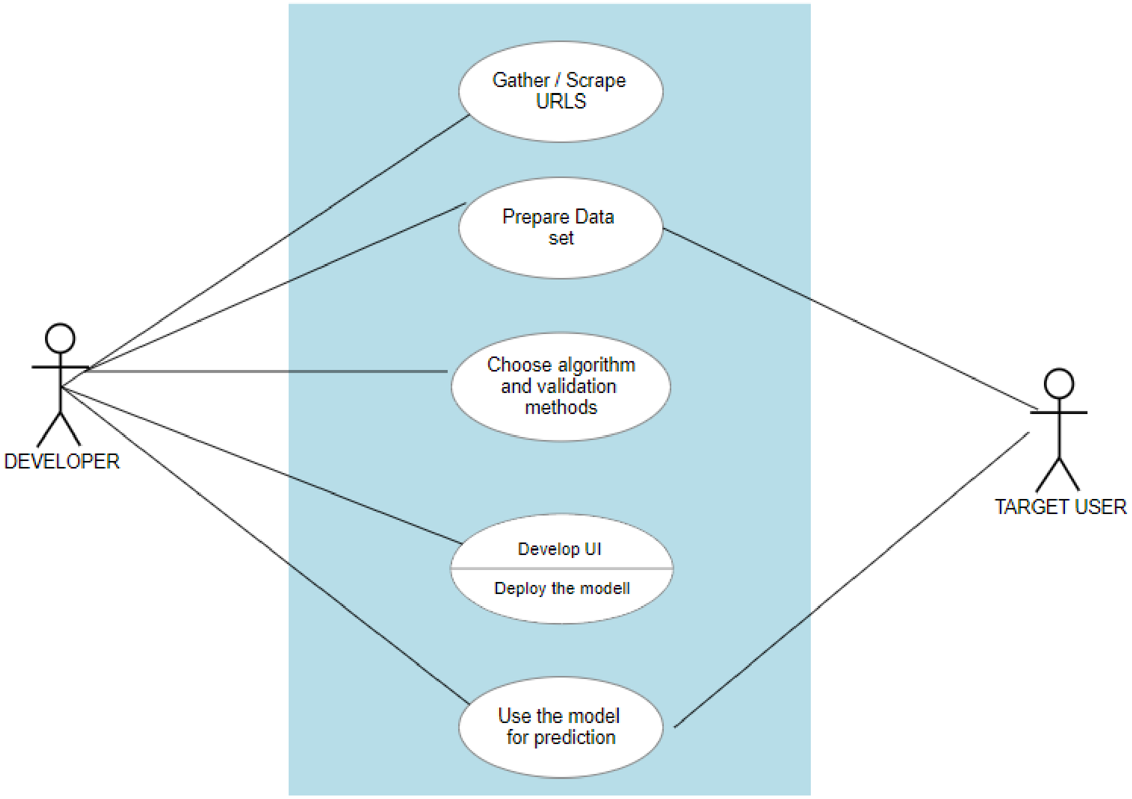


Figure 2 Use Case Diagram

3.2 ARCHITECTURE DIAGRAM

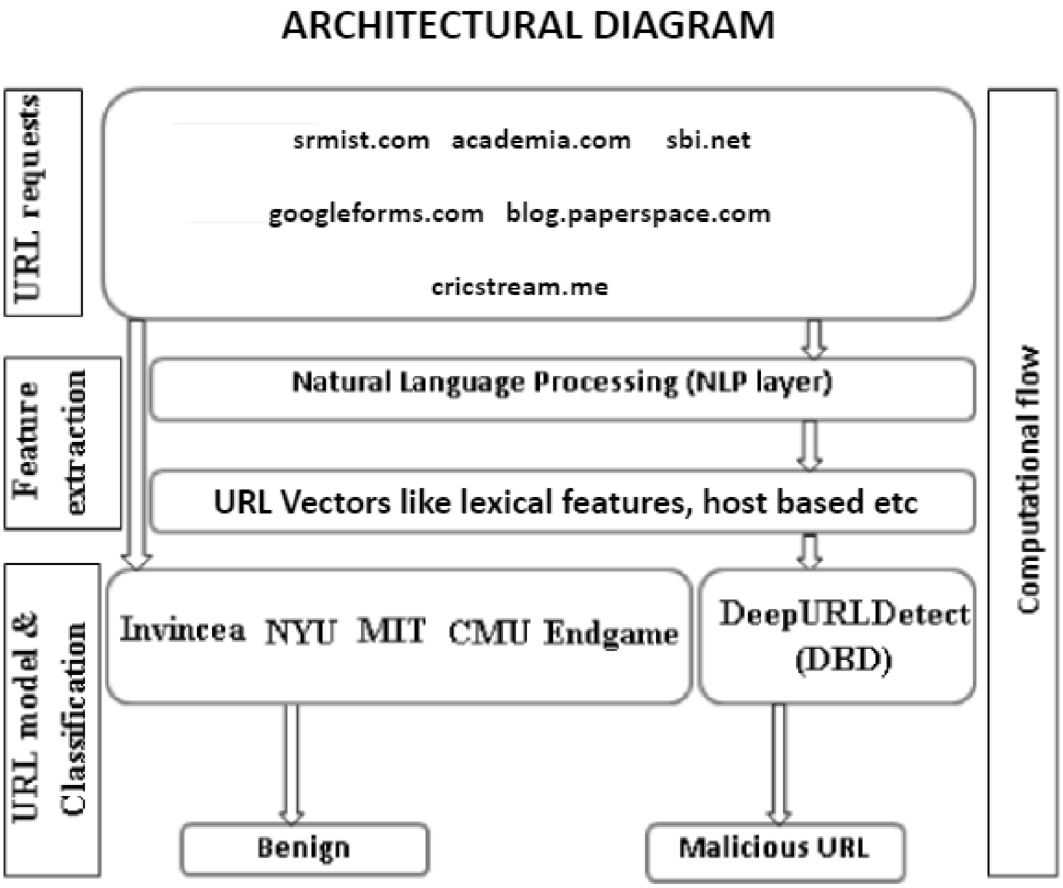

ARCHITECTURAL DIAGRAM
URL requests
srmist.com academia.com sbi.net
googleforms.com blog.paperspace.com
cricstream.me
Feature extraction
Natural Language Processing (NLP layer)
URL Vectors like lexical features, host based etc
URL model & Classification
Invincea NYU MIT CMU Endgame
DeepURLDetect (DBD)
Benign
Malicious URL
Computational flow

Figure 3 Architecture Diagram

3.3 ARCHITECTURE DIAGRAM FOR PROPOSED VOTING SYSTEM

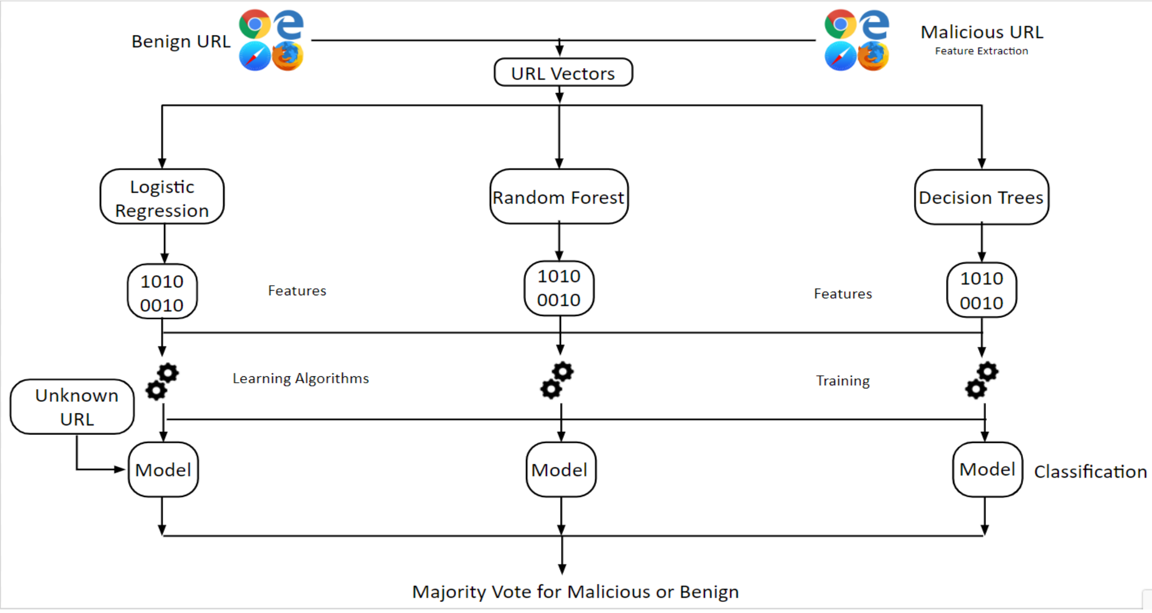


Figure 4 Architectural Diagram for Proposed Voting System

3.4 ALGORITHM WORKFLOW

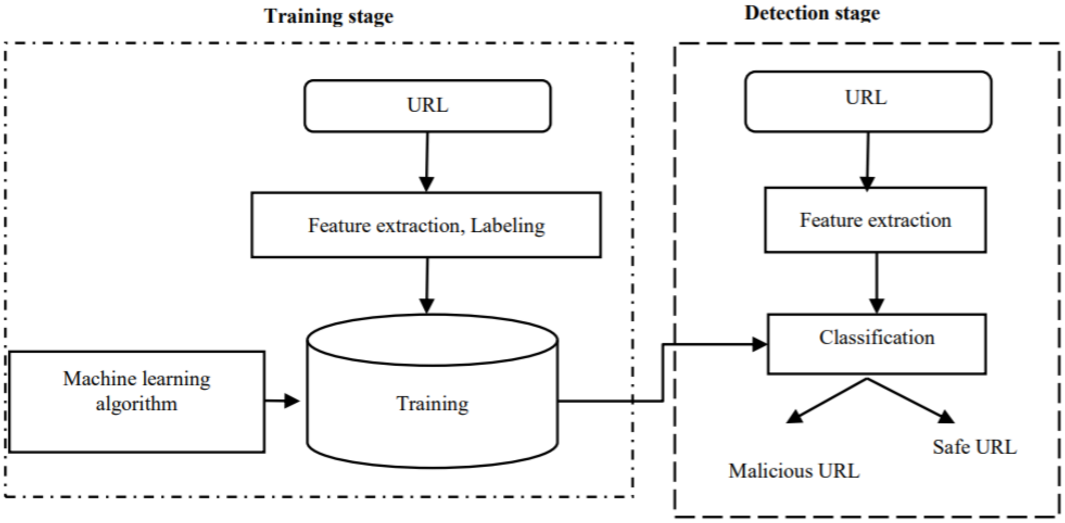


Figure 5 Algorithm Workflow

CHAPTER 4

IMPLEMENTATION

4.1 MODULES IN THE PROJECT

1. **Database Gathering & Collection** - A pool of URLs with specific tags from Kaggle and other web scraped URLs that will be used to train our model.
2. **Data Engineering** - It involves analyzing, preprocessing, extraction of features and data cleaning, normalization and encoding
3. **Model Training & Testing** - A variety of algorithms are used to train our model and it is tested on the validation data to obtain various parameters.
4. **Model Comparison** - The accuracies for different models are obtained and the best one is selected.
5. **Model deployment** - The combination of selected models and blacklisting processes that would work on a double fast and slower layer model is then deployed along with UI on a cloud server which is now ready for the world to test.
6. **Voting System Model** -This was our proposed system which utilizes 3 algorithms and improves upon the accuracy, and is still lightweight.

4.2 ALGORITHMS USED

1) **Logistic Regression :-**

Logistic regression is a statistical analysis method used to predict a data value based on prior observations of a data set.

2) **Random forest Classifier :-**

Random forest is a supervised ML model which basically uses decision trees and then takes as a vote/average for classification/regression.

Here the accuracy is increased due to the combination of multiple classifiers. Also, the training time is lesser in random forests model.

3) **Decision Trees Classifier :-**

Decision Trees are the only supervised learning machine algorithm that can be used both for classification and regression.Their structure matches a lot with the classic Data Structure Trees, which consist of internal and leaf nodes and a root node.

Decision tree learning employs a divide and conquer strategy by conducting a greedy search to identify the optimal split points within a tree. This process of splitting is then repeated in a top-down, recursive manner until all, or the majority of records have been classified under specific class labels.

4) **CNNs :-**

CNNs are deep learning algorithms where the images/data fed to the input layer are assigned importance in terms of biases/weights and then these features are learnt to recognize their importance in the given data.

We have used character level embedding here to use CNNs for text level filtering.

4.3 MODULES DESCRIPTION AND IMPLEMENTATION

The main modules used here are –

- **CNN**
- **URL Features and Extraction**
- **Our proposed Voting System**

4.3.1 CNN MODELS AND HOW WE UTILIZED THEM

CNNs are deep learning algorithms where the images/data fed to the input layer are assigned importance in terms of biases/weights and then these features are learnt to recognize their importance in the given data.

We have used character level embedding here to use CNNs for text level filtering.

In this paper we have utilized a 1D CNN character embedded model with two layers of convolution with 64 filters each having a kernel size of 5 and 3 each with padding specified as same ("zeros added evenly to left/right and up/down") using the elu activation.

As CNNs tend to overfit textual data as their primary purpose is for images, we have used the early stopping feature from keras, where we monitored the values of val_precision parameter over 5 epochs and bring the model to an early stop if no significant improvements are made over that course of processing.

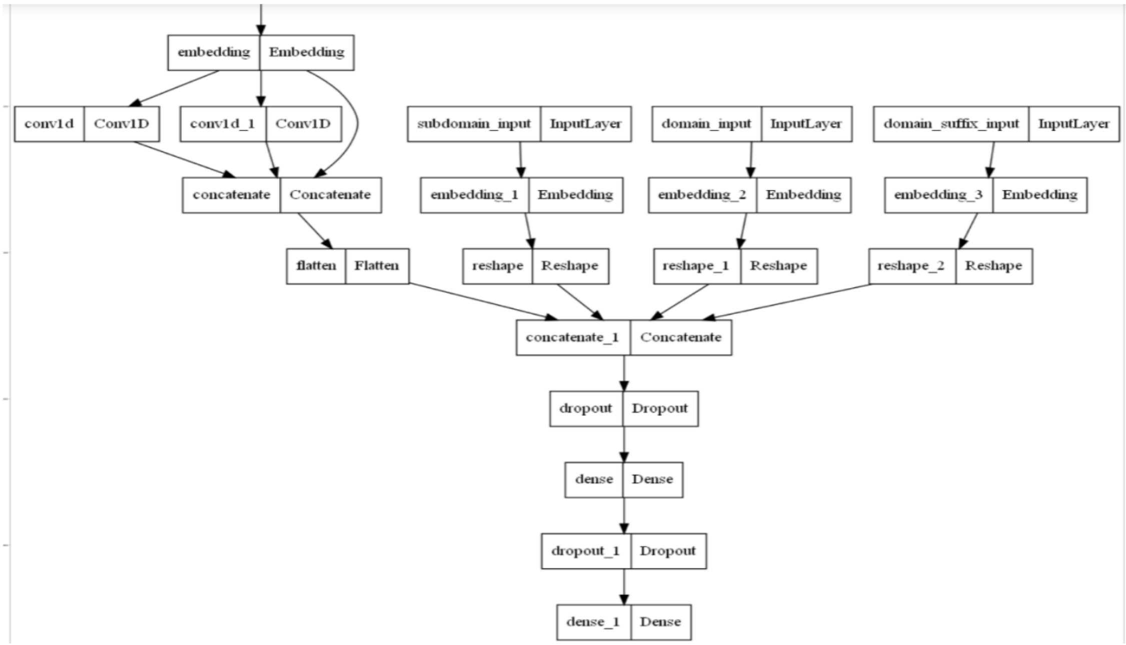
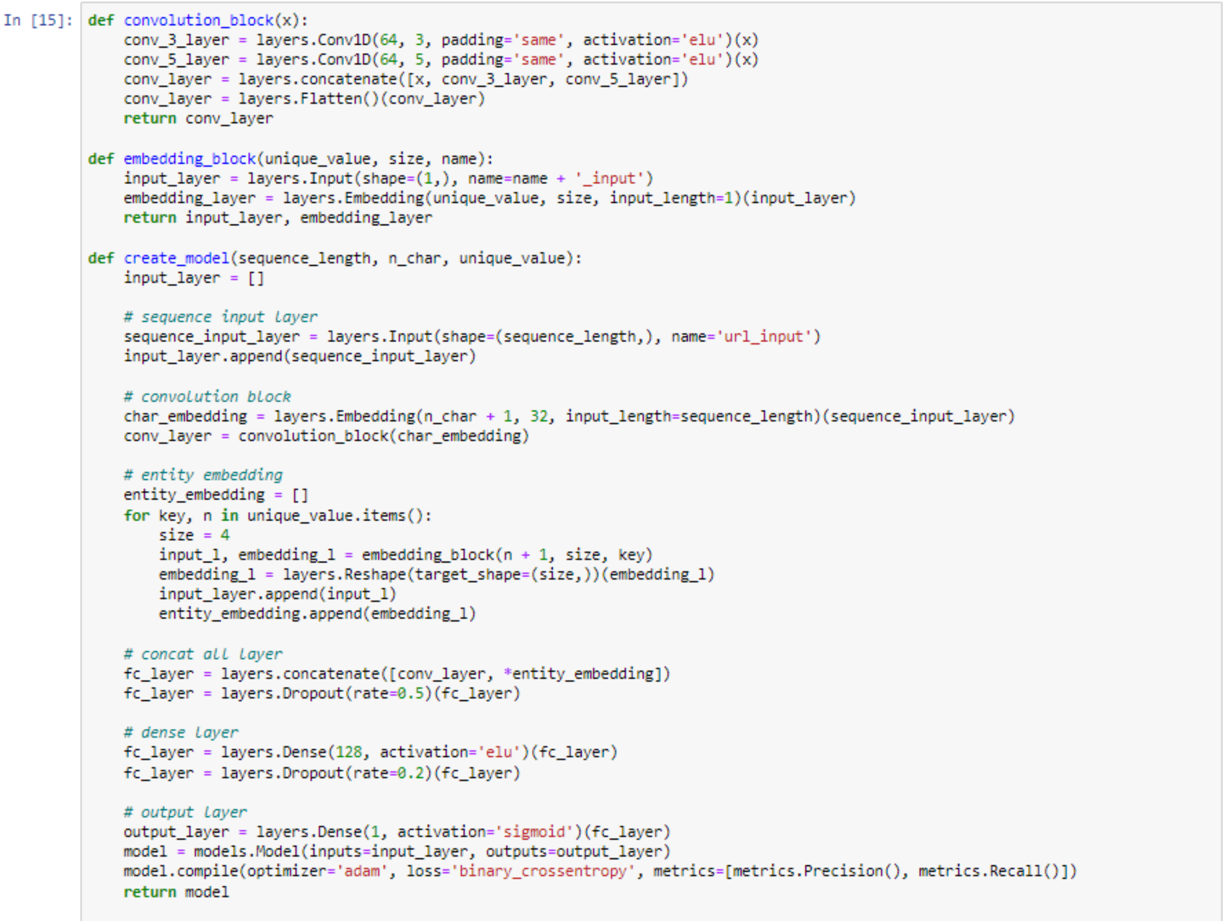


Figure 6 CNN Layers Overview



In [15]:

```python
def convolution_block(x):
    conv_3_layer = layers.Conv1D(64, 3, padding='same', activation='elu')(x)
    conv_5_layer = layers.Conv1D(64, 5, padding='same', activation='elu')(x)
    conv_layer = layers.concatenate([x, conv_3_layer, conv_5_layer])
    conv_layer = layers.Flatten()(conv_layer)
    return conv_layer

def embedding_block(unique_value, size, name):
    input_layer = layers.Input(shape=(1,), name=name + '_input')
    embedding_layer = layers.Embedding(unique_value, size, input_length=1)(input_layer)
    return input_layer, embedding_layer

def create_model(sequence_length, n_char, unique_value):
    input_layer = []

    # sequence input layer
    sequence_input_layer = layers.Input(shape=(sequence_length,), name='url_input')
    input_layer.append(sequence_input_layer)

    # convolution block
    char_embedding = layers.Embedding(n_char + 1, 32, input_length=sequence_length)(sequence_input_layer)
    conv_layer = convolution_block(char_embedding)

    # entity embedding
    entity_embedding = []
    for key, n in unique_value.items():
        size = 4
        input_l, embedding_l = embedding_block(n + 1, size, key)
        embedding_l = layers.Reshape(target_shape=(size,))(embedding_l)
        input_layer.append(input_l)
        entity_embedding.append(embedding_l)

    # concat all layer
    fc_layer = layers.concatenate([conv_layer, *entity_embedding])
    fc_layer = layers.Dropout(rate=0.5)(fc_layer)

    # dense layer
    fc_layer = layers.Dense(128, activation='elu')(fc_layer)
    fc_layer = layers.Dropout(rate=0.2)(fc_layer)

    # output layer
    output_layer = layers.Dense(1, activation='sigmoid')(fc_layer)
    model = models.Model(inputs=input_layer, outputs=output_layer)
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=[metrics.Precision(), metrics.Recall()])
    return model
```

4.3.2 URL FEATURES AND EXTRACTION

URL features are a major part of our data processing system. We needed to extract the main features from the URL database, so that our ML models can learn from these.

URLs mainly constitute these features :-

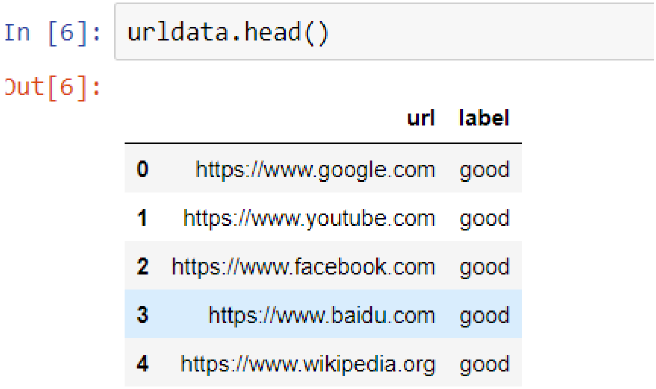
- Lexical
- Host Based
- Content Based Features

But after carefully researching these features we knew that taking every feature into account would be redundant. Lexical features were the most important features to be considered for our project.

So for our project we mainly divided our URL features into 3 main parts:-

1. Length features
2. Count features
3. Binary features

We began with a normal dataset which looked as follows.

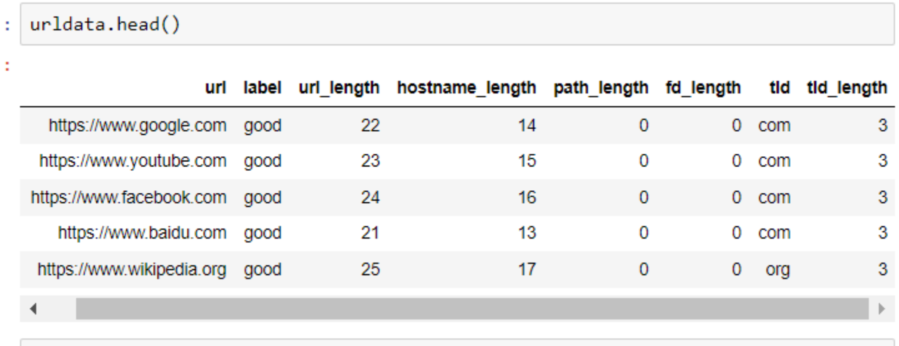


```
In [6]: urldata.head()
Out[6]:
```

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

Table 2 Original Dataset

Then after our length feature extraction we got to the following results.

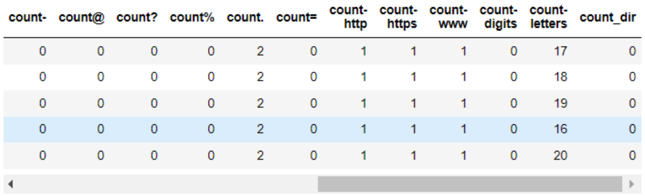


```
: urldata.head()
:
```

url	label	url_length	hostname_length	path_length	fd_length	tld	tld_length
https://www.google.com	good	22	14	0	0	com	3
https://www.youtube.com	good	23	15	0	0	com	3
https://www.facebook.com	good	24	16	0	0	com	3
https://www.baidu.com	good	21	13	0	0	com	3
https://www.wikipedia.org	good	25	17	0	0	org	3

Table 3 Length Features

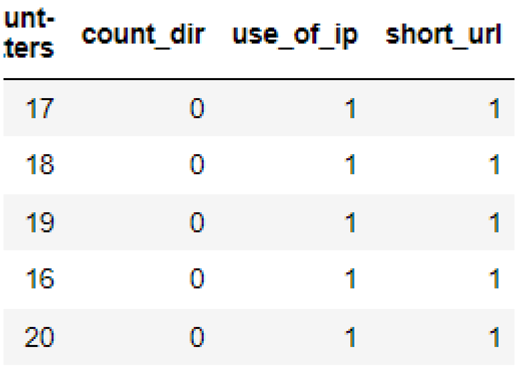
Further we went on to take into account the count features like number of digits, letters, counts of wwws etc.



count-	count@	count?	count%	count.	count=	count-http	count-https	count-www	count-digits	count-letters	count_dir
0	0	0	0	2	0	1	1	1	0	17	0
0	0	0	0	2	0	1	1	1	0	18	0
0	0	0	0	2	0	1	1	1	0	19	0
0	0	0	0	2	0	1	1	1	0	16	0
0	0	0	0	2	0	1	1	1	0	20	0

Table 4 Count Features

Now we were left with binary features like checking shortening services or checking if ip addresses are used in the URL.



unt-ters	count_dir	use_of_ip	short_url
17	0	1	1
18	0	1	1
19	0	1	1
16	0	1	1
20	0	1	1

All these features were utilized to feed to both our models specified in the later stages.

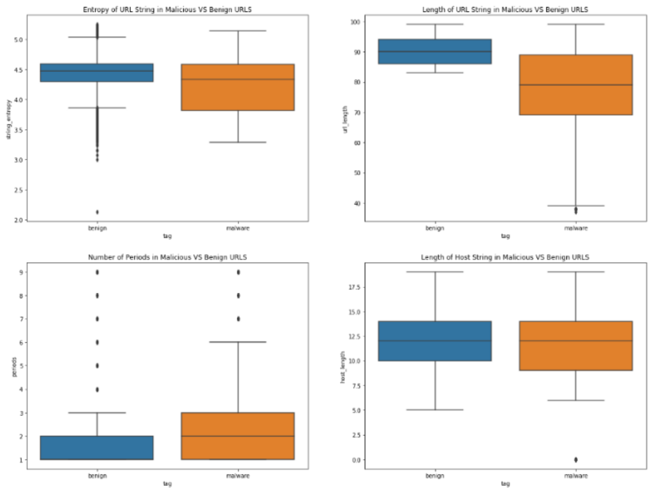


Figure 7 Lexical features comparison plots

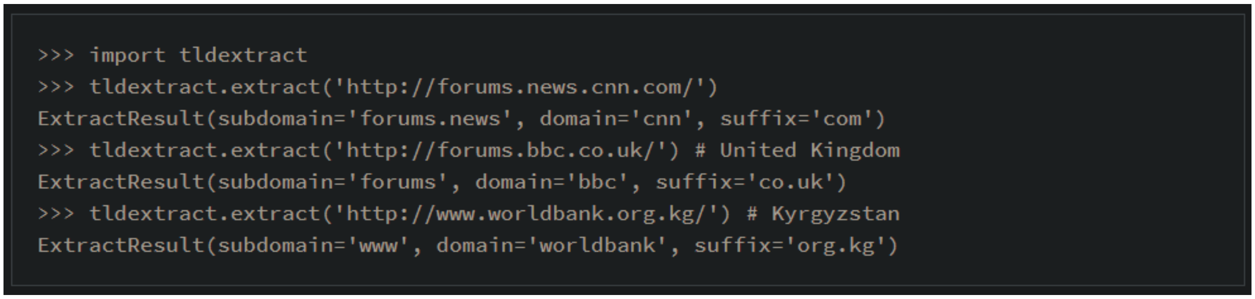
The following features will be extracted from the URL for classification.

1. Length Features
    1.1. Length Of Url
    1.2. Length of Hostname
    1.3. Length Of Path
    1.4. Length Of First Directory
    1.5. Length Of Top Level Domain

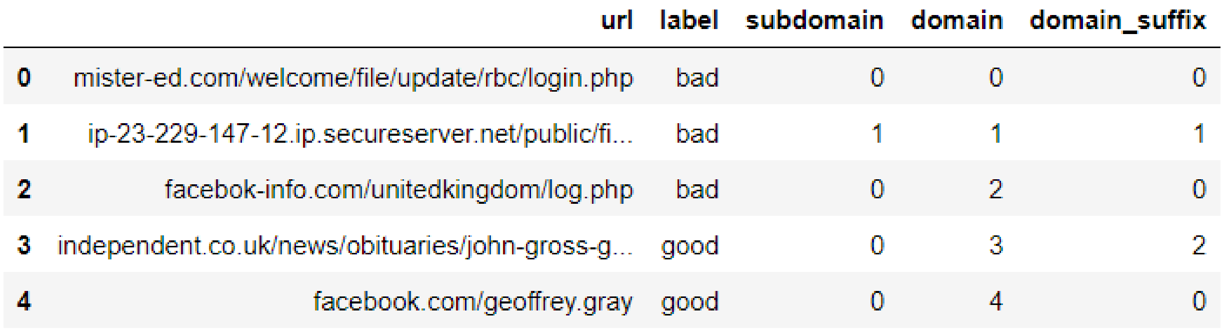
2. Count Features
    2.1. Count Of '-'
    2.2. Count Of '@'
    2.3. Count Of '?'
    2.4. Count Of '%'
    2.5. Count Of '.'
    2.6. Count Of '='
    2.7. Count Of 'http'
    2.8. Count Of 'www'
    2.9. Count Of Digits
    2.10. Count Of Letters
    2.11. Count Of Number Of Directories

3. Binary Features
    3.1. Use of IP or not
    3.2. Use of Shortening URL or not

Apart from the lexical features, we will use TFID - Term Frequency Inverse Document as well.



```
>>> import tldextract
>>> tldextract.extract('http://forums.news.cnn.com/')
ExtractResult(subdomain='forums.news', domain='cnn', suffix='com')
>>> tldextract.extract('http://forums.bbc.co.uk/') # United Kingdom
ExtractResult(subdomain='forums', domain='bbc', suffix='co.uk')
>>> tldextract.extract('http://www.worldbank.org.kg/') # Kyrgyzstan
ExtractResult(subdomain='www', domain='worldbank', suffix='org.kg')
```



	url	label	subdomain	domain	domain_suffix
0	mister-ed.com/welcome/file/update/rbc/login.php	bad	0	0	0
1	ip-23-229-147-12.ip.secureserver.net/public/fi...	bad	1	1	1
2	facebok-info.com/unitedkingdom/log.php	bad	0	2	0
3	independent.co.uk/news/obituaries/john-gross-g...	good	0	3	2
4	facebook.com/geoffrey.gray	good	0	4	0

Table 5 TLD Extract Features

4.3.3 VOTING SYSTEM

Here our proposed voting system model combines the 3 given models - Random Forest, Decision Trees and Logistic Regression. These 3 were the best fit for a lightweight and efficient model.

We have trained and used the system on large and variable datasets containing around 4.5 lakhs of URLs each and discovered major URL vectors and accuracy findings.

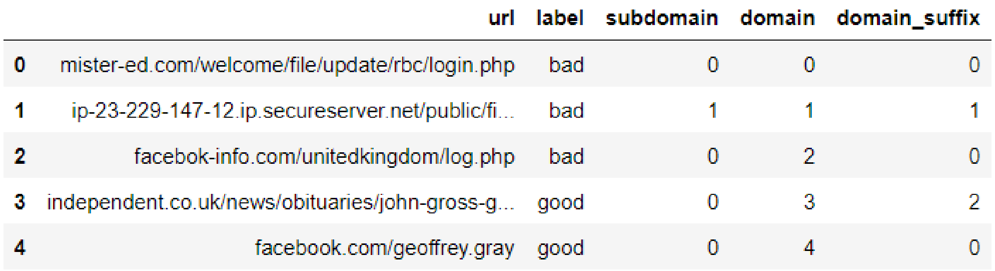
As with the original paper we had started with establishing a method to differentiate and predict different URLs being good or bad, or speaking in a little bit more technical sense to being malicious or benign, but at the end along with achieving our original plan we also created a new algorithm that utilized all the good features of the old three and gave a better result at the end.

CHAPTER 5

RESULTS AND DISCUSSIONS

As with the original paper we had started with establishing a method to differentiate and predict different URLs being good or bad, or speaking in a little bit more technical sense to being malicious or benign, but at the end along with achieving our original plan we also created a new algorithm that utilized all the good features of the old three and gave a better result at the end.

Apart from the aforementioned futuristic approach to the problem we also thought of finding out which URL features are more relevant than the others and we concluded that the https/http feature of the URL are significantly important in a sense that they can impact about 15% of the accuracy of almost all algorithms when not taken into account as opposed to the fact when taken into account for the prediction of the project.

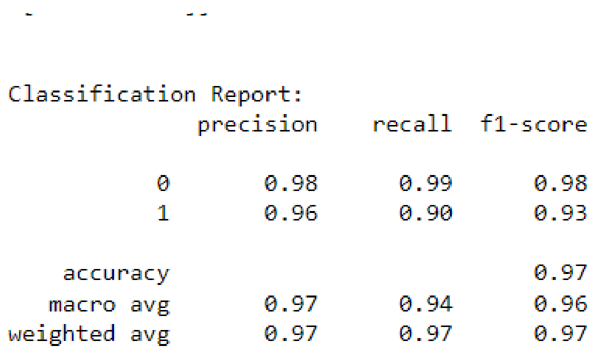


	url	label	subdomain	domain	domain_suffix
0	mister-ed.com/welcome/file/update/rbc/login.php	bad	0	0	0
1	ip-23-229-147-12.ip.secureserver.net/public/fi...	bad	1	1	1
2	facebok-info.com/unitedkingdom/log.php	bad	0	2	0
3	independent.co.uk/news/obituaries/john-gross-g...	good	0	3	2
4	facebook.com/geoffrey.gray	good	0	4	0

Figure 8 Representation of TLD Extract Features

The comparisons made are between the Logistic Regression, Decision Tree and the Rainforest algorithm, which have been later combined to create the voting system at the end of the project as the last step. Two different databases were utilized to get to the results.

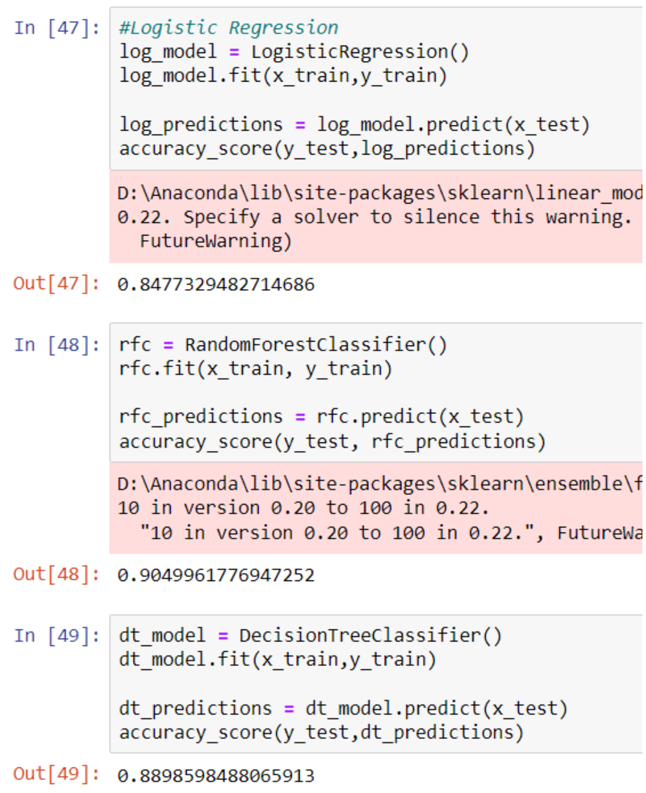
The convoluted neural networks had the following output -



```
Classification Report:
              precision    recall  f1-score

           0       0.98      0.99      0.98
           1       0.96      0.90      0.93

    accuracy                           0.97
   macro avg       0.97      0.94      0.96
weighted avg       0.97      0.97      0.97
```

Table 6 CNN Classification Report

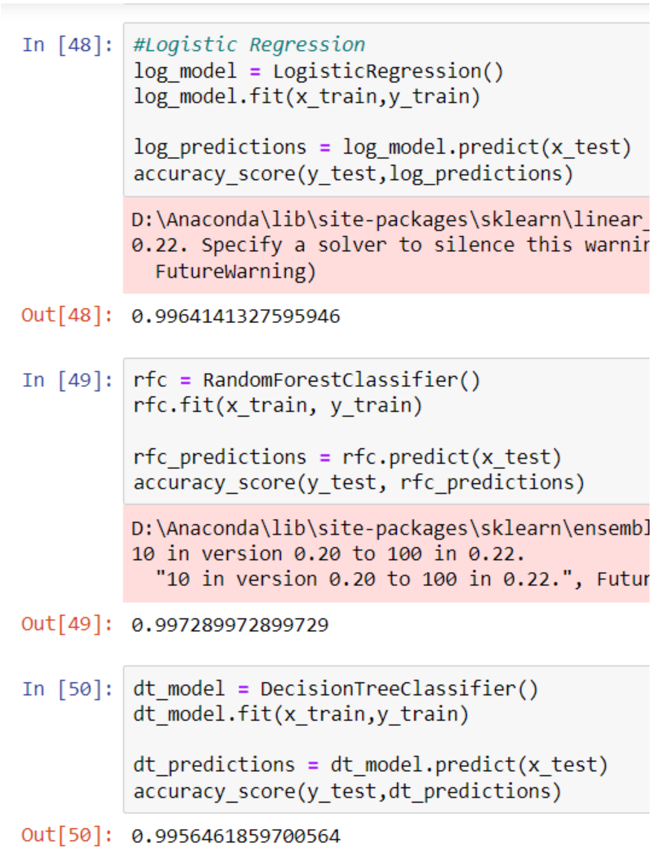
Now, coming to the URL vectors, the output received are as follows -



```
In [47]: #Logistic Regression
         log_model = LogisticRegression()
         log_model.fit(x_train,y_train)

         log_predictions = log_model.predict(x_test)
         accuracy_score(y_test,log_predictions)

         D:\Anaconda\lib\site-packages\sklearn\linear_mod
         0.22. Specify a solver to silence this warning.
           FutureWarning)

Out[47]: 0.8477329482714686

In [48]: rfc = RandomForestClassifier()
         rfc.fit(x_train, y_train)

         rfc_predictions = rfc.predict(x_test)
         accuracy_score(y_test, rfc_predictions)

         D:\Anaconda\lib\site-packages\sklearn\ensemble\f
         10 in version 0.20 to 100 in 0.22.
           "10 in version 0.20 to 100 in 0.22.", FutureWa

Out[48]: 0.9049961776947252

In [49]: dt_model = DecisionTreeClassifier()
         dt_model.fit(x_train,y_train)

         dt_predictions = dt_model.predict(x_test)
         accuracy_score(y_test,dt_predictions)

Out[49]: 0.8898598488065913
```

The accuracy achieved here using the URL vectors is lesser as compared to the CNN model.

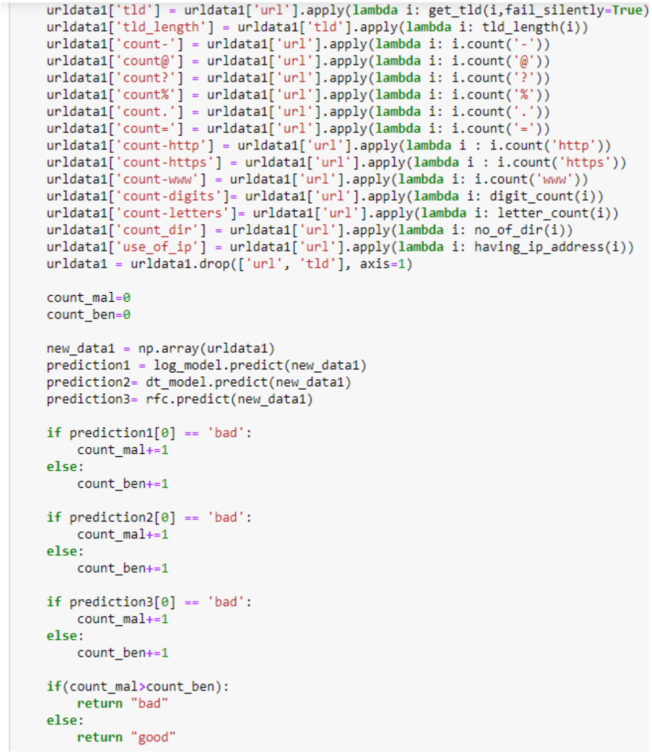
But during our testing we found that, if we take a dataset containing HTTPS and HTTP in the majority of URLs we end up with a significantly larger accuracy for each of these models.



```
In [48]: #Logistic Regression
         log_model = LogisticRegression()
         log_model.fit(x_train,y_train)

         log_predictions = log_model.predict(x_test)
         accuracy_score(y_test,log_predictions)

         D:\Anaconda\lib\site-packages\sklearn\linear_
         0.22. Specify a solver to silence this warnin
           FutureWarning)

Out[48]: 0.9964141327595946

In [49]: rfc = RandomForestClassifier()
         rfc.fit(x_train, y_train)

         rfc_predictions = rfc.predict(x_test)
         accuracy_score(y_test, rfc_predictions)

         D:\Anaconda\lib\site-packages\sklearn\ensembl
         10 in version 0.20 to 100 in 0.22.
           "10 in version 0.20 to 100 in 0.22.", Futur

Out[49]: 0.997289972899729

In [50]: dt_model = DecisionTreeClassifier()
         dt_model.fit(x_train,y_train)

         dt_predictions = dt_model.predict(x_test)
         accuracy_score(y_test,dt_predictions)

Out[50]: 0.9956461859700564
```

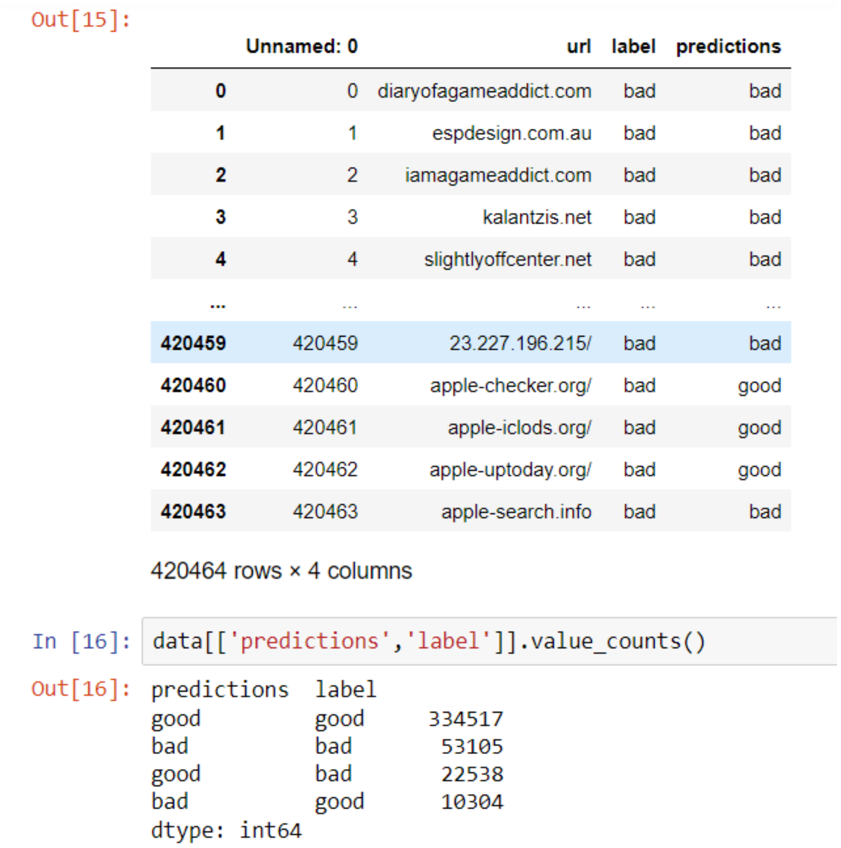
So, we can infer that the most important URL vector is the presence of HTTPS/HTTP.

As we could find the accuracy lacking for our 3 models, we tried to create a voting pipeline for these 3.



```
urldata1['tld'] = urldata1['url'].apply(lambda i: get_tld(i,fail_silently=True)
urldata1['tld_length'] = urldata1['tld'].apply(lambda i: tld_length(i))
urldata1['count-'] = urldata1['url'].apply(lambda i: i.count('-'))
urldata1['count@'] = urldata1['url'].apply(lambda i: i.count('@'))
urldata1['count?'] = urldata1['url'].apply(lambda i: i.count('?'))
urldata1['count%'] = urldata1['url'].apply(lambda i: i.count('%'))
urldata1['count.'] = urldata1['url'].apply(lambda i: i.count('.'))
urldata1['count='] = urldata1['url'].apply(lambda i: i.count('='))
urldata1['count-http'] = urldata1['url'].apply(lambda i : i.count('http'))
urldata1['count-https'] = urldata1['url'].apply(lambda i : i.count('https'))
urldata1['count-www'] = urldata1['url'].apply(lambda i: i.count('www'))
urldata1['count-digits']= urldata1['url'].apply(lambda i: digit_count(i))
urldata1['count-letters']= urldata1['url'].apply(lambda i: letter_count(i))
urldata1['count_dir'] = urldata1['url'].apply(lambda i: no_of_dir(i))
urldata1['use_of_ip'] = urldata1['url'].apply(lambda i: having_ip_address(i))
urldata1 = urldata1.drop(['url', 'tld'], axis=1)

count_mal=0
count_ben=0

new_data1 = np.array(urldata1)
prediction1 = log_model.predict(new_data1)
prediction2= dt_model.predict(new_data1)
prediction3= rfc.predict(new_data1)

if prediction1[0] == 'bad':
    count_mal+=1
else:
    count_ben+=1

if prediction2[0] == 'bad':
    count_mal+=1
else:
    count_ben+=1

if prediction3[0] == 'bad':
    count_mal+=1
else:
    count_ben+=1

if(count_mal>count_ben):
    return "bad"
else:
    return "good"
```

We utilize this voting system and we have improved on the accuracy to 92.18% from individual models. Also, this result helps in concluding that the most important URL vector turns out to be HTTPS when it comes to categorizing malicious and benign websites through lightweight and fast ML models.

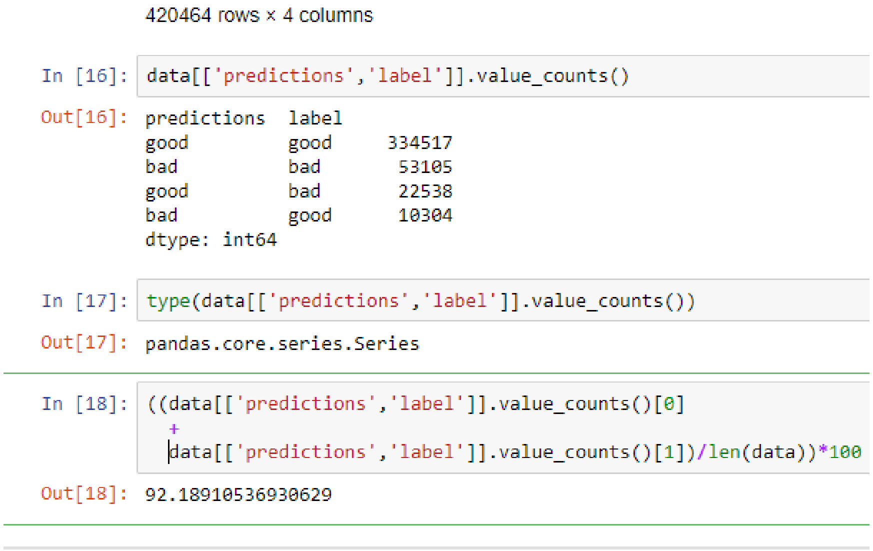


Out[15]:

	Unnamed: 0	url	label	predictions
0	0	diaryofagameaddict.com	bad	bad
1	1	espdesign.com.au	bad	bad
2	2	iamagameaddict.com	bad	bad
3	3	kalantzis.net	bad	bad
4	4	slightlyoffcenter.net	bad	bad
...	...	...	...	...
420459	420459	23.227.196.215/	bad	bad
420460	420460	apple-checker.org/	bad	good
420461	420461	apple-iclods.org/	bad	good
420462	420462	apple-uptoday.org/	bad	good
420463	420463	apple-search.info	bad	bad

420464 rows × 4 columns

```
In [16]: data[['predictions','label']].value_counts()

Out[16]: predictions  label
         good         good     334517
         bad          bad       53105
         good         bad       22538
         bad          good      10304
         dtype: int64
```



420464 rows × 4 columns

```
In [16]: data[['predictions','label']].value_counts()

Out[16]: predictions  label
         good         good     334517
         bad          bad       53105
         good         bad       22538
         bad          good      10304
         dtype: int64
```

```
In [17]: type(data[['predictions','label']].value_counts())

Out[17]: pandas.core.series.Series

In [18]: ((data[['predictions','label']].value_counts()[0]
           +
          data[['predictions','label']].value_counts()[1])/len(data))*100

Out[18]: 92.18910536930629
```

CHAPTER 6

CONCLUSION

As it is evident from the classification report that the CNN model provides us with great accuracy. Also looking at the confusion matrix the false prediction for both the positive and negative values is low as compared to the correct predictions.

The accuracy achieved on URL vectors was on par with the CNN model which was questionable as the models used were basic ones.

Thus we went on to try different databases and datasets to confirm our predictions. The models like logistic regression, Decision Trees and Random Forests gave us great accuracies when trained on datasets containing HTTPs or HTTP signatures.

On bringing up another dataset, where we gathered a labeled dataset without HTTP features at all, we could see that these same models didn’t stand up to the 97% accuracy the CNN model gave us with the same dataset.

Then we tried to improve the accuracy of these models, and as they were lightweight we made a voting system method where the accuracy for the same dataset rose upto 92%.

Thus we also proved that the main URL feature is HTTP feature and a mix of 3 lightweight models should be perfect for deployment, and yet CNN is the most accurate model.

REFERENCES

[1]. https://www.vircom.com/blog/what-is-a-malicious-url/ - Joey Tanny

[2]. https://purplesec.us/resources/cyber-security-statistics/ -Jason, Purplesec author

[3]. Vazhayil, Anu & Ravi, Vinayakumar & Kp, Soman. (2018). Comparative Study of the Detection of Malicious URLs Using Shallow and Deep Networks. 1-6. 10.1109/ICCCNT.2018.8494159.

[4]. B. Liang, J. Huang, F. Liu, D. Wang, D. Dong, and Z. Liang, “Maliciousweb pages detection based on abnormal visibility recognition,” in E-Business and Information System Security, 2009. EBISS’09. Interna-tional Conference on. IEEE, 2009, pp. 1–5.

[5]. S. Garera, N. Provos, M. Chew, and A. D. Rubin, “A framework for detection and measurement of phishing attacks,” in Proceedings of the2007 ACM workshop on Recurring malcode. ACM, 2007, pp. 1–8

[6]. A. Crisan, G. Florea, L. Halasz, C. Lemnaru, and C. Oprisa, “Detecting malicious URLs based on machine learning algorithms and word embeddings,” in Proceedings of the 2020 IEEE 16th International Conference on Intelligent Computer Communication and Processing (ICCP), pp. 187–193, IEEE, Cluj-Napoca, Romania, September 2020.

[7]. S. Singhal, U. Chawla, and R. Shorey, “Machine learning & concept drift based approach for malicious website detection,” in Proceedings of the 2020 International Conference on COMmunication Systems & NETworkS (COMSNETS), pp. 582–585, IEEE, Bengaluru, India, January 2020.

Conferences on Web Intelligence and Intelligent Agent Technology-Volume 01. IEEE Computer Society

[8]. A. Das, A. Das, A. Datta, S. Si, and S. Barman, “Deep approaches on malicious URL classification,” in Proceedings of the 2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT), pp. 1–6, IEEE, Kharagpur, India, 2020, July.

[9]. S. Tiwari, H. Rizvi and K. Kalaiselvi, "Malicious Website Navigation Prevention Using CNNs and URL Vectors: A Study," 2022 International Conference on Computer Communication and Informatics (ICCCI), 2022, pp. 1-6, doi: 10.1109/ICCCI54379.2022.9741056

[10]. Dataset sources -1 https://www.unb.ca/cic/datasets/url-2016.html

-2 https://www.kaggle.com/sid321axn/malicious-urls-dataset

APPENDICES A

Methodologies Used

Terms Used:

1) True Positives (TP): Correct malicious URLs prediction.
2) True Negatives (TN): Correct benign URLs prediction.
3) False Positives (FP): Incorrect malicious URLs prediction.
4) False Negatives (FN): Incorrect benign URLs prediction.

Formulas Used:

1) Precision = $\frac{True\ Positive}{Actual\ Results}$ or $\frac{True\ Positive}{True\ Positive + False\ Positive}$

Precision is the proportion of positive identifications which were truly predicted.

2) Recall $\frac{True\ Positive}{Predicted\ Results}$ or $\frac{True\ Positive}{True\ Positive + False\ Negative}$

Recall is the proportion of actual positives that were identified correctly.

3) Accuracy = $\frac{True\ Positive + True\ Negatives}{All\ Samples}$

Accuracy is the ratio of correct predictions to total samples.

4) F1 score – It’s a statistical measure of the harmonic mean between precision and recall.

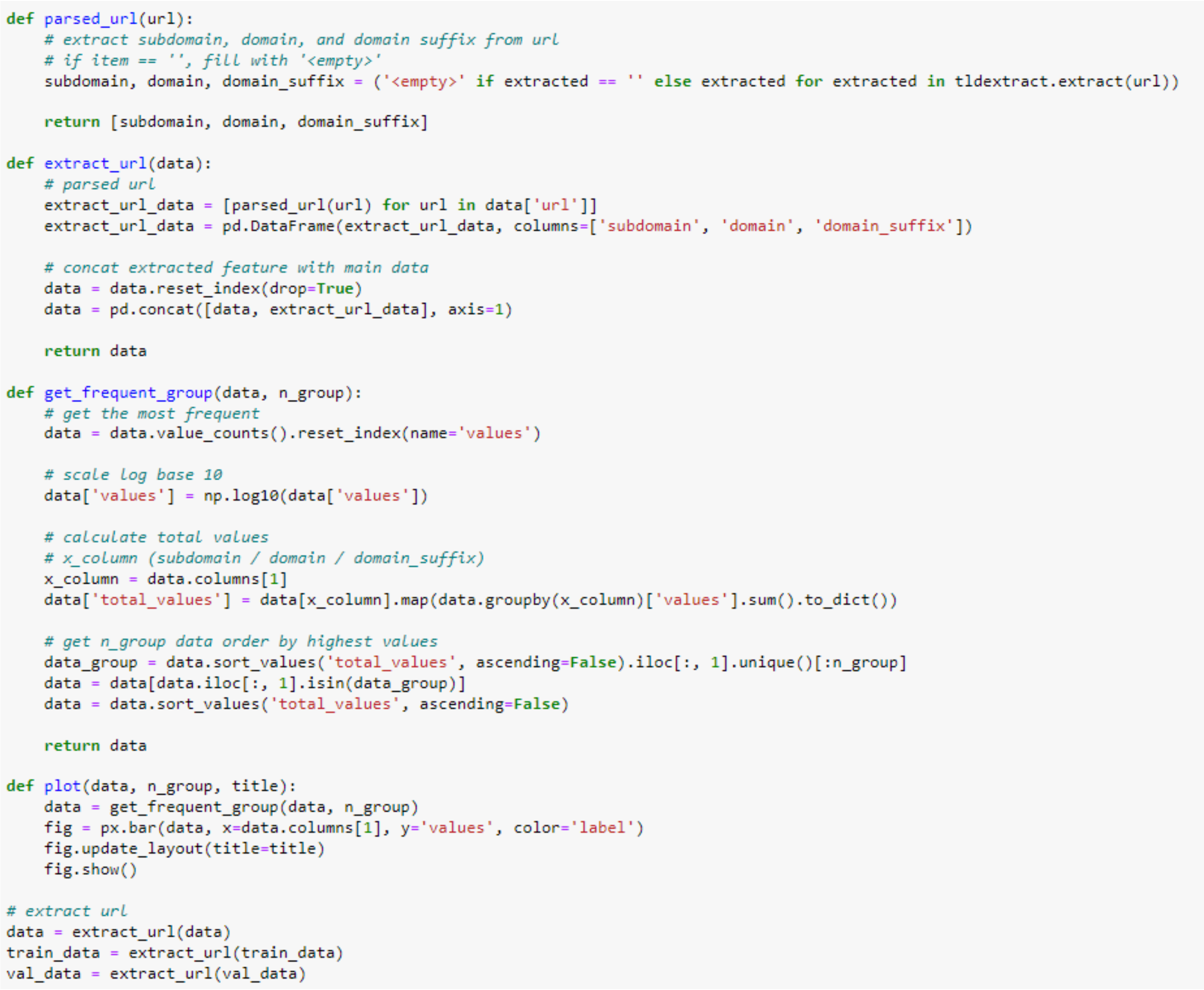
Confusion Matrix:

	Negative	Positive
Negative	TN	FP
Positive	FN	TP

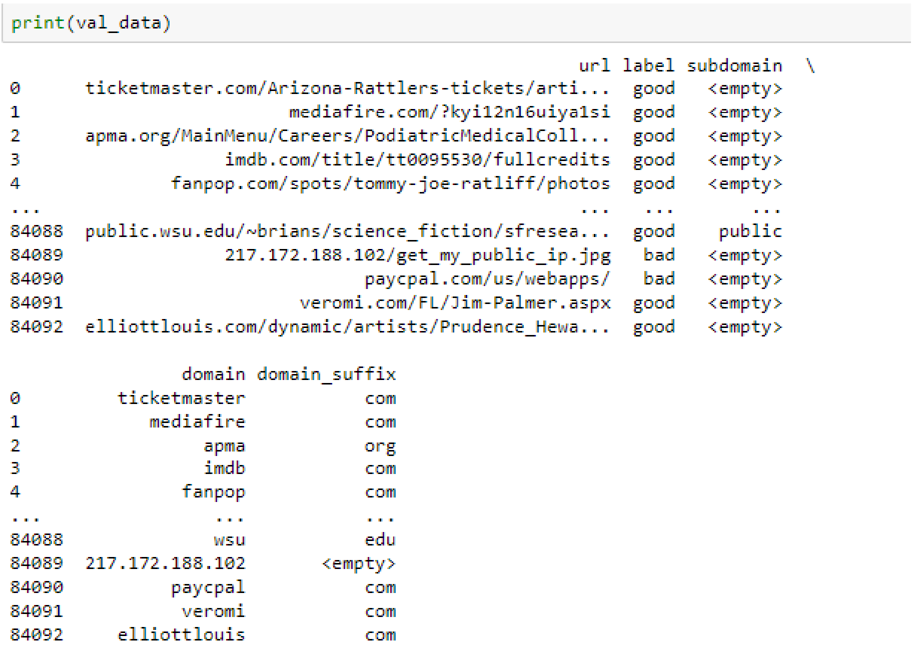
Table 7 Confusion Matrix Terminology

APPENDICES B

Code

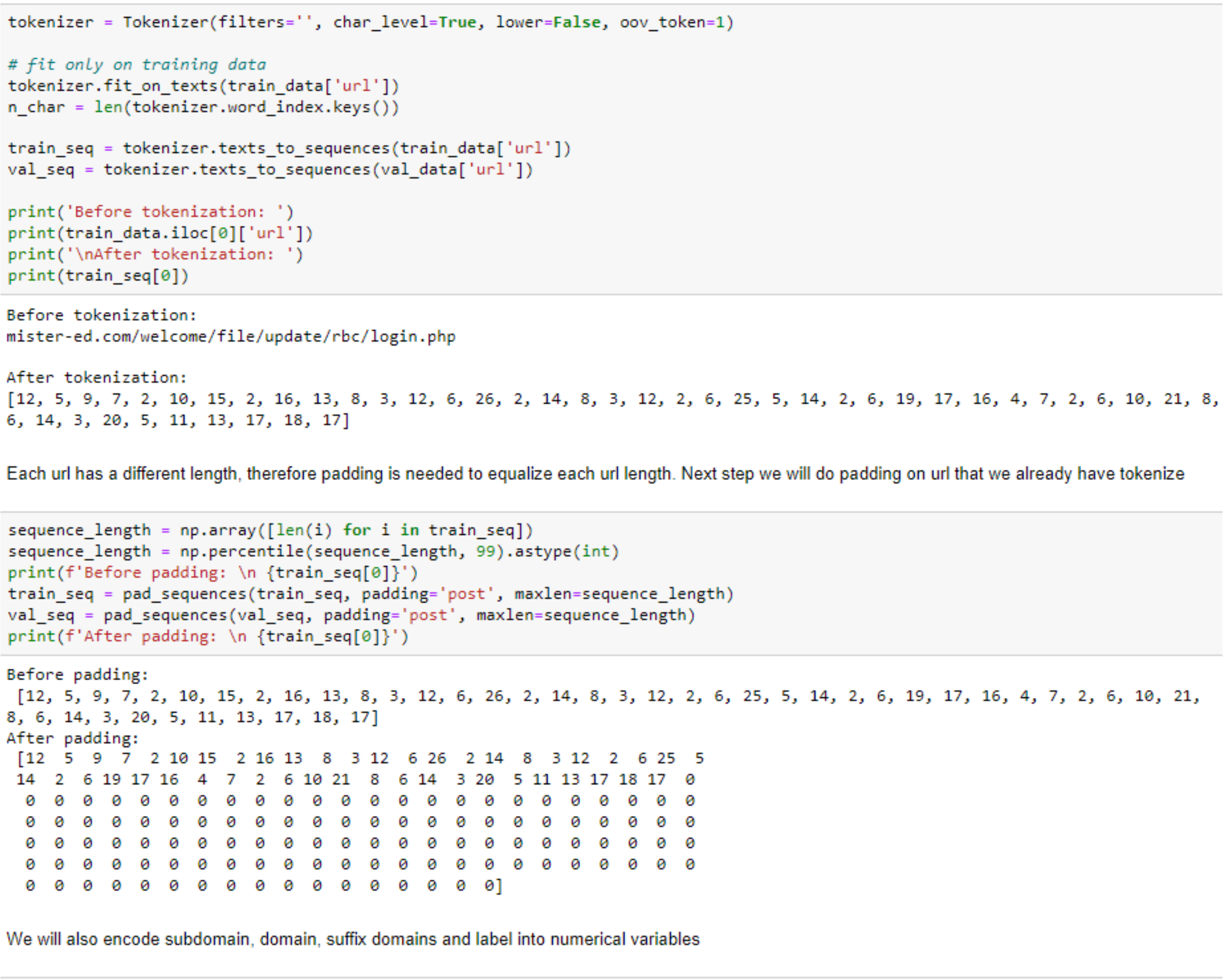


```
def parsed_url(url):
    # extract subdomain, domain, and domain suffix from url
    # if item == '', fill with '<empty>'
    subdomain, domain, domain_suffix = ('<empty>' if extracted == '' else extracted for extracted in tldextract.extract(url))

    return [subdomain, domain, domain_suffix]

def extract_url(data):
    # parsed url
    extract_url_data = [parsed_url(url) for url in data['url']]
    extract_url_data = pd.DataFrame(extract_url_data, columns=['subdomain', 'domain', 'domain_suffix'])

    # concat extracted feature with main data
    data = data.reset_index(drop=True)
    data = pd.concat([data, extract_url_data], axis=1)

    return data

def get_frequent_group(data, n_group):
    # get the most frequent
    data = data.value_counts().reset_index(name='values')

    # scale log base 10
    data['values'] = np.log10(data['values'])

    # calculate total values
    # x_column (subdomain / domain / domain_suffix)
    x_column = data.columns[1]
    data['total_values'] = data[x_column].map(data.groupby(x_column)['values'].sum().to_dict())

    # get n_group data order by highest values
    data_group = data.sort_values('total_values', ascending=False).iloc[:, 1].unique()[:n_group]
    data = data[data.iloc[:, 1].isin(data_group)]
    data = data.sort_values('total_values', ascending=False)

    return data

def plot(data, n_group, title):
    data = get_frequent_group(data, n_group)
    fig = px.bar(data, x=data.columns[1], y='values', color='label')
    fig.update_layout(title=title)
    fig.show()

# extract url
data = extract_url(data)
train_data = extract_url(train_data)
val_data = extract_url(val_data)
```



```
print(val_data)
```

```
                                                     url label subdomain  \
0      ticketmaster.com/Arizona-Rattlers-tickets/arti...  good   <empty>
1                         mediafire.com/?kyi12n16uiya1si  good   <empty>
2      apma.org/MainMenu/Careers/PodiatricMedicalColl...  good   <empty>
3                   imdb.com/title/tt0095530/fullcredits  good   <empty>
4             fanpop.com/spots/tommy-joe-ratliff/photos  good   <empty>
...                                                  ...   ...       ...
84088  public.wsu.edu/~brians/science_fiction/sfresea...  good    public
84089               217.172.188.102/get_my_public_ip.jpg   bad   <empty>
84090                             paycpal.com/us/webapps/   bad   <empty>
84091                       veromi.com/FL/Jim-Palmer.aspx  good   <empty>
84092  elliottlouis.com/dynamic/artists/Prudence_Hewa...  good   <empty>

                domain domain_suffix
0         ticketmaster           com
1            mediafire           com
2                 apma           org
3                 imdb           com
4               fanpop           com
...                ...           ...
84088              wsu           edu
84089  217.172.188.102       <empty>
84090          paycpal           com
84091           veromi           com
84092     elliottlouis           com
```



```
tokenizer = Tokenizer(filters='', char_level=True, lower=False, oov_token=1)

# fit only on training data
tokenizer.fit_on_texts(train_data['url'])
n_char = len(tokenizer.word_index.keys())

train_seq = tokenizer.texts_to_sequences(train_data['url'])
val_seq = tokenizer.texts_to_sequences(val_data['url'])

print('Before tokenization: ')
print(train_data.iloc[0]['url'])
print('\nAfter tokenization: ')
print(train_seq[0])
```

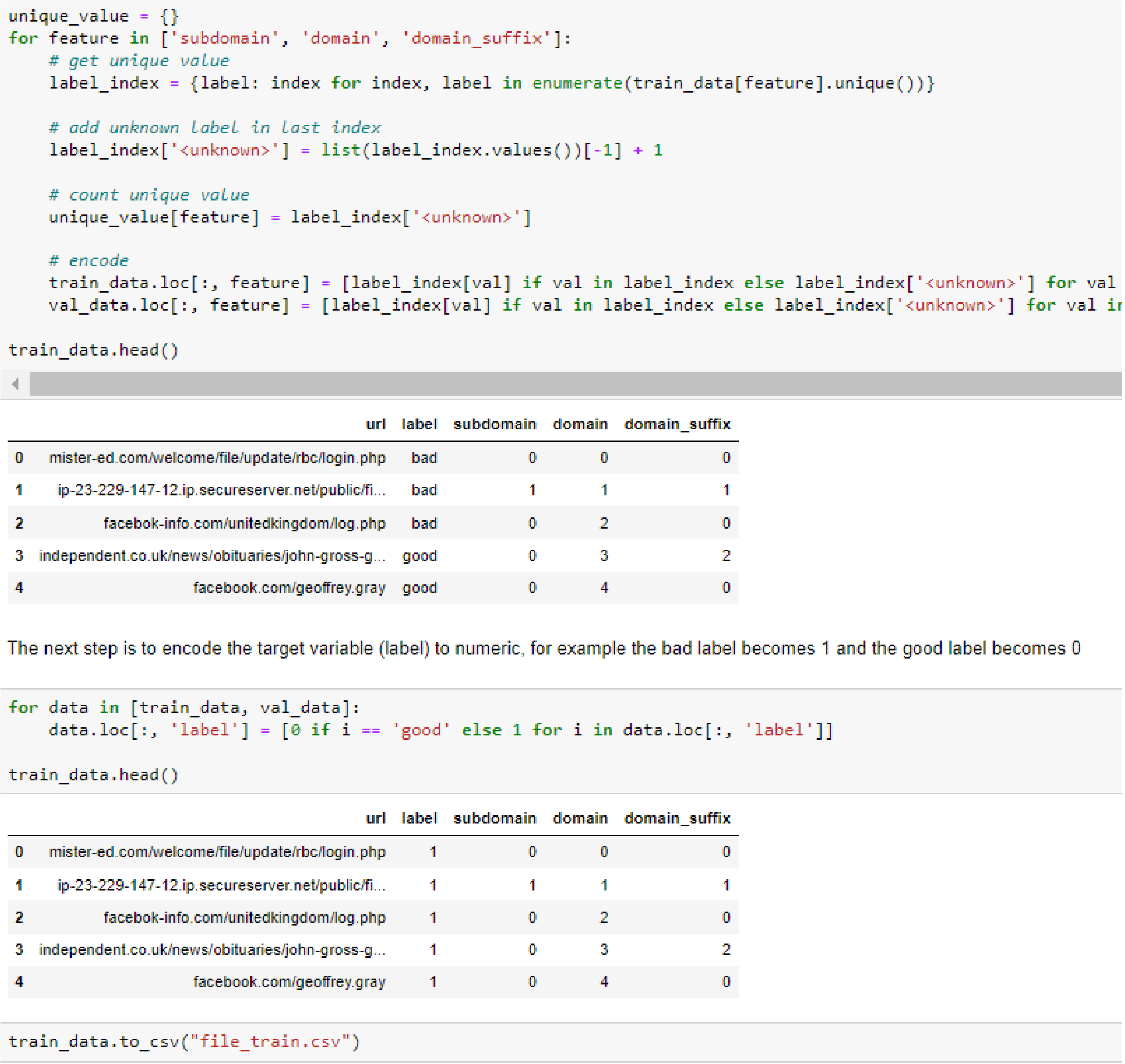
```
Before tokenization:
mister-ed.com/welcome/file/update/rbc/login.php

After tokenization:
[12, 5, 9, 7, 2, 10, 15, 2, 16, 13, 8, 3, 12, 6, 26, 2, 14, 8, 3, 12, 2, 6, 25, 5, 14, 2, 6, 19, 17, 16, 4, 7, 2, 6, 10, 21, 8,
6, 14, 3, 20, 5, 11, 13, 17, 18, 17]
```

Each url has a different length, therefore padding is needed to equalize each url length. Next step we will do padding on url that we already have tokenize

```
sequence_length = np.array([len(i) for i in train_seq])
sequence_length = np.percentile(sequence_length, 99).astype(int)
print(f'Before padding: \n {train_seq[0]}')
train_seq = pad_sequences(train_seq, padding='post', maxlen=sequence_length)
val_seq = pad_sequences(val_seq, padding='post', maxlen=sequence_length)
print(f'After padding: \n {train_seq[0]}')
```

```
Before padding:
 [12, 5, 9, 7, 2, 10, 15, 2, 16, 13, 8, 3, 12, 6, 26, 2, 14, 8, 3, 12, 2, 6, 25, 5, 14, 2, 6, 19, 17, 16, 4, 7, 2, 6, 10, 21,
8, 6, 14, 3, 20, 5, 11, 13, 17, 18, 17]
After padding:
 [12  5  9  7  2 10 15  2 16 13  8  3 12  6 26  2 14  8  3 12  2  6 25  5
 14  2  6 19 17 16  4  7  2  6 10 21  8  6 14  3 20  5 11 13 17 18 17  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
```

We will also encode subdomain, domain, suffix domains and label into numerical variables



```
unique_value = {}
for feature in ['subdomain', 'domain', 'domain_suffix']:
    # get unique value
    label_index = {label: index for index, label in enumerate(train_data[feature].unique())}

    # add unknown label in last index
    label_index['<unknown>'] = list(label_index.values())[-1] + 1

    # count unique value
    unique_value[feature] = label_index['<unknown>']

    # encode
    train_data.loc[:, feature] = [label_index[val] if val in label_index else label_index['<unknown>'] for val
    val_data.loc[:, feature] = [label_index[val] if val in label_index else label_index['<unknown>'] for val i

train_data.head()
```

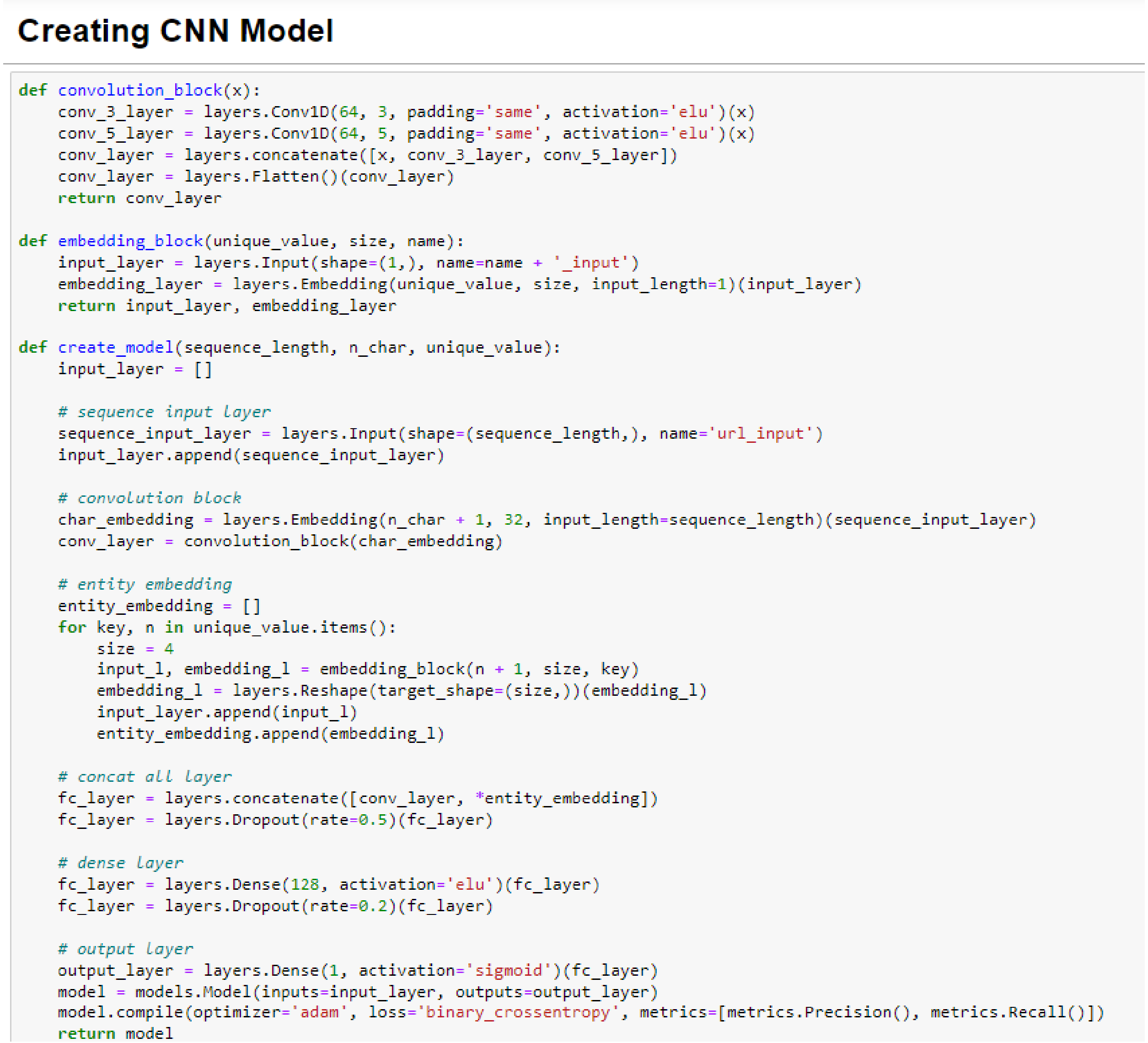
	url	label	subdomain	domain	domain_suffix
0	mister-ed.com/welcome/file/update/rbc/login.php	bad	0	0	0
1	ip-23-229-147-12.ip.secureserver.net/public/fi...	bad	1	1	1
2	facebok-info.com/unitedkingdom/log.php	bad	0	2	0
3	independent.co.uk/news/obituaries/john-gross-g...	good	0	3	2
4	facebook.com/geoffrey.gray	good	0	4	0

The next step is to encode the target variable (label) to numeric, for example the bad label becomes 1 and the good label becomes 0

```
for data in [train_data, val_data]:
    data.loc[:, 'label'] = [0 if i == 'good' else 1 for i in data.loc[:, 'label']]

train_data.head()
```

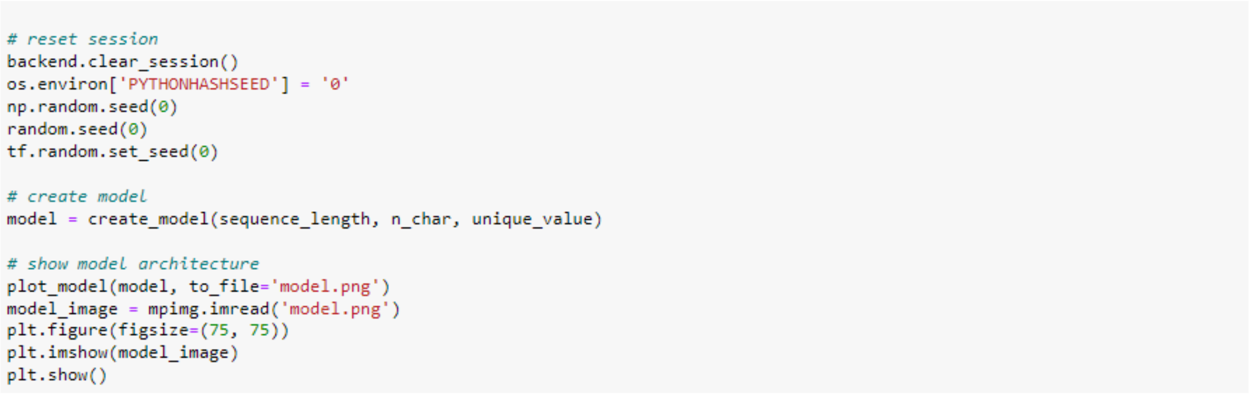
	url	label	subdomain	domain	domain_suffix
0	mister-ed.com/welcome/file/update/rbc/login.php	1	0	0	0
1	ip-23-229-147-12.ip.secureserver.net/public/fi...	1	1	1	1
2	facebok-info.com/unitedkingdom/log.php	1	0	2	0
3	independent.co.uk/news/obituaries/john-gross-g...	1	0	3	2
4	facebook.com/geoffrey.gray	1	0	4	0

```
train_data.to_csv("file_train.csv")
```



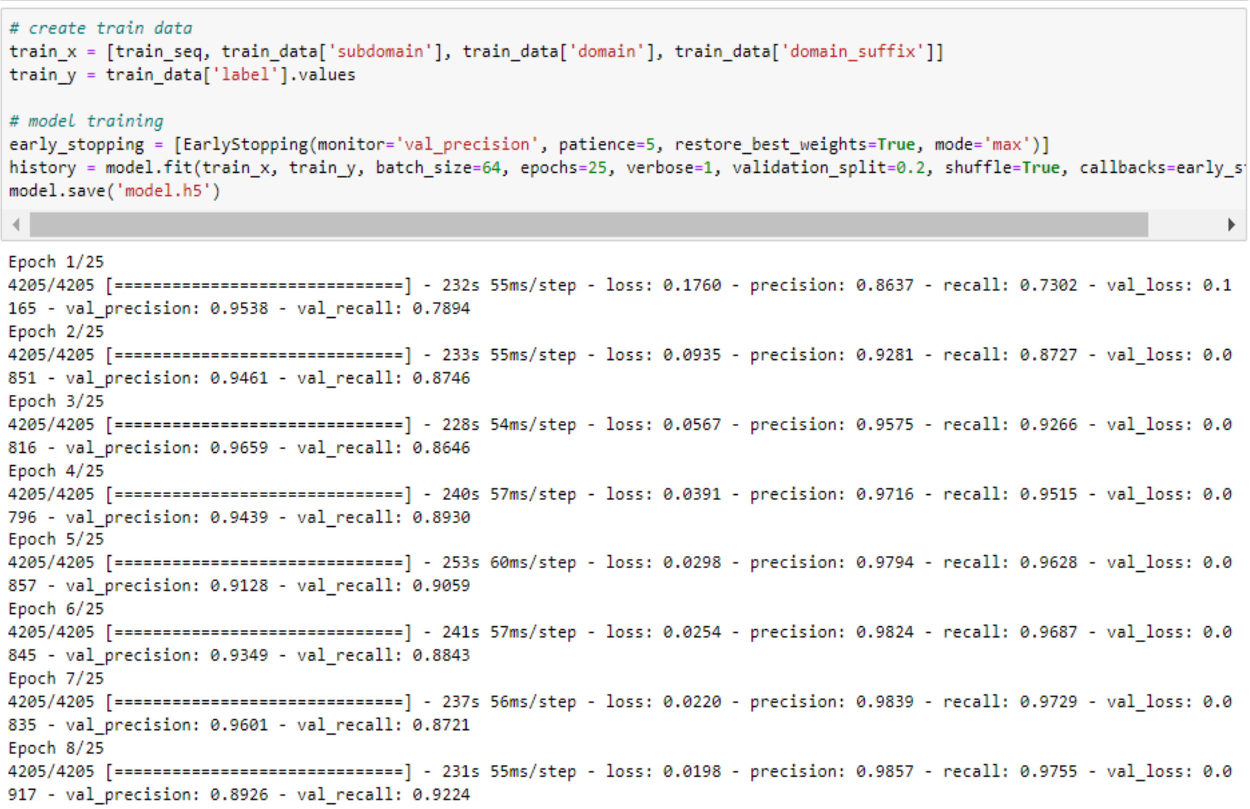
Creating CNN Model

```
def convolution_block(x):
    conv_3_layer = layers.Conv1D(64, 3, padding='same', activation='elu')(x)
    conv_5_layer = layers.Conv1D(64, 5, padding='same', activation='elu')(x)
    conv_layer = layers.concatenate([x, conv_3_layer, conv_5_layer])
    conv_layer = layers.Flatten()(conv_layer)
    return conv_layer

def embedding_block(unique_value, size, name):
    input_layer = layers.Input(shape=(1,), name=name + '_input')
    embedding_layer = layers.Embedding(unique_value, size, input_length=1)(input_layer)
    return input_layer, embedding_layer

def create_model(sequence_length, n_char, unique_value):
    input_layer = []

    # sequence input layer
    sequence_input_layer = layers.Input(shape=(sequence_length,), name='url_input')
    input_layer.append(sequence_input_layer)

    # convolution block
    char_embedding = layers.Embedding(n_char + 1, 32, input_length=sequence_length)(sequence_input_layer)
    conv_layer = convolution_block(char_embedding)

    # entity embedding
    entity_embedding = []
    for key, n in unique_value.items():
        size = 4
        input_l, embedding_l = embedding_block(n + 1, size, key)
        embedding_l = layers.Reshape(target_shape=(size,))(embedding_l)
        input_layer.append(input_l)
        entity_embedding.append(embedding_l)

    # concat all layer
    fc_layer = layers.concatenate([conv_layer, *entity_embedding])
    fc_layer = layers.Dropout(rate=0.5)(fc_layer)

    # dense layer
    fc_layer = layers.Dense(128, activation='elu')(fc_layer)
    fc_layer = layers.Dropout(rate=0.2)(fc_layer)

    # output layer
    output_layer = layers.Dense(1, activation='sigmoid')(fc_layer)
    model = models.Model(inputs=input_layer, outputs=output_layer)
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=[metrics.Precision(), metrics.Recall()])
    return model
```



```
# reset session
backend.clear_session()
os.environ['PYTHONHASHSEED'] = '0'
np.random.seed(0)
random.seed(0)
tf.random.set_seed(0)

# create model
model = create_model(sequence_length, n_char, unique_value)

# show model architecture
plot_model(model, to_file='model.png')
model_image = mpimg.imread('model.png')
plt.figure(figsize=(75, 75))
plt.imshow(model_image)
plt.show()
```

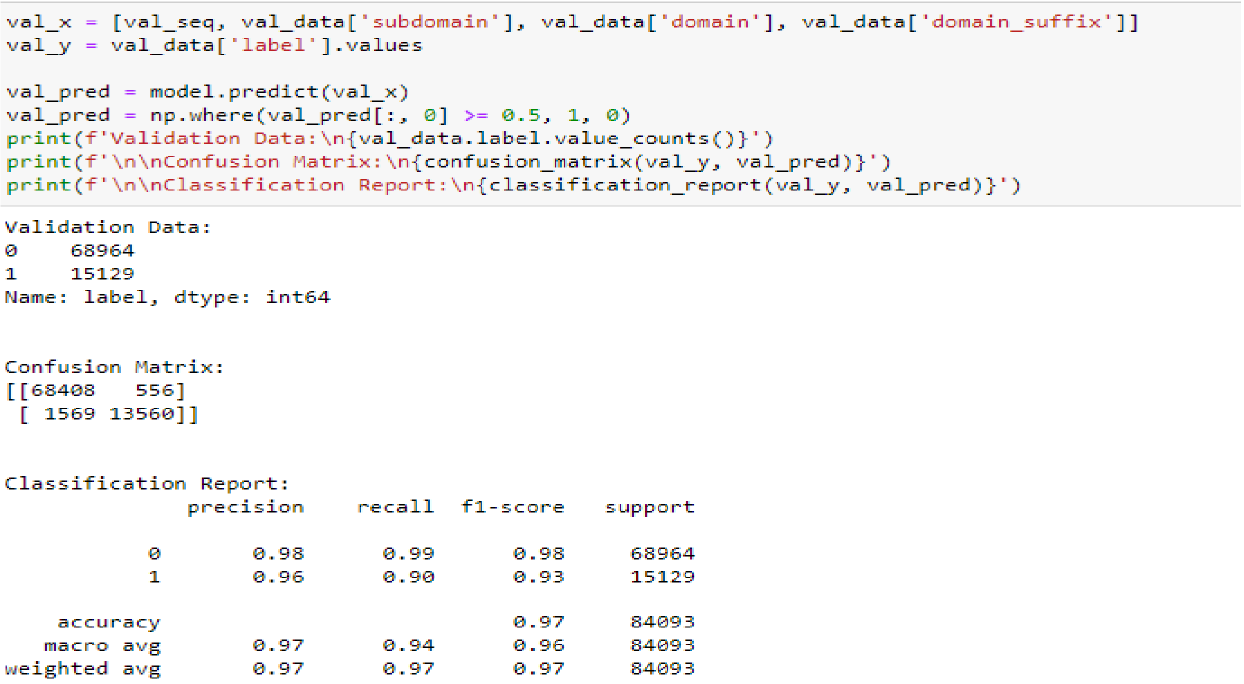
Model Training with early stopping parameter



```
# create train data
train_x = [train_seq, train_data['subdomain'], train_data['domain'], train_data['domain_suffix']]
train_y = train_data['label'].values

# model training
early_stopping = [EarlyStopping(monitor='val_precision', patience=5, restore_best_weights=True, mode='max')]
history = model.fit(train_x, train_y, batch_size=64, epochs=25, verbose=1, validation_split=0.2, shuffle=True, callbacks=early_s
model.save('model.h5')
```

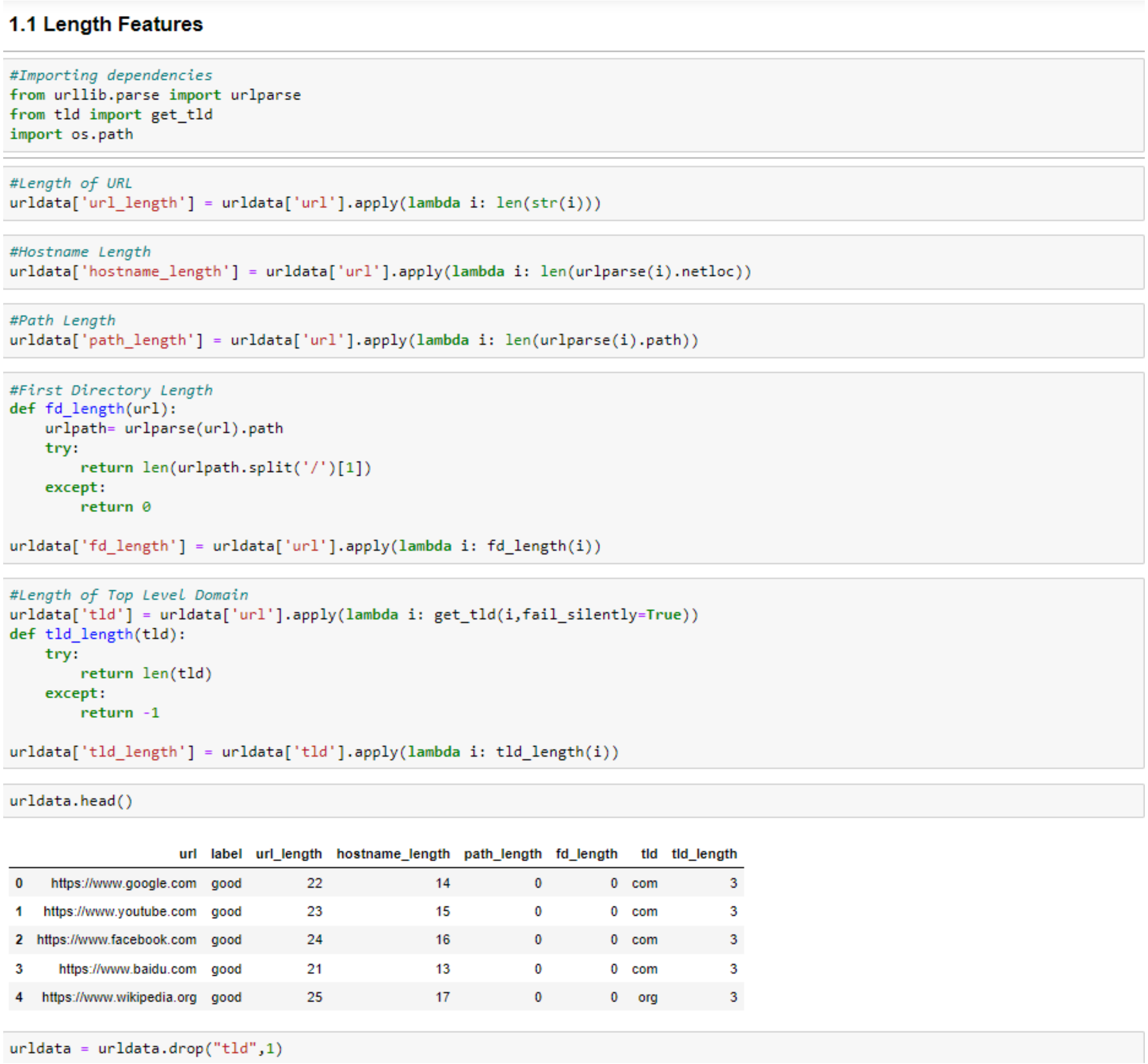
```
Epoch 1/25
4205/4205 [==============================] - 232s 55ms/step - loss: 0.1760 - precision: 0.8637 - recall: 0.7302 - val_loss: 0.1
165 - val_precision: 0.9538 - val_recall: 0.7894
Epoch 2/25
4205/4205 [==============================] - 233s 55ms/step - loss: 0.0935 - precision: 0.9281 - recall: 0.8727 - val_loss: 0.0
851 - val_precision: 0.9461 - val_recall: 0.8746
Epoch 3/25
4205/4205 [==============================] - 228s 54ms/step - loss: 0.0567 - precision: 0.9575 - recall: 0.9266 - val_loss: 0.0
816 - val_precision: 0.9659 - val_recall: 0.8646
Epoch 4/25
4205/4205 [==============================] - 240s 57ms/step - loss: 0.0391 - precision: 0.9716 - recall: 0.9515 - val_loss: 0.0
796 - val_precision: 0.9439 - val_recall: 0.8930
Epoch 5/25
4205/4205 [==============================] - 253s 60ms/step - loss: 0.0298 - precision: 0.9794 - recall: 0.9628 - val_loss: 0.0
857 - val_precision: 0.9128 - val_recall: 0.9059
Epoch 6/25
4205/4205 [==============================] - 241s 57ms/step - loss: 0.0254 - precision: 0.9824 - recall: 0.9687 - val_loss: 0.0
845 - val_precision: 0.9349 - val_recall: 0.8843
Epoch 7/25
4205/4205 [==============================] - 237s 56ms/step - loss: 0.0220 - precision: 0.9839 - recall: 0.9729 - val_loss: 0.0
835 - val_precision: 0.9601 - val_recall: 0.8721
Epoch 8/25
4205/4205 [==============================] - 231s 55ms/step - loss: 0.0198 - precision: 0.9857 - recall: 0.9755 - val_loss: 0.0
917 - val_precision: 0.8926 - val_recall: 0.9224
```

Model Validation



```
val_x = [val_seq, val_data['subdomain'], val_data['domain'], val_data['domain_suffix']]
val_y = val_data['label'].values

val_pred = model.predict(val_x)
val_pred = np.where(val_pred[:, 0] >= 0.5, 1, 0)
print(f'Validation Data:\n{val_data.label.value_counts()}')
print(f'\n\nConfusion Matrix:\n{confusion_matrix(val_y, val_pred)}')
print(f'\n\nClassification Report:\n{classification_report(val_y, val_pred)}')
```

```
Validation Data:
0    68964
1    15129
Name: label, dtype: int64


Confusion Matrix:
[[68408   556]
 [ 1569 13560]]


Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.99      0.98     68964
           1       0.96      0.90      0.93     15129

    accuracy                           0.97     84093
   macro avg       0.97      0.94      0.96     84093
weighted avg       0.97      0.97      0.97     84093
```



1.1 Length Features

```python
#Importing dependencies
from urllib.parse import urlparse
from tld import get_tld
import os.path
```

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

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

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

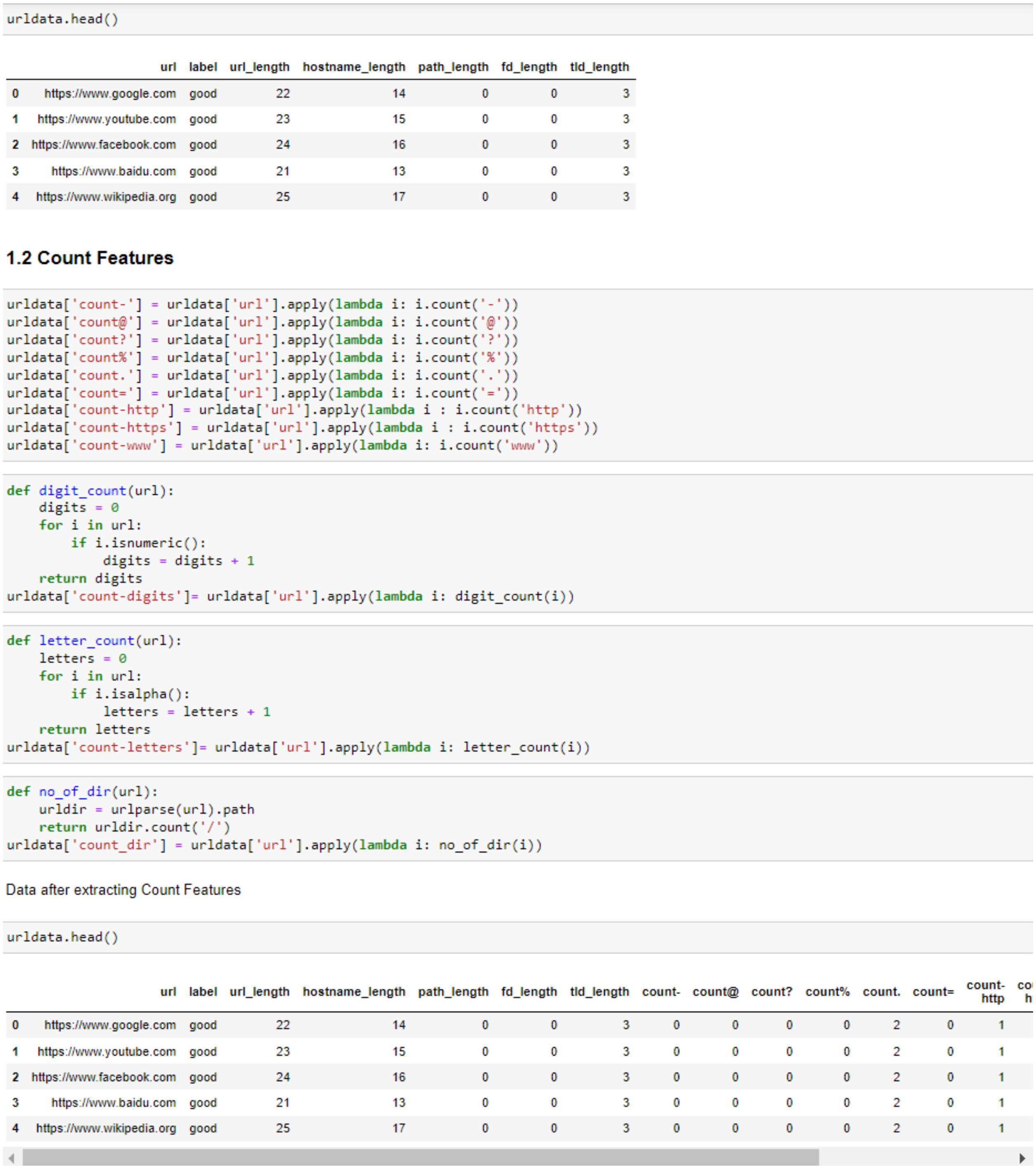
```python
#First Directory Length
def fd_length(url):
    urlpath= urlparse(url).path
    try:
        return len(urlpath.split('/')[1])
    except:
        return 0

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

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

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

```python
urldata.head()
```

	url	label	url_length	hostname_length	path_length	fd_length	tld	tld_length
0	https://www.google.com	good	22	14	0	0	com	3
1	https://www.youtube.com	good	23	15	0	0	com	3
2	https://www.facebook.com	good	24	16	0	0	com	3
3	https://www.baidu.com	good	21	13	0	0	com	3
4	https://www.wikipedia.org	good	25	17	0	0	org	3

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



```
urldata.head()
```

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

1.2 Count Features

```
urldata['count-'] = urldata['url'].apply(lambda i: i.count('-'))
urldata['count@'] = urldata['url'].apply(lambda i: i.count('@'))
urldata['count?'] = urldata['url'].apply(lambda i: i.count('?'))
urldata['count%'] = urldata['url'].apply(lambda i: i.count('%'))
urldata['count.'] = urldata['url'].apply(lambda i: i.count('.'))
urldata['count='] = urldata['url'].apply(lambda i: i.count('='))
urldata['count-http'] = urldata['url'].apply(lambda i : i.count('http'))
urldata['count-https'] = urldata['url'].apply(lambda i : i.count('https'))
urldata['count-www'] = urldata['url'].apply(lambda i: i.count('www'))
```

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

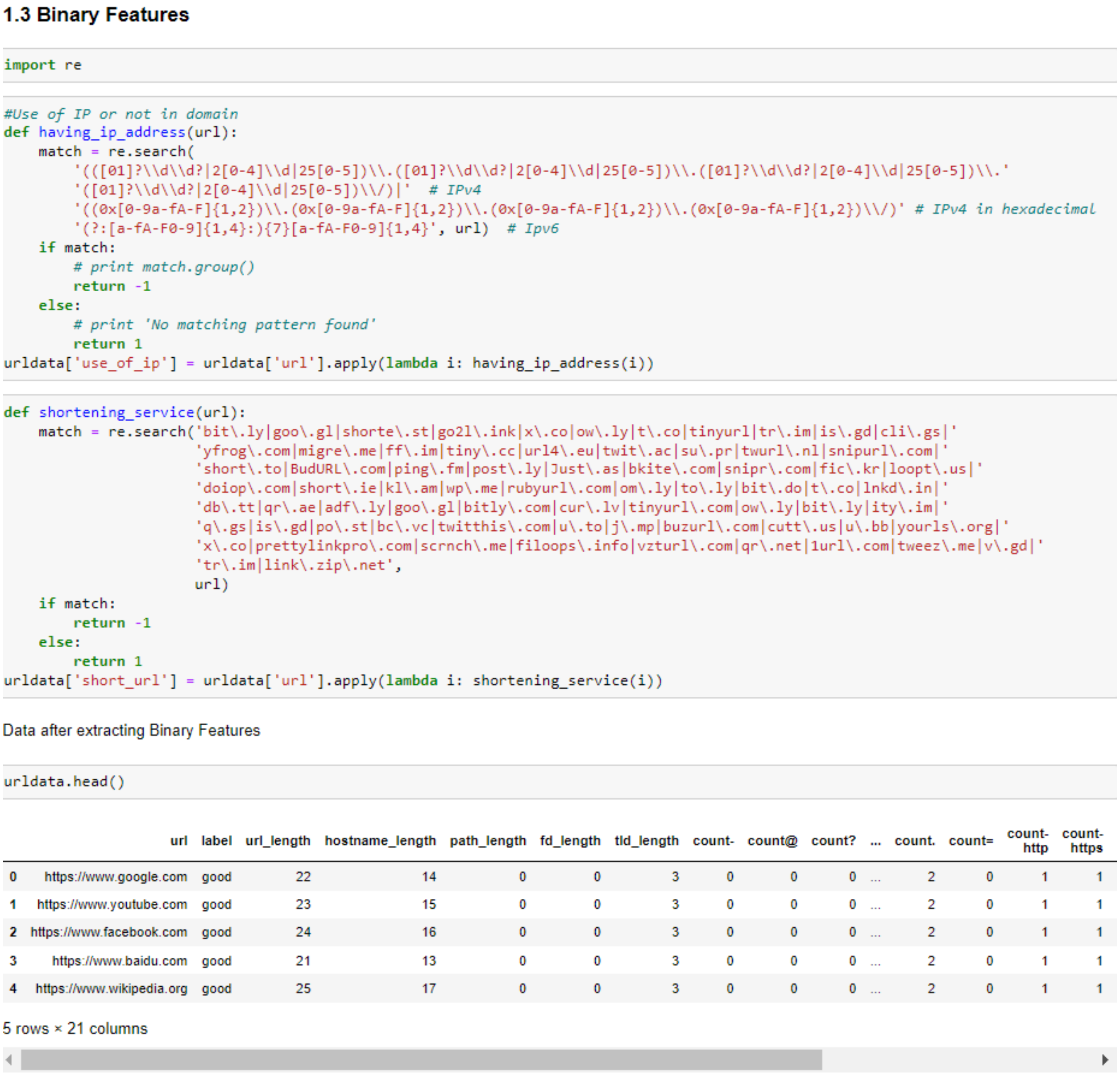
```
def letter_count(url):
    letters = 0
    for i in url:
        if i.isalpha():
            letters = letters + 1
    return letters
urldata['count-letters']= urldata['url'].apply(lambda i: letter_count(i))
```

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

Data after extracting Count Features

```
urldata.head()
```

	url	label	url_length	hostname_length	path_length	fd_length	tld_length	count-	count@	count?	count%	count.	count=	count- http	co h
0	https://www.google.com	good	22	14	0	0	3	0	0	0	0	2	0	1	
1	https://www.youtube.com	good	23	15	0	0	3	0	0	0	0	2	0	1	
2	https://www.facebook.com	good	24	16	0	0	3	0	0	0	0	2	0	1	
3	https://www.baidu.com	good	21	13	0	0	3	0	0	0	0	2	0	1	
4	https://www.wikipedia.org	good	25	17	0	0	3	0	0	0	0	2	0	1	



1.3 Binary Features

```
import re
```

```
#Use of IP or not in domain
def having_ip_address(url):
    match = re.search(
        '(([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.'
        '([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\/)|'  # IPv4
        '((0x[0-9a-fA-F]{1,2})\\.(0x[0-9a-fA-F]{1,2})\\.(0x[0-9a-fA-F]{1,2})\\.(0x[0-9a-fA-F]{1,2})\\/)' # IPv4 in hexadecimal
        '(?:[a-fA-F0-9]{1,4}:){7}[a-fA-F0-9]{1,4}', url)  # Ipv6
    if match:
        # print match.group()
        return -1
    else:
        # print 'No matching pattern found'
        return 1
urldata['use_of_ip'] = urldata['url'].apply(lambda i: having_ip_address(i))
```

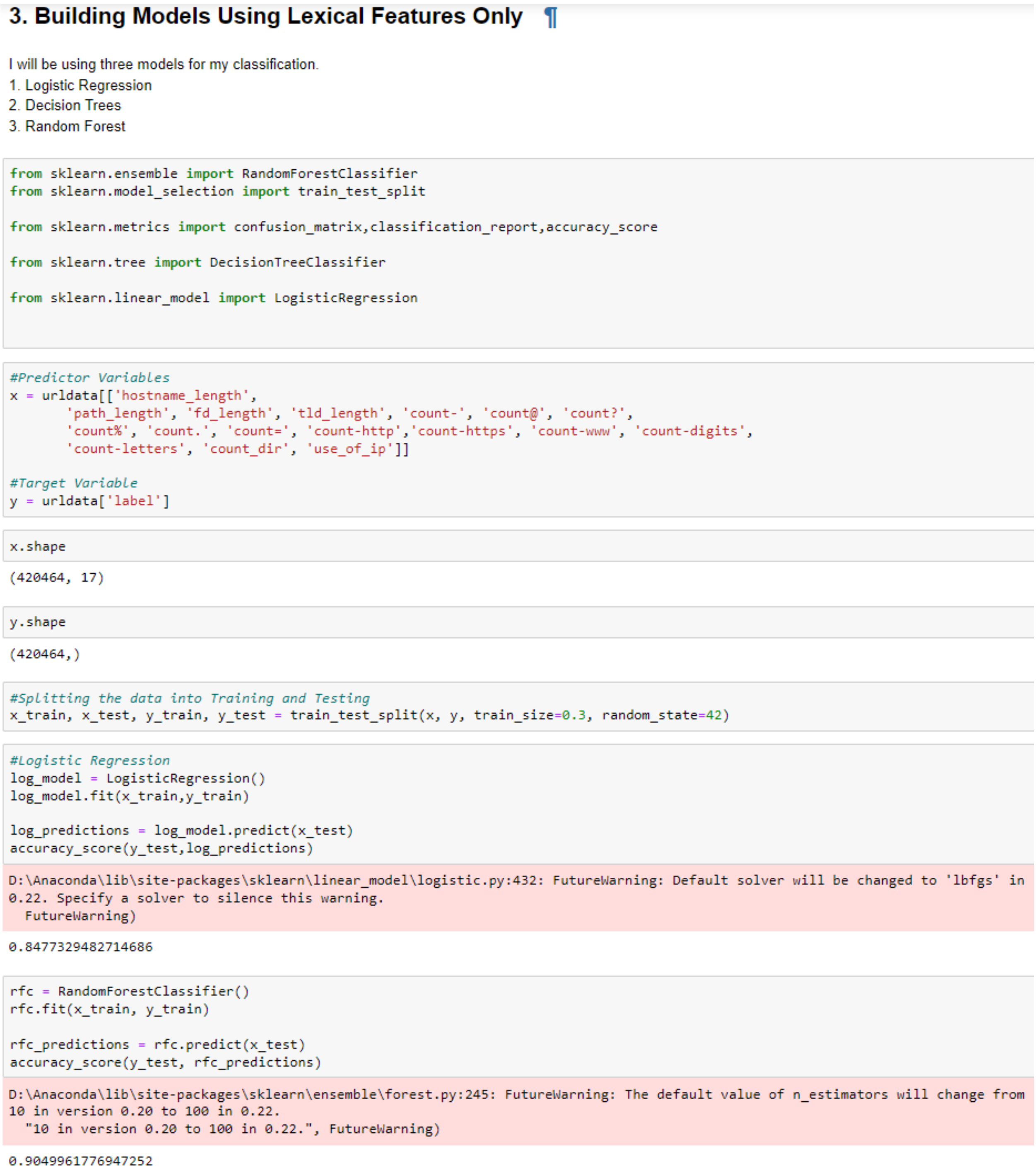
```
def shortening_service(url):
    match = re.search('bit\.ly|goo\.gl|shorte\.st|go2l\.ink|x\.co|ow\.ly|t\.co|tinyurl|tr\.im|is\.gd|cli\.gs|'
                      'yfrog\.com|migre\.me|ff\.im|tiny\.cc|url4\.eu|twit\.ac|su\.pr|twurl\.nl|snipurl\.com|'
                      'short\.to|BudURL\.com|ping\.fm|post\.ly|Just\.as|bkite\.com|snipr\.com|fic\.kr|loopt\.us|'
                      'doiop\.com|short\.ie|kl\.am|wp\.me|rubyurl\.com|om\.ly|to\.ly|bit\.do|t\.co|lnkd\.in|'
                      'db\.tt|qr\.ae|adf\.ly|goo\.gl|bitly\.com|cur\.lv|tinyurl\.com|ow\.ly|bit\.ly|ity\.im|'
                      'q\.gs|is\.gd|po\.st|bc\.vc|twitthis\.com|u\.to|j\.mp|buzurl\.com|cutt\.us|u\.bb|yourls\.org|'
                      'x\.co|prettylinkpro\.com|scrnch\.me|filoops\.info|vzturl\.com|qr\.net|1url\.com|tweez\.me|v\.gd|'
                      'tr\.im|link\.zip\.net',
                      url)
    if match:
        return -1
    else:
        return 1
urldata['short_url'] = urldata['url'].apply(lambda i: shortening_service(i))
```

Data after extracting Binary Features

```
urldata.head()
```

	url	label	url_length	hostname_length	path_length	fd_length	tld_length	count-	count@	count?	...	count.	count=	count-http	count-https
0	https://www.google.com	good	22	14	0	0	3	0	0	0	...	2	0	1	1
1	https://www.youtube.com	good	23	15	0	0	3	0	0	0	...	2	0	1	1
2	https://www.facebook.com	good	24	16	0	0	3	0	0	0	...	2	0	1	1
3	https://www.baidu.com	good	21	13	0	0	3	0	0	0	...	2	0	1	1
4	https://www.wikipedia.org	good	25	17	0	0	3	0	0	0	...	2	0	1	1

5 rows × 21 columns



3. Building Models Using Lexical Features Only ¶

I will be using three models for my classification.
1. Logistic Regression
2. Decision Trees
3. Random Forest

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

from sklearn.metrics import confusion_matrix,classification_report,accuracy_score

from sklearn.tree import DecisionTreeClassifier

from sklearn.linear_model import LogisticRegression
```

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

#Target Variable
y = urldata['label']
```

```
x.shape
```

```
(420464, 17)
```

```
y.shape
```

```
(420464,)
```

```
#Splitting the data into Training and Testing
x_train, x_test, y_train, y_test = train_test_split(x, y, train_size=0.3, random_state=42)
```

```
#Logistic Regression
log_model = LogisticRegression()
log_model.fit(x_train,y_train)

log_predictions = log_model.predict(x_test)
accuracy_score(y_test,log_predictions)
```

```
D:\Anaconda\lib\site-packages\sklearn\linear_model\logistic.py:432: FutureWarning: Default solver will be changed to 'lbfgs' in
0.22. Specify a solver to silence this warning.
  FutureWarning)
```

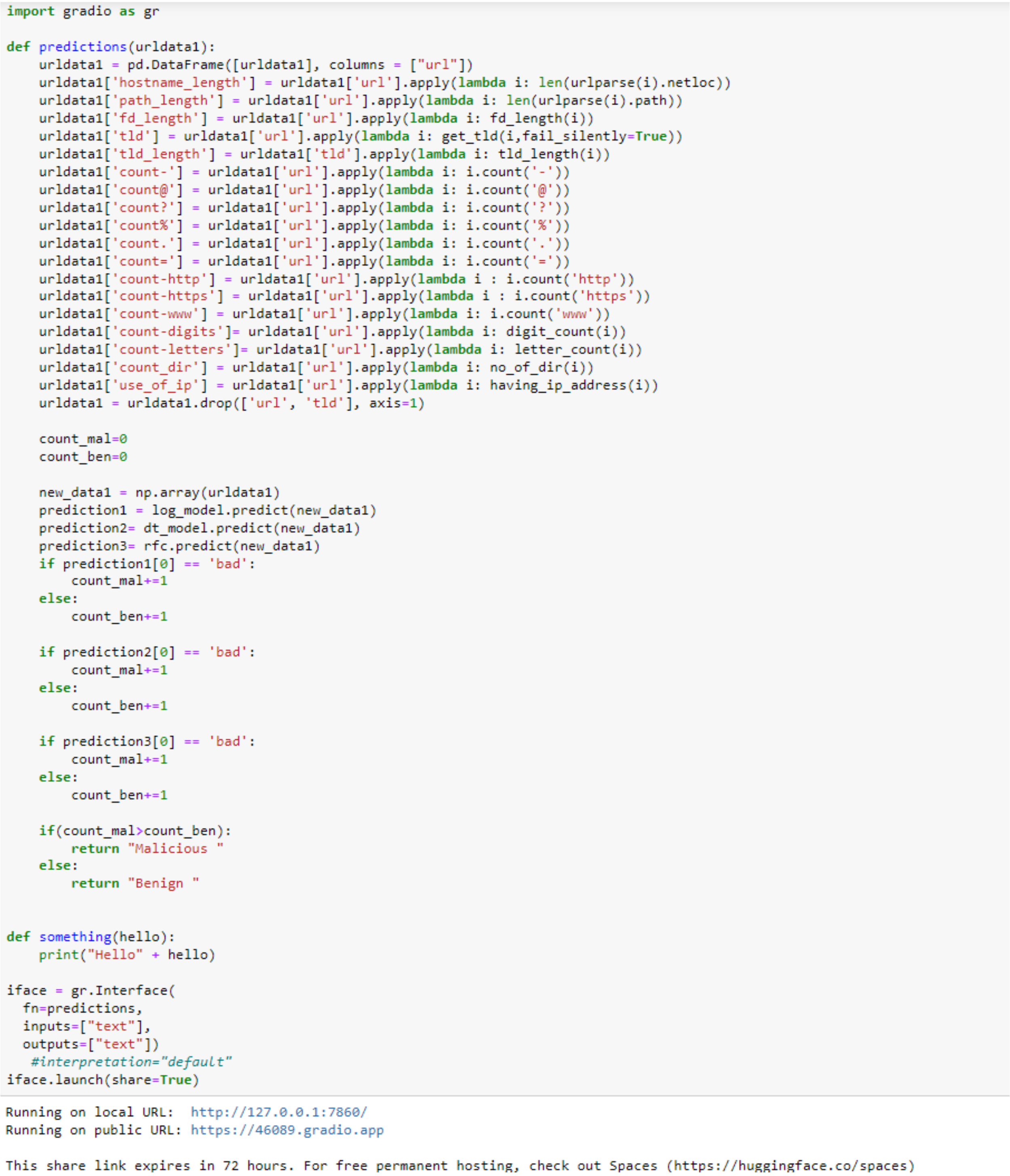
```
0.8477329482714686
```

```
rfc = RandomForestClassifier()
rfc.fit(x_train, y_train)

rfc_predictions = rfc.predict(x_test)
accuracy_score(y_test, rfc_predictions)
```

```
D:\Anaconda\lib\site-packages\sklearn\ensemble\forest.py:245: FutureWarning: The default value of n_estimators will change from
10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

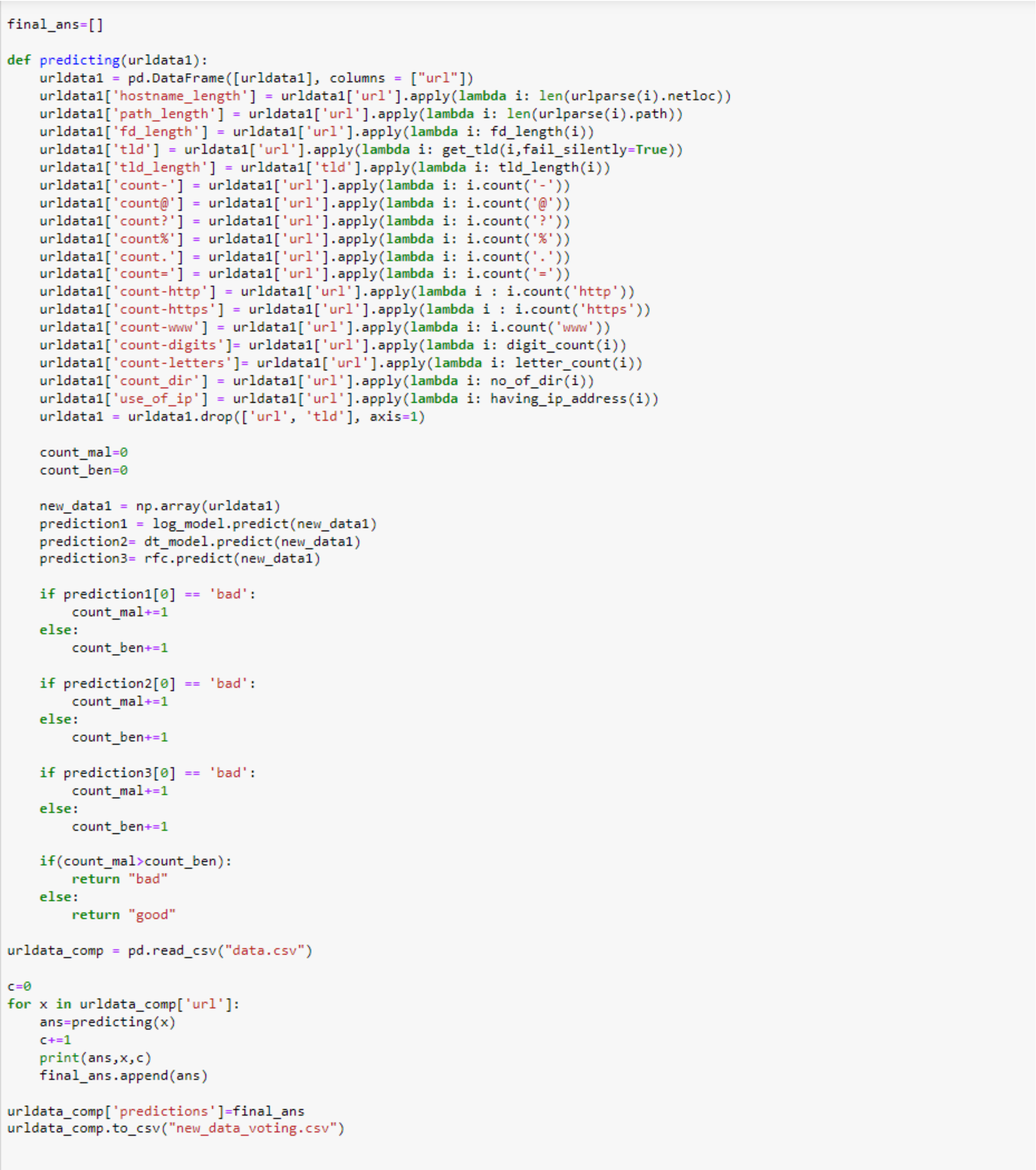
```
0.9049961776947252
```

VOTING SYSTEM CODE



```
import gradio as gr

def predictions(urldata1):
    urldata1 = pd.DataFrame([urldata1], columns = ["url"])
    urldata1['hostname_length'] = urldata1['url'].apply(lambda i: len(urlparse(i).netloc))
    urldata1['path_length'] = urldata1['url'].apply(lambda i: len(urlparse(i).path))
    urldata1['fd_length'] = urldata1['url'].apply(lambda i: fd_length(i))
    urldata1['tld'] = urldata1['url'].apply(lambda i: get_tld(i,fail_silently=True))
    urldata1['tld_length'] = urldata1['tld'].apply(lambda i: tld_length(i))
    urldata1['count-'] = urldata1['url'].apply(lambda i: i.count('-'))
    urldata1['count@'] = urldata1['url'].apply(lambda i: i.count('@'))
    urldata1['count?'] = urldata1['url'].apply(lambda i: i.count('?'))
    urldata1['count%'] = urldata1['url'].apply(lambda i: i.count('%'))
    urldata1['count.'] = urldata1['url'].apply(lambda i: i.count('.'))
    urldata1['count='] = urldata1['url'].apply(lambda i: i.count('='))
    urldata1['count-http'] = urldata1['url'].apply(lambda i : i.count('http'))
    urldata1['count-https'] = urldata1['url'].apply(lambda i : i.count('https'))
    urldata1['count-www'] = urldata1['url'].apply(lambda i: i.count('www'))
    urldata1['count-digits']= urldata1['url'].apply(lambda i: digit_count(i))
    urldata1['count-letters']= urldata1['url'].apply(lambda i: letter_count(i))
    urldata1['count_dir'] = urldata1['url'].apply(lambda i: no_of_dir(i))
    urldata1['use_of_ip'] = urldata1['url'].apply(lambda i: having_ip_address(i))
    urldata1 = urldata1.drop(['url', 'tld'], axis=1)

    count_mal=0
    count_ben=0

    new_data1 = np.array(urldata1)
    prediction1 = log_model.predict(new_data1)
    prediction2= dt_model.predict(new_data1)
    prediction3= rfc.predict(new_data1)
    if prediction1[0] == 'bad':
        count_mal+=1
    else:
        count_ben+=1

    if prediction2[0] == 'bad':
        count_mal+=1
    else:
        count_ben+=1

    if prediction3[0] == 'bad':
        count_mal+=1
    else:
        count_ben+=1

    if(count_mal>count_ben):
        return "Malicious "
    else:
        return "Benign "


def something(hello):
    print("Hello" + hello)

iface = gr.Interface(
  fn=predictions,
  inputs=["text"],
  outputs=["text"])
   #interpretation="default"
iface.launch(share=True)
```

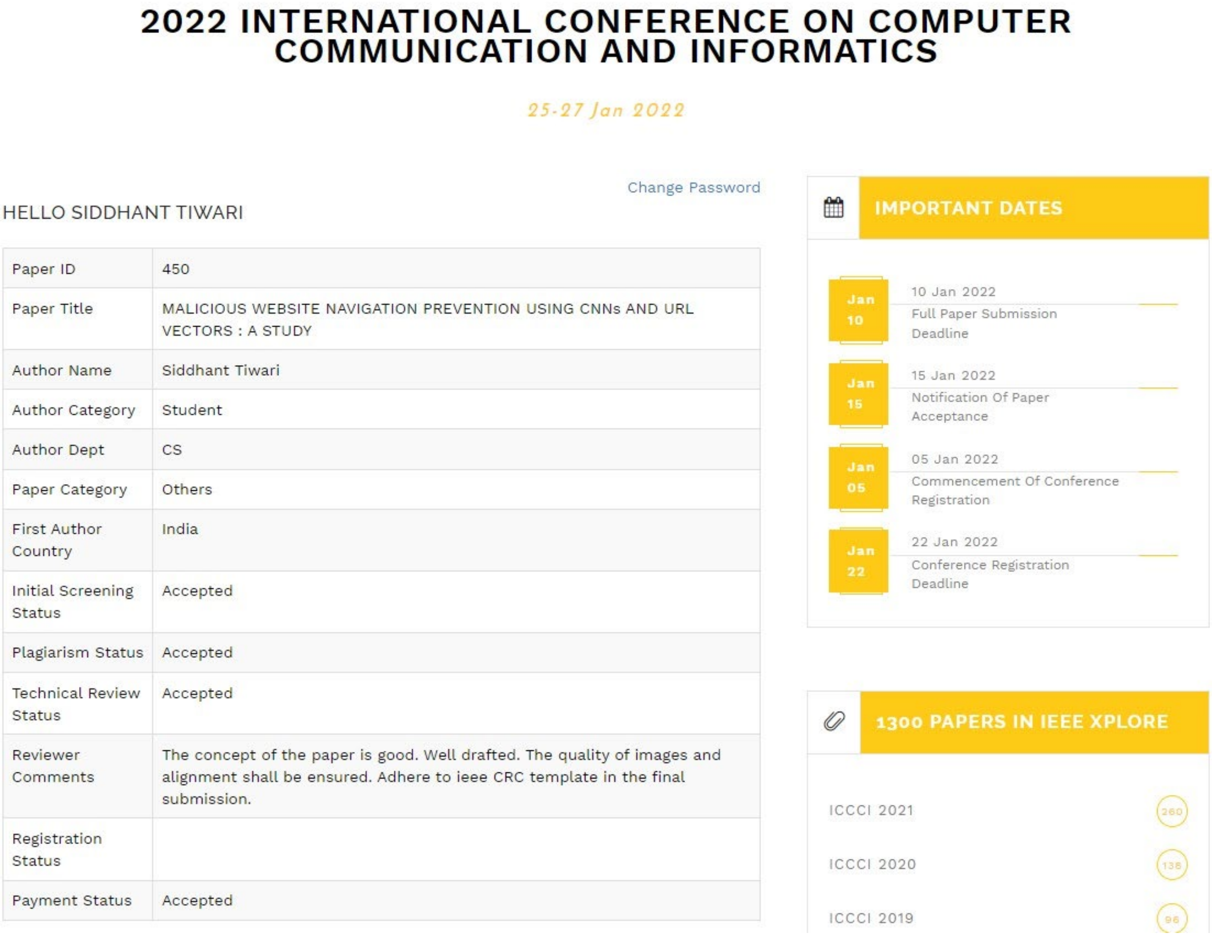
```
Running on local URL:  http://127.0.0.1:7860/
Running on public URL: https://46089.gradio.app

This share link expires in 72 hours. For free permanent hosting, check out Spaces (https://huggingface.co/spaces)
```



```
final_ans=[]

def predicting(urldata1):
    urldata1 = pd.DataFrame([urldata1], columns = ["url"])
    urldata1['hostname_length'] = urldata1['url'].apply(lambda i: len(urlparse(i).netloc))
    urldata1['path_length'] = urldata1['url'].apply(lambda i: len(urlparse(i).path))
    urldata1['fd_length'] = urldata1['url'].apply(lambda i: fd_length(i))
    urldata1['tld'] = urldata1['url'].apply(lambda i: get_tld(i,fail_silently=True))
    urldata1['tld_length'] = urldata1['tld'].apply(lambda i: tld_length(i))
    urldata1['count-'] = urldata1['url'].apply(lambda i: i.count('-'))
    urldata1['count@'] = urldata1['url'].apply(lambda i: i.count('@'))
    urldata1['count?'] = urldata1['url'].apply(lambda i: i.count('?'))
    urldata1['count%'] = urldata1['url'].apply(lambda i: i.count('%'))
    urldata1['count.'] = urldata1['url'].apply(lambda i: i.count('.'))
    urldata1['count='] = urldata1['url'].apply(lambda i: i.count('='))
    urldata1['count-http'] = urldata1['url'].apply(lambda i : i.count('http'))
    urldata1['count-https'] = urldata1['url'].apply(lambda i : i.count('https'))
    urldata1['count-www'] = urldata1['url'].apply(lambda i: i.count('www'))
    urldata1['count-digits']= urldata1['url'].apply(lambda i: digit_count(i))
    urldata1['count-letters']= urldata1['url'].apply(lambda i: letter_count(i))
    urldata1['count_dir'] = urldata1['url'].apply(lambda i: no_of_dir(i))
    urldata1['use_of_ip'] = urldata1['url'].apply(lambda i: having_ip_address(i))
    urldata1 = urldata1.drop(['url', 'tld'], axis=1)

    count_mal=0
    count_ben=0

    new_data1 = np.array(urldata1)
    prediction1 = log_model.predict(new_data1)
    prediction2= dt_model.predict(new_data1)
    prediction3= rfc.predict(new_data1)

    if prediction1[0] == 'bad':
        count_mal+=1
    else:
        count_ben+=1

    if prediction2[0] == 'bad':
        count_mal+=1
    else:
        count_ben+=1

    if prediction3[0] == 'bad':
        count_mal+=1
    else:
        count_ben+=1

    if(count_mal>count_ben):
        return "bad"
    else:
        return "good"

urldata_comp = pd.read_csv("data.csv")

c=0
for x in urldata_comp['url']:
    ans=predicting(x)
    c+=1
    print(ans,x,c)
    final_ans.append(ans)

urldata_comp['predictions']=final_ans
urldata_comp.to_csv("new_data_voting.csv")
```

PROOF OF SUBMISSION

ICCCI Jan 2022 Submission



2022 INTERNATIONAL CONFERENCE ON COMPUTER COMMUNICATION AND INFORMATICS

25-27 Jan 2022

Change Password

HELLO SIDDHANT TIWARI

Paper ID	450
Paper Title	MALICIOUS WEBSITE NAVIGATION PREVENTION USING CNNs AND URL VECTORS : A STUDY
Author Name	Siddhant Tiwari
Author Category	Student
Author Dept	CS
Paper Category	Others
First Author Country	India
Initial Screening Status	Accepted
Plagiarism Status	Accepted
Technical Review Status	Accepted
Reviewer Comments	The concept of the paper is good. Well drafted. The quality of images and alignment shall be ensured. Adhere to ieee CRC template in the final submission.
Registration Status	
Payment Status	Accepted

IMPORTANT DATES

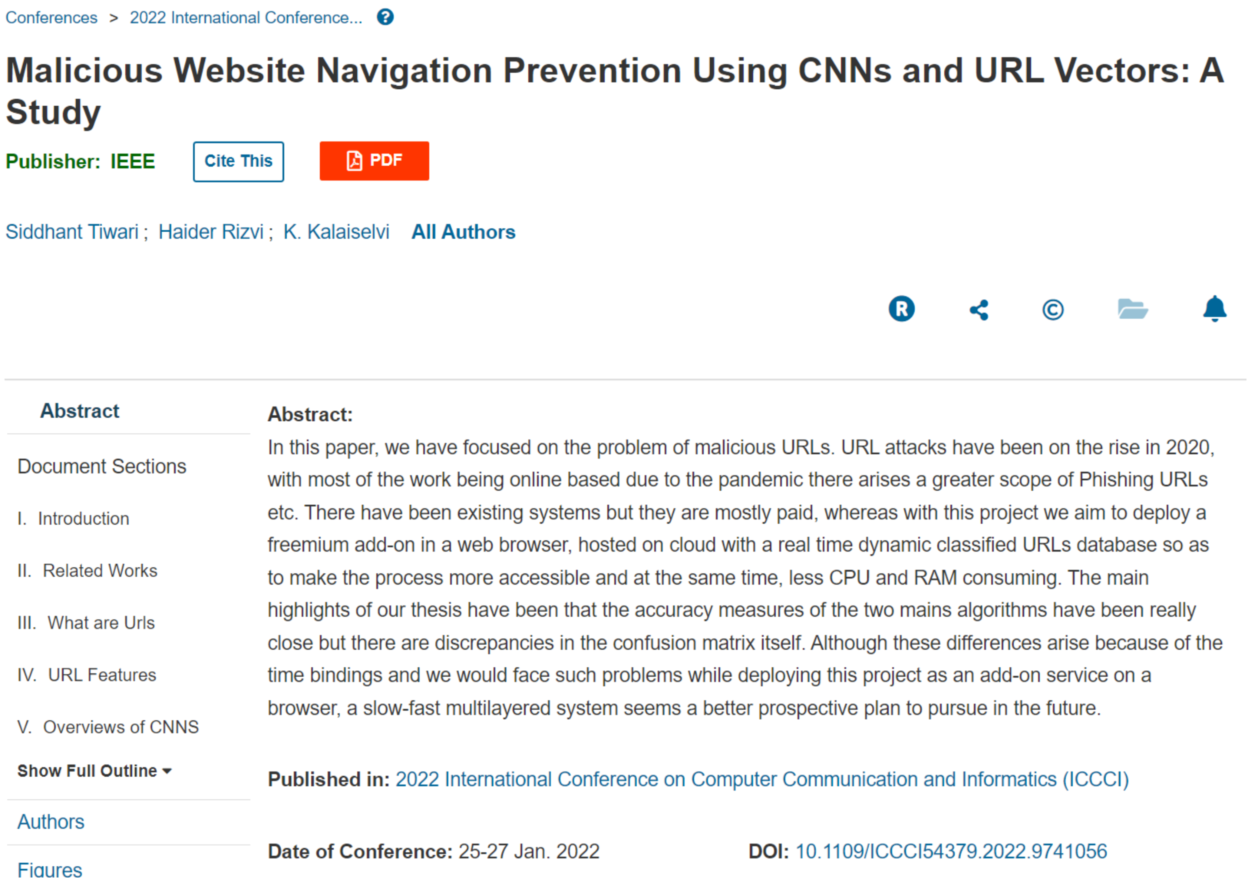
Jan 10	10 Jan 2022 Full Paper Submission Deadline
Jan 15	15 Jan 2022 Notification Of Paper Acceptance
Jan 05	05 Jan 2022 Commencement Of Conference Registration
Jan 22	22 Jan 2022 Conference Registration Deadline

1300 PAPERS IN IEEE XPLORE

ICCCI 2021	260
ICCCI 2020	136
ICCCI 2019	96

LIST OF PUBLICATIONS

ICCCI Jan 2022 Publication



Malicious Website Navigation Prevention Using CNNs and URL Vectors: A Study

Publisher: IEEE

Siddhant Tiwari ; Haider Rizvi ; K. Kalaiselvi **All Authors**

Abstract:

In this paper, we have focused on the problem of malicious URLs. URL attacks have been on the rise in 2020, with most of the work being online based due to the pandemic there arises a greater scope of Phishing URLs etc. There have been existing systems but they are mostly paid, whereas with this project we aim to deploy a freemium add-on in a web browser, hosted on cloud with a real time dynamic classified URLs database so as to make the process more accessible and at the same time, less CPU and RAM consuming. The main highlights of our thesis have been that the accuracy measures of the two mains algorithms have been really close but there are discrepancies in the confusion matrix itself. Although these differences arise because of the time bindings and we would face such problems while deploying this project as an add-on service on a browser, a slow-fast multilayered system seems a better prospective plan to pursue in the future.

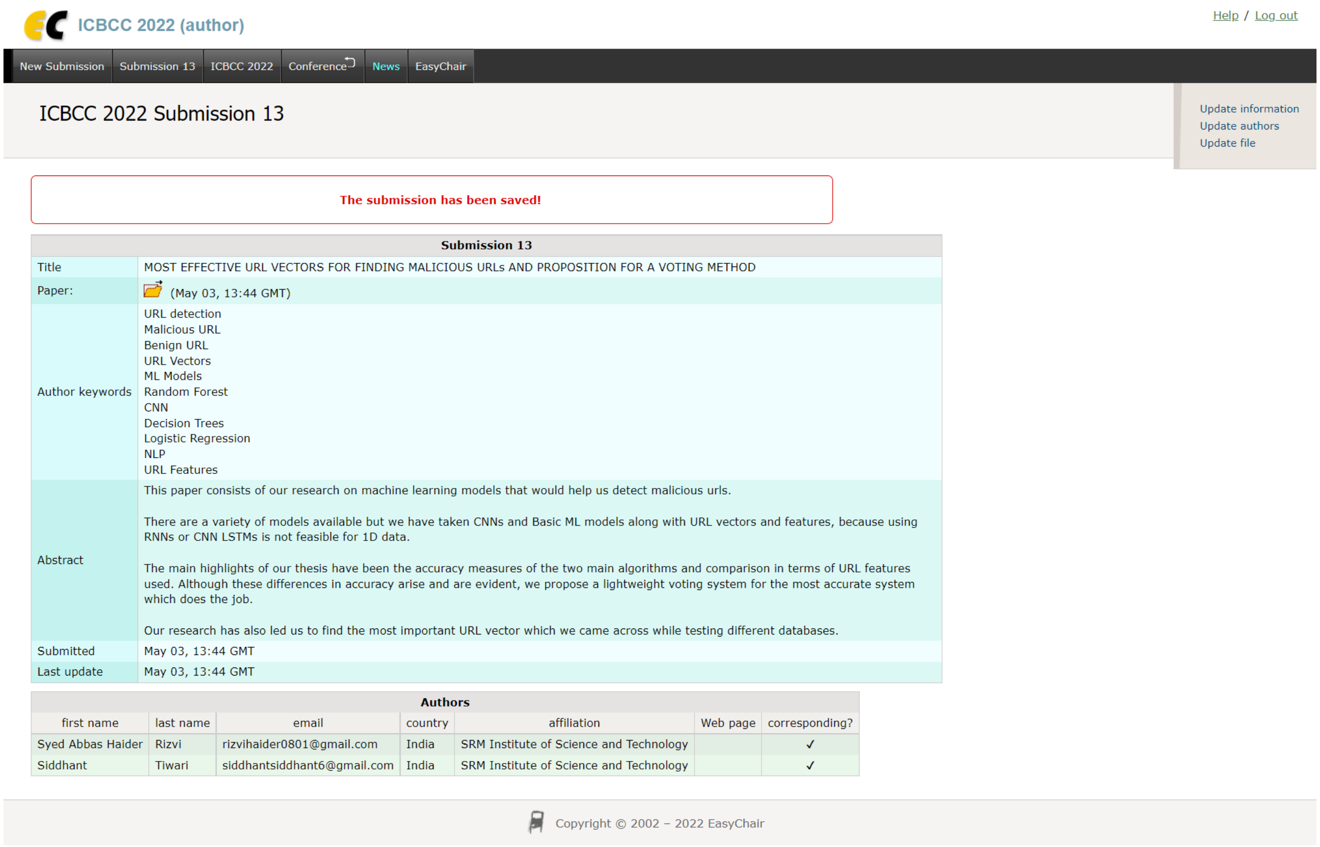
Published in: 2022 International Conference on Computer Communication and Informatics (ICCCI)

Date of Conference: 25-27 Jan. 2022

DOI: 10.1109/ICCCI54379.2022.9741056

Paper Publishment Link: https://ieeexplore.ieee.org/document/9741056

ICBCC May 2022 Submission



ICBCC 2022 Submission 13

The submission has been saved!

Submission 13	
Title	MOST EFFECTIVE URL VECTORS FOR FINDING MALICIOUS URLs AND PROPOSITION FOR A VOTING METHOD
Paper:	(May 03, 13:44 GMT)
Author keywords	URL detection Malicious URL Benign URL URL Vectors ML Models Random Forest CNN Decision Trees Logistic Regression NLP URL Features
Abstract	This paper consists of our research on machine learning models that would help us detect malicious urls. There are a variety of models available but we have taken CNNs and Basic ML models along with URL vectors and features, because using RNNs or CNN LSTMs is not feasible for 1D data. The main highlights of our thesis have been the accuracy measures of the two main algorithms and comparison in terms of URL features used. Although these differences in accuracy arise and are evident, we propose a lightweight voting system for the most accurate system which does the job. Our research has also led us to find the most important URL vector which we came across while testing different databases.
Submitted	May 03, 13:44 GMT
Last update	May 03, 13:44 GMT

Authors

first name	last name	email	country	affiliation	Web page	corresponding?
Syed Abbas Haider	Rizvi	rizvihaider0801@gmail.com	India	SRM Institute of Science and Technology		✓
Siddhant	Tiwari	siddhantsiddhant6@gmail.com	India	SRM Institute of Science and Technology		✓

Copyright © 2002 – 2022 EasyChair

Format - I

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Deemed to be University u/s 3 ofUGC Act, 1956)

Office of Controller of Examinations

REPORT FOR PLAGIARISM CHECK ON THE DISSERTATION/PROJECT REPORTS FOR UG/PG PROGRAMMES

(To be attached in the dissertation/ project report)

1	Name of the Candidate **(IN BLOCK LETTERS)**	SYED ABBAS HAIDER RIZVI
2	Address of the Candidate	A1/601, Sakshi Pride Paradise, Nakshatrawadi, Aurangabad, Maharashtra - 431005 **Mobile Number :** +91 - 95950 32334
3	Registration Number	RA1811030010082
4	Date of Birth	08/01/2000
5	Department	COMPUTER SCIENCE AND ENGINEERING with specialization in CYBER SECURITY
6	Faculty	Networking and Communications
7	Title of the Dissertation/Project	MOST EFFECTIVE URL VECTORS AND METHODS FOR FINDING MALICIOUS URLS: A RESEARCH
8	Whether the above project/dissertation is done by	~~Individual~~ or group : (Strike whichever is not applicable) a) If the project/ dissertation is done in group, then how many students together completed the project: 2 b) Mention the Name & Register number of other candidates : Siddhant Tiwari RA1811030010066
9	Name and address of the Supervisor / Guide	Dr K. Kalaiselvi, Department of Networking and Communications SRM Institute of Science and Technology **Mail ID :** kalaisek2@srmist.edu.in **Mobile Number :** +91 - 99625 72473
10	Name and address of the Co-Supervisor / Co- Guide (if any)	

11	Software Used	Turnitin		
12	Date of Verification	04/05/2022		
13	**Plagiarism Details: (to attach the final report from the software)**			
Chapter	**Title of the Chapter**	**Percentage of similarity index (including self citation)**	**Percentage of similarity index (Excluding self citation)**	**% of plagiarism after excluding Quotes, Bibliography, etc.,**
1	**INTRODUCTION**	**1%**	**<1%**	**0%**
2	**LITERATURE SURVEY**	**1%**	**1%**	**0%**
3	**UML DIAGRAMS**	**0%**	**0%**	**0%**
4	**IMPLEMENTATION**	**2%**	**1%**	**1%**
5	**RESULTS AND DISCUSSIONS**	**2%**	**0%**	**0%**
6	**CONCLUSION**	**1%**	**0%**	**0%**
7	**REFERENCES**	**0%**	**1%**	**0%**
8				
9				
10				
Appendices		**1%**	**0%**	**0%**
I / We declare that the above information have been verified and found true to the best of my / our knowledge.				
Signature of the Candidate		**Name & Signature of the Staff (Who uses the plagiarism check software}**		
Name & Signature of the Sunervisor/Guide		**Name & Signature of the Co-Supervisor/Co-Guide**		
Name & Signature of the HOD				

Project_check_plag.docx

ORIGINALITY REPORT

10%	5%	5%	2%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Siddhant Tiwari, Haider Rizvi, K. Kalaiselvi. "Malicious Website Navigation Prevention Using CNNs and URL Vectors: A Study", 2022 International Conference on Computer Communication and Informatics (ICCCI), 2022 Publication	4%
2	www.ibm.com Internet Source	2%
3	Submitted to University of Liverpool Student Paper	1%
4	www.business-standard.com Internet Source	1%
5	uniaobahia.org Internet Source	1%
6	"Malware Analysis Using Artificial Intelligence and Deep Learning", Springer Science and Business Media LLC, 2021 Publication	1%
7	thesai.org Internet Source	<1%

8	www.coursehero.com Internet Source	<1 %
9	ijsr.net Internet Source	<1 %

Exclude quotes	On	Exclude matches	< 10 words
Exclude bibliography	On		