Real Time Filtering of Malicious URLs

Ishwarya S - 2017103537
Harshini T - 2017103532
Gokul K - 2017103526

Table of Contents

1. Introduction

The rapid development of communication technologies and broadband speeds has greatly democratized the internet. The number of internet users has grown exponentially in the past few years. This has caused a sharp increase in the global internet traffic. This has also opened up several avenues for online fraud. There are several malicious URLs on the internet that are used to scam and cheat people. In many cases privacy of the person is violated and this can also lead to money thefts. Therefore it is of great importance to filter the malicious URLs on the internet. But this process is a bottleneck because of the high internet traffic. Hence it is essential to develop a lightweight filtering system to filter out the harmful URLs.

2. Abstract

In this project we aim to develop a robust and scalable classifier that can perform URL filtering in real-time using lightweight features so that this reduces the processing pressure of the back-end malicious URL detection systems based on content analysis. We will be exploring 3 different algorithms for this classifier and provide a comparative study on the performance of the algorithms using Precision, Recall and Accuracy metrics. The different URL features like URL Length, number of special characters in the URL, origin server, origin country, charset, content length, registration date of the website, last modified date have been used for the classification process. This will be performed as a binary classification problem.

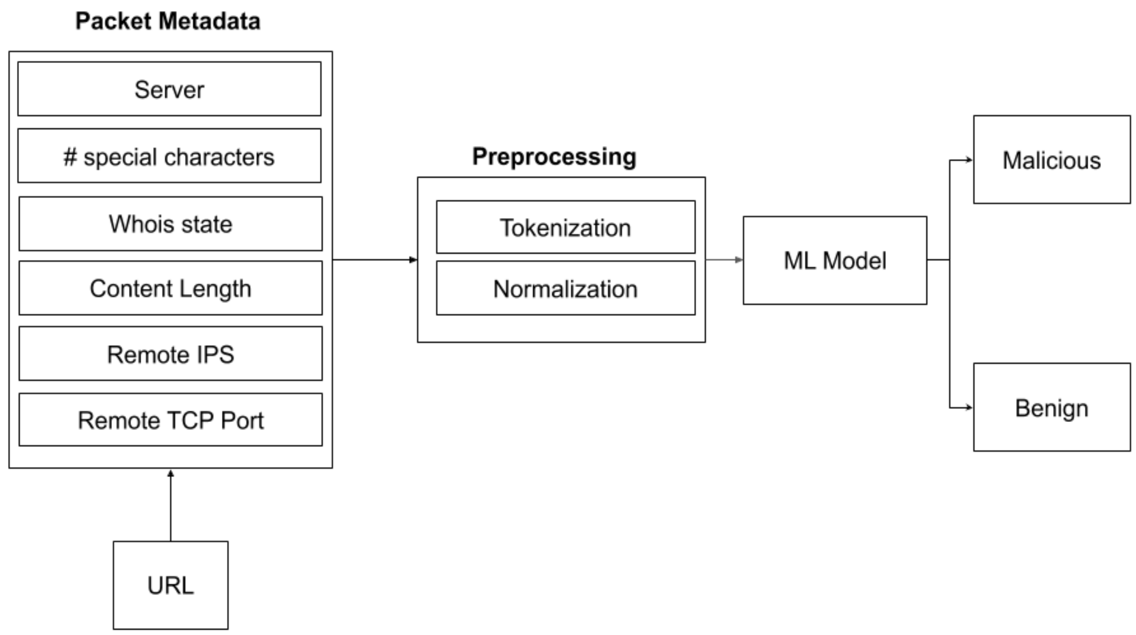
3. Objective

The main aim of the project is to build a lightweight filtering system to filter out the malicious URLs by using Machine Learning Algorithms and methodologies. This process should be lightweight so that the browsing experience of the user is affected.

4. Dataset

The Malicious and Benign websites dataset from Kaggle will be used for this project.

5. System Architecture



6. Algorithms Used

Three Machine Learning Algorithms have been used for this project

1. Multi-Layer Perceptron
2. Random Forest Classifier
3. Support Vector Machine

7. Language and Frameworks

The project was built using the Python programming language. Frameworks like Scikit-Learn, Tensor flow, Keras, Numpy and Pandas were used for this project.

8. Modules

8.1 Pre-processing

8.1.1 Feature Selection

There are a total of 20 features in this dataset. This will cause the overfitting problem and the curse of dimensionality problem. Hence to solve this, we did a feature selection and selected 6 features from the total 20 features. These 6 features were selected based on their covariance value with the output label.

```
#selecting particular columns
df_part=
df[['NUMBER_SPECIAL_CHARACTERS','SERVER','CONTENT_LENGTH','WHOIS_STATEPRO','DIST_REMOTE_TCP_PORT','REMOTE_IPS','Type' ]]
```

8.1.2 Tokenization

This is the process of converting the strings to numeric tokens.

```
#mapping the strings to integers or tokens
server_names = df.SERVER.unique()
server_names_map = {k:v for v,k in enumerate(server_names)}
```

8.1.3 Normalization

Since the range of the numeric data is very large, the variance increases and this affects the training process. To reduce variance, Normalization is done.

```
making the NaN values as 0
df_part['CONTENT_LENGTH'] = df_part['CONTENT_LENGTH'].fillna(0)
#making the content_length column as type int
df_part['CONTENT_LENGTH'] = df_part['CONTENT_LENGTH'].astype('int')
```

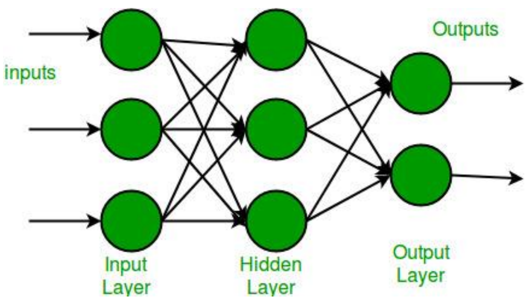
```
from sklearn.model_selection import train_test_split
#splitting into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

```
from sklearn.preprocessing import StandardScaler
#scaling the values in X_train
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

8.2 Building Machine Learning Models and Testing

8.2.1 Multi-Layer Perceptron

A Multi-Layer Perceptron (MLP) or Multi-Layer Neural Network contains one or more hidden layers (apart from one input and one output layer). While a single layer perceptron can only learn linear functions, a multi-layer perceptron can also learn non – linear functions.



8.2.1.1 ALGORITHM

```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(10, activation='relu'))
model.add(tf.keras.layers.Dense(16, activation='relu'))
model.add(tf.keras.layers.Dense(32, activation='relu'))
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
```

```
adam_optimizer = tf.keras.optimizers.Adam(learning_rate=0.01)
model.compile(optimizer=adam_optimizer, loss='binary_crossentropy', metrics=['accuracy'])
```

```
history = model.fit(X_train, y_train, batch_size=32, epochs=30, verbose=1,
validation_split=0.1, shuffle=True)
```

8.2.1.2 TRAINING

```
Epoch 1/30
41/41 [==============================] - 0s 5ms/step - loss: 0.1526 - accuracy: 0.9258 - val_loss: 0.1826 - val_accuracy: 0.9301
Epoch 2/30
41/41 [==============================] - 0s 2ms/step - loss: 0.1255 - accuracy: 0.9438 - val_loss: 0.1675 - val_accuracy: 0.9231
Epoch 3/30
41/41 [==============================] - 0s 2ms/step - loss: 0.1285 - accuracy: 0.9399 - val_loss: 0.1877 - val_accuracy: 0.9021
Epoch 4/30
41/41 [==============================] - 0s 2ms/step - loss: 0.1258 - accuracy: 0.9485 - val_loss: 0.1650 - val_accuracy: 0.9301
Epoch 5/30
41/41 [==============================] - 0s 2ms/step - loss: 0.1392 - accuracy: 0.9399 - val_loss: 0.1656 - val_accuracy: 0.9371
Epoch 6/30
41/41 [==============================] - 0s 2ms/step - loss: 0.1216 - accuracy: 0.9454 - val_loss: 0.1766 - val_accuracy: 0.9371
Epoch 7/30
41/41 [==============================] - 0s 2ms/step - loss: 0.1218 - accuracy: 0.9485 - val_loss: 0.1792 - val_accuracy: 0.9021
Epoch 8/30
41/41 [==============================] - 0s 2ms/step - loss: 0.1172 - accuracy: 0.9485 - val_loss: 0.1880 - val_accuracy: 0.9091
Epoch 9/30
41/41 [==============================] - 0s 2ms/step - loss: 0.1237 - accuracy: 0.9407 - val_loss: 0.1596 - val_accuracy: 0.9301
Epoch 10/30
41/41 [==============================] - 0s 2ms/step - loss: 0.1152 - accuracy: 0.9461 - val_loss: 0.1724 - val_accuracy: 0.9231
Epoch 11/30
41/41 [==============================] - 0s 2ms/step - loss: 0.1149 - accuracy: 0.9461 - val_loss: 0.1871 - val_accuracy: 0.9021
Epoch 12/30
41/41 [==============================] - 0s 2ms/step - loss: 0.1156 - accuracy: 0.9477 - val_loss: 0.1625 - val_accuracy: 0.9161
Epoch 13/30
41/41 [==============================] - 0s 2ms/step - loss: 0.1117 - accuracy: 0.9500 - val_loss: 0.1698 - val_accuracy: 0.9231
Epoch 14/30
41/41 [==============================] - 0s 2ms/step - loss: 0.1152 - accuracy: 0.9493 - val_loss: 0.1794 - val_accuracy: 0.8951
Epoch 15/30
41/41 [==============================] - 0s 2ms/step - loss: 0.1158 - accuracy: 0.9508 - val_loss: 0.2255 - val_accuracy: 0.9021
Epoch 16/30
41/41 [==============================] - 0s 2ms/step - loss: 0.1279 - accuracy: 0.9438 - val_loss: 0.1673 - val_accuracy: 0.9231
Epoch 17/30
41/41 [==============================] - 0s 2ms/step - loss: 0.1187 - accuracy: 0.9477 - val_loss: 0.1905 - val_accuracy: 0.9021
Epoch 18/30
41/41 [==============================] - 0s 2ms/step - loss: 0.1227 - accuracy: 0.9485 - val_loss: 0.1805 - val_accuracy: 0.9231
Epoch 19/30
41/41 [==============================] - 0s 2ms/step - loss: 0.1190 - accuracy: 0.9438 - val_loss: 0.1504 - val_accuracy: 0.9231
Epoch 20/30
41/41 [==============================] - 0s 2ms/step - loss: 0.1300 - accuracy: 0.9415 - val_loss: 0.1751 - val_accuracy: 0.9161
Epoch 21/30
41/41 [==============================] - 0s 2ms/step - loss: 0.1130 - accuracy: 0.9477 - val_loss: 0.1474 - val_accuracy: 0.9091
Epoch 22/30
41/41 [==============================] - 0s 2ms/step - loss: 0.1084 - accuracy: 0.9493 - val_loss: 0.1924 - val_accuracy: 0.9161
Epoch 23/30
41/41 [==============================] - 0s 2ms/step - loss: 0.1090 - accuracy: 0.9563 - val_loss: 0.1662 - val_accuracy: 0.9161
Epoch 24/30
41/41 [==============================] - 0s 2ms/step - loss: 0.1074 - accuracy: 0.9524 - val_loss: 0.1905 - val_accuracy: 0.9161
Epoch 25/30
41/41 [==============================] - 0s 2ms/step - loss: 0.1036 - accuracy: 0.9563 - val_loss: 0.1650 - val_accuracy: 0.9231
Epoch 26/30
41/41 [==============================] - 0s 2ms/step - loss: 0.1080 - accuracy: 0.9477 - val_loss: 0.1675 - val_accuracy: 0.9161
Epoch 27/30
41/41 [==============================] - 0s 2ms/step - loss: 0.1048 - accuracy: 0.9508 - val_loss: 0.1938 - val_accuracy: 0.9091
Epoch 28/30
41/41 [==============================] - 0s 2ms/step - loss: 0.1024 - accuracy: 0.9485 - val_loss: 0.1880 - val_accuracy: 0.9091
Epoch 29/30
41/41 [==============================] - 0s 2ms/step - loss: 0.1165 - accuracy: 0.9500 - val_loss: 0.2058 - val_accuracy: 0.9231
Epoch 30/30
41/41 [==============================] - 0s 2ms/step - loss: 0.1202 - accuracy: 0.9469 - val_loss: 0.1857 - val_accuracy: 0.9091
```

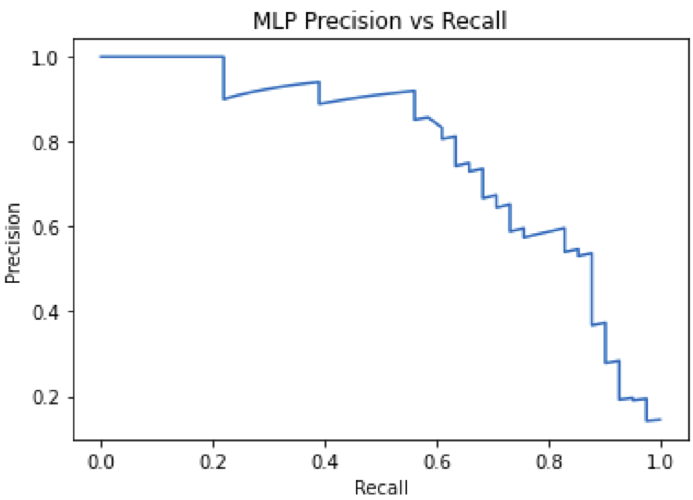
8.2.1.3 RESULT

Accuracy: 93%

class	precision	recall	F1 - Score
0	0.96	0.97	0.96
1	0.73	0.66	0.69

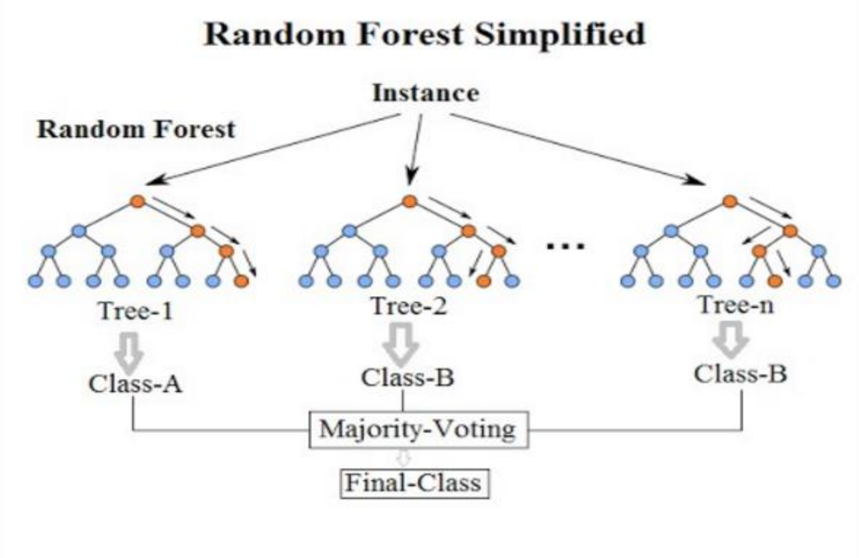
```
[30] y_pred_mlp = model.predict_classes(X_test)

[31] print(accuracy_score(y_test, y_pred_mlp))

     0.9327731092436975

[32] print(classification_report(y_test, y_pred_mlp))

                   precision    recall  f1-score   support

               0       0.96      0.97      0.96       316
               1       0.73      0.66      0.69        41

        accuracy                           0.93       357
       macro avg       0.84      0.81      0.83       357
    weighted avg       0.93      0.93      0.93       357
```

8.2.1.4 PRECISION-RECALL GRAPH



8.2.2 Random Forest

Random forest is a supervised learning algorithm which is used for both classification as well as regression. It creates decision trees on data samples and then gets the prediction from each of them and finally selects the best solution by means of voting. It is an ensemble method which is better than a single decision tree because it reduces the over-fitting by averaging the result.



8.2.2.1 ALGORITHM

```
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators=30, random_state=42)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))
print(accuracy_score(y_test, y_pred))
```

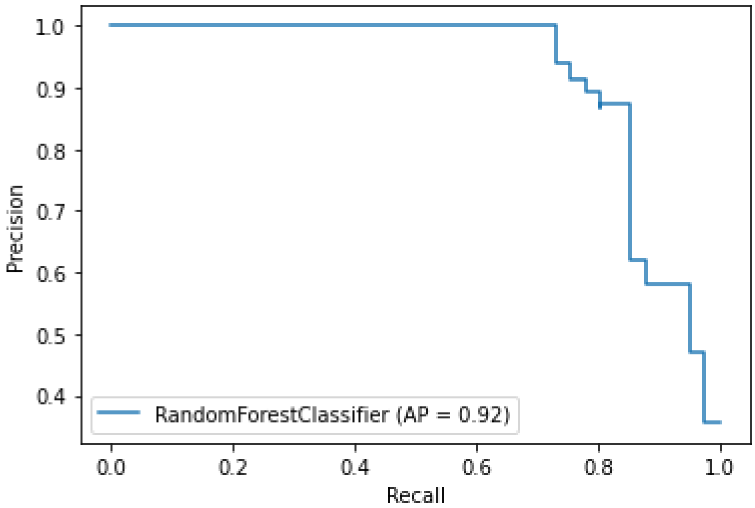
8.2.2.2 RESULT

Accuracy: 96.6%

Class	Precision	Recall	F1 - Score
0	0.97	0.99	0.98
1	0.89	0.80	0.85

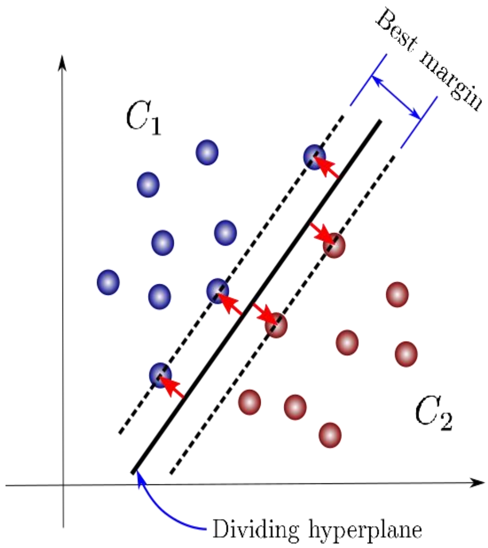
```
[[312   4]
 [  8  33]]
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       316
           1       0.89      0.80      0.85        41

    accuracy                           0.97       357
   macro avg       0.93      0.90      0.91       357
weighted avg       0.97      0.97      0.97       357

0.9663865546218487
```

8.2.2.3 PRECISION-RECALL GRAPH



8.2.3 Support Vector Machine

Support Vector Machine is a supervised learning algorithm which is used for both classification as well as regression. Basically, SVM finds a hyper-plane that creates a boundary between the types of data. In 2-dimensional space, this hyper-plane is nothing but a line. In SVM, we plot each data item in the dataset in an N-dimensional space, where N is the number of features/attributes in the data. Next, find the optimal hyperplane to separate the data. SVM can only perform binary classification (i.e., choose between two classes).



8.2.3.1 ALGORITHM

```
clf = svm.SVC(kernel='rbf')
clf.fit(X_train, y_train)
```

8.2.3.2 PARAMETERS

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

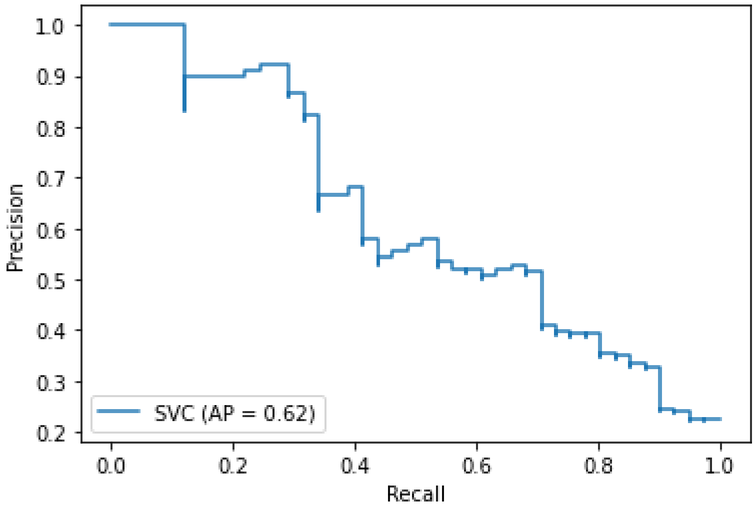
8.2.3.3 RESULT

Accuracy: 91.6%

Class	Precision	Recall	F1 - Score
0	0.92	1	0.95
1	0.92	0.29	0.44

```
[38] y_pred_svm = clf.predict(X_test)

[39] print(accuracy_score(y_test, y_pred_svm))

     0.9159663865546218

[40] print(classification_report(y_test, y_pred_svm))

                  precision    recall  f1-score   support

               0       0.92      1.00      0.95       316
               1       0.92      0.29      0.44        41

        accuracy                           0.92       357
       macro avg       0.92      0.64      0.70       357
    weighted avg       0.92      0.92      0.90       357
```

8.2.3.4 PRECISION-RECALL GRAPH



8.2.4 Final Comparison Results

Algorithm	Accuracy	Precision
Multi-Layer Perceptron	92.4%	96%
Random Forest	96.6%	97%
Support Vector Machine	91.6%	92%

8.3 Real time filtering output

8.3.1.1 Code for processing the input links

```
def pred_link(server = "None", whois_state="None",  \
          nos_spl_chars = 0, content_len = 0, \
          remote_ips = 0, remote_tcp=0):
  server_token = server_names_map[server]
  whois_state_token = state_names_map[whois_state]
  inp_arr = [[nos_spl_chars, server_token, content_len,\
          whois_state_token, remote_tcp, remote_ips]]
  inp_arr = sc.transform(inp_arr)
  result = classifier.predict(inp_arr)

  if result == 0:
    return "The link is not safe"
  return "The link is safe"
```

9. Outputs

```
#actual label 0
pred_link("Microsoft-HTTPAPI/2.0","Arizona", 10, 324, 2,13)
```

'The link is not safe'

```
#actual label 0
pred_link("nginx","PANAMA", 11, 0, 14,46)
```

'The link is not safe'

```
#actual label 1
pred_link("Apache/2.2.14 (FreeBSD) mod_ssl/2.2.14 OpenSSL/0.9.8y DAV/2 PHP/5.2.12 with Suhosin-Patch","Utah", 10, 2516, 2,
```

'The link is safe'

```
#actual label 1
pred_link("nginx", "Novosibirskaya obl.",7,686,2,0 )
```

'The link is safe'

```
#actual label 1
pred_link("nginx/1.10.1", "None", 5, 0, 0,0)
```

'The link is safe'

```
#actual label 0
pred_link("None", "None", 7, 13716, 8, 6)
```

'The link is not safe'

10. Conclusion

Thus through this project we conclude that the Random Forest Classifier is best suited for the real-time filtering of the malicious URLs. It is fast and also accurate compared to the MLP and SVM models.

11. References

Research paper:
https://ieeexplore.ieee.org/document/8672274

Dataset:
https://www.kaggle.com/xwolf12/malicious-and-benign-websites

Random Forest:
https://www.tutorialspoint.com/machine_learning_with_python/machine_learning_with python_classification_algorithms_random_forest.htm

Multi-Layer Perceptron:
https://medium.com/@AI_with_Kain/understanding-of-multilayer-perceptron-mlp-8f179c4a135f

Support Vector Machine:
https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

Model:
https://towardsdatascience.com/machine-learning-general-process-8f1b510bd8af