Modelling Enhanced Phishing detection using XGBoost

[Data Security and Privacy]

MSc Internship

Cybersecurity

Nishant Nityanand Naik

19138342

School of Computing

National college of Ireland

Supervisor- Ross Spelman

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Nishant Nityanand Naik
Student ID:	19138342
Programme:	MSc CyberSecurity
Year:	2020-2021
Module:	MSc Internship
Supervisor:	Ross Spelman
Submission Due Date:	17-08-2020
Project Title:	Modelling Enhanced Phishing Detection using XGBoost
Word Count:	7064
Page Count:	21

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

I agree to an electronic copy of my thesis being made publicly available on NORMA the National College of Ireland's Institutional Repository for consultation.

Signature:	Nishant Nityanand Naik
Date:	15th August 2021

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	Q
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	Q
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	Q

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Modelling Enhanced Phishing detection using XGBoost

[Data Security and Privacy]

This configuration manual consists of in detail specification of the setup and a walkthrough the entire model.

1. Setup

EC2 instance was created with amazon linux2 with t2.micro. The details ae given below.

Platform	Amazon Linux
Instance ID	i-0f5627e1e6ed8212c
Public IPv4 DNS	ec2-54-80-45-230.compute-1.amazonaws.com
Instance type	T2.micro
No of PU	1
Inbound rules for incoming data	Type-SSH, Port-22, Protocol-TCP
Outbound rules for outgoing traffic	Type-All traffic, Port-All, Protocol-All
Public IP address	54.80.45.230
Storage	8Gb

Table 1: Environment Setup

Putty is used to emulate and transfer files since it is compatible with the application and supports majority of network protocols including SSH. Private key of the domain is download and converted to .ppk file and connected to aws. The session is saved in the name of thesis.

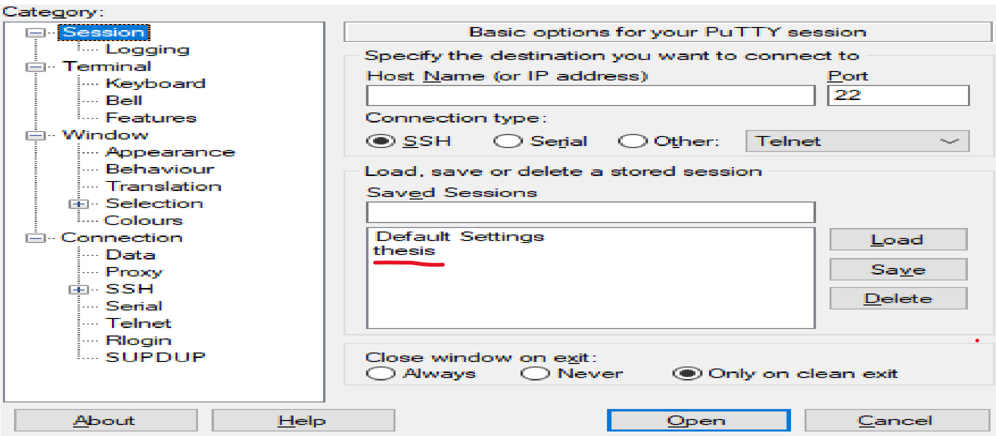

Figure 1: Putty Software Terminal Emulator

Connection is made and logged in as 'ec2-user'. Status check is made just to ensure that everything is upto date [1].

```
https://aws.amazon.com/amazon-linux-2/
(base) [ec2-user@ip-172-31-47-169 ~]$ aws --version
aws-cli/2.2.28 Python/3.8.8 Linux/4.14.241-184.433.amzn2.x86_64 exe/x86_64.amzn.
2 prompt/off
(base) [ec2-user@ip-172-31-47-169 ~]$ python --version
Python 3.8.8
(base) [ec2-user@ip-172-31-47-169 ~]$ anaconda --version
anaconda Command line client (version 1.7.2)
(base) [ec2-user@ip-172-31-47-169 ~]$
```

Figure 2: SSH Connection and versions of Installed Applications

Jupyter notebook is installed and run in the EC2 [2]. now when the initial setup is all finished, we can start with the implementation of the project.

2. Data collection

At first, the dataset is collected from the URLs

Phishing dataset- https://www.phishtank.com/developer_info.php

Legitimate dataset- https://www.unb.ca/cic/datasets/url-2016.html

And stored in the name of phishingURL.csv and LegitimateURL.csv

```
In [3]: import pandas as pd

In [4]: #loading the phishing URLs data to dataframe
   ...: data0 = pd.read_csv("phishingURL.csv")
   ...: data0.head()
Out[4]:
   phish_id                                     url  ... online              targ
et
0   7263022         https://amazon.click-cosme3.com/_  ...    yes               Oth
er
1   7263020        http://www.americanexpzzess.xyz/  ...    yes  American Expre
ss
2   7263018              https://www.amazon-jo.icu/  ...    yes               Oth
er
3   7263017          https://co-jp.rakuteine.shop/  ...    yes               Oth
er
4   7263014  http://mbankaccon.temp.swtest.ru/Mbnki/  ...    yes               Oth
er

[5 rows x 8 columns]

In [5]: data0.shape
Out[5]: (11130, 8)
```

Figure 3: Phishing Dataset

From this we take the desired number of URLs for the training. In this case lets consider 5000, and store it in phishurl.

```
In [6]: #Collecting 5,000 Phishing URLs randomly
   ...: phishurl = data0.sample(n = 5000, random_state = 12).copy()
   ...: phishurl = phishurl.reset_index(drop=True)
   ...: phishurl.head()
Out[6]:
   phish_id                                                url  ... online  target
0   7181927    https://sites.google.com/view/drakio/bt-business  ...    yes   Other
1   7222603   http://timeline.fbcom-qh7cnn8u.kets.sd/connect...  ...    yes   Other
2   6125677   https://bowmanconsultinggroup-my.sharepoint.co...  ...    yes   Other
3   7259201          https://linktr.ee/microsoftonliineservice  ...    yes   Other
4   7065310                  http://realestate-page-homes.com/  ...    yes   Other

[5 rows x 8 columns]

In [7]: phishurl.shape
Out[7]: (5000, 8)
```

Figure 4: Randomly selected 500 URLs

Repeating the same thing with legitimate URL and storing them in the file “legiurl,csv”.

```
In [8]: #Loading legitimate files
   ...: data1 = pd.read_csv("legitimateURL.csv")
   ...: data1.columns = ['URLs']
   ...: data1.head()
Out[8]:
                                                URLs
0  http://1337x.to/torrent/1110018/Blackhat-2015-...
1  http://1337x.to/torrent/1122940/Blackhat-2015-...
2  http://1337x.to/torrent/1124395/Fast-and-Furio...
3  http://1337x.to/torrent/1145504/Avengers-Age-o...
4  http://1337x.to/torrent/1160078/Avengers-age-o...

In [9]: data1.shape
Out[9]: (35377, 1)

In [10]: #Collecting 5,000 Legitimate URLs randomly
    ...: legiurl = data1.sample(n = 5000, random_state = 12).copy()
    ...: legiurl = legiurl.reset_index(drop=True)
    ...: legiurl.head()
Out[10]:
                                                URLs
0  http://graphicriver.net/search?date=this-month...
1  http://ecnavi.jp/redirect/?url=http://www.cros...
2  https://hubpages.com/signin?explain=follow+Hub...
3  http://extratorrent.cc/torrent/4190536/AOMEI+B...
4  http://icicibank.com/Personal-Banking/offers/o...

In [11]: legiurl.shape
Out[11]: (5000, 1)
```

Figure 5: Random 500 legitimate URL

3. Feature Extraction

Now we have the set of URL available. Its time to specify the features on which they must be divided and to extract these features from the 10,000 URLs present. The codes for the features are given in a separate file written in python. A list is created to call the functions. All the features od URLs are extracted and appended to this list.

```
In [13]: #Function to extract features
    ...: def featureExtraction(url,label):
    ...:
    ...:     features = []
    ...:     #Address bar based features (10)
    ...:     features.append(getDomain(url))
    ...:     features.append(havingIP(url))
    ...:     features.append(haveAtSign(url))
    ...:     features.append(getLength(url))
    ...:     features.append(getDepth(url))
    ...:     features.append(redirection(url))
    ...:     features.append(httpDomain(url))
    ...:     features.append(tinyURL(url))
    ...:     features.append(prefixSuffix(url))
    ...:
    ...:     #Domain based features (4)
    ...:     dns = 0
    ...:     try:
    ...:       domain_name = whois.whois(urlparse(url).netloc)
    ...:     except:
    ...:       dns = 1
    ...:
    ...:     features.append(dns)
    ...:     features.append(web_traffic(url))
    ...:     features.append(1 if dns == 1 else domainAge(domain_name))
    ...:     features.append(1 if dns == 1 else domainEnd(domain_name))
    ...:
    ...:     # HTML & Javascript based features (4)
    ...:     try:
    ...:       response = requests.get(url)
    ...:     except:
    ...:       response = ""
    ...:     features.append(iframe(response))
    ...:     features.append(mouseOver(response))
    ...:     features.append(rightClick(response))
    ...:     features.append(forwarding(response))
    ...:     features.append(label)
    ...:
    ...:     return features
    ...:
```

Figure 6: Feature Extraction and file storing

After applying the feature extraction, the final dataset is saved as "finaldataset.csv"

4. Model training and Testing

Necessary packages are imported which are required for the project and the final dataset is loaded into the dataframe. The dataset consists of 10,000 URLs and 18 features.

```
In [4]: #importing basic packages
   ...: import pandas as pd
   ...: import numpy as np
   ...: import seaborn as sns
   ...: import matplotlib.pyplot as plt


In [5]:

In [5]: #Loading the data
   ...: data0 = pd.read_csv('finaldataset.csv')
   ...: data0.head()
Out[5]:
             Domain  Have_IP  Have_At  URL_Length  URL_Depth  Redirection  https_Domain  ...  Domain_Age  Domain_End  iFrame  Mouse_Over  Right_Click  Web_Forwards  Lab
el
0  graphicriver.net        0        0           1          1            0             0  ...           1           1       0           0            1             0
 0
1         ecnavi.jp        0        0           1          1            1             0  ...           1           1       0           0            1             0
 0
2      hubpages.com        0        0           1          1            0             0  ...           0           1       0           0            1             0
 0
3   extratorrent.cc        0        0           1          3            0             0  ...           0           1       0           0            1             0
 0
4     icicibank.com        0        0           1          3            0             0  ...           0           1       0           0            1             0
 0

[5 rows x 18 columns]

In [6]: data0.shape
Out[6]: (10000, 18)
```

Figure 7: URLs and their Feature

Figure 8 and 9 gives the clear picture of the features

```
In [8]: data0.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 18 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Domain         10000 non-null  object
 1   Have_IP        10000 non-null  int64
 2   Have_At        10000 non-null  int64
 3   URL_Length     10000 non-null  int64
 4   URL_Depth      10000 non-null  int64
 5   Redirection    10000 non-null  int64
 6   https_Domain   10000 non-null  int64
 7   TinyURL        10000 non-null  int64
 8   Prefix/Suffix  10000 non-null  int64
 9   DNS_Record     10000 non-null  int64
 10  Web_Traffic    10000 non-null  int64
 11  Domain_Age     10000 non-null  int64
 12  Domain_End     10000 non-null  int64
 13  iFrame         10000 non-null  int64
 14  Mouse_Over     10000 non-null  int64
 15  Right_Click    10000 non-null  int64
 16  Web_Forwards   10000 non-null  int64
 17  Label          10000 non-null  int64
dtypes: int64(17), object(1)
memory usage: 1.4+ MB
```

Figure 8: Selected set of Features

```
In [10]: data0.describe()
    ...:
Out[10]:
            Have_IP       Have_At    URL_Length     URL_Depth   Redirection  ...        iFrame   Mouse_Over  Right_Click  Web_Forwards         Label
count  10000.000000  10000.000000  10000.000000  10000.000000  10000.000000  ...  10000.000000  10000.00000  10000.00000  10000.000000  10000.000000
mean       0.005500      0.022600      0.773400      3.072000      0.013500  ...      0.090900      0.06660      0.99930      0.105300      0.500000
std        0.073961      0.148632      0.418653      2.128631      0.115408  ...      0.287481      0.24934      0.02645      0.306955      0.500025
min        0.000000      0.000000      0.000000      0.000000      0.000000  ...      0.000000      0.00000      0.00000      0.000000      0.000000
25%        0.000000      0.000000      1.000000      2.000000      0.000000  ...      0.000000      0.00000      1.00000      0.000000      0.000000
50%        0.000000      0.000000      1.000000      3.000000      0.000000  ...      0.000000      0.00000      1.00000      0.000000      0.500000
75%        0.000000      0.000000      1.000000      4.000000      0.000000  ...      0.000000      0.00000      1.00000      0.000000      1.000000
max        1.000000      1.000000      1.000000     20.000000      1.000000  ...      1.000000      1.00000      1.00000      1.000000      1.000000

[8 rows x 17 columns]
```

Figure 9: URL features

The data is divided into 80-20% for training and testing the models. This should be randomly shuffled so that its distributed equally.

```
In [14]: # Sepratating & assigning features and target columns to X & y
    ...: y = data['Label']
    ...: X = data.drop('Label',axis=1)
    ...: X.shape, y.shape
Out[14]: ((10000, 16), (10000,))

In [15]: # Splitting the dataset into train and test sets: 80-20 split
    ...: from sklearn.model_selection import train_test_split
    ...:
    ...: X_train, X_test, y_train, y_test = train_test_split(X, y,
    ...:                                                     test_size = 0.2, random_state = 12)
    ...: X_train.shape, X_test.shape
Out[15]: ((8000, 16), (2000, 16))
```

Figure 10: Splitting Data

Models and their values

Decision Tree classifier.

The results for Decision Tree classifier is as follows.

```
In [18]: # Decision Tree model
    ...: from sklearn.tree import DecisionTreeClassifier
    ...:
    ...: # instantiate the model
    ...: tree = DecisionTreeClassifier(max_depth = 5)
    ...: # fit the model
    ...: tree.fit(X_train, y_train)
Out[18]: DecisionTreeClassifier(max_depth=5)

In [19]: #predicting the target value from the model for the samples
    ...: y_test_tree = tree.predict(X_test)
    ...: y_train_tree = tree.predict(X_train)

In [20]: #computing the accuracy of the model performance
    ...: acc_train_tree = accuracy_score(y_train,y_train_tree)
    ...: acc_test_tree = accuracy_score(y_test,y_test_tree)
    ...:
    ...: print("Decision Tree: Accuracy on training Data: {:.3f}".format(acc_train_tree))
    ...: print("Decision Tree: Accuracy on test Data: {:.3f}".format(acc_test_tree))
Decision Tree: Accuracy on training Data: 0.812
Decision Tree: Accuracy on test Data: 0.819

In [26]: def mean_score(scoring):
    ...:     return {i:j.mean() for i,j in scoring.items()}
    ...:     dtree_clf=DecisionTreeClassifier()
    ...: cross_val_scores = cross_validate(dtree_clf, X, y, cv=fold_count, scoring=scoring)
    ...: dtree_score = mean_score(cross_val_scores)
    ...: print(dtree_score)
{'fit_time': 0.010121846199035644, 'score_time': 0.007004547119140625, 'test_accuracy': 0.8594000000000002, 'test_recall': 0.7834000000000001, 'test_precision': 0.92403
91183398575, 'test_f1': 0.8477868190444694}
```

Figure 11: test results of Decision Tree

The overall test accuracy 85.94, Recall- 0.783, Precision 0.924. F1-0.847

Random Forest Classifier

The results for RF classifier are as follows.

```
In [27]: # Random Forest model
    ...: from sklearn.ensemble import RandomForestClassifier
    ...:
    ...: # instantiate the model
    ...: forest = RandomForestClassifier(max_depth=5)
    ...:
    ...: # fit the model
    ...: forest.fit(X_train, y_train)
Out[27]: RandomForestClassifier(max_depth=5)

In [28]:
    ...: #predicting the target value from the model for the samples
    ...: y_test_forest = forest.predict(X_test)
    ...: y_train_forest = forest.predict(X_train)

In [29]: #computing the accuracy of the model performance
    ...: acc_train_forest = accuracy_score(y_train,y_train_forest)
    ...: acc_test_forest = accuracy_score(y_test,y_test_forest)
    ...:
    ...: print("Random forest: Accuracy on training Data: {:.3f}".format(acc_train_forest))
    ...: print("Random forest: Accuracy on test Data: {:.3f}".format(acc_test_forest))
Random forest: Accuracy on training Data: 0.819
Random forest: Accuracy on test Data: 0.822

In [30]: rforest_clf=RandomForestClassifier()
    ...: cross_val_scores = cross_validate(rforest_clf, X, y, cv=fold_count, scoring=scoring)
    ...: rforest_clf_score = mean_score(cross_val_scores)
    ...: print(rforest_clf_score)
{'fit_time': 0.4056638240814209, 'score_time': 0.03227367401123047, 'test_accuracy': 0.8592000000000001, 'test_recall': 0.7864, 'test_precision': 0.9207002974976115, 't
est_f1': 0.8480872282484098}
```

Figure 12: Test results of Random Forest

Accuracy- 0.859, Recall- 0.786, Precision- 0.920, F score- 0.848

Multilayer Perceptron

The results for multilayer perceptron are as follows.

```
In [31]: # Multilayer Perceptrons model
    ...: from sklearn.neural_network import MLPClassifier
    ...:
    ...: # instantiate the model
    ...: mlp = MLPClassifier(alpha=0.001, hidden_layer_sizes=([100,100,100]))
    ...:
    ...: # fit the model
    ...: mlp.fit(X_train, y_train)
Out[31]: MLPClassifier(alpha=0.001, hidden_layer_sizes=[100, 100, 100])

In [32]:

In [32]: #predicting the target value from the model for the samples
    ...: y_test_mlp = mlp.predict(X_test)
    ...: y_train_mlp = mlp.predict(X_train)

In [33]: #computing the accuracy of the model performance
    ...: acc_train_mlp = accuracy_score(y_train,y_train_mlp)
    ...: acc_test_mlp = accuracy_score(y_test,y_test_mlp)
    ...:
    ...: print("Multilayer Perceptrons: Accuracy on training Data: {:.3f}".format(acc_train_mlp))
    ...: print("Multilayer Perceptrons: Accuracy on test Data: {:.3f}".format(acc_test_mlp))
Multilayer Perceptrons: Accuracy on training Data: 0.861
Multilayer Perceptrons: Accuracy on test Data: 0.863
```

Figure 13: Test results of Multilayer Perceptron

```
In [34]: MLP_clf=MLPClassifier()
    ...: cross_val_scores = cross_validate(MLP_clf, X, y, cv=fold_count, scoring=scoring)
    ...: MLP_clf_score = mean_score(cross_val_scores)
    ...: print(MLP_clf_score)
/home/ec2-user/anaconda3/lib/python3.8/site-packages/sklearn/neural_network/_multilayer_perceptron.py:614: ConvergenceWarning: Stochastic Optimizer: Maximum iterations
(200) reached and the optimization hasn't converged yet.
  warnings.warn(
/home/ec2-user/anaconda3/lib/python3.8/site-packages/sklearn/neural_network/_multilayer_perceptron.py:614: ConvergenceWarning: Stochastic Optimizer: Maximum iterations
(200) reached and the optimization hasn't converged yet.
  warnings.warn(
/home/ec2-user/anaconda3/lib/python3.8/site-packages/sklearn/neural_network/_multilayer_perceptron.py:614: ConvergenceWarning: Stochastic Optimizer: Maximum iterations
(200) reached and the optimization hasn't converged yet.
  warnings.warn(
/home/ec2-user/anaconda3/lib/python3.8/site-packages/sklearn/neural_network/_multilayer_perceptron.py:614: ConvergenceWarning: Stochastic Optimizer: Maximum iterations
(200) reached and the optimization hasn't converged yet.
  warnings.warn(
/home/ec2-user/anaconda3/lib/python3.8/site-packages/sklearn/neural_network/_multilayer_perceptron.py:614: ConvergenceWarning: Stochastic Optimizer: Maximum iterations
(200) reached and the optimization hasn't converged yet.
  warnings.warn(
{'fit_time': 7.444262552261352, 'score_time': 0.008514833450317384, 'test_accuracy': 0.8518000000000001, 'test_recall': 0.7536000000000002, 'test_precision': 0.93817276
8216813, 'test_f1': 0.8355259253437023}
```

Figure 13: Test results of Multilayer Perceptron

Accuracy- 0.851, Recall- 0.753, Precision- 0.938, F score- 0.835

XGBoost Classifier

The results for XGBoost Classifier are as follows.

```
In [36]: #predicting the target value from the model for the samples
    ...: y_test_xgb = xgb.predict(X_test)
    ...: y_train_xgb = xgb.predict(X_train)

In [37]: #computing the accuracy of the model performance
    ...: acc_train_xgb = accuracy_score(y_train,y_train_xgb)
    ...: acc_test_xgb = accuracy_score(y_test,y_test_xgb)
    ...:
    ...: print("XGBoost: Accuracy on training Data: {:.3f}".format(acc_train_xgb))
    ...: print("XGBoost : Accuracy on test Data: {:.3f}".format(acc_test_xgb))
XGBoost: Accuracy on training Data: 0.867
XGBoost : Accuracy on test Data: 0.862
```

```
{'fit_time': 0.798342227935791, 'score_time': 0.010219621658325195, 'test_accuracy': 0.8588999999999999, 'test_recall': 0.7864, 'test_precision': 0.9203212333796943, 't
est_f1': 0.8477419622573172}
```

Figure 14: Test results for XGBoost Classifier

Accuracy- 0.858, Recall- 0.786, Precision- 0.920, F1 score- 0.847

Support Vector Machines

The results for SVM are as follows

```
In [41]: from sklearn.svm import SVC
    ...:
    ...: # instantiate the model
    ...: svm = SVC(kernel='linear', C=1.0, random_state=12)
    ...: #fit the model
    ...: svm.fit(X_train, y_train)
Out[41]: SVC(kernel='linear', random_state=12)

In [42]: #predicting the target value from the model for the samples
    ...: y_test_svm = svm.predict(X_test)
    ...: y_train_svm = svm.predict(X_train)
    ...:

In [43]: #computing the accuracy of the model performance
    ...: acc_train_svm = accuracy_score(y_train,y_train_svm)
    ...: acc_test_svm = accuracy_score(y_test,y_test_svm)
    ...:
    ...: print("SVM: Accuracy on training Data: {:.3f}".format(acc_train_svm))
    ...: print("SVM : Accuracy on test Data: {:.3f}".format(acc_test_svm))
SVM: Accuracy on training Data: 0.802
SVM : Accuracy on test Data: 0.804
```

```
{'fit_time': 2.229040098190308, 'score_time': 0.11832234859466553, 'test_accuracy': 0.8019000000000001, 'test_recall': 0.6248, 'test_precision': 0.9674808580150357, 'te
st_f1': 0.7590945541363523}
```

Figure 15: Test results of SVM

Accuracy- 0.801, Reall- 0.624, Precision- 0.967, F score- 0.759

Conclusion

The experiment was completed in full with successful feature extraction procedure and model training. However, the results obtained were not as planned.

References

[1] *Youtube.com*, 2021. [Online]. Available: https://www.youtube.com/watch?v=ulprqHHWlng&t=52s.

[2] "Run Project Jupyter Notebooks On Amazon EC2", *Chrisalbon.com*, 2021. [Online]. Available: https://chrisalbon.com/code/aws/basics/run_project_jupyter_on_amazon_ec2/.