Detecting Phishing Websites Using Machine Learning

1. Introduction

Phishing is a type of social engineering where an attacker sends a fraudulent (e.g., spoofed, fake, or otherwise deceptive) message designed to trick a human victim into revealing sensitive information to the attacker or to deploy malicious software on the victim's infrastructure like ransomware. Phishing attacks have become increasingly sophisticated and often transparently mirror the site being targeted, allowing the attacker to observe everything while the victim is navigating the site, and transverse any additional security boundaries with the victim.

It is a method of identity theft that relies on individuals unwittingly volunteering personal details or information that can then be used for nefarious purposes. It is often carried out through the creation of a fraudulent website, email, or text appearing to represent a legitimate firm.

It involves compromising legitimate web pages in order to redirect users to a malicious website or an exploit kit via cross-site scripting. A hacker may compromise a website and insert an exploit kit such as MPack in order to compromise legitimate users who visit the now compromised web server. One of the simplest forms of page hijacking involves altering a webpage to contain a malicious inline frame which can allow an exploit kit to load. Page hijacking is frequently used in tandem with a watering hole attack on corporate entities in order to compromise targets.

A scammer may use a fraudulent website that appears on the surface to look the same as a legitimate website. Visitors to the site, thinking they are interacting with a real business, may submit their personal information, such as social security numbers, account numbers, login IDs, and passwords, to this site. The scammers then use the information submitted to steal visitors' money, identity, or both; or to sell the information to other criminal parties.

Malicious links will lead to a website that often steals login credentials or financial information like credit card numbers. Attachments from phishing emails can contain malware that once opened can leave the door open to the attacker to perform malicious behavior from the user’s computer.

The importance of safeguarding online users from becoming victims of online fraud, divulging confidential information to an attacker among other effective uses of phishing as an attacker’s tool, phishing detection tools play a vital role in ensuring a secure online experience for users. Unfortunately, many of the existing phishing-detection tools, especially those that depend on an

existing blacklist, suffer limitations such as low detection accuracy and high false alarm that is often caused by either a delay in blacklist update as a result of the human verification process involved in classification or perhaps, it can be attributed to human error in classification which may lead to improper classification of the classes.

2. Dataset Overview

The Data I am using consists of Phishing Websites and benign websites. The Phishing Websites Data Set was collected mainly from the PhishTank archive, MillerSmiles archive, Googleâ€™s searching operators. The benign websites were collected from Alexa.com.

The algorithms used on this dataset have never been used on it. The few other projects done are on different datasets with not many attributes, I even found out missing attributes from other datasets that were very important to the decision making. It's a new data with new attributes, I believe my contributions are using this data and using the classifier algorithms and post and pre pruning the data to avoid overfitting was one of the challenges. Other projects and literature reviews I have read have not applied this classification problems to this kind of data, with this many attributes, they have not used this new and updated dataset and have not applied Pre-Pruning and Post Pruning. So that was a first and it really made the data easy to apply my models onto it.

There are a lot of algorithms and a wide variety of data types of phishing detection out there. A phishing URL and the corresponding page have several features which can be differentiated from a malicious URL. The data I have chosen has 11,055 instances and 32 attributes.based on certain features:

1. Address-based features:

- IP Address in URL
- Length of URL
- Using URL shortening services like "TinyURL"
- URL's having "@" symbol
- Redirecting using "//"
- Sub Domain and Multi-Sub Domains
- HTTPS in Domain name
- Domain Registration Length
- Favicon
- Using Non-Standard Port
- The existence of "HTTPS" token in the domain part of the URL

- Adding prefix or suffix separated by (-) to the Domain

2. Abnormal-based Features:

- Request URL
- URL of anchor
- Link in <Meta>, <Script> and <Link> tags
- Server Form Handler (SFH)
- Submitting information to an email
- Abnormal URL

3. HTML and JavaScript-based features:

- Website Forwarding
- Status bar customization
- Disabling right-click
- Using pop-up window
- IFrame redirection

4. Domain-based features:

- Age of domain
- DNS record
- Website traffic
- PageRank
- Google index
- Number of links pointing to the page
- Statistical reports based feature

- There are in total 28 features that were selected from the dataset for this model.

3. Loading the Data

The first step is to load the data and see what the first few instances and some of the columns are

Loading the Data

```
# Load the csv data file
dataset = pd.read_csv('Training Dataset.csv')
# dataset = pd.DataFrame(data[0])
dataset.head()
```

Figure 1: Loading the data

	id	having_IP_Address	URL_Length	Shortining_Service	having_At_Symbol	double_slash_redirecting
0	1	-1	1	1	1	-1
1	2	1	1	1	1	1
2	3	1	0	1	1	1
3	4	1	0	1	1	1
4	5	1	0	-1	1	1

Figure 2: Loading the data output

4. Familiarizing with the Data

Now that we have loaded the data, we try to look into the data and get familiar with it. Gain information about the data, if it has any missing values; all the data types of the attributes are the same, and get the shape of the data.

Information about the data and its attribute types.

```
# Displays all the info about the dataset
dataset.info()
```

✓ 0.8s

```
Output exceeds the size limit. Open the full output data in a text
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11055 entries, 0 to 11054
Data columns (total 32 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   id                          11055 non-null  int64
 1   having_IP_Address           11055 non-null  int64
 2   URL_Length                  11055 non-null  int64
 3   Shortining_Service          11055 non-null  int64
 4   having_At_Symbol            11055 non-null  int64
 5   double_slash_redirecting    11055 non-null  int64
 6   Prefix_Suffix               11055 non-null  int64
 7   having_Sub_Domain           11055 non-null  int64
 8   SSLfinal_State              11055 non-null  int64
```

Figure 3: Display info

We can see the data types of all the attributes are int64, no categorical values or strings, so no need to change attribute values

Figure 4

The shape of the data:

```
# Display the shape of the dataset
dataset.shape

✓ 0.4s

(11055, 32)
```

Figure 5: Shape of the data

Displaying all the columns in the dataset

```
# Display all the attributes of the dataset
dataset.columns

✓ 0.3s

Index(['id', 'having_IP_Address', 'URL_Length', 'Shortining_Service',
       'having_At_Symbol', 'double_slash_redirecting', 'Prefix_Suffix',
       'having_Sub_Domain', 'SSLfinal_State', 'Domain_registeration_length',
       'Favicon', 'port', 'HTTPS_token', 'Request_URL', 'URL_of_Anchor',
       'Links_in_tags', 'SFH', 'Submitting_to_email', 'Abnormal_URL',
       'Redirect', 'on_mouseover', 'RightClick', 'popUpWidnow', 'Iframe',
       'age_of_domain', 'DNSRecord', 'web_traffic', 'Page_Rank',
       'Google_Index', 'Links_pointing_to_page', 'Statistical_report',
       'Result'],
      dtype='object')
```

Figure 6: Shape of the data output

Displaying all attributes values data type to see if there are types that need to be changed.

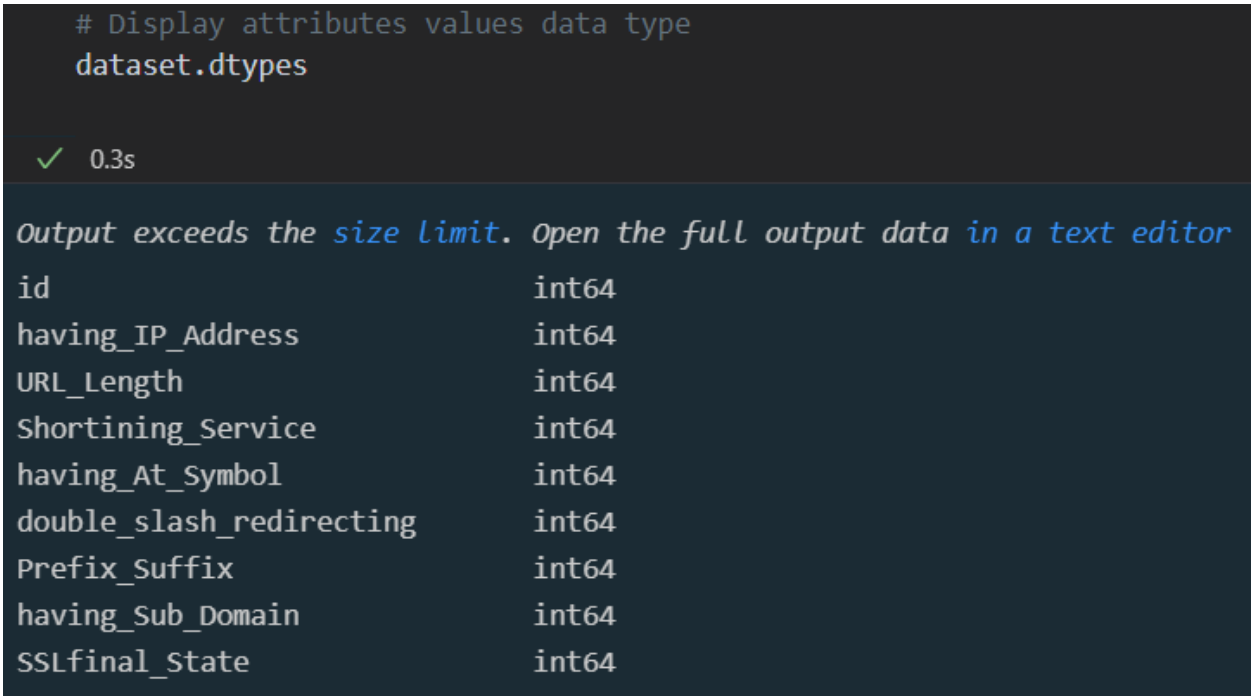

Figure 7: Display dtypes

In this data all the values are of type int64 there are no categorical values that need to be changed.

5. Visualizing the Data

The next step is to visualize the data and see if there is a clear idea of what the information means by giving it visual context through maps or graphs. This makes it easier for us to comprehend and therefore makes it easier to identify trends, patterns, and outliers within large data sets. Plots and graphs like histograms and heatmaps are displayed to find how the data is distributed and how the features are related to each other.

- **Histogram**: Plotting the histogram of the data to summarise discrete or continuous data that are measured on an interval scale and to illustrate the major features of the distribution of the data in a convenient form.

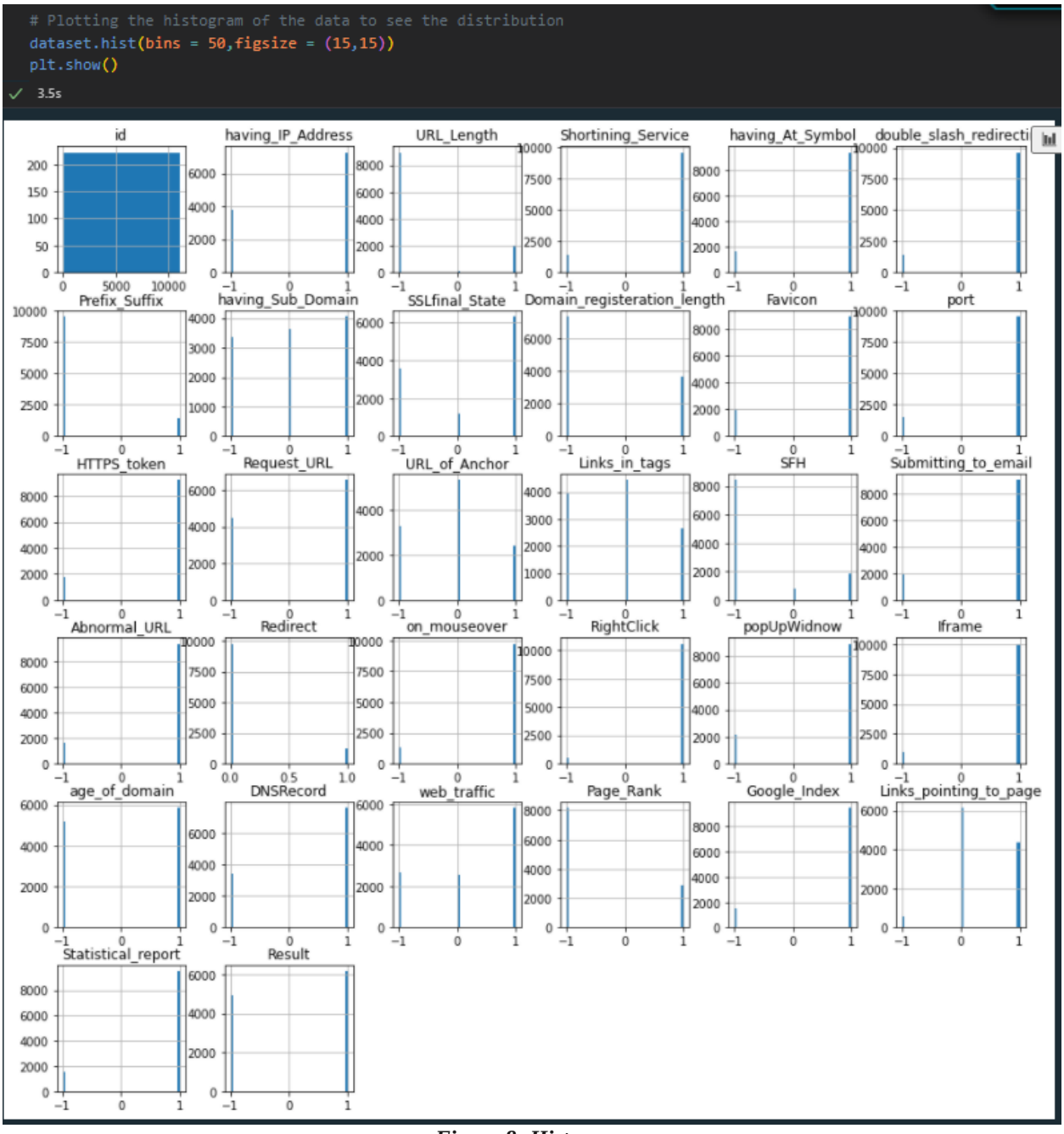

Figure 8: Histogram

- Heatmap: Plotting the heatmap for the data to better visualize the volume of locations/events within a dataset and show relationships and correlation between the attributes.

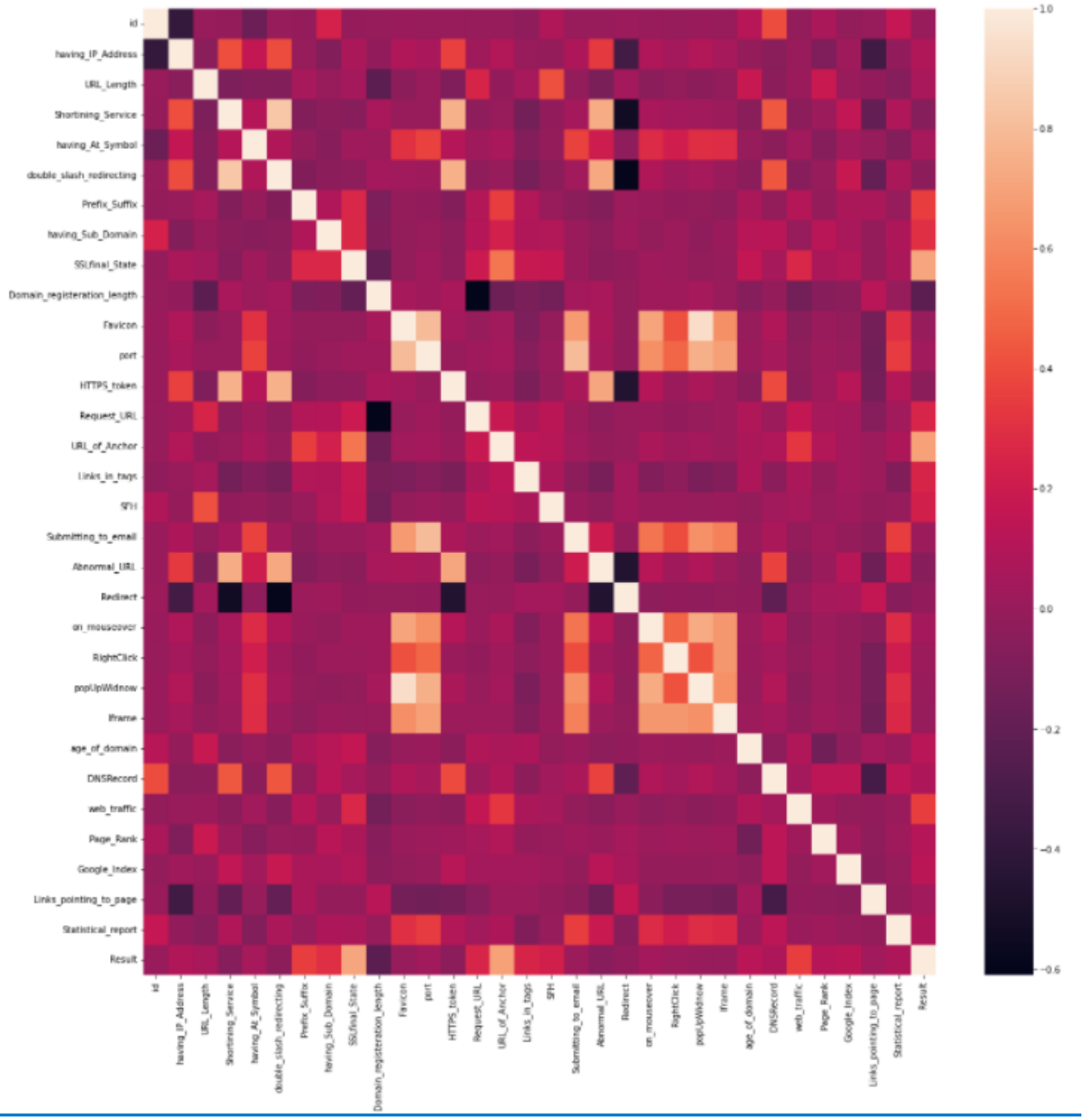

Figure 9: Heatmap

6. Data Preprocessing and EDA

The next step is to clean and use data preprocessing techniques and transform the data to use it in the models. The data may have many irrelevant and missing parts. To handle it we need to find out if the dataset has missing values and noisy data; if it needs to be normalized or discretized; if the data needs to be reduced; or if there are any continuous variables.

```
dataset.describe()
```
✓ 0.1s Python

	id	having_IP_Address	URL_Length	Shortining_Service	having_At_Symbol	double_slash_redirecting	Prefix_Suffix	having_Sub_Domain	SSLfinal_State
count	11055.000000	11055.000000	11055.000000	11055.000000	11055.000000	11055.000000	11055.000000	11055.000000	11055.000000
mean	5528.000000	0.313795	-0.633198	0.738761	0.700588	0.741474	-0.734962	0.063953	0.250927
std	3191.447947	0.949534	0.766095	0.673998	0.713598	0.671011	0.678139	0.817518	0.911892
min	1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000
25%	2764.500000	-1.000000	-1.000000	1.000000	1.000000	1.000000	-1.000000	-1.000000	-1.000000
50%	5528.000000	1.000000	-1.000000	1.000000	1.000000	1.000000	-1.000000	0.000000	1.000000
75%	8291.500000	1.000000	-1.000000	1.000000	1.000000	1.000000	-1.000000	1.000000	1.000000
max	11055.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

Figure 10: Describe

We use “describe” here to see the count, mean, std, minimum values, quartiles (25%, 50%, 75%), and maximum values for all the attributes.

From this, we learned that most of the data is made of {-1,1} values except for “id” which is the number of instances, and we are gonna drop that attribute cause it has no significance to our ML learning model.

```
# Dropping the id column cause it is not useful for the model
dataset1 = dataset.drop(['id'], axis = 1).copy()
dataset1.head()
```

	having_IP_Address	URL_Length	Shortining_Service	having_At_Symbol	double_slash_redi
0	-1	1	1	1	
1	1	1	1	1	
2	1	0	1	1	
3	1	0	1	1	
4	1	0	-1	1	

Figure 11: Dropping Id

We dropped the “id” attribute and its values and assigned it to a new dataset variable, called “dataset1”.

Next, we check if the data has any null or missing values and we do that by:

```
# Checking the data for null or missing values
dataset1.isnull().sum()
```

Figure 12: isnull()

Output:

```
having_IP_Address                0
URL_Length                       0
Shortining_Service               0
having_At_Symbol                 0
double_slash_redirecting         0
Prefix_Suffix                    0
having_Sub_Domain                0
SSLfinal_State                   0
Domain_registeration_length      0
Favicon                          0
port                             0
HTTPS_token                      0
Request_URL                      0
URL_of_Anchor                    0
Links_in_tags                    0
SFH                              0
Submitting_to_email              0
Abnormal_URL                     0
Redirect                         0
on_mouseover                     0
RightClick                       0
popUpWidnow                      0
Iframe                           0
age_of_domain                    0
DNSRecord                        0
...
Page_Rank                        0
Google_Index                     0
Links_pointing_to_page           0
Statistical_report               0
Result                           0
dtype: int64
```

Figure 13: isnull() output

From the above figure, we can see the data doesn't have any null or missing values. Now the data is ready to be trained.

7. Splitting the Data

Here we split the data into training and test data 80-20 for our models to use. We first assign the class “Result”, into a variable to compare when we apply the models, and then we drop the class from our variable that’s gonna be split into training and test data.

```
# Sepratating & assigning features and target columns to x & y
y = dataset1['Result']
x = dataset1.drop('Result', axis=1)
x.shape, y.shape

✓ 0.7s
((11055, 30), (11055,))
```

Figure 14:Splitting the data

We used train_test_split to split the dataset, while also shuffling the data, i.e the training and test data are randomly selected

```
# 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=42, shuffle=True)

x_train.shape, y_train.shape, x_test.shape, y_test.shape

# Display the shuffled dataset
x_train.head()
```

Figure 15

We randomly assign the data into the train and test sets.
Output:

	having_IP_Address	URL_Length	Shortining_Service	having_At_Symbol	double_slash_redirecting	Prefix
480	1	-1	1	1	1	
10812	-1	-1	1	1	1	
4064	1	1	1	1	1	
8225	-1	-1	-1	-1	-1	
9432	-1	-1	1	1	1	

Figure 16

8. Machine Learning Models and Training Algorithm

This is a supervised machine learning task, so Classification and Regression are the options to use. This data set comes under a classification problem, as the input URL is classified as

phishing or legitimate. The classification techniques considered to train the data are:

- Decision Tree Algorithm
- Random Forest Algorithm
- Support Vector Machine Algorithm

a. Decision Tree Algorithm

A Decision Tree is a supervised machine learning algorithm that can be used for both Regression and Classification problem statements. It divides the complete dataset into smaller subsets while at the same time an associated Decision Tree is incrementally developed. The data is continuously split according to a certain parameter. The tree can be explained by two entities, namely decision nodes and leaves. The leaves are the decisions or the final outcomes. And the decision nodes are where the data is split.

In a Decision Tree Algorithm, the first step is to split the data into training and test, which was already done above. Next step is to use the classifier algorithm on the training sets without any parameters and see the results.

```
# Default Decision Model
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.tree import DecisionTreeClassifier

classfier = DecisionTreeClassifier(random_state=42)
classfier.fit(x_train, y_train)

#predicting the target value from the model for the samples with default values
y_test_tree = classfier.predict(x_test)
y_train_tree = classfier.predict(x_train)

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))
print("Classification report - \n", classification_report(y_test, y_test_tree))
print(confusion_matrix(y_test, y_test_tree))
```

Figure 17

Output:

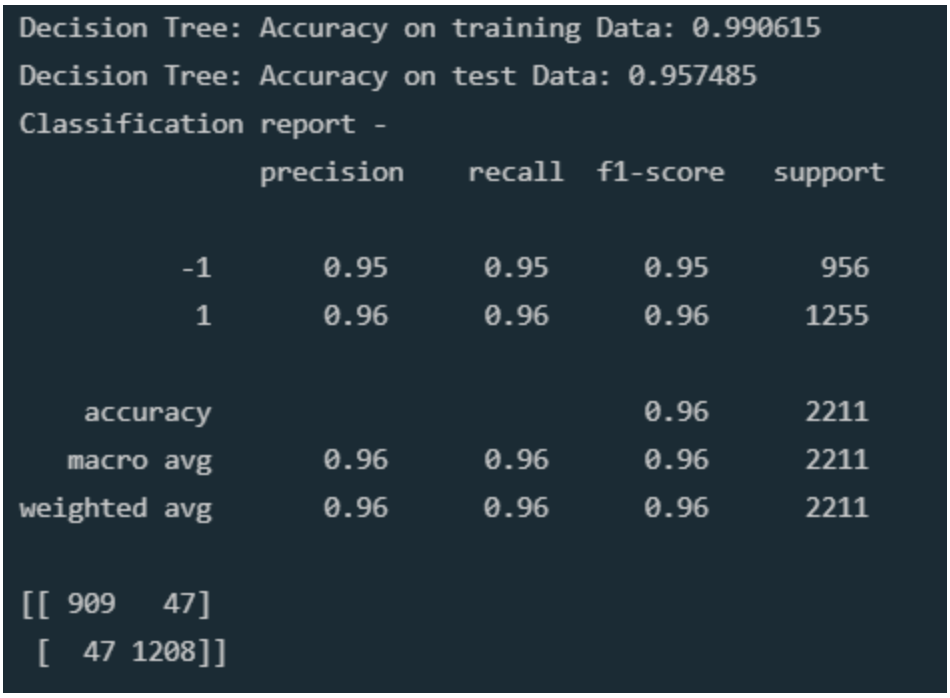

Figure 18

Displayed are the confusion matrix and the classification report, which consists of accuracy, recall, f1-score, and support.

The model's accuracy on training data is 99% and the accuracy for the test data is 95.7%. By looking at this report it can be seen the model overfits for the data since the train data accuracy is bigger than the test data accuracy.

Next step is to apply pre pruning, to determine the maximum depth the tree can go, the minimum samples split, and the minimum samples leaf.

Pre Pruning:

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score

# instatiate the model
# max_depth represents the maximum depth the tree can go

# Sample max_depth, min_samples_split, and min_samples_leaf to test out and find the best one to avoid overfitting
param_grid = {
    "max_depth": [3,5,10,15,20,None],
    "min_samples_split": [2,5,7,10],
    "min_samples_leaf": [1,2,5]
}

# Make a default decision tree

classfier = DecisionTreeClassifier(random_state=42)
grid_cv = GridSearchCV(classfier, param_grid, scoring="roc_auc", n_jobs = -1, cv=3).fit(x_train, y_train)

# Check the AUC ROC score to decide the max_depth, min_samples_split, and min_samples_leaf values

print("Param for GS", grid_cv.best_params_)
print("CV score for GS", grid_cv.best_score_)
print("Train AUC ROC Score for GS: ", roc_auc_score(y_train, grid_cv.predict(x_train)))
print("Test AUC ROC Score for GS: ", roc_auc_score(y_test, grid_cv.predict(x_test)))
print(confusion_matrix(y_test, y_test_tree))
```

Figure 19: Pre-Pruning

Output:

```
Param for GS {'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 10}
CV score for GS 0.9817258448233783
Train AUC ROC Score for GS:  0.9599976898814947
Test AUC ROC Score for GS:  0.9420014502658822
[[ 909   47]
 [  47 1208]]
```

Figure 20: Pre-Pruning output

The Pre-Pruning result outputs the best values of max_depth, min_samples_leaf, and the min_samples_split. These are values set to stop the tree from overfitting.

The Pre-Pruning accuracy score after putting the values from the above figure are:

```
classfier = DecisionTreeClassifier(max_depth=10, min_samples_leaf=1, min_samples_split=10)
classfier.fit(x_train, y_train)

#predicting the target value from the model for the samples with default values
y_test_tree = classfier.predict(x_test)
y_train_tree = classfier.predict(x_train)

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))
print("Classification report - \n", classification_report(y_test, y_test_tree))
```

Figure 21

Output:

```
Decision Tree: Accuracy on training Data: 0.960086
Decision Tree: Accuracy on test Data: 0.940751
Classification report -
               precision    recall  f1-score   support

          -1       0.92      0.94      0.93       956
           1       0.96      0.94      0.95      1255

    accuracy                           0.94      2211
   macro avg       0.94      0.94      0.94      2211
weighted avg       0.94      0.94      0.94      2211
```

Figure 22: max_depth

From the above figure, it is seen that the model has improved but still needs more values to improve the accuracy for the test data.

The next step is to post prune it and find the best ccp_alpha, Cost complexity pruning. Cost complexity pruning provides another option to control the size of a tree. In DecisionTreeClassifier, this pruning technique is parameterized by the cost complexity parameter, ccp_alpha. Greater values of ccp_alpha increase the number of nodes pruned.

To find the best value of ccp_alpha, it is required to plot the AUC-ROC score vs the alpha value to see where the accuracy is maximum for both training and test datas.

```
classfier = DecisionTreeClassifier(random_state=42)
classfier.fit(x_train, y_train)

# compute ccp_alpha values
path = classfier.cost_complexity_pruning_path(x_train, y_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities


# train DT classifier for each ccp_alpha value
clfs = []
for ccp_alpha in ccp_alphas:
    classfier = DecisionTreeClassifier(random_state=42, ccp_alpha=ccp_alpha)
    classfier.fit(x_train, y_train)
    clfs.append(classfier)

# Plot train and test score for each of the above trained model
clfs = clfs[:-1]
ccp_alphas = ccp_alphas[:-1]

train_scores = [roc_auc_score(y_train, classfier.predict(x_train)) for classfier in clfs]
test_scores = [roc_auc_score(y_test, classfier.predict(x_test)) for classfier in clfs]

fig, ax = plt.subplots(figsize = (12,6))
ax.set_xlabel("alpha")
ax.set_ylabel("accuracy")
ax.set_title("AUC-ROC score vs alpha")
ax.plot(ccp_alphas, train_scores, marker='o', label="train")
ax.plot(ccp_alphas, test_scores, marker='o', label="test")
ax.legend()
plt.show()
```

Figure 23: ccp_alpha

Output:

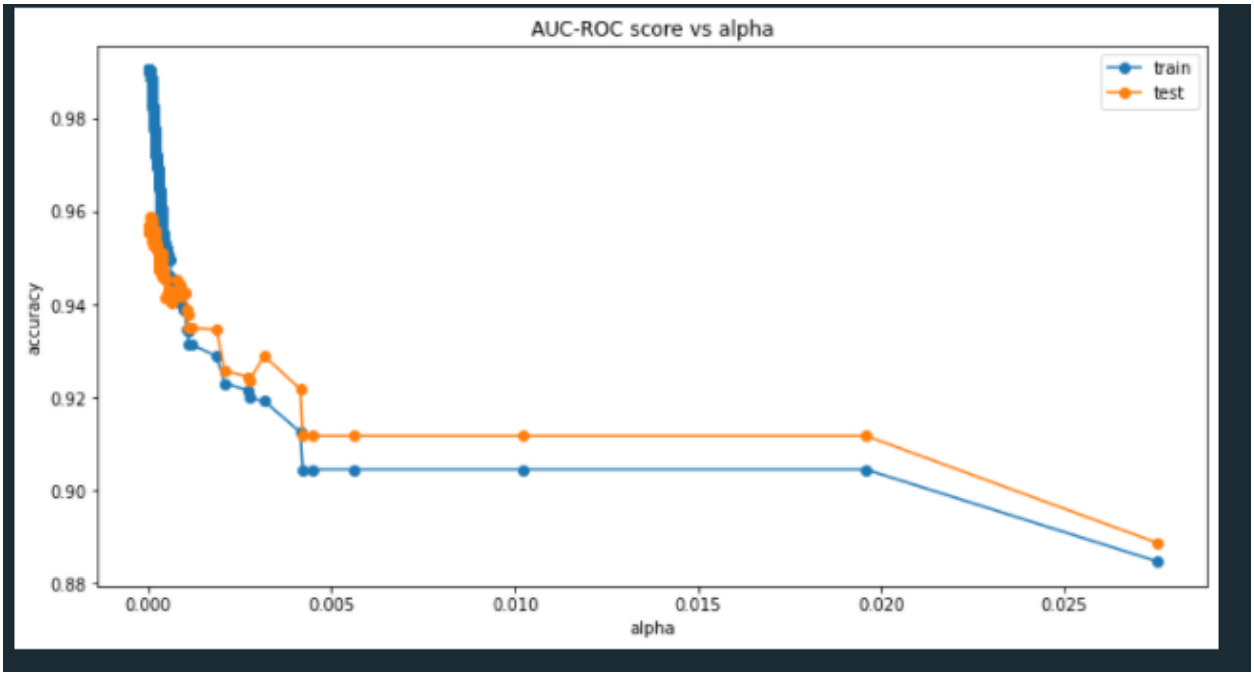

Figure 23: ccp_alpha to accuracy graph

From the above figure the maximum accuracy the model can reach is when the alpha value is at 0.001. Now combining all the values gives us an output of:

```
# Fit is called to train the algorithm on the trainig data which is passed as parameter to the "fit" method
# Classify the dataset using the above given values

classfier = DecisionTreeClassifier(max_depth=10, min_samples_leaf=2, min_samples_split=10, ccp_alpha=0.001)
classfier.fit(x_train, y_train)
```

✓ 0.7s

```
DecisionTreeClassifier(ccp_alpha=0.001, max_depth=10, min_samples_leaf=2,
                       min_samples_split=10)
```

Figure 24: ccp_alpha evaluation

```
from sklearn.metrics import classification_report, confusion_matrix

#predicting the target value from the model for the samples after post-pruning
y_test_tree = classfier.predict(x_test)
y_train_tree = classfier.predict(x_train)

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))
print("Classification report - \n", classification_report(y_test, y_test_tree))
print(confusion_matrix(y_test, y_test_tree))
```

Figure 25

Performance Evaluation:

```
Decision Tree: Accuracy on training Data: 0.938602
Decision Tree: Accuracy on test Data: 0.942108
Classification report -
              precision    recall  f1-score   support

          -1       0.92      0.95      0.93       956
           1       0.96      0.94      0.95      1255

    accuracy                           0.94      2211
   macro avg       0.94      0.94      0.94      2211
weighted avg       0.94      0.94      0.94      2211

[[ 904   52]
 [  76 1179]]
```

Figure 26: Performance Evaluation

Now the accuracy for the test data is greater than the accuracy for the training data, which significantly increases the model's capability of classifying new data.

Visualizing the tree:

```
# Decision tree visualization
from sklearn import tree

# Garbage Collector to handle Out of Memory Error when visualizing
import gc
gc.collect()


# Visualizing decision tree using "plot_tree"
fn=['having_IP_Address',\
'URL_Length','Shortining_Service','having_At_Symbol','double_slash_redirecting','Prefix_Suffix\
','having_Sub_Domain \
','SSLfinal_State \
','Domain_registeration_length \
','Favicon \
','port \
','HTTPS_token \
','Request_URL \
','URL_of_Anchor \
','Links_in_tags \
','SFH \
','Submitting_to_email \
','Abnormal_URL \
','Redirect \
','on_mouseover \
','RightClick \
','popUpWidnow \
','Iframe \
','age_of_domain \
','DNSRecord \
','web_traffic \
','Page_Rank \
','Google_Index \
','Links_pointing_to_page \
','Statistical_report' \
]

cn = ['-1','1']
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (10,10), dpi=300)

tree.plot_tree(classfier, feature_names=fn, class_names=cn, filled=True, fontsize=6, rounded=True)
plt.show()

plt.savefig('Output2.png')
```

Figure 27: Plotting the Decision Tree

Output:

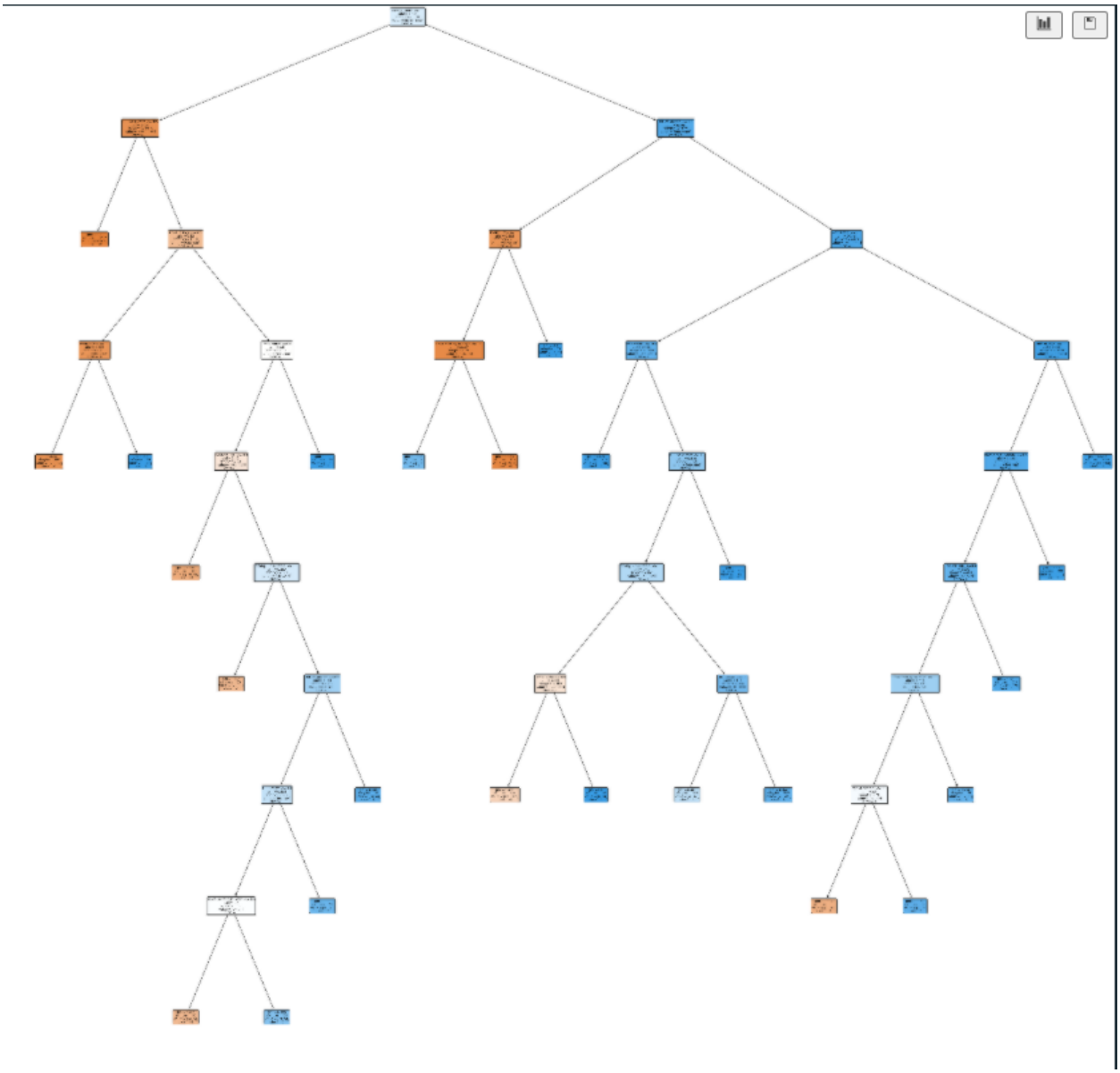

Figure 28: Decision Tree

This is what the tree looks like, as it can be seen the maximum depth it has undergone is 10 and the root node in the tree looks like the figure below:

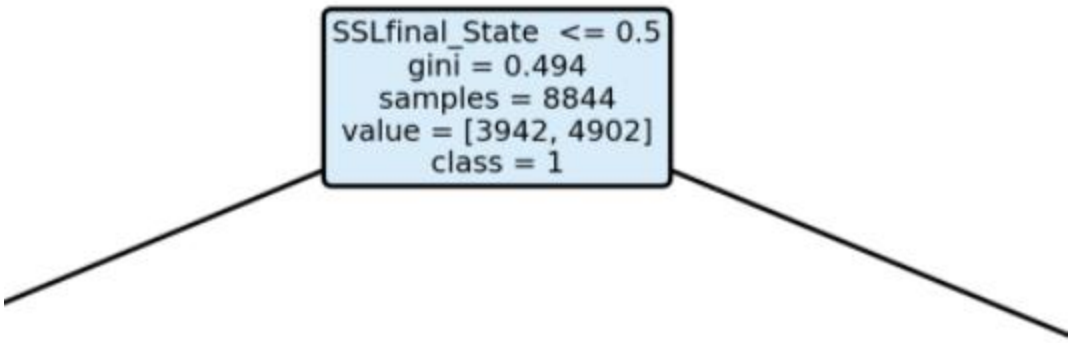

Figure 29: Root Node

The last step is to check the feature importance in the model:

```
plt.figure(figsize=(8,7))
n_features = x_train.shape[1]
plt.barh(range(n_features), classfier.feature_importances_, align="center")
plt.yticks(np.arange(n_features), x_train.columns)
plt.xlabel("Feature Importance")
plt.ylabel("Features")
plt.show()
```

Figure 30: Plotting Feature Importance

Output:

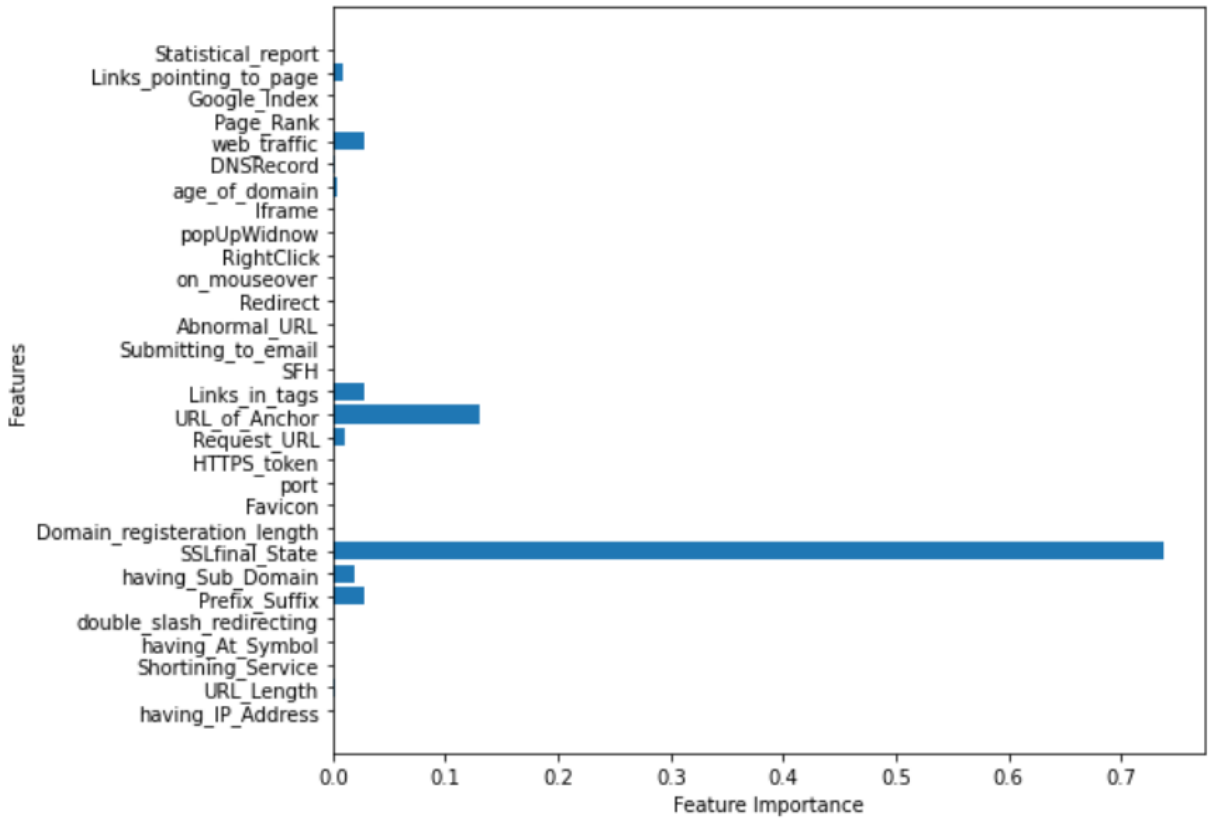

Figure 31: Feature Importance

b. Random Forest Algorithm

Random forest is a flexible, easy-to-use machine learning algorithm that produces, even without hyper-parameter tuning, a great result most of the time. It is also one of the most used algorithms, because of its simplicity and diversity (it can be used for both classification and regression tasks). It is a supervised learning algorithm. The "forest" it builds is an ensemble of decision trees, usually trained with the “bagging” method. The general idea of the bagging method is that a combination of learning models increases the overall result.

The first step for Random Forest is to determine the best possible values of its parameters, i.e. n_estimators (This is the number of trees you want to build before taking the maximum voting or averages of predictions. A higher number of trees give you better performance but makes your code slower.), max_depth, and min_samples_leaf.

To determine the best value of n_estimators we use GridSearchCv to plot the AUC_Score with respect to n_estimators.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score, roc_curve, auc

n_estimators = [1, 2, 4, 8, 16, 32, 64, 100, 200]
train_results = []
test_results = []
for estimator in n_estimators:
   rf = RandomForestClassifier(n_estimators=estimator, n_jobs=-1)
   rf.fit(x_train, y_train)
   train_pred = rf.predict(x_train)
   false_positive_rate, true_positive_rate, thresholds = roc_curve(y_train, train_pred)
   roc_auc = auc(false_positive_rate, true_positive_rate)
   train_results.append(roc_auc)
   y_pred = rf.predict(x_test)
   false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pred)
   roc_auc = auc(false_positive_rate, true_positive_rate)
   test_results.append(roc_auc)
from matplotlib.legend_handler import HandlerLine2D
line1, = plt.plot(n_estimators, train_results, 'b', label='Train AUC')
line2, = plt.plot(n_estimators, test_results, 'r', label='Test AUC')
plt.legend(handler_map={line1: HandlerLine2D(numpoints=2)})
plt.ylabel('AUC score')
plt.xlabel('n_estimators')
plt.show()
```

Figure 32: n_estimators

Output:

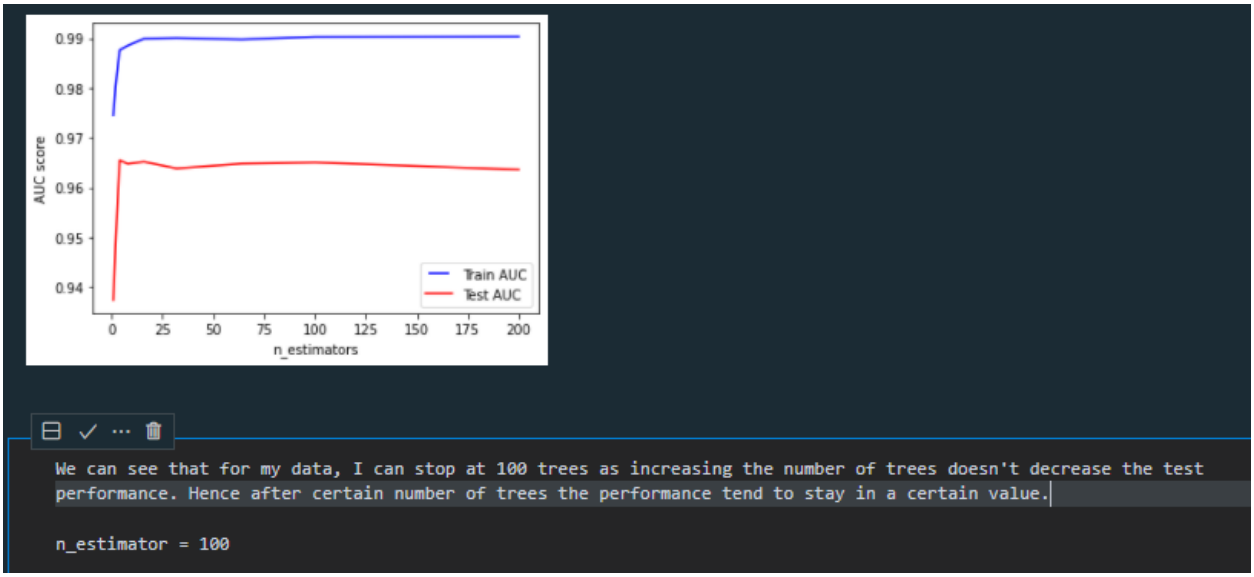

Figure 33: n_estimators to AUC-Score

The next step is to find out the best value for max_depth. That is also done using GridSearchCv in the following way:

```
max_depths = np.linspace(1, 32, 32, endpoint=True)
train_results = []
test_results = []
for max_depth in max_depths:
   rf = RandomForestClassifier(max_depth=max_depth, n_jobs=-1)
   rf.fit(x_train, y_train)
   train_pred = rf.predict(x_train)
   false_positive_rate, true_positive_rate, thresholds = roc_curve(y_train, train_pred)
   roc_auc = auc(false_positive_rate, true_positive_rate)
   train_results.append(roc_auc)
   y_pred = rf.predict(x_test)
   false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pred)
   roc_auc = auc(false_positive_rate, true_positive_rate)
   test_results.append(roc_auc)
from matplotlib.legend_handler import HandlerLine2D
line1, = plt.plot(max_depths, train_results, 'b', label="Train AUC")
line2, = plt.plot(max_depths, test_results, 'r', label="Test AUC")
plt.legend(handler_map={line1: HandlerLine2D(numpoints=2)})
plt.ylabel('AUC score')
plt.xlabel('Tree depth')
plt.show()
```

Figure 34: max_depths

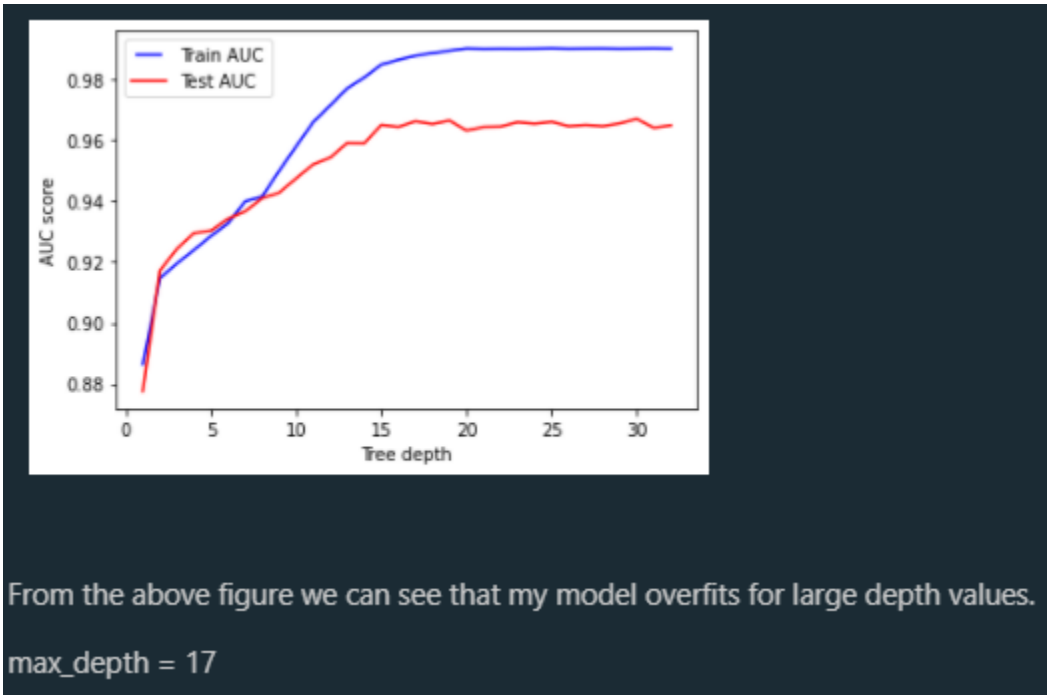

Figure 35: max_depths to AUC Score

Now that we know the best values for max_depth and n_estimator, we can use the Random Forest Classifier to use those values and predict the target values.

```python
# Random Forest model
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import export_graphviz
from subprocess import CalledProcessError
from IPython import display

# instantiate the model
# Giving the maximum n_estimators value after trying different values for max accuracy
forest = RandomForestClassifier(max_depth=17, n_estimators=100)

# fit the model
forest.fit(x_train, y_train)
```

Figure 36: Random Forest Classifier

```python
#predicting the target value from the model for the samples
y_test_forest = forest.predict(x_test)
y_train_forest = forest.predict(x_train)

#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))
print("Classification report - \n", classification_report(y_test, y_test_forest))
print(confusion_matrix(y_test, y_test_forest))
```

✓ 0.1s

Figure 37

Performance Evaluation:

```
Random forest: Accuracy on training Data: 0.989
Random forest: Accuracy on test Data: 0.968
Classification report -
               precision    recall  f1-score   support

          -1       0.98      0.95      0.96       956
           1       0.96      0.98      0.97      1255

    accuracy                           0.97      2211
   macro avg       0.97      0.97      0.97      2211
weighted avg       0.97      0.97      0.97      2211

[[ 905   51]
 [  19 1236]]
```

Figure 38: Random Forest Classifier Performance Evaluation

Plotting a Random Tree is computationally costly cause it computes and calculates 100 trees:

```
fig, axes = plt.subplots(nrows = 1,ncols = 5,figsize = (10,2), dpi=900)
for index in range(0,5):
    tree.plot_tree(forest.estimators_[index],
                   feature_names = fn,
                   class_names=cn,
                   filled = True,
                   ax = axes[index]);
    axes[index].set_title('Estimator: ' + str(index), fontsize=11)
fig.savefig('forest_individualtree.png')
```

✓ 5m 13.7s

Figure 39: Plotting Random Forest

Output:

The tree is big and takes quite a long time to run, this is some of the root nodes on the tree:

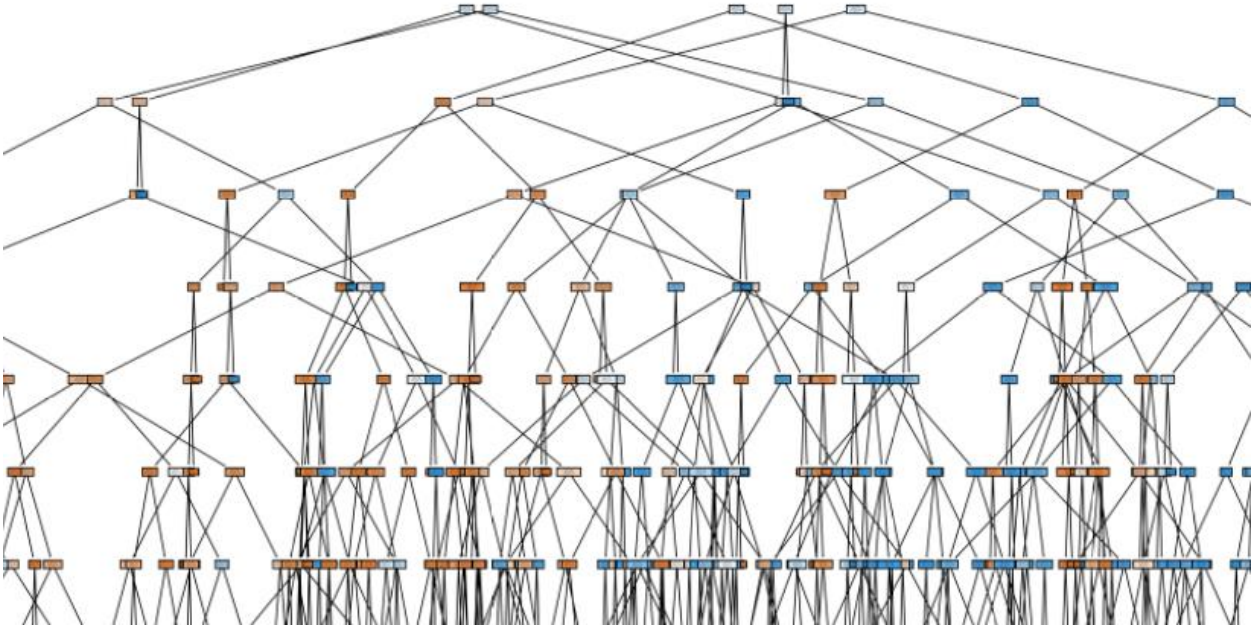

Figure 40: Random Forest Tree

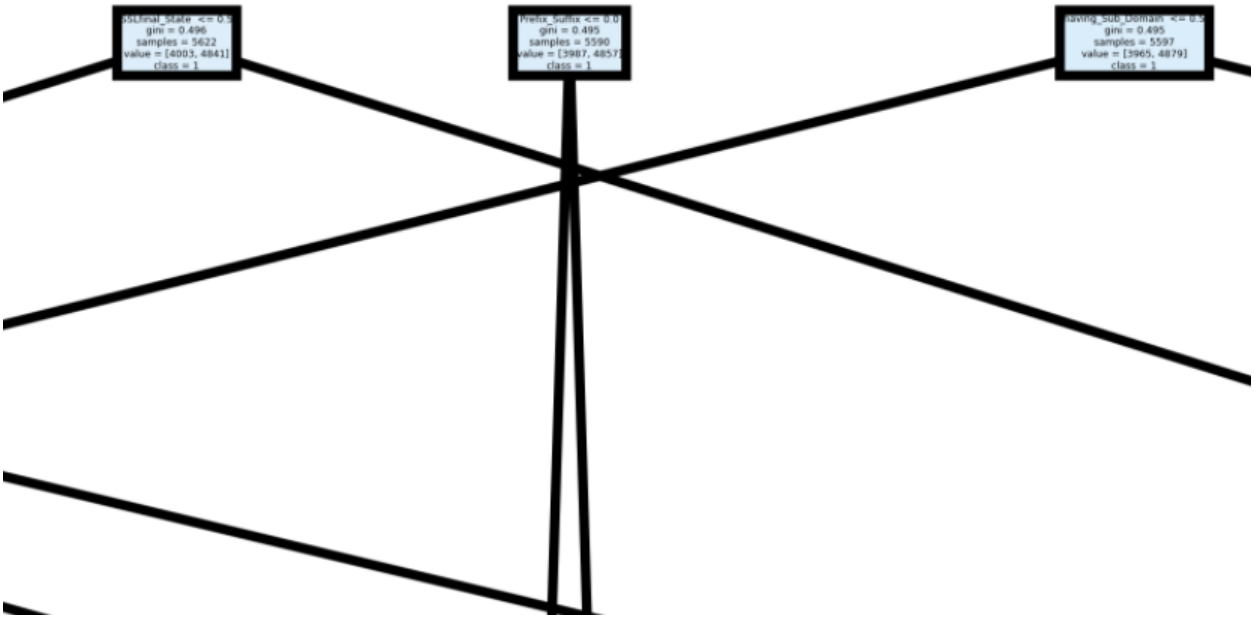

Figure 41: Random Forest Tree sample

The last step is to check the feature importance in the model:

```
# checking the feature importance in the model
plt.figure(figsize=(9,7))
n_features = x_train.shape[1]
plt.barh(range(n_features), forest.feature_importances_, align='center')
plt.yticks(np.arange(n_features), x_train.columns)
plt.xlabel("Feature importance")
plt.ylabel("Feature")
plt.show()
```

Figure 42: Plotting Random Forest Feature Importance

Output:

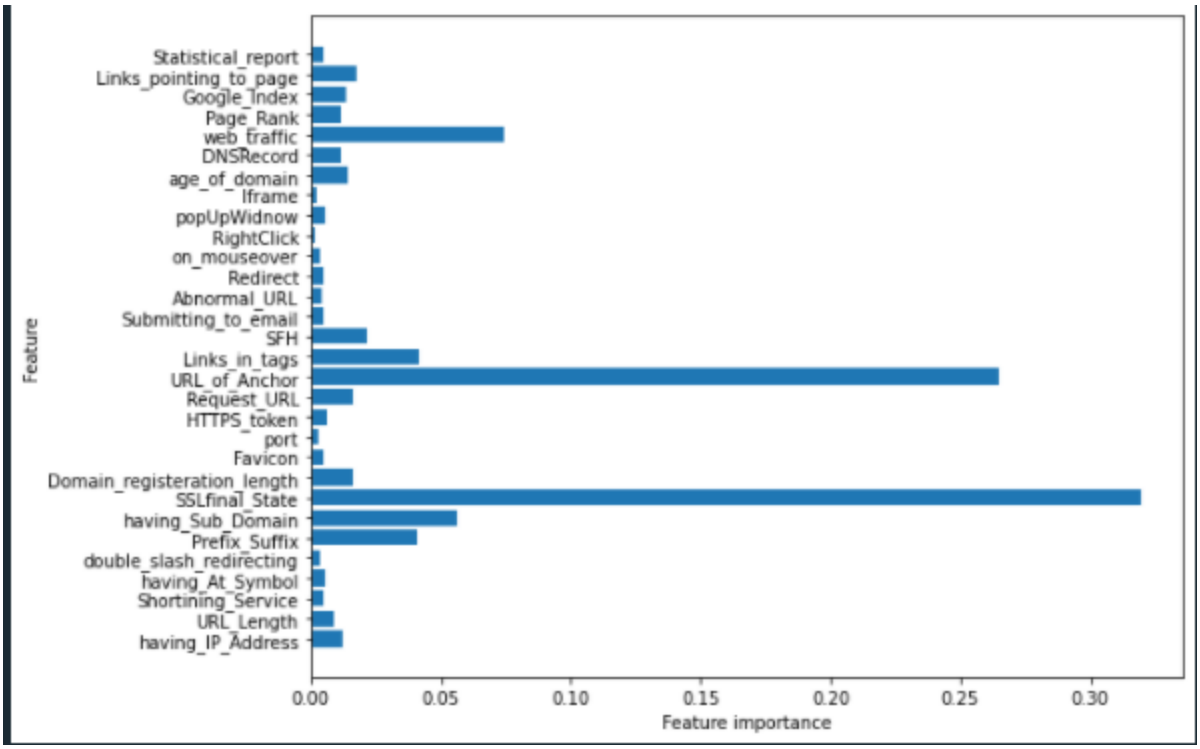

Figure 43: Random Forest Feature Importance

c. SVM Algorithm

In machine learning, support vector machines (SVMs, also support vector networks) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier.

```
# Support vector machine model
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score, LeaveOneOut

# instantiate the model
svm = SVC(kernel='linear', C=1.0, random_state=12)

# fit the model
svm.fit(x_train, y_train)
```

Figure 44: svm.SVC

After we fit the model, we are ready to use it to predict the test data.

```
#predicting the target value from the model for the samples
y_test_svm = svm.predict(x_test)
y_train_svm = svm.predict(x_train)

#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))
print("Classification report - \n", classification_report(y_test, y_test_svm))
print(confusion_matrix(y_test, y_test_svm))
```

Figure 45: SVM accuracy score

Performance Evaluation:

```
SVM: Accuracy on training Data: 0.928
SVM : Accuracy on test Data: 0.929
Classification report -
              precision    recall  f1-score   support

          -1       0.93      0.90      0.92       956
           1       0.93      0.95      0.94      1255

    accuracy                           0.93      2211
   macro avg       0.93      0.93      0.93      2211
weighted avg       0.93      0.93      0.93      2211

[[ 861   95]
 [  63 1192]]
```

Figure 46: SVM performance evaluation

9. Comparison of Models

To compare the models' performance, a data frame is created. The columns of this data frame are the lists created to store the results of the model.

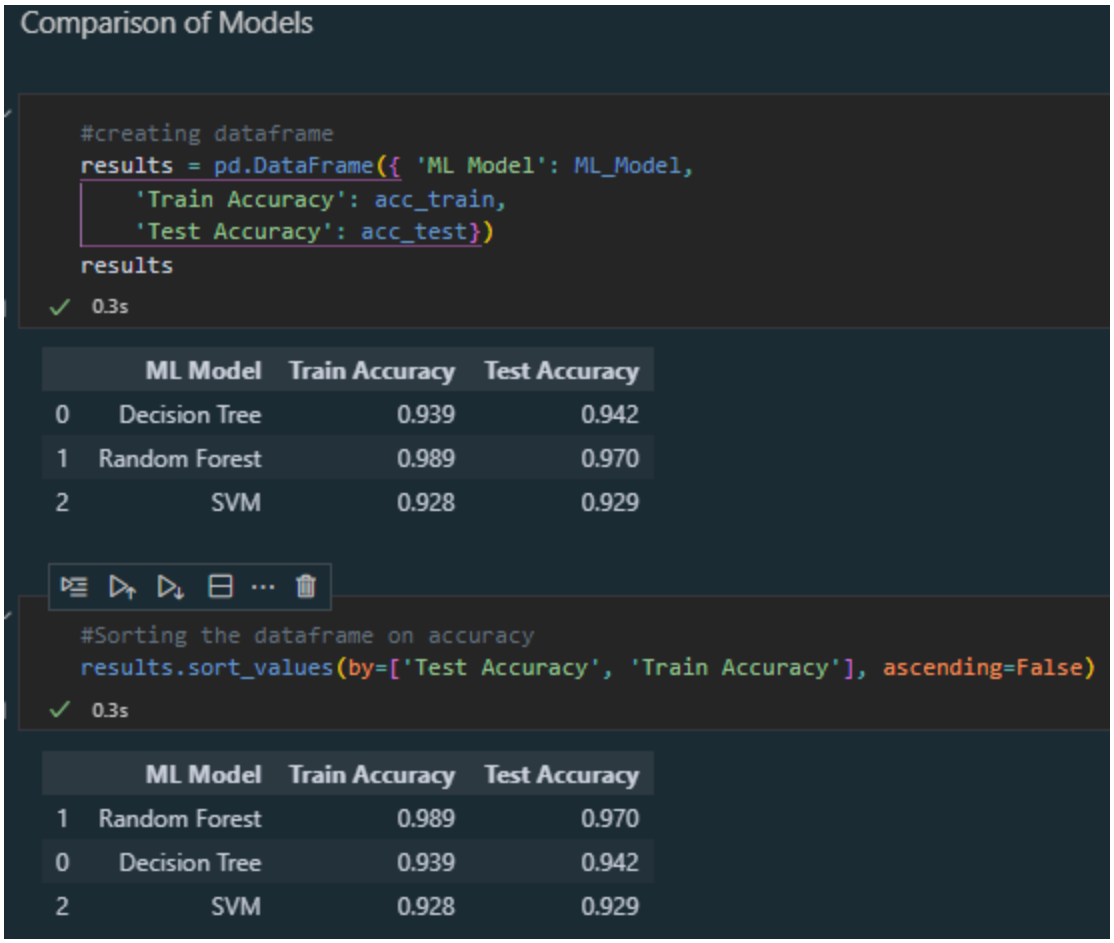

Figure 47: Comparisons of Model

The models are evaluated, and the considered metric is accuracy. From the above figures, it is shown that Random Forest gives better performance. The last step is to save and load the model and test out the model on some sample data.

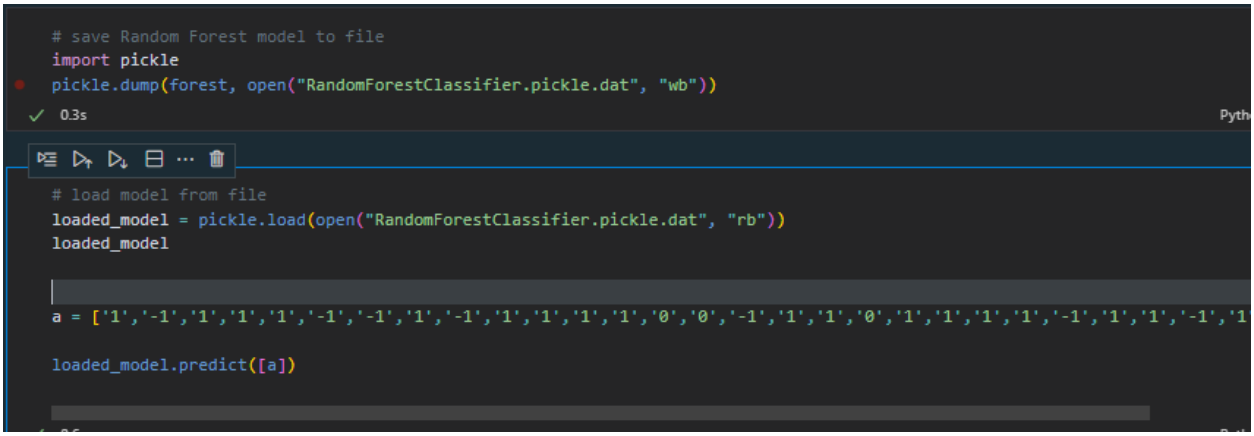

Figure 48: Loading and testing Model

Predicting the data:

```
array([1], dtype=int64)
```

Figure 49: Predicting Result

This test was conducted on an 80-20 split, further splitting testing was done. The comparison of the models when the data is split on an 80-20, 60-40, and 50-50 are:

	ML Model	Train Accuracy	Test Accuracy
1	Random Forest	0.989	0.970
0	Decision Tree	0.939	0.942
2	SVM	0.928	0.929

Figure 50: 80-20 split data performance evaluation

	ML Model	Train Accuracy	Test Accuracy
1	Random Forest	0.989	0.966
0	Decision Tree	0.942	0.937
2	SVM	0.930	0.926

Figure 51: 60-40 split data performance evaluation

	ML Model	Train Accuracy	Test Accuracy
1	Random Forest	0.990	0.964
0	Decision Tree	0.942	0.938
2	SVM	0.931	0.926

Figure 52: 50-50 split data performance evaluation

Table1: Comparison of Models

In all cases, the Random Forest algorithm performs much better than the rest.

10. Conclusion

This project aims to enhance detection methods to detect phishing websites using machine learning technology. Detection accuracy of 97.0% was achieved using the Random Forest algorithm with the lowest false positive rate. The result shows that classifiers give better performance when more instances and more attributes was used.

The problem of phishing cannot be eradicated, nonetheless can be reduced by combating it in two ways, improving targeted anti-phishing procedures and techniques and informing the public on how fraudulent phishing websites can be detected and identified. To combat the ever evolving and complexity of phishing attacks and tactics, ML anti-phishing techniques are essential.

The outcome of this project reveals that the proposed method presents superior results rather than the existing deep learning methods. It has achieved better accuracy and F1—score with a limited amount of time. The future direction of this project is to develop an unsupervised deep learning method to generate insight from a URL.

The algorithms used on this dataset have never been used on it. The few other projects done are on different datasets with not many attributes, I even found out missing attributes from other datasets that were very important to the decision making. It's a new data with new attributes, I believe my contributions are using this data and using the classifier algorithms and post and pre pruning the data to avoid overfitting was one of the challenges. Other projects and literature reviews I have read have not applied this classification problems to this kind of data, with this many attributes, they have not used this new and updated dataset and have not applied Pre-Pruning and Post Pruning. So that was a first and it really made the data easy to apply my models onto it.

Working on this project was very knowledgeable and worth the effort. In the future, this project could be used as a browser extension or an application with a user interface that users can use to detect phishing websites more easily.

11. References

[1] (PDF) Phishing Website Detection using Machine Learning Algorithms (researchgate.net)

[2] Phishing URL Detection with ML. Phishing is a form of fraud in which… | by Ebubekir Büber | Towards Data Science

[3] Phishing-Website-Detection-by-Machine-Learning-Techniques

[4] UCI Machine Learning Repository: Phishing Websites Data Set

[5] Detecting phishing websites using machine learning technique (plos.org)

[6] Phishing - Wikipedia

[7] Phishing Definition (investopedia.com)