PHISHING WEBSITES (Decision Tree Classisfiers)

INTRODUCTION

This discussion is set to explore a dataset(phishing_dataset.csv) to determine whether a website is a phishing website(websites that masquarade as other know or trusted websites to get personal data from the end user.) In this analysis a class of -1 is used to represent a non-phishing website and a 1 represents a phishing website.

DATA EXPLORATION

Below is a summary of the data:

```
from google.colab import files
data_to_load = files.upload()
```


Choose Files phishing_dataset.csv

- **phishing_dataset.csv**(application/vnd.ms-excel) - 855445 bytes, last modified: 30/07/2020 - 100% done

Saving phishing_dataset.csv to phishing_dataset (1).csv

```
import pandas as pd
import numpy as np
```

```
import io
theData = pd.read_csv(io.BytesIO(data_to_load['phishing_dataset.csv']))
```

```
theData.head().T
```

index	1	2	3	4	5
having_IPhaving_IP_Address	-1	1	1	1	1
URLURL_Length	1	1	0	0	0
Shortining_Service	1	1	1	1	-1
having_At_Symbol	1	1	1	1	1
double_slash_redirecting	-1	1	1	1	1
Prefix_Suffix	-1	-1	-1	-1	-1
having_Sub_Domain	-1	0	-1	-1	1
SSLfinal_State	-1	1	-1	-1	1
Domain_registeration_length	-1	-1	-1	1	-1
Favicon	1	1	1	1	1
port	1	1	1	1	1
HTTPS_token	-1	-1	-1	-1	1
Request_URL	1	1	1	-1	1
URL_of_Anchor	-1	0	0	0	0
Links_in_tags	1	-1	-1	0	0
SFH	-1	-1	-1	-1	-1
Submitting_to_email	-1	1	-1	1	1
Abnormal_URL	-1	1	-1	1	1
Redirect	0	0	0	0	0
on_mouseover	1	1	1	1	-1
RightClick	1	1	1	1	1
popUpWidnow	1	1	1	1	-1

```
theData.describe().T
```

index	11055.0	5528.000000	3191.447947	1.0	2764.5	5528.0	829
having_IPhaving_IP_Address	11055.0	0.313795	0.949534	-1.0	-1.0	1.0	
URLURL_Length	11055.0	-0.633198	0.766095	-1.0	-1.0	-1.0	-
Shortining_Service	11055.0	0.738761	0.673998	-1.0	1.0	1.0	
having_At_Symbol	11055.0	0.700588	0.713598	-1.0	1.0	1.0	
double_slash_redirecting	11055.0	0.741474	0.671011	-1.0	1.0	1.0	
Prefix_Suffix	11055.0	-0.734962	0.678139	-1.0	-1.0	-1.0	-
having_Sub_Domain	11055.0	0.063953	0.817518	-1.0	-1.0	0.0	
SSLfinal_State	11055.0	0.250927	0.911892	-1.0	-1.0	1.0	
Domain_registeration_length	11055.0	-0.336771	0.941629	-1.0	-1.0	-1.0	
Favicon	11055.0	0.628584	0.777777	-1.0	1.0	1.0	
port	11055.0	0.728268	0.685324	-1.0	1.0	1.0	
HTTPS_token	11055.0	0.675079	0.737779	-1.0	1.0	1.0	
Request_URL	11055.0	0.186793	0.982444	-1.0	-1.0	1.0	
URL_of_Anchor	11055.0	-0.076526	0.715138	-1.0	-1.0	0.0	
Links_in_tags	11055.0	-0.118137	0.763973	-1.0	-1.0	0.0	
SFH	11055.0	-0.595749	0.759143	-1.0	-1.0	-1.0	-
Submitting_to_email	11055.0	0.635640	0.772021	-1.0	1.0	1.0	
Abnormal_URL	11055.0	0.705292	0.708949	-1.0	1.0	1.0	
Redirect	11055.0	0.115694	0.319872	0.0	0.0	0.0	
on_mouseover	11055.0	0.762099	0.647490	-1.0	1.0	1.0	
RightClick	11055.0	0.913885	0.405991	-1.0	1.0	1.0	
popUpWidnow	11055.0	0.613388	0.789818	-1.0	1.0	1.0	
Iframe	11055.0	0.816915	0.576784	-1.0	1.0	1.0	
age_of_domain	11055.0	0.061239	0.998168	-1.0	-1.0	1.0	

We will then rename the column 'Result' to 'Class' and transform the data so that the column doesn't have any negative numbers because that slows down the performance of the model

```
theData.rename(columns={'Result':'Class'}, inplace=True)
theData['Class']=theData['Class'].map({-1:0,1:1})
```

Links_pointing_to_page	11055.0	0.344007	0.569944	-1.0	0.0	0.0	

```
theData.head().T
```

index	1	2	3	4	5
having_IPhaving_IP_Address	-1	1	1	1	1
URLURL_Length	1	1	0	0	0
Shortining_Service	1	1	1	1	-1
having_At_Symbol	1	1	1	1	1
double_slash_redirecting	-1	1	1	1	1
Prefix_Suffix	-1	-1	-1	-1	-1
having_Sub_Domain	-1	0	-1	-1	1
SSLfinal_State	-1	1	-1	-1	1
Domain_registeration_length	-1	-1	-1	1	-1
Favicon	1	1	1	1	1
port	1	1	1	1	1
HTTPS_token	-1	-1	-1	-1	1
Request_URL	1	1	1	-1	1
URL_of_Anchor	-1	0	0	0	0
Links_in_tags	1	-1	-1	0	0
SFH	-1	-1	-1	-1	-1
Submitting_to_email	-1	1	-1	1	1
Abnormal_URL	-1	1	-1	1	1
Redirect	0	0	0	0	0
on_mouseover	1	1	1	1	-1
RightClick	1	1	1	1	1
popUpWidnow	1	1	1	1	-1
Iframe	1	1	1	1	1
age_of_domain	-1	-1	1	-1	-1
DNSRecord	-1	-1	-1	-1	-1
web_traffic	-1	0	1	1	0
Page_Rank	-1	-1	-1	-1	-1
Google_Index	1	1	1	1	1
Links_pointing_to_page	1	1	0	-1	1
Statistical_report	-1	1	-1	1	1

```
theData['Class'].unique()
```

```
array([0, 1])
```

CREATING OUR MODEL

We first need to split the data into test and training with the 20% to 80% ratio respectively, which is normally most ideal

```
from sklearn.model_selection import train_test_split


x=theData.iloc[:,0:30].values.astype(int)
y=theData.iloc[:,30].values.astype(int)


x
```

```
array([[    1,    -1,     1, ...,    -1,     1,     1],
       [    2,     1,     1, ...,    -1,     1,     1],
       [    3,     1,     0, ...,    -1,     1,     0],
       ...,
       [11053,     1,    -1, ...,    -1,     1,     0],
       [11054,    -1,    -1, ...,    -1,     1,     1],
       [11055,    -1,    -1, ...,    -1,    -1,     1]])
```

```
y
```

```
array([-1,  1, -1, ...,  1,  1, -1])
```

```
np.random.seed(7)


x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2, random_state=1)


#Installing weights and bases
!pip install wandb
```

```
     |████████████████████████████████| 112kB 19.6MB/s
Requirement already satisfied: nvidia-ml-py3>=7.352.0 in /usr/local/lib/python3.6/dis
Requirement already satisfied: psutil>=5.0.0 in /usr/local/lib/python3.6/dist-package
Collecting configparser>=3.8.1
  Downloading https://files.pythonhosted.org/packages/4b/6b/01baa293090240cf0562cc5ec
Collecting gql==0.2.0
  Downloading https://files.pythonhosted.org/packages/c4/6f/cf9a3056045518f06184e804b
Collecting shortuuid>=0.5.0
  Downloading https://files.pythonhosted.org/packages/25/a6/2ecc1daa6a304e7f1b216f089
Requirement already satisfied: requests>=2.0.0 in /usr/local/lib/python3.6/dist-packa
Requirement already satisfied: PyYAML>=3.10 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.6/dist-packages
Collecting subprocess32>=3.5.3
  Downloading https://files.pythonhosted.org/packages/32/c8/564be4d12629b912ea431f1a5
     |████████████████████████████████| 102kB 8.2MB/s
Requirement already satisfied: Click>=7.0 in /usr/local/lib/python3.6/dist-packages (
Collecting GitPython>=1.0.0
  Downloading https://files.pythonhosted.org/packages/f9/1e/a45320cab182bf1c8656107b3
     |████████████████████████████████| 163kB 16.7MB/s
Collecting docker-pycreds>=0.4.0
  Downloading https://files.pythonhosted.org/packages/f5/e8/f6bd1eee09314e7e6dee49cbe
Collecting watchdog>=0.8.3
  Downloading https://files.pythonhosted.org/packages/0e/06/121302598a4fc01aca942d937
     |████████████████████████████████| 102kB 9.0MB/s
Requirement already satisfied: certifi in /usr/local/lib/python3.6/dist-packages (fro
Requirement already satisfied: urllib3>=1.10.0 in /usr/local/lib/python3.6/dist-packa
Collecting graphql-core<2,>=0.5.0
  Downloading https://files.pythonhosted.org/packages/b0/89/00ad5e07524d8c523b14d70c6
     |████████████████████████████████| 71kB 6.5MB/s
Requirement already satisfied: promise<3,>=2.0 in /usr/local/lib/python3.6/dist-packa
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.6/dist-pac
Collecting gitdb<5,>=4.0.1
  Downloading https://files.pythonhosted.org/packages/48/11/d1800bca0a3bae820b84b7d81
     |████████████████████████████████| 71kB 7.9MB/s
Collecting pathtools>=0.1.1
  Downloading https://files.pythonhosted.org/packages/e7/7f/470d6fcdf23f9f3518f6b0b76
Collecting smmap<4,>=3.0.1
  Downloading https://files.pythonhosted.org/packages/b0/9a/4d409a6234eb940e6a78dfdfc
Building wheels for collected packages: gql, subprocess32, watchdog, graphql-core, pa
  Building wheel for gql (setup.py) ... done
  Created wheel for gql: filename=gql-0.2.0-cp36-none-any.whl size=7630 sha256=fcd4a8
  Stored in directory: /root/.cache/pip/wheels/ce/0e/7b/58a8a5268655b3ad74feef5aa9794
  Building wheel for subprocess32 (setup.py) ... done
  Created wheel for subprocess32: filename=subprocess32-3.5.4-cp36-none-any.whl size=
  Stored in directory: /root/.cache/pip/wheels/68/39/1a/5e402bdfdf004af1786c8b853fd92
  Building wheel for watchdog (setup.py) ... done
```

Create an instance of the Decision Tree Classifier model and fit it onto the trainging data

```
  Building wheel for graphql-core (setup.py) ... done
```

```
from sklearn.metrics import accuracy_score, precision_recall_fscore_support,classification
from sklearn import tree
import wandb
import time
```

```
Installing collected packages: sentry-sdk, configparser, graphql-core, gql, shortuuid
```

We'll first create a Decision Tree with the entropy criterion and a max deth of # to see what exactly we're working with

```
clf=tree.DecisionTreeClassifier(criterion='entropy',max_depth=3)
```

```
clf=clf.fit(x,y)
clf
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=3, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

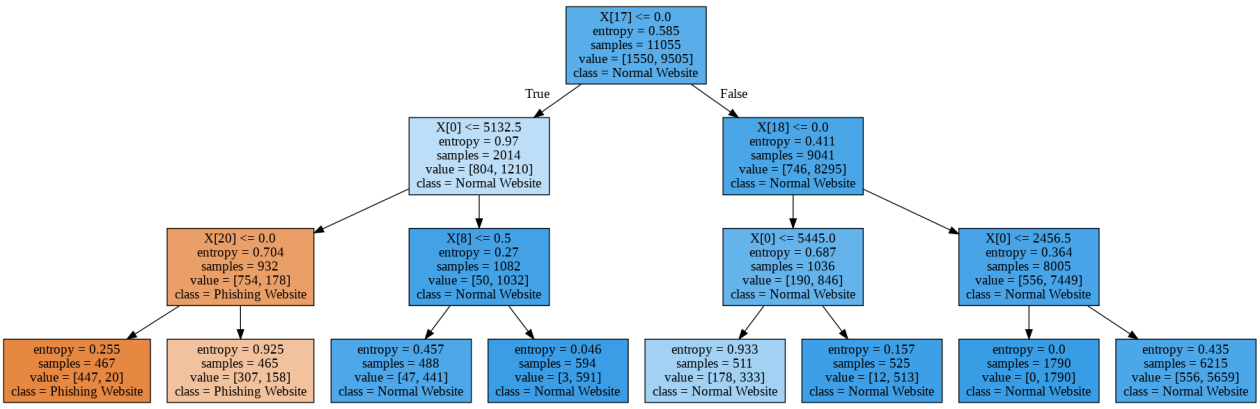
VISUALIZING THE DECISION TREE

A visual representation of the model we created above will make it easier to understand the classification

```
import pydotplus
from IPython.display import Image
```

```
dot_data=tree.export_graphviz(clf, class_names=['Phishing Website','Normal Website'],fille
```

```
graph=pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())
```



MAKING SENSE OF THE MODEL

We can now analyse our model's performance from Weights and Biases with the default hyperparameters set in place

```
def train_eval_pipeline(model,train_data,test_data,name):
```

```
  #Initialize Weights and Biases
  wandb.init(project="Decision Tree Example Using Phishing Dataset", name=name)
  #segregate the datasets
  (x_train,y_train)=train_data
  (x_test,y_test)=test_data

  # train and log all the necessary metrics
  start=time.time()
  model.fit(x_train,y_train)
  end=time.time()-start
  prediction=model.predict(x_test)
  wandb.log({"accuracy": accuracy_score(y_test,prediction)*100.0,
             "precision": precision_recall_fscore_support(y_test,prediction,average='macro
             'recall': precision_recall_fscore_support(y_test,prediction,average='macro')[
             'Training_time':end})
  print("Accuracy score of the Decision Tree Classifier with default hyperparameters {0:.2
  .format(accuracy_score(y_test,prediction)*100.0))
  print('\n')
  print("---Classification report of the Decision Tree Classifier with default hyperparame
  print('\n')
  print(classification_report(y_test,prediction,target_names=['Phishing Websites','Normal

train_eval_pipeline(clf, (x_train,y_train),(x_test,y_test),'Decision Tree Classifier')
```

```
Logging results to Weights & Biases (Documentation).
Project page:
https://app.wandb.ai/shelton17/Decision%20Tree%20Example%20Using%20Phishing%20Dataset
Run page:
https://app.wandb.ai/shelton17/Decision%20Tree%20Example%20Using%20Phishing%20Dataset/runs/1
Accuracy score of the Decision Tree Classifier with default hyperparameters 91.41%


---Classification report of the Decision Tree Classifier with default hyperparameters


                   precision    recall  f1-score   support

Phishing Websites       0.84      0.50      0.63       318
  Normal Websites       0.92      0.98      0.95      1893

         accuracy                           0.91      2211
        macro avg       0.88      0.74      0.79      2211
     weighted avg       0.91      0.91      0.90      2211
```

Ata a maximum depth of 3 and using the set default hyperparameters, we get an accuracy level of ***91.41%***

MODEL OPTIMIZATION

Optimization by maximum depth increase

Find attached a pdf file with the weights and biases image screenshot

Increasing the max_depth of our tree canincrease our overall accuracy but er risk overfitting the model which inturn will cause the accuracy to be lower.

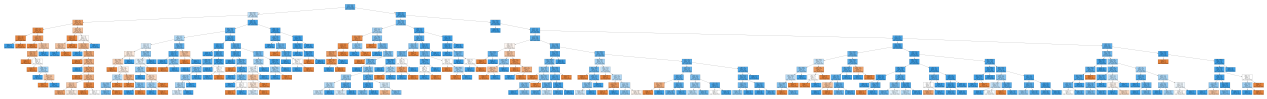
To test the most optimum of maximum depth we will use 11 and 27 as our max depths and

```
clf2=tree.DecisionTreeClassifier(criterion='entropy',max_depth=11)
```

```
clf2=clf2.fit(x,y)
clf2
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=11, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

```
dot_data=tree.export_graphviz(clf2, class_names=['Phishing Website','Normal Website'],fill
graph=pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())
```



```
train_eval_pipeline(clf2, (x_train,y_train),(x_test,y_test),'Decision Tree Classifier 2')
```

```
Logging results to Weights & Biases (Documentation).
Project page:
https://app.wandb.ai/shelton17/Decision%20Tree%20Example%20Using%20Phishing%20Dataset
Run page:
https://app.wandb.ai/shelton17/Decision%20Tree%20Example%20Using%20Phishing%20Dataset/runs/2
Accuracy score of the Decision Tree Classifier with default hyperparameters 92.99%


---Classification report of the Decision Tree Classifier with default hyperparameters


                    precision    recall  f1-score   support

Phishing Websites       0.84      0.63      0.72       318
  Normal Websites       0.94      0.98      0.96      1893

         accuracy                           0.93      2211
        macro avg       0.89      0.80      0.84      2211
     weighted avg       0.93      0.93      0.93      2211
```

The accuracy using a maximum depth of 11 is:* 92.99%*

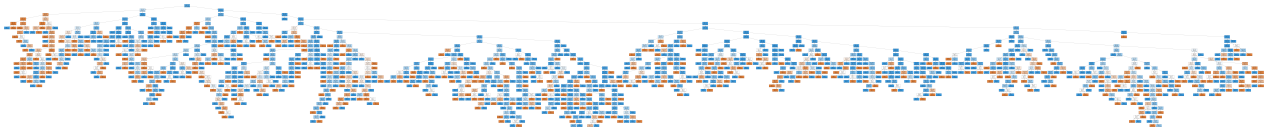
```
clf3=tree.DecisionTreeClassifier(criterion='entropy',max_depth=27)
```

```
clf3=clf3.fit(x,y)
clf3
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=27, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

```
dot_data=tree.export_graphviz(clf3, class_names=['Phishing Website','Normal Website'],fill
graph=pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())
```

```
dot: graph is too large for cairo-renderer bitmaps. Scaling by 0.720756 to fit
```



```
train_eval_pipeline(clf3, (x_train,y_train),(x_test,y_test),'Decision Tree Classifier 3')
```

```
Logging results to Weights & Biases (Documentation).
Project page:
https://app.wandb.ai/shelton17/Decision%20Tree%20Example%20Using%20Phishing%20Dataset
Run page:
https://app.wandb.ai/shelton17/Decision%20Tree%20Example%20Using%20Phishing%20Dataset/runs/2
Accuracy score of the Decision Tree Classifier with default hyperparameters 92.31%


---Classification report of the Decision Tree Classifier with default hyperparameters


                    precision    recall  f1-score   support

Phishing Websites       0.73      0.75      0.74       318
  Normal Websites       0.96      0.95      0.95      1893

         accuracy                           0.92      2211
        macro avg       0.84      0.85      0.85      2211
     weighted avg       0.92      0.92      0.92      2211
```

The accuracy using a maximum depth of 11 is:* 92.31%*

Optimization by Tuning hyperparameters

```
#Using Random Search instead
#Import the GridSearchCV
```

```
from sklearn.model_selection import RandomizedSearchCV


min_weight_fraction_leaf=[0.1,0.01,0.001]
max_depth=[3,11,37]
ccp_alpha=[0.1,0.01,0.001]
#Create a dictionary where tol and max_iteration are keys and the list of their
#values are the corresponding values. Key -> value
param_grid=dict(min_weight_fraction_leaf=min_weight_fraction_leaf,max_depth=max_depth,ccp_


#Creating an instance of RandomSearchCV with the required parameters
random_model=RandomizedSearchCV(estimator=clf2,param_distributions=param_grid,cv=5)


random_model_result=random_model.fit(x_train,y_train)
#summary
best_score, best_params=random_model_result.best_score_, random_model_result.best_params_
print("Best score: %.2f using %s" %(best_score*100, best_params))
```

```
Best score: 92.80 using {'min_weight_fraction_leaf': 0.001, 'max_depth': 37, 'ccp_alp
```

CONCLUSION

At the beginning of the discussion and analysis we ended up with a Decision Tree Model with an accuracy of 91.41%. We were however not satisfied with this and then took it a step further to try using two methods to optimize our model for better accuracy without overfitting the model on the training set.

When we opted to change the default hyperparameters and go for a random search we got an accuracy score of 92.80% as our best score. This was a *1.59* improvement from our original.

With the maximum depths however, we witnessed both an increase in accuracy and a decrease at the different levels we set. On the Second Decision Tree (depth 11) we were able to achieve an improved accuracy by *1.54%* a significant leap in statisctis but this improvement wasn't kept with additional depths, at a depth of 27 we scored a 92.31%, still better than the original but an alarm that we'd left safe territories and were on the edge of overfitting.

It is important to note that we achieved our most optimised level by setting our max depth to a level around optimum, which in this case was 11, scoring us an accuracy of 92.99%

This model performed better than the previous classification model using logistic regression that gave us a 91.72% after optimization during class.

Weights And Biases Screenshot

Find attached a pdf file with the weights and biases image screenshot