Decision Tree Classifiers Using the Phishing Dataset

Introduction

The main objective of this analysis is to create a machine learning model that uses Decision Trees to determine whether a website is malicious (a phishing website). We are going to use the dataset "phishing.csv". It has several features that are used to determine the class (known as "result" in this dataset), which is binary in this scenario. A class of -1 is a website that is not malicious (Normal) while a class of 1 is a malicious website (Phishing).

Summary of Data

We can summarize the data as follows:

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

```
myData=pd.read_csv("/content/phishing.csv")
```

```
myData.head().T
```

	0	1	2	3	4
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

```
myData.describe().T
```

	count	mean	std	min	25%	50%	
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	
RightClick	11055.0	0.913885	0.405991	-1.0	1.0	1.0	

We will rename **result** column to **class** and transform the data so the column does not have negative numbers. This is because negative numbers reduce performance

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

```
myData.head().T
```

	0	1	2	3	4
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

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

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

Model Creation

```
#We will split the data into training set and testing set to 80% and 20% respectively
from sklearn.model_selection import train_test_split
```

```
x=myData.iloc[:,0:30].values.astype(int)
```

```
y=myData.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)
```

```
!pip install wandb
```

```
Requirement already satisfied: wandb in /usr/local/lib/python3.6/dist-packages (0.9.
Requirement already satisfied: configparser>=3.8.1 in /usr/local/lib/python3.6/dist-
Requirement already satisfied: shortuuid>=0.5.0 in /usr/local/lib/python3.6/dist-pac
Requirement already satisfied: nvidia-ml-py3>=7.352.0 in /usr/local/lib/python3.6/di
Requirement already satisfied: GitPython>=1.0.0 in /usr/local/lib/python3.6/dist-pac
Requirement already satisfied: subprocess32>=3.5.3 in /usr/local/lib/python3.6/dist-
Requirement already satisfied: Click>=7.0 in /usr/local/lib/python3.6/dist-packages 
Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: psutil>=5.0.0 in /usr/local/lib/python3.6/dist-packag
Requirement already satisfied: PyYAML>=3.10 in /usr/local/lib/python3.6/dist-package
Requirement already satisfied: requests>=2.0.0 in /usr/local/lib/python3.6/dist-pack
Requirement already satisfied: docker-pycreds>=0.4.0 in /usr/local/lib/python3.6/dis
Requirement already satisfied: python-dateutil>=2.6.1 in /usr/local/lib/python3.6/di
Requirement already satisfied: watchdog>=0.8.3 in /usr/local/lib/python3.6/dist-pack
Requirement already satisfied: sentry-sdk>=0.4.0 in /usr/local/lib/python3.6/dist-pa
Requirement already satisfied: gql==0.2.0 in /usr/local/lib/python3.6/dist-packages 
Requirement already satisfied: gitdb<5,>=4.0.1 in /usr/local/lib/python3.6/dist-pack
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.6/dist-package
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.6/dist-pa
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-p
Requirement already satisfied: pathtools>=0.1.1 in /usr/local/lib/python3.6/dist-pac
Requirement already satisfied: promise<3,>=2.0 in /usr/local/lib/python3.6/dist-pack
Requirement already satisfied: graphql-core<2,>=0.5.0 in /usr/local/lib/python3.6/di
Requirement already satisfied: smmap<4,>=3.0.1 in /usr/local/lib/python3.6/dist-pack
```

```
#instantiate the Decision Tree Classifier model and fit it to the training data

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

We will create a Decision Tree Classifier **clf** that uses the **entropy** criterion and has a max depth of 3

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

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

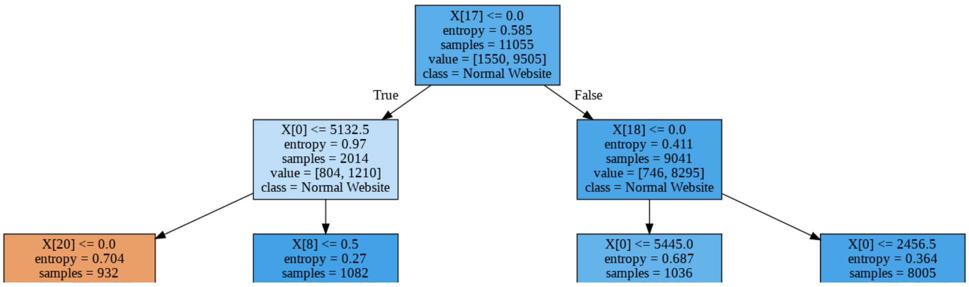
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=3, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
   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

Data Visualization

```
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())
```



Analyzing the Model

We can analyze how well our model performed by using the train_eval_pipeline model we have created along with Weights and Biases. Default hyperparameters have been used on the wandb function.

```
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 hyperparameter values
  .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')
```

```
wandb: ERROR Not authenticated.  Copy a key from https://app.wandb.ai/authorize
API Key: ··········
wandb: Appending key for api.wandb.ai to your netrc file: /root/.netrc
```

Logging results to [Weights & Biases](https://app.wandb.ai) [(Documentation)](https://docs.wandb.com).
Project page:
https://app.wandb.ai/rvpatel/Decision%20Tree%20Example%20Using%20Phishing%20Dataset
Run page:
https://app.wandb.ai/rvpatel/Decision%20Tree%20Example%20Using%20Phishing%20Dataset/runs/kenl

```
Accuracy score of the Decision Tree Classifier with default hyperparameter values 91
```

From the function we have created we can see that the default hyperparameter values produce an accuracy score of 91.41%.

Model Optimization

```
    Normal websites       0.92      0.98      0.95      1895
```

While the accuracy of our model with default parameters is high, we can potentially improve upon the model using several methods. We will test two optimization methods:

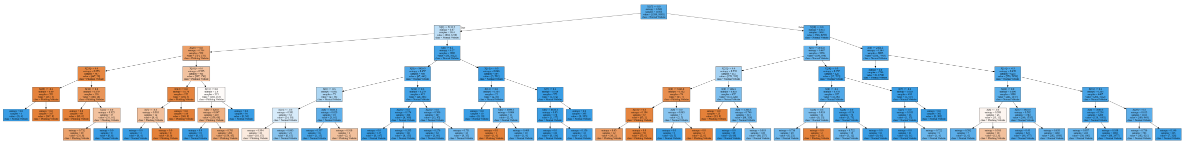
-Increasing the max depth of the decision tree. -Uploading the hyperparameters on Wandb.

```
#we will use the max depths 6 and 10  to see if there is an improvement
clf2=tree.DecisionTreeClassifier(criterion='entropy',max_depth=6)
```

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

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=6, 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')
```

Logging results to Weights & Biases (Documentation).
Project page:
https://app.wandb.ai/rvpatel/Decision%20Tree%20Example%20Using%20Phishing%20Dataset
Run page:
https://app.wandb.ai/rvpatel/Decision%20Tree%20Example%20Using%20Phishing%20Dataset/runs/1cz8

```
Accuracy score of the Decision Tree Classifier with default hyperparameter values 92


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


                   precision    recall  f1-score   support

Phishing Websites       0.89      0.58      0.70       318
  Normal Websites       0.93      0.99      0.96      1893

         accuracy                           0.93      2211
        macro avg       0.91      0.78      0.83      2211
     weighted avg       0.93      0.93      0.92      2211
```

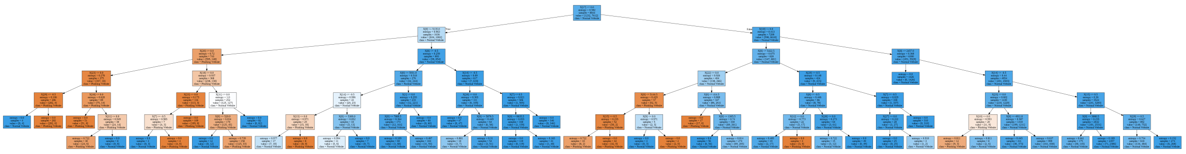
the accuracy score for the max depth of 6 is 92.90%. this is more than the score of the max depth in 3

```
clf3=tree.DecisionTreeClassifier(criterion='entropy',max_depth=10)
```

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

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=10, 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')
```

Logging results to Weights & Biases (Documentation).
Project page:
https://app.wandb.ai/rvpatel/Decision%20Tree%20Example%20Using%20Phishing%20Dataset
Run page:
https://app.wandb.ai/rvpatel/Decision%20Tree%20Example%20Using%20Phishing%20Dataset/runs/1k7

```
Accuracy score of the Decision Tree Classifier with default hyperparameter values 92


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


                   precision    recall  f1-score   support

Phishing Websites       0.89      0.58      0.70       318
  Normal Websites       0.93      0.99      0.96      1893

         accuracy                           0.93      2211
        macro avg       0.91      0.78      0.83      2211
     weighted avg       0.93      0.93      0.92      2211
```

the accuracy score for max depth 6 and 10 is similar

Conclusion

From the logistic regression we had an accuracy score of 91.68% while in the decision tree we had an accuracy score of 92.90% thus proving that decision trees are more accurate