Source Code

In[1]: importing required libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')
import seaborn as sns
from sklearn import metrics
import warnings
warnings.filterwarnings('ignore')
```

In[2]: Loading data into dataframe

```
data = pd.read_csv("phishing.csv")
data.head()
```

In[3]: Shape of dataframe

```
data.shape
```

In[4]: Listing the features of the dataset

```
data.columns
```

In[5]: pairplot for particular features

```
df = data[['PrefixSuffix-', 'SubDomains', 'HTTPS','AnchorURL','WebsiteTraffic','class']]
sns.pairplot(data = df,hue="class",corner=True);
```

In[6]: Splitting the dataset into dependant and independant fetature

```
X = data.drop(["class"],axis =1)
y = data["class"]
```

In[7]: 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)
X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

In[8]: Creating holders to store the model performance results

```
ML_Model = []
accuracy = []
f1_score = []
recall = []
precision = []

#function to call for storing the results
def storeResults(model, a,b,c,d):
  ML_Model.append(model)
  accuracy.append(round(a, 3))
  f1_score.append(round(b, 3))
  recall.append(round(c, 3))
  precision.append(round(d, 3))
```

In[9]: Linear regression model

```
from sklearn.linear_model import LogisticRegression
#from sklearn.pipeline import Pipeline

# instantiate the model
log = LogisticRegression()

# fit the model
log.fit(X_train,y_train)
```

In[10]: predicting the target value from the model for the samples

```
y_train_log = log.predict(X_train)
y_test_log = log.predict(X_test)
```

In[11]: computing the accuracy, f1_score, Recall, precision of the model performance

```
acc_train_log = metrics.accuracy_score(y_train,y_train_log)
acc_test_log = metrics.accuracy_score(y_test,y_test_log)
print("Logistic Regression : Accuracy on training Data: {:.3f}".format(acc_train_log))
print("Logistic Regression : Accuracy on test Data: {:.3f}".format(acc_test_log))
print()

f1_score_train_log = metrics.f1_score(y_train,y_train_log)
f1_score_test_log = metrics.f1_score(y_test,y_test_log)
print("Logistic Regression : f1_score on training Data: {:.3f}".format(f1_score_train_log))
print("Logistic Regression : f1_score on test Data: {:.3f}".format(f1_score_test_log))
print()

recall_score_train_log = metrics.recall_score(y_train,y_train_log)
recall_score_test_log = metrics.recall_score(y_test,y_test_log)
print("Logistic Regression : Recall on training Data: {:.3f}".format(recall_score_train_log))
print("Logistic Regression : Recall on test Data: {:.3f}".format(recall_score_test_log))
print()

precision_score_train_log = metrics.precision_score(y_train,y_train_log)
precision_score_test_log = metrics.precision_score(y_test,y_test_log)
print("Logistic Regression : precision on training Data:
{:.3f}".format(precision_score_train_log))
print("Logistic Regression : precision on test Data: {:.3f}".format(precision_score_test_log))
```

In[12]: computing the classification report of the model

```
print(metrics.classification_report(y_test, y_test_log))
```

In[13]: storing the results. The below mentioned order of parameter passing is important.

```
storeResults('Logistic Regression',acc_test_log,f1_score_test_log,
             recall_score_train_log,precision_score_train_log)
```

In[14]: K-Nearest Neighbors Classifier model

```
from sklearn.neighbors import KNeighborsClassifier

# instantiate the model
knn = KNeighborsClassifier(n_neighbors=1)

# fit the model
knn.fit(X_train,y_train)
```

In[15]: predicting the target value from the model for the samples

```
y_train_knn = knn.predict(X_train)
y_test_knn = knn.predict(X_test)
```

In[16]: computing the accuracy,f1_score,Recall,precision of the model performance

```
acc_train_knn = metrics.accuracy_score(y_train,y_train_knn)
acc_test_knn = metrics.accuracy_score(y_test,y_test_knn)
print("K-Nearest Neighbors : Accuracy on training Data: {:.3f}".format(acc_train_knn))
print("K-Nearest Neighbors : Accuracy on test Data: {:.3f}".format(acc_test_knn))
print()

f1_score_train_knn = metrics.f1_score(y_train,y_train_knn)
```

```
f1_score_test_knn = metrics.f1_score(y_test,y_test_knn)
print("K-Nearest Neighbors : f1_score on training Data: {:.3f}".format(f1_score_train_knn))
print("K-Nearest Neighbors : f1_score on test Data: {:.3f}".format(f1_score_test_knn))
print()

recall_score_train_knn = metrics.recall_score(y_train,y_train_knn)
recall_score_test_knn = metrics.recall_score(y_test,y_test_knn)
print("K-Nearest Neighborsn : Recall on training Data:
{:.3f}".format(recall_score_train_knn))
print("Logistic Regression : Recall on test Data: {:.3f}".format(recall_score_test_knn))
print()

precision_score_train_knn = metrics.precision_score(y_train,y_train_knn)
precision_score_test_knn = metrics.precision_score(y_test,y_test_knn)
print("K-Nearest Neighbors : precision on training Data:
{:.3f}".format(precision_score_train_knn))
print("K-Nearest Neighbors : precision on test Data:
{:.3f}".format(precision_score_test_knn))
```

In[17]: computing the classification report of the model

```
print(metrics.classification_report(y_test, y_test_knn))
```

In[18]:

```
training_accuracy = []
test_accuracy = []
# try max_depth from 1 to 20
depth = range(1,20)
for n in depth:
    knn = KNeighborsClassifier(n_neighbors=n)

    knn.fit(X_train, y_train)
    # record training set accuracy
    training_accuracy.append(knn.score(X_train, y_train))
```

```
    # record generalization accuracy
    test_accuracy.append(knn.score(X_test, y_test))


#plotting the training & testing accuracy for n_estimators from 1 to 20
plt.plot(depth, training_accuracy, label="training accuracy")
plt.plot(depth, test_accuracy, label="test accuracy")
plt.ylabel("Accuracy")
plt.xlabel("n_neighbors")
plt.legend();
```

In[19]: Decision Tree Classifier model

```
from sklearn.tree import DecisionTreeClassifier

# instantiate the model
tree = DecisionTreeClassifier(max_depth=30)

# fit the model
tree.fit(X_train, y_train)
```

In[20]: predicting the target value from the model for the samples

```
y_train_tree = tree.predict(X_train)
y_test_tree = tree.predict(X_test)
```

In[21]: computing the accuracy, f1_score, Recall, precision of the model performance

```
acc_train_tree = metrics.accuracy_score(y_train,y_train_tree)
acc_test_tree = metrics.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()
```

```
f1_score_train_tree = metrics.f1_score(y_train,y_train_tree)
f1_score_test_tree = metrics.f1_score(y_test,y_test_tree)
print("Decision Tree : f1_score on training Data: {:.3f}".format(f1_score_train_tree))
print("Decision Tree : f1_score on test Data: {:.3f}".format(f1_score_test_tree))
print()

recall_score_train_tree = metrics.recall_score(y_train,y_train_tree)
recall_score_test_tree = metrics.recall_score(y_test,y_test_tree)
print("Decision Tree : Recall on training Data: {:.3f}".format(recall_score_train_tree))
print("Decision Tree : Recall on test Data: {:.3f}".format(recall_score_test_tree))
print()

precision_score_train_tree = metrics.precision_score(y_train,y_train_tree)
precision_score_test_tree = metrics.precision_score(y_test,y_test_tree)
print("Decision Tree : precision on training Data: {:.3f}".format(precision_score_train_tree))
print("Decision Tree : precision on test Data: {:.3f}".format(precision_score_test_tree))
```

In[22]: computing the classification report of the model

```
print(metrics.classification_report(y_test, y_test_tree))
```

In[23]:

```
training_accuracy = []
test_accuracy = []
# try max_depth from 1 to 30
depth = range(1,30)
for n in depth:
    tree_test = DecisionTreeClassifier(max_depth=n)

    tree_test.fit(X_train, y_train)
    # record training set accuracy
```

```
    training_accuracy.append(tree_test.score(X_train, y_train))
    # record generalization accuracy
    test_accuracy.append(tree_test.score(X_test, y_test))


#plotting the training & testing accuracy for max_depth from 1 to 30
plt.plot(depth, training_accuracy, label="training accuracy")
plt.plot(depth, test_accuracy, label="test accuracy")
plt.ylabel("Accuracy")
plt.xlabel("max_depth")
plt.legend();
```

In[24]: storing the results. The below mentioned order of parameter passing is important.

```
storeResults('Decision Tree',acc_test_tree,f1_score_test_tree,
             recall_score_train_tree,precision_score_train_tree)
```

In[25]: Random Forest Classifier Model

```
from sklearn.ensemble import RandomForestClassifier

# instantiate the model
forest = RandomForestClassifier(n_estimators=10)

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

In[26]: predicting the target value from the model for the samples

```
y_train_forest = forest.predict(X_train)
y_test_forest = forest.predict(X_test)
```

In[27]: computing the accuracy, f1_score, Recall, precision of the model performance

```
acc_train_forest = metrics.accuracy_score(y_train,y_train_forest)
acc_test_forest = metrics.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()

f1_score_train_forest = metrics.f1_score(y_train,y_train_forest)
f1_score_test_forest = metrics.f1_score(y_test,y_test_forest)
print("Random Forest : f1_score on training Data: {:.3f}".format(f1_score_train_forest))
print("Random Forest : f1_score on test Data: {:.3f}".format(f1_score_test_forest))
print()

recall_score_train_forest = metrics.recall_score(y_train,y_train_forest)
recall_score_test_forest = metrics.recall_score(y_test,y_test_forest)
print("Random Forest : Recall on training Data: {:.3f}".format(recall_score_train_forest))
print("Random Forest : Recall on test Data: {:.3f}".format(recall_score_test_forest))
print()

precision_score_train_forest = metrics.precision_score(y_train,y_train_forest)
precision_score_test_forest = metrics.precision_score(y_test,y_test_tree)
print("Random Forest : precision on training Data:
{:.3f}".format(precision_score_train_forest))
print("Random Forest : precision on test Data: {:.3f}".format(precision_score_test_forest))
```

In[28]: computing the classification report of the model

```
print(metrics.classification_report(y_test, y_test_forest))
```

In[29]:

```
training_accuracy = []
test_accuracy = []
# try max_depth from 1 to 20
depth = range(1,20)
for n in depth:
    forest_test =  RandomForestClassifier(n_estimators=n)

    forest_test.fit(X_train, y_train)
    # record training set accuracy
    training_accuracy.append(forest_test.score(X_train, y_train))
    # record generalization accuracy
    test_accuracy.append(forest_test.score(X_test, y_test))



#plotting the training & testing accuracy for n_estimators from 1 to 20
plt.figure(figsize=None)
plt.plot(depth, training_accuracy, label="training accuracy")
plt.plot(depth, test_accuracy, label="test accuracy")
plt.ylabel("Accuracy")
plt.xlabel("n_estimators")
plt.legend();
```

In[30]: storing the results. The below mentioned order of parameter passing is important.

```
storeResults('Random Forest',acc_test_forest,f1_score_test_forest,
        recall_score_train_forest,precision_score_train_forest)
```

In[31]: Gradient Boosting Classifier Model

```
from sklearn.ensemble import GradientBoostingClassifier


# instantiate the model
```

```
gbc = GradientBoostingClassifier(max_depth=4,learning_rate=0.7)

# fit the model
gbc.fit(X_train,y_train)
```

In[32]: predicting the target value from the model for the samples

```
y_train_gbc = gbc.predict(X_train)
y_test_gbc = gbc.predict(X_test)
```

In[33]: computing the accuracy, f1_score, Recall, precision of the model performance

```
acc_train_gbc = metrics.accuracy_score(y_train,y_train_gbc)
acc_test_gbc = metrics.accuracy_score(y_test,y_test_gbc)
print("Gradient Boosting Classifier : Accuracy on training Data:
{:.3f}".format(acc_train_gbc))
print("Gradient Boosting Classifier : Accuracy on test Data: {:.3f}".format(acc_test_gbc))
print()

f1_score_train_gbc = metrics.f1_score(y_train,y_train_gbc)
f1_score_test_gbc = metrics.f1_score(y_test,y_test_gbc)
print("Gradient Boosting Classifier : f1_score on training Data:
{:.3f}".format(f1_score_train_gbc))
print("Gradient Boosting Classifier : f1_score on test Data:
{:.3f}".format(f1_score_test_gbc))
print()

recall_score_train_gbc = metrics.recall_score(y_train,y_train_gbc)
recall_score_test_gbc =  metrics.recall_score(y_test,y_test_gbc)
print("Gradient Boosting Classifier : Recall on training Data:
{:.3f}".format(recall_score_train_gbc))
print("Gradient Boosting Classifier : Recall on test Data:
{:.3f}".format(recall_score_test_gbc))
print()
```

```
precision_score_train_gbc = metrics.precision_score(y_train,y_train_gbc)

precision_score_test_gbc = metrics.precision_score(y_test,y_test_gbc)

print("Gradient Boosting Classifier : precision on training Data:
{:.3f}".format(precision_score_train_gbc))

print("Gradient Boosting Classifier : precision on test Data:
{:.3f}".format(precision_score_test_gbc))
```

In[34]: computing the classification report of the model

```
print(metrics.classification_report(y_test, y_test_gbc))
```

In[35]:

```
training_accuracy = []

test_accuracy = []

# try learning_rate from 0.1 to 0.9

depth = range(1,10)

for n in depth:

    forest_test =  GradientBoostingClassifier(learning_rate = n*0.1)


    forest_test.fit(X_train, y_train)

    # record training set accuracy

    training_accuracy.append(forest_test.score(X_train, y_train))

    # record generalization accuracy

    test_accuracy.append(forest_test.score(X_test, y_test))



#plotting the training & testing accuracy for n_estimators from 1 to 50

plt.figure(figsize=None)

plt.plot(depth, training_accuracy, label="training accuracy")

plt.plot(depth, test_accuracy, label="test accuracy")

plt.ylabel("Accuracy")

plt.xlabel("learning_rate")

plt.legend();
```

In[36]:

```
training_accuracy = []
test_accuracy = []
# try learning_rate from 0.1 to 0.9
depth = range(1,10,1)
for n in depth:
    forest_test =  GradientBoostingClassifier(max_depth=n,learning_rate = 0.7)

    forest_test.fit(X_train, y_train)
    # record training set accuracy
    training_accuracy.append(forest_test.score(X_train, y_train))
    # record generalization accuracy
    test_accuracy.append(forest_test.score(X_test, y_test))



#plotting the training & testing accuracy for n_estimators from 1 to 50
plt.figure(figsize=None)
plt.plot(depth, training_accuracy, label="training accuracy")
plt.plot(depth, test_accuracy, label="test accuracy")
plt.ylabel("Accuracy")
plt.xlabel("max_depth")
plt.legend();
```

In[37]: storing the results. The below mentioned order of parameter passing is important.

```
storeResults('Gradient Boosting Classifier',acc_test_gbc,f1_score_test_gbc,
          recall_score_train_gbc,precision_score_train_gbc)
```

In[38]: Multi-layer Perceptron Classifier Model

```
from sklearn.neural_network import MLPClassifier


# instantiate the model
```

```
mlp = MLPClassifier()
#mlp = GridSearchCV(mlpc, parameter_space)

# fit the model
mlp.fit(X_train,y_train)
```

In[39]: predicting the target value from the model for the samples

```
y_train_mlp = mlp.predict(X_train)
y_test_mlp = mlp.predict(X_test)
```

In[40]: computing the accuracy, f1_score, Recall, precision of the model performance

```
acc_train_mlp  = metrics.accuracy_score(y_train,y_train_mlp)
acc_test_mlp = metrics.accuracy_score(y_test,y_test_mlp)
print("Multi-layer Perceptron : Accuracy on training Data: {:.3f}".format(acc_train_mlp))
print("Multi-layer Perceptron : Accuracy on test Data: {:.3f}".format(acc_test_mlp))
print()

f1_score_train_mlp = metrics.f1_score(y_train,y_train_mlp)
f1_score_test_mlp = metrics.f1_score(y_test,y_test_mlp)
print("Multi-layer Perceptron : f1_score on training Data:
{:.3f}".format(f1_score_train_mlp))
print("Multi-layer Perceptron : f1_score on test Data: {:.3f}".format(f1_score_train_mlp))
print()

recall_score_train_mlp = metrics.recall_score(y_train,y_train_mlp)
recall_score_test_mlp = metrics.recall_score(y_test,y_test_mlp)
print("Multi-layer Perceptron : Recall on training Data:
{:.3f}".format(recall_score_train_mlp))
print("Multi-layer Perceptron : Recall on test Data: {:.3f}".format(recall_score_test_mlp))
print()
```

```
precision_score_train_mlp = metrics.precision_score(y_train,y_train_mlp)

precision_score_test_mlp = metrics.precision_score(y_test,y_test_mlp)

print("Multi-layer Perceptron : precision on training Data:
{:.3f}".format(precision_score_train_mlp))

print("Multi-layer Perceptron : precision on test Data:
{:.3f}".format(precision_score_test_mlp))
```

In[41]: storing the results. The below mentioned order of parameter passing is important.

```
storeResults('Multi-layer Perceptron',acc_test_mlp,f1_score_test_mlp,

        recall_score_train_mlp,precision_score_train_mlp)
```

In[42]: creating dataframe

```
result = pd.DataFrame({ 'ML Model' : ML_Model,

                'Accuracy' : accuracy,

                'f1_score' : f1_score,

                'Recall'   : recall,

                'Precision': precision,

               })
```

In[43]: dispalying total result

```
Result
```

In[44]: Sorting the datafram on accuracy

```
sorted_result=result.sort_values(by=['Accuracy',
'f1_score'],ascending=False).reset_index(drop=True)
```

In[45]: dispalying total result

```
sorted_result
```

In[46]: checking the feature improtance in the model

```
plt.figure(figsize=(9,7))
n_features = X_train.shape[1]
plt.barh(range(n_features), gbc.feature_importances_, align='center')
plt.yticks(np.arange(n_features), X_train.columns)
plt.title("Feature importances using permutation on full model")
plt.xlabel("Feature importance")
plt.ylabel("Feature")
plt.show()
```