KNN.py

```
#Import Libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix , accuracy_score
from sklearn.model_selection import GridSearchCV

FILENAME="Final_Refined_Encoded_Normalized.csv"
FILEPATH='DATASET/' + FILENAME;

##Load Dataset
def loadDataSet():

   dataset = pd.read_csv(FILEPATH)
   return dataset

##Split Dataset
def splitDataSet(dataSet,testSize):

   #Split dataset into its attributes X and labels y
   X = dataSet.iloc[:, :-1].values
   y = dataSet.iloc[:, 102].values

   #Splits the dataset into  train data and  test data by test size
   X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=testSize)
   return X_train, X_test, y_train, y_test

## Applying Grid Search on KNN to get best ore, best parameters and best estimator
def applyGridSeacrh(X_train,y_train):
   print('\nApplying GridSearch on kNN...\n')
   grid_params={'n_neighbors':range(1,40),
            'weights':['uniform','distance'],
            'metric':['euclidean','manhattan']
            }
   gs=GridSearchCV(
         KNeighborsClassifier(),
         grid_params,
         verbose=1,
         cv=3,
```

```
        n_jobs=-1
        )
    gs_results=gs.fit(X_train,y_train)
    return gs_results.best_score_,gs_results.best_estimator_,gs_results.best_params_

## Applying Knn Algorithm
def applyKNN(X_train, X_test, y_train, y_test):

    print("\nApplying KNN...")

    #Scaling data
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    #Apply GridSearch to find best score, estimator and best parameters
    best_score,best_estimator,best_params=applyGridSeacrh(X_train,y_train)
    print('\nBest Score:' +str(best_score))
    print('Best Estimator: '+ str(best_estimator))
    print('Best Parameters: '+str(best_params))

    # instantiate learning model with best estimator
    knn = best_estimator

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

    # predict the response
    pred = knn.predict(X_test)

    #print('\nConfusion Matrix:')
    #print(confusion_matrix(y_test, pred))
    # evaluate accuracy
    print('\nTrain Score: '+ str(knn.score(X_train,y_train)))
    print('Test Score: '+ str(knn.score(X_test,y_test)))
    print('\nAccuracy:')
    print (accuracy_score(y_test, pred))

    # Calculating error for K values between 1 and 40
    error = []
    for i in range(1, 40):
        knn = KNeighborsClassifier(n_neighbors=i)
        knn.fit(X_train, y_train)
        pred_i = knn.predict(X_test)
```

```
    error.append(np.mean(pred_i != y_test))
  #print(error)
  #plot the error values against K values
  plt.figure(figsize=(12, 6))
  plt.plot(range(1, 40), error, color='red', linestyle='dashed', marker='o',markerfacecolor='blue',
markersize=10)
  plt.title('Error Rate K Value')
  plt.xlabel('K Value')
  plt.ylabel('Mean Error')

if __name__ == "__main__":
  dataSet= loadDataSet();
  X_train, X_test, y_train, y_test=splitDataSet(dataSet,0.20)
  applyKNN(X_train, X_test, y_train, y_test)
```

Output:

```
In [5]: runfile('/Users/gopi/Desktop/CS-513-FinalCode--/KNN.py', wdir='/Users/gopi/Desktop/CS-513-FinalCode--')

Applying KNN...

Applying GridSearch on kNN...

Fitting 3 folds for each of 156 candidates, totalling 468 fits
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    8.4s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:   48.1s
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed:  1.9min
[Parallel(n_jobs=-1)]: Done 468 out of 468 | elapsed:  2.1min finished

Best Score:0.6461769115442278
Best Estimator: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='manhattan',
           metric_params=None, n_jobs=1, n_neighbors=8, p=2,
           weights='distance')
Best Parameters: {'metric': 'manhattan', 'n_neighbors': 8, 'weights': 'distance'}

Train Score: 0.9975012493753124
Test Score: 0.6473526473526473

Accuracy:
0.6473526473526473
```