Implementing K-Nearest Neighbors (KNN)

Shimul Paul
Department of Computer Science and Engineering
Ahsanullah University of Science and Technology
Dhaka, Bangladesh
160204014@aust.edu

***Abstract*—KNN (K Nearest Neighbors) is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure.It is used to obtain a modified training database for the posterior learning of the classification tree structure. The same class category products are homogeneous, then the nearest assignment. Since assigning unlabeled cases to their most close labeled neighbors is relatively simple.Because of its non-parametric approach, the methods depend heavily on the training instances.**

***Index Terms*—KNN, Nearest Neighbor, Euclidean**

I. Introduction

KNN is a type of classification where the function is only approximated locally and all computation is deferred until function evaluation.It is a non-parametric, lazy learning algorithm. Its main function is to predict the classification of a new sample point using data from several groups. The limitation of KNN is it cant predict the data being analyzed. The model is built from the data. As KNN is the most basic and most straightforward classification technique during learning, this rule essentially keeps the entire training set. The assigned classes are based on the majority mark of its k-nearest neighbors in the training set. The equation of euclidean distance is shown below:

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

II. Experimental Design / Methodology

- Firstly, I read all the training data using pandas read_csv() function and store it in a array.
- Then, I convert it in numpy arrays.
- Then, I make two class Class1 and Class2 for visualize.
- Then, I plot the train data with the help of matplotlib with marker and color.
- Then, for the test data I read all the data and apply the euclidean distance eqation for calculate the distance and sort them according ascending order and also take the value of k from user.
- After that, if the number of occurrences for counting class1 is greater than class2 then it is in Class1 and vice verse.
- Then, I plot the predicted test class with test data with the help of matplotlib with marker and color.
- After that I print and save the distances from test data.

III. Algorithm Implementation / Code

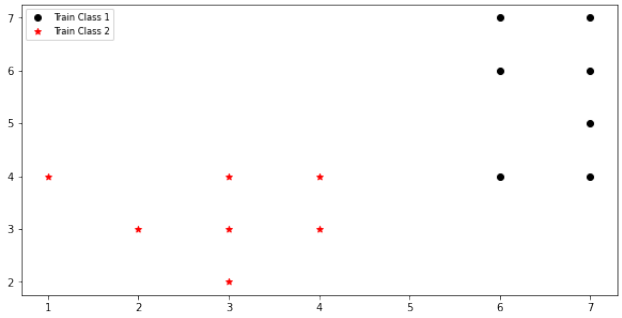
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math

#read the file
traindata = pd.read_csv('train_knn.txt', sep=","
,header=None)
traindata=traindata.to_numpy()
#if the class value is 1 then store it in class 1
else class 2
class1=[[x[0],x[1]] for x in traindata
if x[2]==1]
class2=[[x[0],x[1]] for x in traindata
if x[2]==2]
class1 = np.array(class1)
class2 = np.array(class2)
#plot the training data
fig, ax = plt.subplots()
fig.set_figheight(5)
fig.set_figwidth(10)
ax.scatter(class1[:,0],class1[:,1],marker='o'
,color='k',label='Train Class 1')
ax.scatter(class2[:,0],class2[:,1],marker='*'
,color='r',label='Train Class 2')
legend = ax.legend(loc='upper left', shadow=False,
fontsize='small',labelspacing=0.5)
legend.get_frame().set_facecolor('None')
plt.show()
testdata = pd.read_csv('test_knn.txt', sep=","
,header=None)
testdata=testdata.to_numpy()
distances = []
prediction_list = []
final_dis = []
k = int(input())
for i in testdata:
    predicted = 5
    count1 = 0
    count2 = 0
    for j in traindata:
        dis = math.sqrt(((i[0] - j[0]) **2)
```

```
            + ((i[1] - j[1]) **2))
            distances.append([dis,j[2]])
        distances.sort(key = lambda x: x[0])
        for u in distances[:k]:
            if(u[1] == 1):
                count1 = count1 + 1
            else:
                count2 = count2 + 1
        if(count1 > count2):
            predicted = 1
        else:
            predicted = 2

        distances.insert(k,[i[0],i[1]])
        print(distances[:k])
        final_dis.append(distances[:k+1])

        prediction_list.append([i[0],i[1]
        ,predicted])
        distances = []

class1=[[x[0],x[1]] for x in prediction_list
if x[2]==1]
class2=[[x[0],x[1]] for x in prediction_list
if x[2]==2]
class1 = np.array(class1)
class2 = np.array(class2)

f, ax = plt.subplots()
f.set_figheight(5)
f.set_figwidth(10)
ax.scatter(class1[:,0],class1[:,1],
marker='o',color='k',label='Test Class 1')
ax.scatter(class2[:,0],class2[:,1],marker='*',
color='r',label='Test Class 2')
legend = ax.legend(loc='upper left',
shadow=False,fontsize='small',labelspacing=0.5)
legend.get_frame().set_facecolor('None')
plt.show()

prediction = open('prediction.txt','w')
for j in (final_dis):
    predicted = ''
    count1 = 0
    count2 = 0
    print(j[len(j)-1])
    stest = str(j[len(j)-1])+'\n'
    prediction.write(stest)
    n =1
    for u in j[:-1]:
        print('Distance ',n,':',u[0]
        ,'\tClass',u[1])
        sdis = 'Distance '+ str(n)+ ':'+
        str( u[0] ) +'\tClass'+str(u[1])+'\n'
        prediction.write(sdis)
        if(u[1] == 1):
            count1 = count1 + 1
        else:
            count2 = count2 + 1
        n+=1
    if(count1 > count2):
        predicted = 'Class 1'
    else:
        predicted = 'Class 2'
    print('Predicted Class: ',predicted,'\n')
    spredict = 'Predicted Class: '+
    str(predicted)+'\n'
    prediction.write(spredict)
prediction.close()
```

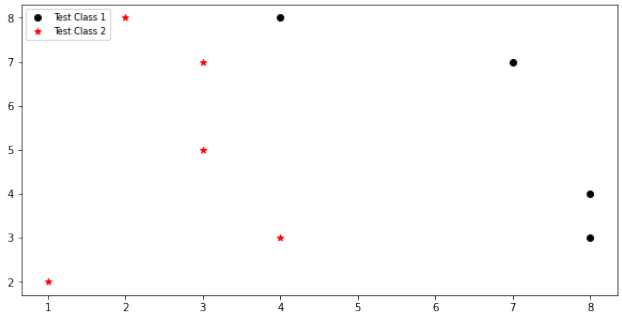
IV. Result Analysis

The results is shown below:

- Firstly, I plot the train data with the help of matplotlib and the output shown below:



- Secondly, I calculate the test data with the help of euclidean distance equation and classify the data and plot the test data using matplotlib and got this output:



- Thirdly, I saved the output as per the question wanted for k = 5:

```
[3, 7]
Distance 1:3.0   Class1
Distance 2:3.0   Class2
Distance 3:3.1622776601683795    Class1
Distance 4:3.1622776601683795    Class2
Distance 5:3.605551275463989     Class2
Predicted Class: Class 2
[7, 7]
Distance 1:0.0   Class1
Distance 2:1.0   Class1
Distance 3:1.0   Class1
```

```
Distance 4:1.4142135623730951    Class1
Distance 5:2.0   Class1
Predicted Class: Class 1
[4, 3]
Distance 1:0.0   Class2
Distance 2:1.0   Class2
Distance 3:1.0   Class2
Distance 4:1.4142135623730951    Class2
Distance 5:1.4142135623730951    Class2
Predicted Class: Class 2
[2, 8]
Distance 1:4.123105625617661     Class1
Distance 2:4.123105625617661     Class2
Distance 3:4.123105625617661     Class2
Distance 4:4.47213595499958      Class1
Distance 5:4.47213595499958      Class2
Predicted Class: Class 2
[3, 5]
Distance 1:1.0   Class2
Distance 2:1.4142135623730951    Class2
Distance 3:2.0   Class2
Distance 4:2.23606797749979      Class2
Distance 5:2.23606797749979      Class2
Predicted Class: Class 2
[1, 2]
Distance 1:1.4142135623730951    Class2
Distance 2:2.0   Class2
Distance 3:2.0   Class2
Distance 4:2.23606797749979      Class2
Distance 5:2.8284271247461903    Class2
Predicted Class: Class 2
[4, 8]
Distance 1:2.23606797749979      Class1
Distance 2:2.8284271247461903    Class1
Distance 3:3.1622776601683795    Class1
Distance 4:3.605551275463989     Class1
Distance 5:4.0   Class2
Predicted Class: Class 1
[8, 3]
Distance 1:1.4142135623730951    Class1
Distance 2:2.23606797749979      Class1
Distance 3:2.23606797749979      Class1
Distance 4:3.1622776601683795    Class1
Distance 5:3.605551275463989     Class1
Predicted Class: Class 1
[8, 4]
Distance 1:1.0   Class1
Distance 2:1.4142135623730951    Class1
Distance 3:2.0   Class1
Distance 4:2.23606797749979      Class1
Distance 5:2.8284271247461903    Class1
Predicted Class: Class 1
```

CONCLUSION

The purpose of the k Nearest Neighbors (KNN) algorithm is to use a database in which the data points are separated into several separate classes to predict the classification of a new sample point.