Implementing K-Nearest Neighbors (KNN)

Mayeesha Humaira
dept. Computer Science and Engineering
Ahsanullah University of Science and Technology
Dhaka, Bangladesh
160204008@aust.edu
section - A1

***Abstract*—The main aim of this experiment is to implement K-Nearest Neighbors algorithm to classify data points into two classes.**

***Index Terms*—Neighbors; Euclidean distance; K-nearest;**

I. Introduction

K-Nearest Neighbors is a algorithm that sorts entire training data to find K-most similar neighbors and make prediction based on these neighbors. This algorithm is performed in three steps they are:

- **Step-1: Calculating the Euclidean Distance**
  The Euclidean Distance is calculated using the following formula:

$$distance = \sqrt{(x_i - x_t)^2 + (y_i - y_t)^2} \quad (1)$$

  Where x_i is the i th x coordinate of training data, y_i is the i th y coordinate of training data, x_t is the x coordinate of testing data and y_t is the y coordinate of testing data.

- **Step-2: Finding K Nearest Neighbors**
  Neighbors for a test data point are the k closest training data points that have minimum Euclidean distance among all training data. After computing euclidean distance for all training data they are sorted and k most similar neighbors to test data is returned.

- **Step-3: Predicting class of test data**
  Among the K nearest neighbors a number of instances of class one and class two are calculated then the class having a maximum number of K neighbors is selected as the predicted class.

II. Experimental Design / Methodology

Two datasets named train_knn.txt and test_knn.txt were provided. In train_knn class labels of the data points were given but in test_knn.txt class labels were not given. train data and test data are shown in Table I and Table II respectively. The following task were performed on these datasets.

1) **Task-1:**
   Take input from "train.txt" and plot the points with different colored markers according to the assigned class label.

TABLE I
Data points in the train_knn.txt dataset.

x1	x2	Class
7	7	1
7	4	1
6	4	1
7	5	1
7	6	1
6	7	1
6	6	1
3	4	2
2	3	2
3	2	2
4	3	2
3	3	2
4	4	2
1	4	2

TABLE II
Data points in the test_knn.txt dataset.

x1	x2
3	7
7	7
4	3
2	8
3	5
1	2
4	8
8	3
8	4

Solution:-
Data points were plotted using matplotlib as shown in Fig 1. Here the red circles represent data points of class-1 and the the blue star marks represent data points of class-2.

2) **Task-2:**
   Implement KNN algorithm. The value of K will be taken from user. Classify the test points from "test.txt" with different colored markers according to the predicted class label. assigned class label.
   Solution:-
   K was a integer variable that was taken as input from

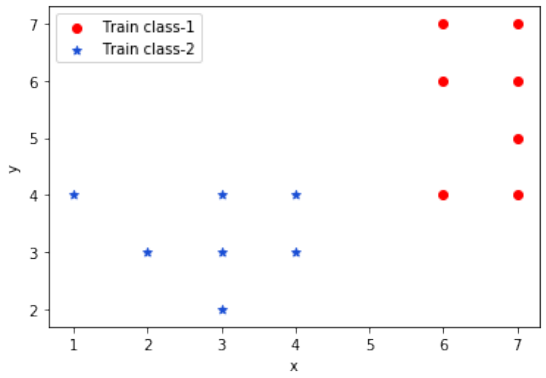

Fig. 1. Graphical representation of training data using matplotlib.

user. Then training dataset were sorted according to euclidean distance in ascending order. Afterwards k training points from the sorted list is chosen these are the nearest neighbor to the test data point used to calculate euclidean distance. Finally, the maximum times a class is in the k neighbors is chosen as the predicted class for that test point. After implementing the algorithm the classified test data points along with the training was plotted using matplotlib and illustrated in Fig 2.

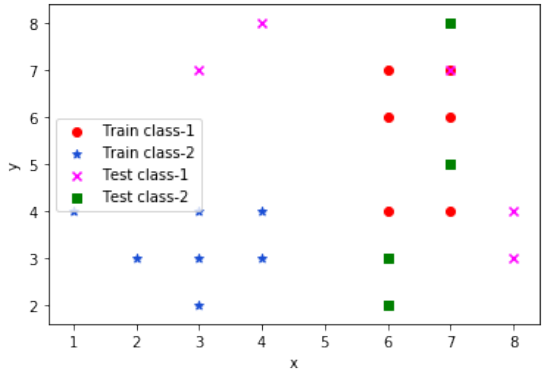

Fig. 2. Graphical representation of test data using matplotlib.

3) **Task-3:**
Print the top K distances along with their class labels and the predicted class to "prediction.txt" for each of the test data. So, for example, if K = 3, for one of the test data (3,7), the "prediction.txt" may look like:
Test point: 3, 7
Distance 1: 2 Class: 1
Distance 2: 4 Class: 0
Distance 3: 5 Class: 1
Predicted class: 1

Solution:-
For K=3 the three neighbors of all test data along with the predicted class is shown in Fig 3.

```
Enter your value: 3
TestPoint:  3.0 , 7.0
Distance 1 :3.000       class: 1.0
Distance 2 :3.000       class: 2.0
Distance 3 :3.162       class: 1.0
Predicted Class: 1

TestPoint:  7.0 , 7.0
Distance 1 :0.000       class: 1.0
Distance 2 :1.000       class: 1.0
Distance 3 :1.000       class: 1.0
Predicted Class: 1

TestPoint:  4.0 , 3.0
Distance 1 :0.000       class: 2.0
Distance 2 :1.000       class: 2.0
Distance 3 :1.000       class: 2.0
Predicted Class: 2

TestPoint:  2.0 , 8.0
Distance 1 :4.123       class: 1.0
Distance 2 :4.123       class: 2.0
Distance 3 :4.123       class: 2.0
Predicted Class: 2

TestPoint:  3.0 , 5.0
Distance 1 :1.000       class: 2.0
Distance 2 :1.414       class: 2.0
Distance 3 :2.000       class: 2.0
Predicted Class: 2

TestPoint:  1.0 , 2.0
Distance 1 :1.414       class: 2.0
Distance 2 :2.000       class: 2.0
Distance 3 :2.000       class: 2.0
Predicted Class: 2

TestPoint:  4.0 , 8.0
Distance 1 :2.236       class: 1.0
Distance 2 :2.828       class: 1.0
Distance 3 :3.162       class: 1.0
Predicted Class: 1

TestPoint:  8.0 , 3.0
Distance 1 :1.414       class: 1.0
Distance 2 :2.236       class: 1.0
Distance 3 :2.236       class: 1.0
Predicted Class: 1

TestPoint:  8.0 , 4.0
Distance 1 :1.000       class: 1.0
Distance 2 :1.414       class: 1.0
Distance 3 :2.000       class: 1.0
Predicted Class: 1
```

Fig. 3. Illustration of K neighbors of all test points and the predicted class of each point.

III. Result Analysis

Changing the value of K changes the the predicted class.Three different plots having K=2, K=3, k=5 and k=8 is shown in Fig 4, Fig 5, Fig 6 and Fig 7 respectively. From these figures it can be seen that changing value of k changes the predicted class of test data.

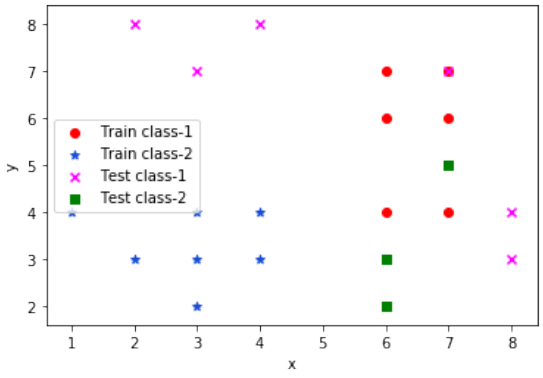

Fig. 4. Graphical representation of test data having K=2.

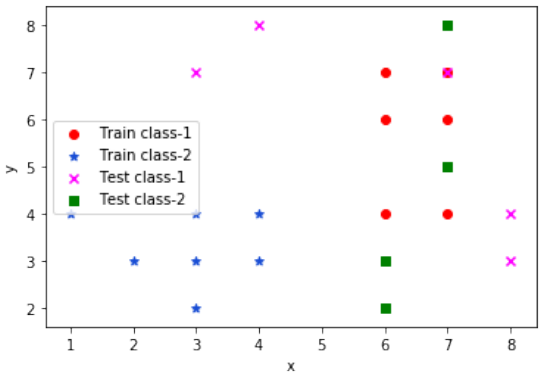

Fig. 5. Graphical representation of test data having K=3.

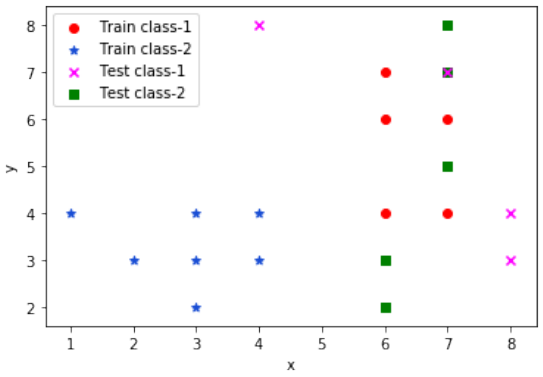

Fig. 6. Graphical representation of test data having K=5.

IV. Conclusion

K-Nearest Neighbor is a simple algorithm that can be easily implemented. But the main problem is choosing the correct value of K that accurately classifies the test data points. To get the right K value, KNN algorithm should be run several times with different values of K and then select the k value that has the least number of errors.

V. Algorithm Implementation / Code

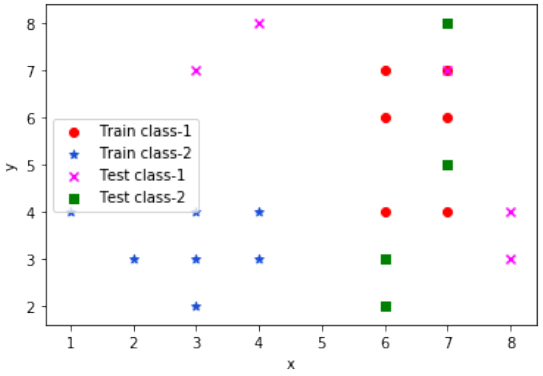

Fig. 7. Graphical representation of test data having K=8.

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

#Plotting training data
xd,yd,zd = np.loadtxt('train_knn.txt',unpack=True,
    delimiter=',')
print(type(xd),xd)
print(yd)
print(zd)

# Train data with class
datacl = np.zeros((len(xd),3))
print(datacl.shape, datacl.ndim)

for h in range(len(xd)):
    datacl[h][0]=xd[h]
    datacl[h][1]=yd[h]
    datacl[h][2]=zd[h]
print(datacl, type(datacl))


plt.xlabel('x')
plt.ylabel('y')
for m in range(len(zd)):
    if datacl[m][2]==1:
        xc1=plt.scatter(datacl[m][0], datacl[m][1],
    color='r')
    elif datacl[m][2]==2:
        xc2=plt.scatter(datacl[m][0], datacl[m][1],
    marker='*', color='#184DD5')

plt.legend([xc1, xc2], ["Train class-1", "Train
    class-2"])
plt.show()

#Test Data before classification
xt,yt = np.loadtxt('test_knn.txt',unpack=True,
    delimiter=',')
data = np.zeros((len(xt),2))
print(data.shape, data.ndim)

for h in range(len(xt)):
    data[h][0]=xt[h]
    data[h][1]=yt[h]
print(data, type(data))

k = input("Enter your value: ")
k=int(k)
cl=[]
# calculate the Euclidean distance between two
    vectors
def euclidean_distance(row1, row2):
    distance = 0.0
    for i in range(len(row1)-1):
        distance += (row1[i] - row2[i])**2
    return np.sqrt(distance)

# Locate the most similar neighbors
def get_neighbors(train, test_row, num_neighbors):
    distances = list()
    for train_row in train:
        dist = euclidean_distance(train_row,
    test_row)
        distances.append((train_row, dist))
        #index[1]=dist
    distances.sort(key=lambda index: index[1])
    neighbors = list()
    for i in range(num_neighbors):
        neighbors.append((distances[i][0],distances[
    i][1]))
    return neighbors

# Make a classification prediction with neighbors
```

```
def predict_classification(train, test_row, num_neighbors):
    neighbors = get_neighbors(train, test_row, num_neighbors)
    cnt=0
    print('TestPoint: ',test_row[0],',',test_row[1])
    for neighbor in neighbors:
        cnt=cnt+1
        print('Distance',cnt,':%.3f'%neighbor[1],'\tclass:',neighbor[0][-1])
    output_values = [row[0][-1] for row in neighbors]
    prediction = max(set(output_values), key=output_values.count)
    return prediction

for n in range(len(data)):
    prediction = predict_classification(datacl, data[n], k)
    cl.append(prediction)
    print('Predicted Class: %d' % (prediction))
    print()


#test data with predicted class
dataP = np.zeros((len(data),3))

for h in range(len(data)):
    dataP[h][0]=data[h][0]
    dataP[h][1]=data[h][1]
    dataP[h][2]=cl[h]
print(dataP, type(dataP))

#Ploting classified points
plt.xlabel('x')
plt.ylabel('y')
for m in range(len(datacl)):
    if datacl[m][2]==1:
        xc1=plt.scatter(datacl[m][0], datacl[m][1], color='red')
    elif datacl[m][2]==2:
        xc2=plt.scatter(datacl[m][0], datacl[m][1], marker='*', color='#184DD5')

for n in range(len(dataP)):
    if dataP[n][2]==1:
        xtc1=plt.scatter(dataP[n][0], dataP[n][1], marker='x', color='magenta')
    elif datacl[m][2]==2:
        xtc2=plt.scatter(datacl[n][0], dataP[n][1], marker='s', color='green')

plt.legend([xc1, xc2, xtc1, xtc2], ["Train class-1", "Train class-2", "Test class-1", "Test class-2"])
plt.show()
```