K-Nearest Neighbor Implementation

Md. Ashraful Haque
dept. Computer Science and Engineering
Ahsanullah University of Science and Technology
Dhaka, Bangladesh
160204101@aust.edu

***Abstract*—K-Nearest Neighbor is a famous algorithm to classify data based on distance of the data points plotted in the dimension of the data.**

I. Introduction

The algorithm Calculates the euclidean distance between the selected test data and all the training data. Then ranks the training points according to their distance from the test data. Then top k points are selected and the class that the majority of the points belong to is selected to be the class of the test data. this is how a prediction is made.

II. Experimental Design / Methodology

step 1 Collecting the test data. In our case since it is a supervised learning technique, we have train data points with labels and test data points.

step 2

An iteration goes through all the test data points (xi, yi). For each data point of the test data, the distances between all training data and the test data are calculated.

step 3

Then the distances are sorted in an ascending order and first k distances are selected from the sorted list of distances.

step 4

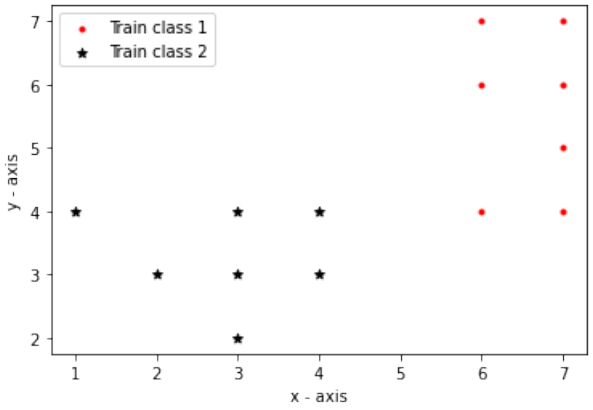
Among the K nearest neighbors of the test data point the class that the majority of the neighbors belong to is predicted to be the class of the test data.

step 5

The same process is repeated through out the whole test set and thus the predictions for the test data are maid.

III. Result Analysis

Task 1 Ploting the input

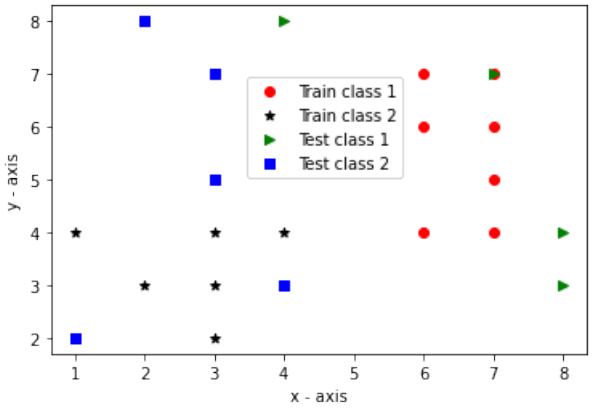


Task 2 Classification results of the test set (k = 5)
This is the content of "**prediction.txt**"

```
Test point : (3, 7)
Distance 1: 3.00   class: 1
Distance 2: 3.00   class: 2
Distance 3: 3.16   class: 1
Distance 4: 3.16   class: 2
Distance 5: 3.61   class: 2
Predicted class:2

Test point : (7, 7)
Distance 1: 0.00   class: 1
Distance 2: 1.00   class: 1
Distance 3: 1.00   class: 1
Distance 4: 1.41   class: 1
Distance 5: 2.00   class: 1
Predicted class:1

Test point : (4, 3)
Distance 1: 0.00   class: 2
Distance 2: 1.00   class: 2
Distance 3: 1.00   class: 2
Distance 4: 1.41   class: 2
Distance 5: 1.41   class: 2
Predicted class:2

Test point : (2, 8)
Distance 1: 4.12   class: 1
Distance 2: 4.12   class: 2
Distance 3: 4.12   class: 2
Distance 4: 4.47   class: 1
Distance 5: 4.47   class: 2
Predicted class:2
```

```
Test point : (3, 5)
Distance 1: 1.00   class: 2
Distance 2: 1.41   class: 2
Distance 3: 2.00   class: 2
Distance 4: 2.24   class: 2
Distance 5: 2.24   class: 2
Predicted class:2

Test point : (1, 2)
Distance 1: 1.41   class: 2
Distance 2: 2.00   class: 2
Distance 3: 2.00   class: 2
Distance 4: 2.24   class: 2
Distance 5: 2.83   class: 2
Predicted class:2

Test point : (4, 8)
Distance 1: 2.24   class: 1
Distance 2: 2.83   class: 1
Distance 3: 3.16   class: 1
Distance 4: 3.61   class: 1
Distance 5: 4.00   class: 2
Predicted class:1

Test point : (8, 3)
Distance 1: 1.41   class: 1
Distance 2: 2.24   class: 1
Distance 3: 2.24   class: 1
Distance 4: 3.16   class: 1
Distance 5: 3.61   class: 1
Predicted class:1

Test point : (8, 4)
Distance 1: 1.00   class: 1
Distance 2: 1.41   class: 1
Distance 3: 2.00   class: 1
Distance 4: 2.24   class: 1
Distance 5: 2.83   class: 1
Predicted class:1
```

Task 3 plot K = 5



IV. Conclusion

So, by these experiments we can come to the conclusion that by variations of the different kind, we can optimize the performance of the classifier.

V. Algorithm Implementation / Code

Task 1:

```
import matplotlib.pyplot as plt

trainData = {}
entry = []

with open("train_knn.txt") as f:
    for line in f:
        entry = line.split(",")
        if (entry[2][-1] == '\n'):
            classInfo =  entry[2][:-1]
        else:
            classInfo =  entry[2]
        trainData[(int(entry[0]),int(entry[1]))] =
    int(classInfo)
    print(trainData)

trainData1x = []
trainData1y = []
trainData2x = []
trainData2y = []

for coor , classinfo in trainData.items():
    if(classinfo == 1):
        trainData1x.append(coor[0]);
        trainData1y.append(coor[1]);
    else:
        trainData2x.append(coor[0]);
        trainData2y.append(coor[1]);

print(trainData1x , trainData1y)
print(trainData2x , trainData2y)


plt.scatter(trainData1x, trainData1y, color = 'r',
    marker = ".", label = 'Train class 1')

plt.scatter(trainData2x, trainData2y, color = 'k',
    marker = "*", label = 'Train class 2')

plt.xlabel('x - axis')
plt.ylabel('y - axis')

plt.legend(loc = 'upper left')

plt.show()
```

Task 2 and 3:

```
testData = {}
entry = []

with open("test_knn.txt") as f:
    for line in f:
        entry = line.split(",")
        if (entry[1][-1] == '\n'):
            entry[1] =  entry[1][:-1]

        classInfo =  0
        testData[(int(entry[0]),int(entry[1]))] =
    int(classInfo)
    print(testData)

import math

def euclidianDistance(queryInstance, trainingSamples
    ):
    distance = math.sqrt(math.pow(queryInstance[0] -
     trainingSamples[0] , 2) + math.pow(
    queryInstance[1] - trainingSamples[1],2 ) )
```

```
    return distance

euclidianDistance((1,1),(2,2) )

def getDistancesForOneTestSample(testSample ,
    trainData):
    distances = {}

    for trainSample in trainData:
        distances[trainSample] = euclidianDistance(
    testSample , trainSample )

    return distances

def getSortedDistancesForOneTestSample(testSample ,
    trainData):
    distances = getDistancesForOneTestSample(
    testSample , trainData)
    #print(distances)

    sortedDistances = {k: v for k, v in sorted(
    distances.items(), key=lambda item: item[1])}

    return sortedDistances

def getKNearestNeighborforOneTestSample(k ,
    testSample , trainData ):
    sortedDistences =
    getSortedDistancesForOneTestSample(testSample ,
    trainData)
    sortedDistenceList = list(sortedDistences.keys()
    )
    return sortedDistenceList[:k], sortedDistences

def getClassForaSingletestSample(k ,testSample ,
    trainData ):
    kNearestNeighbors , Distances  =
    getKNearestNeighborforOneTestSample(k ,
    testSample , trainData )
    kNearestNeighbors , Distances

    classList = []
    f = open("prediction.txt", "a")

    f.write("Test point : " + str(testSample) + "\n"
    )
    print( "Test point : " + str(testSample) + "\n")


    index = 0
    for eachTestItem in kNearestNeighbors:
        index += 1
        string =   "Distance "  + str(index) +": "+
    str("{:.2f}".format(Distances[eachTestItem]))+ "
       " + "class: " +str(trainData[eachTestItem])+"
    \n"
        print( string)

        f.write(string)

        classList.append(trainData[eachTestItem])



    print(classList)

    selectedClass = marorityVote(classList)
    print("Predicted class:" , selectedClass)
    f.write("Predicted class:" + str(selectedClass)
    + "\n"  + "\n")
    f.close()

    return  selectedClass
```

```
def marorityVote(listofNumbers):

    uniqueValues = set(listofNumbers)

    countDictionary = {}

    count = 0

    for eachuniqueValue in uniqueValues:
        for eachElement in listofNumbers:
            if(eachuniqueValue == eachElement):
                count += 1
        countDictionary[eachuniqueValue] = count
        count = 0

    sortedCountDictionary = {k: v for k, v in sorted
    (countDictionary.items(), key=lambda item: item
    [1])}
    decision = list(sortedCountDictionary.keys())
    [-1]
    return decision

marorityVote([1, 2, 1,3])


k = int(input("k = "))

for testSample in testData.keys():
    testData[testSample] =
    getClassForaSingletestSample(k ,testSample ,
    trainData )

testData

**Plotting the result**

testData1x = []
testData1y = []
testData2x = []
testData2y = []

for coor , classinfo in testData.items():
    if(classinfo == 1):
        testData1x.append(coor[0]);
        testData1y.append(coor[1]);
    else:
        testData2x.append(coor[0]);
        testData2y.append(coor[1]);

print(testData1x , testData1y)
print(testData2x , testData2y)


plt.scatter(trainData1x, trainData1y, color = 'r',
    marker = "o", label = 'Train class 1')

plt.scatter(trainData2x, trainData2y, color = 'k',
    marker = "*", label = 'Train class 2')

plt.scatter(testData1x, testData1y, color = 'g',
    marker = ">", label = 'Test class 1')

plt.scatter(testData2x, testData2y, color = 'b',
    marker = "s", label = 'Test class 2')


plt.xlabel('x - axis')
plt.ylabel('y - axis')

plt.legend(loc = 'lower center')

plt.show()
```