Implementing K-Nearest Neighbors (KNN)

Mayisha Farzana
dept.Computer Science and Engineering
Ahsanullah University of Science and Technology
Dhaka, Bangladesh
160204028@aust.edu

***Abstract*—The Nearest Neighbor algorithm is used as a preprocessing algorithm to obtain a modified training database for the posterior learning of the classification tree structure. When the functions and goal classes are multiple, complex, or challenging to comprehend, classification tasks can be complicated. In cases where the same class category products are homogeneous, the nearest neighbor assignment can be sufficient since assigning unlabeled cases to their most close labeled neighbors is relatively simple. Such classification approaches will help us decipher the story behind unlabeled data using identified data while minimizing the analysis of complicated features and goal groups. This is because these methods do not depend on prior distribution assumptions. However, because of this non-parametric approach, the methods depend heavily on the training instances, which are called lazy algorithms.**

***Index Terms*—Machine Learning Supervised Classification Classifier Combination,Euclidean distance**

I. Introduction

KNN is a non-parametric, lazy learning algorithm. It predicts the classification of a new sample point using data from several groups. KNN is non-parametric since it makes no predictions about the data being analyzed, i.e., the model is built from the data. The KNN is the most basic and most straightforward classification technique. During learning, this rule essentially keeps the entire training set and assigns a class to each question based on the majority mark of its k-nearest neighbors in the training set. **Choice of K:**

- The estimate gets less stable as K reaches 1.
- If we increase the value of K, the prediction becomes more stable because of majority of voters.
- Taking a majority vote among labels needs K to be an odd number to have a tiebreaker.

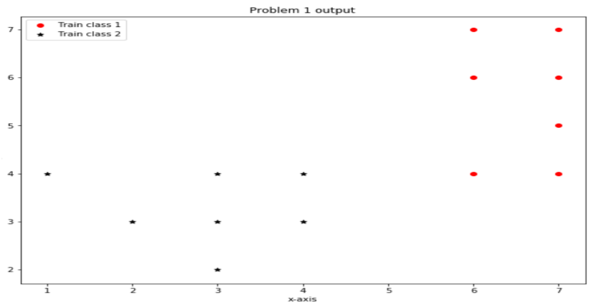
II. Experimental Design / Methodology

A.Classifying the sample points: We will read the trainKNN.txt file. From it, we will classifiy the train data points using different color and markers.

B. Implementing KNN algorithm: In this section, we will implement the KNN algorithm. At first, we will determine the K values. After that, we will calculate the distance between query instances and the training samples. For distance calculation, we will use euclidean distance.

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

After calculating the distance, we will sort the distance ans determine the nearest neighbours based on the K-th minimum



distance. From the sorted array,we will choose the top K rows. Based on the most appearing class of these rows, it will assign a class to the test point.

III. Saving the results in the text file:

After getting all the results, we will store the result and the predicted class in the prediction.txt file.

IV. Result Analysis

As we give nine test points, so we will get the different predicted classes for each test point. If we give K values 3, after sorting it, we will get the top 3 values. If there are two good classes and one bad class, we can predict that the unknown sample will be from the good classes. This is how we can predict each class.

V. Conclusion

K is selected from the database closest to the new sample KNN does not learn any model KNN makes predictions using the similarity between an input sample and each training instance. 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. This sort of situation is best motivated through examples.

VI. Algorithm Implementation / Code

```
'''Assignment 4'''
'''Mayisha Farzana'''
'''ID: 160204028'''
'''Problem 1 '''
import numpy as np
```

```python
import matplotlib.pyplot as plt
ar = np.loadtxt('train_knn.txt',delimiter=",",dtype='int32')
print('Train Data')
br = np.loadtxt('test_knn.txt',delimiter=",",dtype='int32')
for x in  ar:
    print("x = ",x[0],"y = ",x[1],"class = ",x[2]) # print the class names
ar_clas1=np.array([row for row in ar if row[2]==1])
ar_clas2=np.array([row for row in ar if row[2]==2])
x_train_1=ar_clas1[:,0]
y_train_1=ar_clas1[:,1]
x_train_2=ar_clas2[:,0]
y_train_2=ar_clas2[:,1]
fig,ax=plt.subplots()#to show it in the same figure
plt.title("Problem 1 output")
plt.xlabel('x-axis', color='black')
plt.ylabel('y-axis', color='black')
ax.scatter(x_train_1,y_train_1,marker='o',color='r',label='Train class 1')
ax.scatter(x_train_2,y_train_2,marker='*',color='black',label='Train class 2')
fig.set_figheight(8)
fig.set_figwidth(10)
ax.legend()#show the output figure
print('Test Data:')
# Test distance function
for i in br:
    print('x=',i[0],'y=',i[1])
from math import sqrt
#Implementing KNN algorithm
#Calculate the Euclidean distance between two vectors
def euclidean_distance(x1,y1,x2,y2):
    return sqrt((x1-x2)**2+(y1-y2)**2)
def distance(x,y):
    return sqrt((x[0]-y[0])**2+(x[1]-y[1])**2)
print(euclidean_distance(1,2,3,2))
print(distance(br[0],br[1]))
'''KNN ALGORITHM'''
out=open('prediction.txt','a')

def KNN(k,point):
    best=[]
    out.write("Test Point: {},{}\n".format(point[0],point[1]))
    cnt_class1=0
    cnt_class2=0
    for item in ar:
        dis=distance(point,item)
        #print(item,'distance',dis)
        best.append((dis,item[2]))
        #print(best)
    best.sort(key=lambda x:x[0])
    for i in range(k):
        out.write("Distance {}: {:.2f} \t Class:{}\n".format(i+1,best[i][0],best[i][1]))
    for i in range(k):
        if(best[i][1]==1):
            cnt_class1+=1
        else:
            cnt_class2+=1
 #   out.write("cnt = {} - {}\n".format(cnt_class1,cnt_class2))

    if(cnt_class1>=cnt_class2):
        out.write("Predicted Class {}\n".format(1))
    else:
        out.write("Predicted Class {}\n".format(2))
    out.write("\n")
K=int(input('Enter K value:'))
for point in br:
    KNN(K,point)
out.close()
```