Implementing K-Nearest Neighbors (KNN)

Anika Tanzim
Dept. Computer Science and Engineering
Ahsanullah University of Science and Technology
Dhaka, Bangladesh
160204072@aust.edu

***Abstract*—The objective of this experiment is to implement K-Nearest Neighbors (KNN) algorithm. The k-nearest neighbors (KNN) algorithm is a simple, easy-to-implement supervised machine learning algorithm that can be used to solve both classification and regression problems. KNN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using KNN algorithm.**

***Index Terms*—K-Nearest Neighbors, Euclidean Distance, Supervised Machine Learning**

I. Introduction

K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other. It uses test data to make an "educated guess" on what an unclassified point should be classified as.

When implementing KNN, the first step is to transform data points into feature vectors, or their mathematical value. The algorithm then works by finding the Euclidean distance between the mathematical values of these points.

Euclidean Distance: Let point X have Cartesian coordinates (x1,x2) and let point Y have coordinates (y1,y2). Then the distance between X and Y is given by:

$$d(X,Y) = \sqrt{(y1-x1)^2 \ + \ (y2-x2)^2}$$

KNN runs this formula to compute the distance between each data point and the test data. It then finds the majority of the category of these points being similar to the test data and classifies it based on which points share the highest probabilities.

II. Experimental Design / Methodology

A. *Classifying the sample points:*

A train dataset is given. All the points are classified into two classes - Class 1 and Class 2. Using Matplotlib, all the points from Train class 1 are plotted using blue dot marker and the points from train class 2 are plotted using green square marker in figure 1.

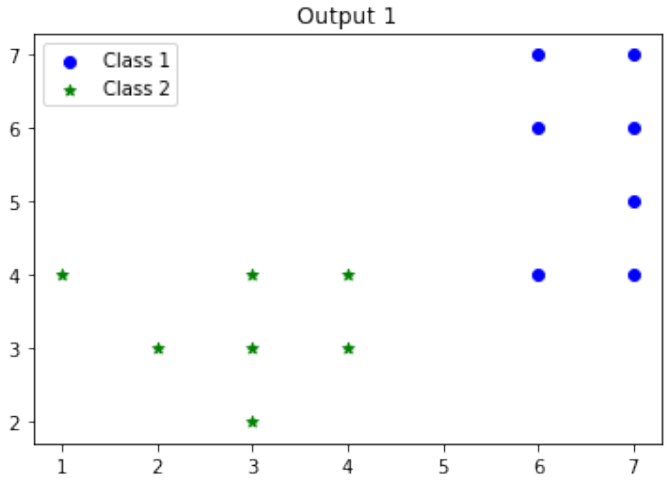

Fig. 1. Task 1 Output: Sample points Classified

B. *Implementing KNN algorithm and Classifying test points:*

Now, The KNN algorithm is implemented. Here is step by step on how we have computed K-nearest neighbors KNN algorithm:

1) At first, we have determined the K value by taken user input. While choosing K value, Here are some things to keep in mind:
   a) As we decrease the value of K to 1, our predictions become less stable.
   b) Inversely, as we increase the value of K, our predictions become more stable due to majority voting / averaging, and thus, more likely to make more accurate predictions (up to a certain point).
   c) In cases where we are taking a majority vote (e.g. picking the mode in a classification problem) among labels, we usually make K an odd number to have a tiebreaker.
2) We have calculated the distance between the query-instance and all the training samples using Euclidean distance.
3) Then, we have sorted the distance and determined nearest neighbors based on the K-th minimum distance.
4) Then we have gathered the category of the nearest neighbors.

5) Finally, we have used simple majority of the category of nearest neighbors as the prediction value of the query instance, i.e from top k values, we have taken most appeared class.

C. *Printing into "prediction.txt"*

We have stored the output into the text file "prediction.txt". We have printed the top K distances along with their class labels and the predicted class to the file for each of the test data.

III. Result Analysis

We get the predicted classes for each points given in the test points. We see while choosing the neighbours, we take top k values from sort distances. after that if majority train points are from class 1 then, the test point will be class1, and it will go to class 2 otherwise. We can see, it is simple to implement. It is robust to the noisy training data. we can also say, it can be more effective if the training data is large.

IV. Conclusion

KNN algorithm is a good algorithm to use when beginning to explore the world of machine learning, but it still has room for improvement and modification. But it also always needs to determine the value of K which may be complex some time. The computation cost is high because of calculating the distance between the data points for all the training samples.

V. Algorithm Implementation / Code

```
import numpy as np
import matplotlib.pyplot as plt
from math import sqrt

a=np.loadtxt('train_knn.txt',delimiter=",")
b=np.loadtxt('test_knn.txt',delimiter=",")

class1=[]
class2=[]
for i in a:
    if i[2]==1:
        class1.append(i)
    else:
        class2.append(i)
class1=np.array(class1)
class2=np.array(class2)

x1=class1[:,0] # 0 no col from every row
y1=class1[:,1] #1 no col from every row
x2=class2[:,0]
y2=class2[:,1]

fig,ax=plt.subplots()
plt.title("Output 1")
ax.scatter(x1, y1, marker='o',color='b',label='Class
    1')
ax.scatter(x2, y2, marker='*',color='g',label='Class
    2')
ax.legend()


#Implementing KNN algorithm
out=open('prediction.txt','a')

def dist(x,y):
    return sqrt((x[0]-y[0])**2+(x[1]-y[1])**2)


def k_nearest_neighbours(k,point):
    out.write("\nTest Point: {},{}".format(point[0],
    point[1]))
    print("Test Point: "+ str(point[0]) + ","+ str(
    point[1]))

    cnt1=0
    cnt2=0
    knn=[]

    #step2. calculating distance between test point
    and each train points
    for item in a:
        dis=dist(point,item)
        knn.append((dis,item[2]))

    #step3. sorting distance and taking k nearest
    neighbours
    knn.sort(key=lambda x:x[0])


    for i in range(k):
        ii=i+1
        #step4. gathering categories of the
    neighbours
        out.write("\nDistance {}: {:.2f} \t Class:{}
    ".format(ii, knn[i][0], knn[i][1]))
        print("Distance " + str(ii) +": " + str(int(
    knn[i][0])) + "\t Class:" + str(int(knn[i][1])))

        #step 5. tking majority of the category
        if(knn[i][1]==1):
            cnt1+=1
        else:
            cnt2+=1

    if(cnt1>=cnt2):
        out.write("\nPredicted Class {}".format(1))
        print("Predicted Class 1")
    else:
        out.write("\nPredicted Class {}".format(2))
        print("Predicted Class 2")
    print("\n")

#step 1. Determining parameter
k=int(input('Enter K [odd number]:'))
out.write("\nk= {}".format(k))
for point in b:
    k_nearest_neighbours(k,point)

out.close()
```