Implementing K-Nearest Neighbors (KNN)

Marufa Kamal
dept. Computer Science and Engineering
Ahsanullah University of Science and Technology
Dhaka, Bangladesh
160204073@aust.edu

***Abstract*—KNN is a neighbor based learning algorithm for classification and regression problems. It is based upon supervised data which helps to classify new data points based on similarity measures. In this experiment we are going to classify new data points by determining the nearest neighbors using K-nearest neighbor algorithm.**

***Index Terms*—Machine learning Classification problem, Nearest Neighbor Algorithm, Supervised Classification, Euclidean Distance, Labeled data.**

I. Introduction

K-Nearest Neighbors is a supervised learning algorithm that classifies data points by measuring the neighbor distance from the new data point. It is a non-parametric, lazy learning algorithm as it does not assume anything about the underlying data and does not have a specialized training period. It uses all the training data to classify the new data points. It also uses feature similarity to predict the values of new data points A new point is classified by the majority votes for its neighbor classes. The 'K' in this algorithm refers to the number of neighbors to be considered while measuring the distance. The distance can be calculated using different formulas such as Euclidian distance, Manhattan distance, Minkowski distance. But in this experiment, the **Euclidean distance** formula is going to be used.

The value of K i.e., the nearest data points is usually taken as an odd number as the majority vote is taken after measuring the nearest neighbor points from the new instance.

II. Experimental Design / Methodology

The following steps were taken in order to complete the 3 tasks that were assigned.

A. Plotting Training Sample Points

The training data set consisted of 6 vector samples of which 3 belonged to class 1 and the remaining to class 2. Using the python package '**matplotlib**' I have plotted all sample points from the '*train-perceptron.txt*' file making sure that the samples from the same class are marked with the same color and marker. Storing the values of the 2 classes in separate **numpy** arrays the points are plotted.

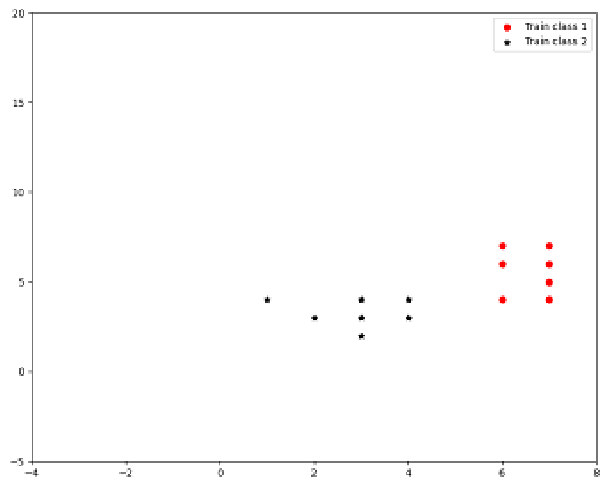

Fig. 1. Plotting training sample points

B. Predicting Test Points Using KNN

Now after plotting the test point the KNN algorithm is to be implemented. To do so at first, the value of 'k' is taken as an input from the user. After that, the euclidean distance is calculated between the neighbors and the test point using the following formula.

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

Next, after calculating the distance, all the neighbors are sorted in ascending order concerning the distance. The top K neighbors are taken and a majority voting is done on the class values. For example, if the value of k=3 then if 2 points belong to class 1 and another one to class 2 the test point will be predicted as a class 1 point based on the majority number of class count. In this way for all the test points the class is predicted using KNN.

C. Plotting the predicted test points

After predicting all the class for the 9 test points we plot them using different colored markers. The value of the predicted class are stored in an array and for each test point the points are plotted using matplotlib python package.

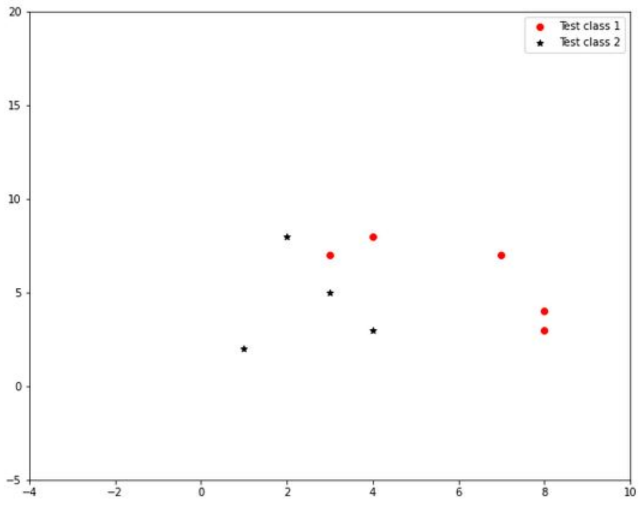

Fig. 2. Plotting testing sample points

III. Result Analysis

KNN uses all the training samples to predict a class and for larger data it can give better results. But at the same time for noisy training data it can be robust. The value of K plays a major role as the class is predicted based on the majority votes. If k=1 then the number of neighbor is quite low, at the same time for higher value of k the neighbor will be larger giving stable results. But it should be optimum enough that unnecessary neighbors are not added up.

IV. Conclusion

Although KNN is easy to implement for larger training data computation can be costly. In this experiment for out small test set the algorithm tried to predict the class simply by measuring the distance and counting the maximum value for a class. Computation was faster making it easier to implement.

V. Algorithm Implementation / Code

```
# -*- coding: utf-8 -*-
"""Pr_assm4.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1kr4-
    eEmpwxcfITRe7ykIG3aq-Kn6ylEz
"""

#1
import matplotlib.pyplot as plt
import numpy as np
train_data =np.loadtxt('/content/drive/MyDrive/4.2/
    pattern /lab5/train_knn.txt',delimiter=",",dtype
    ='int32')
test_data =np.loadtxt('/content/drive/MyDrive/4.2/
    pattern /lab5/test_knn.txt',delimiter=",",dtype=
    'int32')

print("Train data:\n",train_data);


train_data_c1=[];
train_data_c2=[];
for item in train_data:
    if item[2]==1:
        #print(item[2]);
        train_data_c1.append(item)
    elif item[2]==2:
        train_data_c2.append(item)
train_data_class1= np.array(train_data_c1);
train_data_class2= np.array(train_data_c2);

x1,y1= train_data_class1[:,0],train_data_class1
    [:,1];
x2,y2= train_data_class2[:,0],train_data_class2
    [:,1];
print("x1",x1);
print("x2",x2);
fig=plt.figure(1, figsize=(10,8))
chart1= fig.add_subplot()
chart1.scatter(x1,y1,marker='o',color='r',label='
    Train class 1');
chart1.scatter(x2,y2,marker='*',color='k',label='
    Train class 2');

chart1.axis([-4,8,-5,20]);
chart1.legend()#
plt.savefig('TrainClass.png')

print("Test data:\n",test_data);

from math import sqrt
def euclidean_distance(x,y):
  euc_dis=sqrt(pow((x[0]-y[0]),2)+pow((x[1]-y[1]),2)
    )
  return euc_dis

print(euclidean_distance(test_data[0],test_data[1]))

'''KNN ALGORITHM'''
predicted=[]
out=open('prediction.txt','w')
def KNN(k,test_xy):
    ranked_data=[]
    c1=0
    c2=0
    print("Test Point ",test_xy[0],test_xy[1],end="\
    n")
    out.write("Test Point:{},{}\n".format(test_xy
    [0],test_xy[1]))

    for val in train_data:
        dis=euclidean_distance(point,val)
        ranked_data.append((dis,val[2]))
        #print(ranked_data)
    ranked_data.sort(key=lambda x:x[0])
    for i in range(k):
        print("Distance ",i+1,": ","{:.2f}".format(
    ranked_data[i][0]),"\tclass:","{:.2f}".format(
    ranked_data[i][1]),end="\n")
        out.write("Distance "+str(i+1)+": "+"{:.2f}"
    .format(ranked_data[i][0])+"    class:"+"{:.2f}".
    format(ranked_data[i][1])+"\n")
    for i in range(k):
        if(ranked_data[i][1]==1):
            c1+=1
        else:
            c2+=1
    if(c1>=c2):
        print("Predicted Class ","{:.2f}".format(1),
    end="\n")
        out.write("Predicted Class "+"{:.2f}".format
    (1)+"\n")
        predicted.append(1)
    else:
        print("Predicted Class ","{:.2f}".format(2),
    end="\n")
        out.write("Predicted Class "+"{:.2f}".format
    (2)+"\n")
        predicted.append(2)
    out.write("\n")

K=int(input('Enter K value:'))
for point in test_data:
    KNN(K,point)
out.close()

print("Predicted classes: ",predicted)
test_data_c1=[];
test_data_c2=[];
i=0
for item in test_data:
    if predicted[i]==1:
        #print(item[2]);
        test_data_c1.append(item)
    elif predicted[i]==2:
        test_data_c2.append(item)
    i=i+1
test_data_class1= np.array(test_data_c1);
test_data_class2= np.array(test_data_c2);

x_test_1,y_test_1= test_data_class1[:,0],
    test_data_class1[:,1];
x_test_2,y_test_2= test_data_class2[:,0],
    test_data_class2[:,1];
print("x1",x_test_1);
print("x2",x_test_2);
fig=plt.figure(1, figsize=(10,8))
chart1= fig.add_subplot()
chart1.scatter(x_test_1,y_test_1,marker='o',color='r
    ',label='Test class 1');
chart1.scatter(x_test_2,y_test_2,marker='*',color='k
    ',label='Test class 2');

chart1.axis([-4,10,-5,20]);
chart1.legend()#
plt.savefig('TestClass.png')

print("file content")
with open('/content/prediction.txt', 'r') as f:
    print(f.read())
```