Implementing K-Nearest Neighbors (KNN)

Devopriya Tirtho
16.02.04.033
Department of Computer Science and Engineering
Ahsanullah University of Science and Technology
Dhaka, Bangladesh

***Abstract*—In 'Machine Learning', for predicting the accurate class of an unknown sample we need to visualize our training samples whether they are linearly separable or not. K-Nearest Neighbors' classifier helps to classify data points perfectly with the help of euclidean distance between points and classify them according to nearest neighbors. This classifier is a robust model for classifying unknown data points with the help of trained data.**

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

I. Introduction

K-Nearest Neighbors is also known as **KNN** classifier. This type of classifier is used for regression and classification problem. For the both types, this method works with the help of **K** numbered nearest neighbors. For classification, from a supervised dataset the data points are defined properly. For unknown samples, each sample is measured by the distance from the point to each of the training points. After distance measurement, the K-nearest neighbors are picked and the type of class which is in majority define the sample point's class. The main drawback of the classification method is picking the **K** number and problem occurs when the class distribution is skewed. Here, in this assignment, we will implement a **K-Nearest Neighbors** classifier to predict the class of some unknown samples.

II. Task

There are two datasets named 'train' and 'test' where all the datapoints are given. The 'train' dataset is provided with datapoints along with their respective classes. The 'train' datapoints with their respective classes are given :

Then there is a 'test' dataset which is given with the datapoints to predict the accurate classes of those samples. The 'test' datapoints are given below:
$W = \{(3,7),(7,7),(4,3),(2,8),(3,5),(1,2),(4,8),(8,3),(8,4)\}$

Here are the tasks, which we have to do for implementing the **K-Nearest Neighbors** classifier:

- The first task is to take input from the 'train' dataset and plot them with different colored markers according to the assigned class label.
- Then, we have to implement the **KNN** algorithm where the value of 'K' has to be taken from the user.

TABLE I
Table for Sample Training Data

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

- Finally, we have to print the top 'K' distances along with their class labels.

III. Experimental Design

- For the first task, we have to plot the datapoints with according to their class labels with different colored markers.

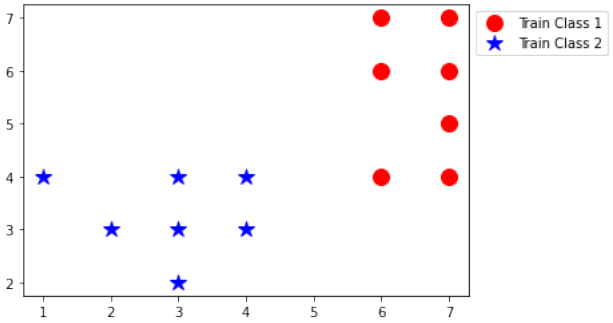

Fig. 1. Visualization of the Training Samples

- For the implementation of the **K-Nearest Neighbors** classifier, we need to take each datapoint and find the **Euclidean Distance** from each training samples. The rule for calculating the **Euclidean Distance** is:

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{1}$$

- Then, we have to take input of 'K' and find the K-nearest neighbors to classify the datapoint. Here, is a example of predicting the class of $(3, 7)$ datapoint in a tabular form:

X1	X2	Squared Distance to (3,7)	Rank Minimum Distance	K=3, is it included?	Class
7	7	(7-3)^2+(7-7)^2 =	16	No	N/A
7	4	(7-3)^2+(4-7)^2 =	25	No	N/A
6	4	(6-3)^2+(4-7)^2 =	18	No	N/A
7	5	(7-3)^2+(5-7)^2 =	20	No	N/A
7	6	(7-3)^2+(6-7)^2 =	17	No	N/A
6	7	(6-3)^2+(7-7)^2 =	9	Yes	1
6	6	(6-3)^2+(6-7)^2 =	10	Yes	1
3	4	(3-3)^2+(4-7)^2 =	9	Yes	2
2	3	(2-3)^2+(3-7)^2 =	17	No	N/A
3	2	(3-3)^2+(2-7)^2 =	25	No	N/A
4	3	(4-3)^2+(3-7)^2 =	17	No	N/A
3	3	(3-3)^2+(3-7)^2 =	16	No	N/A
4	4	(4-3)^2+(4-7)^2 =	10	No	N/A
1	4	(1-3)^2+(4-7)^2 =	13	No	N/A

Here, the predicted class is 1 as the majority of nearest neighbors refer to class 1.

- Finally, we have printed the class labels for each datapoint with the stat.

```
Test Point: 3 7
Distance 1 : 9
Class 1 : 1
Distance 2 : 9
Class 2 : 2
Distance 3 : 10
Class 3 : 1
Predicted Class: 1
-----------------------------------
Test Point: 7 7
Distance 1 : 0
Class 1 : 1
Distance 2 : 1
Class 2 : 1
Distance 3 : 1
Class 3 : 1
Predicted Class: 1
-----------------------------------
Test Point: 4 3
Distance 1 : 0
Class 1 : 2
Distance 2 : 1
Class 2 : 2
Distance 3 : 1
Class 3 : 2
Predicted Class: 2
-----------------------------------
Test Point: 2 8
Distance 1 : 17
Class 1 : 1
Distance 2 : 17
Class 2 : 2
Distance 3 : 17
Class 3 : 2
Predicted Class: 2
```

Fig. 2. Output of Test Samples after Implementing KNN Algorithm

IV. Result Analysis

The classification for sample data is given below:

TABLE II
TABLE FOR CLASSIFICATION OF SAMPLE DATA

X1	*X2*	*Class*
3	**7**	**1**
7	**7**	**1**
4	**3**	**2**
2	**8**	**2**
3	**5**	**2**
1	**2**	**2**
4	**8**	**1**
8	**3**	**1**
8	**4**	**1**

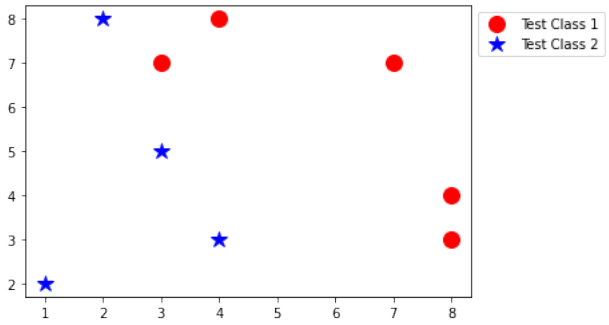

Fig. 3. Visualization of Testing Data

V. Python Code

```
# -*- coding: utf-8 -*-
"""160204033_Assignment04.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/
drive/1IIedNgjzpWxBnt-J25YWg75iCnJ2J5nr
"""

import pandas as pd

import numpy as np
import matplotlib.pyplot as plt

train_data = pd.read_csv('train_knn.txt',
sep=",", header=None)
train_data.columns = ["x1", "y1","class"]

train_data

test_data = pd.read_csv('test_knn.txt',
 sep=",", header=None)
test_data.columns = ["x1", "y1"]
```

```
test_data

b=train_data.shape
print(b[0])
RowNumber=b[0]

c=test_data.shape
print(c[0])
RowNumberTest=c[0]

train_class1_x=[]
train_class1_y=[]
train_class2_x=[]
train_class2_y=[]


for i in range(0,RowNumber):
    y=train_data.at[i,'class']
    if(y==1):
      train_class1_x.
        append(train_data.at[i,'x1'])
      train_class1_y.
        append(train_data.at[i,'y1'])
    else:
      train_class2_x.
        append(train_data.at[i,'x1'])
      train_class2_y.
        append(train_data.at[i,'y1'])


plt.plot(train_class1_x,train_class1_y,
'ro', markersize=12,label="Train Class 1")
plt.plot(train_class2_x,train_class2_y,
'b*', markersize=12,label="Train Class 2")
plt.legend(bbox_to_anchor=(1, 1),
 loc="upper left")

NearestNeighbor=3


class_predict=[]
class_final_predict=[]
distance=[]
for i in range(0,RowNumberTest):
  print("Test Point:",
  test_data.at[i,'x1'],
  test_data.at[i,'y1'])
  for j in range(0,RowNumber):
      distance_cal=(train_data.at[j,'x1']-
        test_data.at[i,'x1'])**2 +
      (train_data.at[j,'y1']-
        test_data.at[i,'y1']) **2
      class_pred=train_data.at[j,'class']
      distance.
        append([distance_cal,class_pred])
  distance.sort()
  for k in range(0,NearestNeighbor):

  dis_array=np.array(distance)

  for k in range (0, NearestNeighbor):
    print("Distance",k+1,":",dis_array[k,0])
    print("Class",k+1,":",dis_array[k,1])
    class_predict.append(dis_array[k,1])


  class1=class_predict.count(1)
  class2=class_predict.count(2)
  if (class1>class2):
    class_final_predict.append(1)
    print("Predicted Class: 1",)
  else:
    class_final_predict.append(2)
    print("Predicted Class: 2",)

  print("-----------------------------------")
  class_predict=[]
  distance=[]


print(class_final_predict)

test_data['class']=class_final_predict

test_data

test_class1_x=[]
test_class1_y=[]
test_class2_x=[]
test_class2_y=[]


for i in range(0,RowNumberTest):
    y=test_data.at[i,'class']
    if(y==1):
      test_class1_x.
        append(test_data.at[i,'x1'])
      test_class1_y.
        append(test_data.at[i,'y1'])
    else:
      test_class2_x.
        append(test_data.at[i,'x1'])
      test_class2_y.
        append(test_data.at[i,'y1'])


plt.plot(test_class1_x,test_class1_y,
'ro', markersize=12,label="Test Class 1")
plt.plot(test_class2_x,test_class2_y,
'b*', markersize=12,label="Test Class 2")
plt.legend(bbox_to_anchor=(1, 1),
 loc="upper left")
```

VI. Conclusion

'KNN Classifier' is a simple classification technique which uses **Euclidean Distancing** method to classify sample points according to number of nearest neighbors.