Implementing K-Nearest Neighbors (KNN)

Ahmad Subaktagin Jabir
Department of Computer Science and Engineering
Ahsanullah University of Science and Technology
Dhaka, Bangladesh.
160204061@aust.edu

***Abstract*—K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique. It assumes the similarity between the new case and available cases and puts the new case into the category that is most similar to the available categories. It is a non-parametric algorithm, which means it does not make any assumption on underlying data. It is also called a lazy learner algorithm. We will see in this experiment how this algorithm works.**

***Index Terms*—Euclidean Distance, Machine Learning, Supervised Classification.**

I. Introduction

K-Nearest Neighbor is an algorithm that predicts the classification of a new sample point using data from several groups. It is a non-parametric and a lazy learner algorithm. This 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 K- NN algorithm. At the training phase this algorithm just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data. In order to work with KNN Algorithm, we will follow below steps:

- Determine parameter K = number of nearest neighbors. Here we will use an odd number so that we can have a tiebreaker.
- Calculate the distance between the query-instance and all the training samples. We will use Euclidean Distance for the calculation.
- Sort the distance and determine nearest neighbors based on the K-th minimum distance.
- Gather the category of the nearest neighbors.
- Use the simple majority of the category of nearest neighbors as the prediction value of the query instance.

II. Experimental Design / Methodology

A. **Classifying Training Points:** We will read from the "$train\_knn.txt$" file and we will classify the points according to classes with different color and markers. After plotting, I got the following figure:

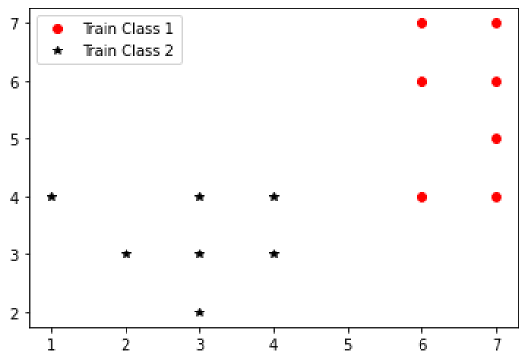


Fig. 1. Classifying Train Data Points

B. **Implementing KNN Algorithm:** In this part, we will implement the KNN Algorithm. First of all, we will determine K values. After that, we will calculate the distance between the training and testing samples. For calculation, we will be using Euclidean Distance and the equation is given below:

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

We will not use square root here so that we can get less computational cost. After calculation, we will sort the distance and determine the nearest neighbours based on the K-th minimum distance. After that, we will choose top K rows from the sorted array. Now we will assign a class to the test point which has the most appearance.

III. Result Analysis

1) **Predicting class of Test Points:** Using k=3 and test samples from "$test\_knn.txt$", I got following figure after applying KNN Algorithm:

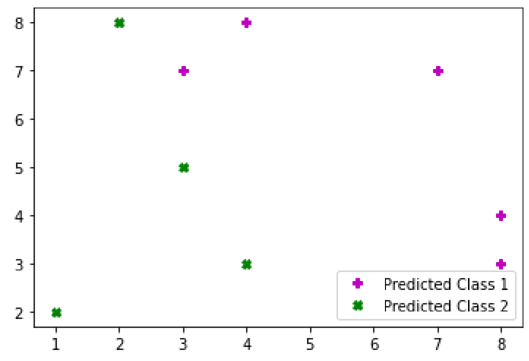
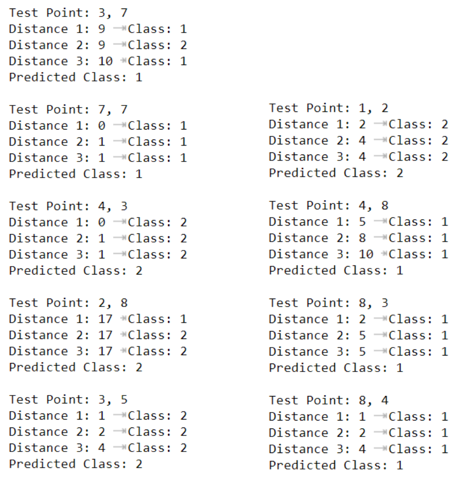


Fig. 2. Prediction class of Test Points

Here, I used different colors and markers to separate one class from another class.

2) **Saving Results in a Text File:** After getting all the results, I stored all results and predicted class in ”*prediction.txt*” file. Output from the text file is given below:



```
Test Point: 3, 7
Distance 1: 9 →Class: 1
Distance 2: 9 →Class: 2
Distance 3: 10 →Class: 1
Predicted Class: 1

Test Point: 7, 7                    Test Point: 1, 2
Distance 1: 0 →Class: 1             Distance 1: 2 →Class: 2
Distance 2: 1 →Class: 1             Distance 2: 4 →Class: 2
Distance 3: 1 →Class: 1             Distance 3: 4 →Class: 2
Predicted Class: 1                  Predicted Class: 2

Test Point: 4, 3                    Test Point: 4, 8
Distance 1: 0 →Class: 2             Distance 1: 5 →Class: 1
Distance 2: 1 →Class: 2             Distance 2: 8 →Class: 1
Distance 3: 1 →Class: 2             Distance 3: 10 →Class: 1
Predicted Class: 2                  Predicted Class: 1

Test Point: 2, 8                    Test Point: 8, 3
Distance 1: 17 →Class: 1            Distance 1: 2 →Class: 1
Distance 2: 17 →Class: 2            Distance 2: 5 →Class: 1
Distance 3: 17 →Class: 2            Distance 3: 5 →Class: 1
Predicted Class: 2                  Predicted Class: 1

Test Point: 3, 5                    Test Point: 8, 4
Distance 1: 1 →Class: 2             Distance 1: 1 →Class: 1
Distance 2: 2 →Class: 2             Distance 2: 2 →Class: 1
Distance 3: 4 →Class: 2             Distance 3: 4 →Class: 1
Predicted Class: 2                  Predicted Class: 1
```

Fig. 3. Outputs from ”*prediction.txt*” file

IV. Conclusion

The purpose of K-Nearest Neighbors 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. But there are some disadvantages in this algorithm also. One of them is having high computation cost because of calculating the distance between the data points for all the training samples. In spite of that, this algorithm is simple to implement and robust to the noisy training datas.

V. Algorithm Implementation / Code

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import csv

### Loading Train Dataset
train_set = []
with open('train_knn.txt','r') as file:
    new_reader = csv.reader(file,delimiter=',')
    for row in new_reader:
        train_set.append(row)

for i in range(len(train_set)):
    for j in range(len(train_set[i])):
        train_set[i][j] = int(train_set[i][j])

### Separating Classes
a = []
b = []
for train in train_set:
    if train[2]==1:
        a.append([train[0], train[1]])
    elif train[2]==2:
        b.append([train[0], train[1]])
class_a = np.array(a)
class_b = np.array(b)

### Plotting Train Dataset Points
plt.plot(class_a[:,0:1],class_a[:, 1:], linestyle =
    '', marker='o', color='r', label ="Train Class 1
    ")
plt.plot(class_b[:,0:1],class_b[:, 1:], linestyle =
    '', marker = '*', color = 'k', label ="Train
    Class 2")
plt.legend()
plt.show()

### Loading Test Dataset
test_set = []
with open('test_knn.txt','r') as file:
    new_reader = csv.reader(file,delimiter=',')
    for row in new_reader:
        test_set.append(row)

for i in range(len(test_set)):
    for j in range(len(test_set[i])):
        test_set[i][j] = int(test_set[i][j])

### Euclidean Distance
def euclidean_distance(row1, row2):
    distance = (row1[0] - row2[0])**2 + (row1[1] -
    row2[1])**2
    return distance

output = open('prediction.txt','a')

### KNN Algorithm
pred_class1 = []
pred_class2 = []

def KNN(k, point):
    top = []
    output.write("Test Point: %d, %d\n"%(point[0],
    point[1]))
    cnt_1 = 0
    cnt_2 = 0
    for item in train_set:
        dist = euclidean_distance(point, item)
        top.append((dist, item[2]))
    top.sort(key=lambda x:x[0])
    for i in range(k):
        output.write("Distance %d: %d \tClass: %d\n"
    %(i+1, top[i][0], top[i][1]))
    for i in range(k):
        if(top[i][1]==1):
            cnt_1 = cnt_1+1
        else:
            cnt_2 = cnt_2+1
    if(cnt_1>=cnt_2):
        output.write("Predicted Class: 1")
        pred_class1.append(point)
    else:
        output.write("Predicted Class: 2")
        pred_class2.append(point)
    output.write("\n\n")

k=int(input('Enter the Value of K: '))
for point in test_set:
    KNN(k, point)
output.close()

### Plotting Predicted Classes
pred_class1 = np.array(pred_class1)
pred_class2 = np.array(pred_class2)

plt.plot(pred_class1[:,0:1],pred_class1[:, 1:],
    linestyle = '', marker='P', color='m', label ="
    Predicted Class 1")
plt.plot(pred_class2[:,0:1],pred_class2[:, 1:],
    linestyle = '', marker = 'X', color = 'g', label
     ="Predicted Class 2")
plt.legend()
plt.show()
```