Implementing K-Nearest Neighbors (KNN)

Anas Sikder
Department of CSE
Ahsanullah University of Science and Technology
Dhaka, Bangladesh
160204021@aust.edu

***Abstract*—My task for this experiment was to classify some unlabeled test set with the help of the KNN algorithm. I implemented the algorithm using the three distance formula i,e. Euclidean, Manhattan, Minkowski and labeled the test data where I vary the number of nearest neighbors value and observe the class label for different settings. It works fine in my experimental test set.**

***Index Terms*—K-Nearest Neighbors, Euclidean distance, Manhatten distance, Minkowski distance, class, training samples, testing samples, etc.**

I. Introduction

K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on the Supervised Learning technique.KNN is a lazy learning algorithm because it does not have a specialized training phase and uses all the data for training while classification. Also, it is a non-parametric learning algorithm because it does not assume anything about the underlying data.

A. Motivation

My main goal is to implementing K-Nearest Neighbors (KNN) on given training sets and labeled the test sets using the KNN algorithm.

B. Theory

1) K-Nearest Neighbour: Here is step by step on how to compute K-nearest neighbors algorithm:

- Firstly I determined parameter K = number of nearest neighbors value.
- Then I calculated the distance between the query-instance and all the training samples.
- Sort the distance and determining nearest neighbors based on the K-th minimum distance.
- Gathering the category of the nearest neighbor.
- Finally using the simple majority of the category or a majority vote of nearest neighbors as the prediction value of the query instance.

2) Euclidean distance: For two data points P1(x_1, y_1) and P2(x_2, y_2) the distance using Euclidean formula calculated as:

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

3) Manhatten distance: For two data points P1(x_1, y_1) and P2(x_2, y_2) the distance using Manhattan formula calculated as:

$$distance = |x_1 - x_2| + |y_1 - y_2|$$

4) Minkowski Distance: For two data points P1(x_1, y_1) and P2(x_2, y_2) the distance using Minkowski formula calculated as:

$$distance = ((x_1 - x_2)^p + (y_1 - y_2)^p)^{\frac{1}{p}}$$

here 'p' is the input order parameter.When $p = 1$ it is called City block distance and for $p = 2$ it is called Euclidean distance.

II. Experimental Design / Methodology

A. Task 1

I took the train data and separated them based on the class label given and plot the points with different color markers with the help of the class label.

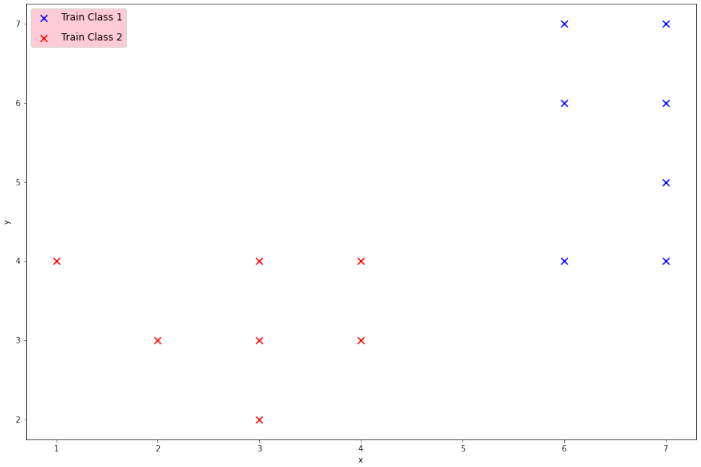

Fig. 1. Train Set

B. Task 2

Then I implement the KNN algorithm described in the theory section and for the distance between query instance and training data I used 3 types of formulas also described in the theory section. Then classify the test points with different color markers according to the class label.

C. Task3

In the final task top, K distances along with their class labels and the predicted class were stored into three text files for three different distance settings.

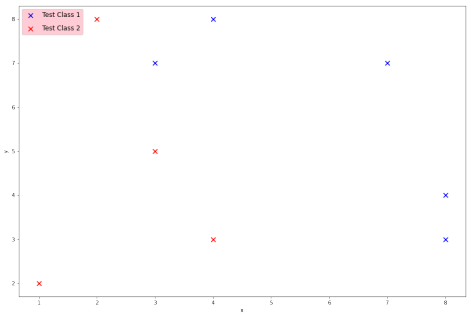

Fig. 2. Test Set with K=3 in Euclidean setting

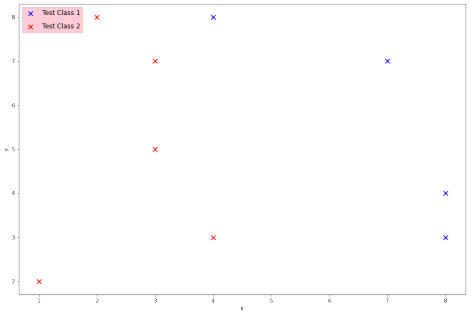

Fig. 3. Test Set with K=7 in Mankowski setting

predict_eu.txt

```
test point [3 7]
distance 1 : 3.0 class 1
distance 2 : 3.0 class 2
distance 3 : 3.16 class 1
predicted  class 1

test point [7 7]
distance 1 : 0.0 class 1
distance 2 : 1.0 class 1
distance 3 : 1.0 class 1
predicted  class 1

test point [4 3]
distance 1 : 0.0 class 2
distance 2 : 1.0 class 2
distance 3 : 1.0 class 2
predicted  class 2

test point [2 8]
distance 1 : 4.12 class 1
distance 2 : 4.12 class 2
distance 3 : 4.12 class 2
predicted  class 2

test point [3 5]
distance 1 : 1.0 class 2
distance 2 : 1.41 class 2
distance 3 : 2.0 class 2
predicted  class 2

test point [1 2]
distance 1 : 1.41 class 2
distance 2 : 2.0 class 2
distance 3 : 2.0 class 2
predicted  class 2

test point [4 8]
distance 1 : 2.24 class 1
distance 2 : 2.83 class 1
distance 3 : 3.16 class 1
predicted  class 1

test point [8 3]
distance 1 : 1.41 class 1
distance 2 : 2.24 class 1
distance 3 : 2.24 class 1
predicted  class 1

test point [8 4]
distance 1 : 1.0 class 1
distance 2 : 1.41 class 1
distance 3 : 2.0 class 1
predicted  class 1
```

Fig. 4. labeling in file for test classes using Euclidean Distance

predict_man.txt

```
test point [3 7]
distance 1 : 3 class 1
distance 2 : 3 class 2
distance 3 : 4 class 1
predicted  class 1

test point [7 7]
distance 1 : 0 class 1
distance 2 : 1 class 1
distance 3 : 1 class 1
predicted  class 1

test point [4 3]
distance 1 : 0 class 2
distance 2 : 1 class 2
distance 3 : 1 class 2
predicted  class 2

test point [2 8]
distance 1 : 5 class 1
distance 2 : 5 class 2
distance 3 : 5 class 2
predicted  class 2

test point [3 5]
distance 1 : 1 class 2
distance 2 : 2 class 2
distance 3 : 2 class 2
predicted  class 2

test point [1 2]
distance 1 : 2 class 2
distance 2 : 2 class 2
distance 3 : 2 class 2
predicted  class 2

test point [4 8]
distance 1 : 3 class 1
distance 2 : 4 class 1
distance 3 : 4 class 1
predicted  class 1

test point [8 3]
distance 1 : 2 class 1
distance 2 : 3 class 1
distance 3 : 3 class 1
predicted  class 1

test point [8 4]
distance 1 : 1 class 1
distance 2 : 2 class 1
distance 3 : 2 class 1
predicted  class 1
```

Fig. 5. labeling in file for test classes using Manhattan Distance

predict_eu.txt

```
test point [3 7]
distance 1 : 3.0 class 1
distance 2 : 3.0 class 2
distance 3 : 3.16 class 1
predicted  class 1

test point [7 7]
distance 1 : 0.0 class 1
distance 2 : 1.0 class 1
distance 3 : 1.0 class 1
predicted  class 1

test point [4 3]
distance 1 : 0.0 class 2
distance 2 : 1.0 class 2
distance 3 : 1.0 class 2
predicted  class 2

test point [2 8]
distance 1 : 4.12 class 1
distance 2 : 4.12 class 2
distance 3 : 4.12 class 2
predicted  class 2

test point [3 5]
distance 1 : 1.0 class 2
distance 2 : 1.41 class 2
distance 3 : 2.0 class 2
predicted  class 2

test point [1 2]
distance 1 : 1.41 class 2
distance 2 : 2.0 class 2
distance 3 : 2.0 class 2
predicted  class 2

test point [4 8]
distance 1 : 2.24 class 1
distance 2 : 2.83 class 1
distance 3 : 3.16 class 1
predicted  class 1

test point [8 3]
distance 1 : 1.41 class 1
distance 2 : 2.24 class 1
distance 3 : 2.24 class 1
predicted  class 1

test point [8 4]
distance 1 : 1.0 class 1
distance 2 : 1.41 class 1
distance 3 : 2.0 class 1
predicted  class 1
```

Fig. 6. labeling in file for test classes using Minkowski Distance

III. Result Analysis

The total number of test sets was nine. My task was to either labeled them class 1 or class 2 depending on the nearest neighbor's value 'K'.For all three distances having different K values, I observed class label. For my experiment I took K as an odd value as having K as an even value sometimes for some of the data points it is not possible to label them any of the classes.

A. Using Euclidean distance

1) Number of nearest neighbors, $K = 3$*:* When I took the number of nearest neighbors 3 in the Euclidean setting, I found **five data points were labeled as class 1 and four data points were labeled as class 2.The decision boundary will be drawn between two classes.**

2) Number of nearest neighbors, $K = 5$*:* When I took the number of nearest neighbors 5 in this setting, I found **four data points are labeled as class 1 and five data points labeled as class 2. The decision boundary will be shifted left in between the two classes.**

3) Number of nearest neighbors, $K = 7$*:* When I took the number of nearest neighbors 7 in this setting, I found **four data points are labeled as class 1 and five data points labeled as class 2. The decision boundary will be drawn between the two classes.**

4) Number of nearest neighbors, $K = 9$*:* When I took the number of nearest neighbors 3 in this setting, I found **four data points are labeled as class 1 and five data points labeled as class 2. The decision boundary will be drawn between 2 classes.**

Using Euclidian distance				
Data points	K=3	K=5	K=7	K=9
$[3, 7]$	class1	class2	class2	class2
$[7, 7]$	class1	class1	class1	class1
$[4, 3]$	class2	class2	class2	class2
$[2, 8]$	class2	class2	class2	class2
$[3, 5]$	class2	class2	class2	class2
$[1, 2]$	class2	class2	class2	class2
$[4, 8]$	class1	class1	class1	class1
$[8, 3]$	class1	class1	class1	class1
$[8, 4]$	class1	class1	class1	class1

For further testing, I can choose K as 11,13, up to training data points.

B. Using Manhattan distance

1) Number of nearest neighbors, $K = 3$*:* When I took the number of nearest neighbors 3 in the Manhattan setting, I found **five data points are labeled as class 1 and four data points labeled as class 2. The decision boundary will be drawn between two classes.**

2) Number of nearest neighbors, $K = 5$*:* When I took the number of nearest neighbors 5 in this setting, I found **five data points are labeled as class 1 and four data points labeled as class 2. The decision boundary will be drawn between the two classes.**

3) Number of nearest neighbors, $K = 7$*:* When I took the number of nearest neighbors 7 in this setting, I found **five data points are labeled as class 1 and four data points labeled as class 2. The decision boundary will be drawn between the two classes.**

4) Number of nearest neighbors, $K = 9$*:* When I took the number of nearest neighbors 9 in this setting, I found **four data points are labeled as class 1 and five data points labeled as class 2. The decision boundary will be shifted left in between the two classes.**

Using Manhattan distance				
Data points	K=3	K=5	K=7	K=9
$[3, 7]$	class1	class1	class1	class2
$[7, 7]$	class1	class1	class1	class1
$[4, 3]$	class2	class2	class2	class2
$[2, 8]$	class2	class2	class2	class2
$[3, 5]$	class2	class2	class2	class2
$[1, 2]$	class2	class2	class2	class2
$[4, 8]$	class1	class1	class1	class1
$[8, 3]$	class1	class1	class1	class1
$[8, 4]$	class1	class1	class1	class1

For further testing, I can choose K as 11,13, up to training data points.

C. Using Minkowski Distance

1) Number of nearest neighbors, $K = 3$*:* When I took the number of nearest neighbors 3 in this setting, I found **five data points are labeled as class 1 and four data points labeled as class 2. The decision boundary will be drawn between the two classes.**

2) Number of nearest neighbors, $K = 5$*:* When I took the number of nearest neighbors 5 in this setting, I found **five data points are labeled as class 1 and four data points labeled as class 2. The decision boundary will be drawn between two classes.**

3) Number of nearest neighbors, $K = 7$*:* When I took the number of nearest neighbors 7 in this setting, I found **four data points are labeled as class 1 and five data points labeled as class 2. The decision boundary will be shifted left in between the two classes.**

4) Number of nearest neighbors, $K = 9$*:* When I took the number of nearest neighbors 9 in this setting, I found **four data points are labeled as class 1 and five data points labeled as class 2. The decision boundary will be drawn between 2 classes.**

Using Minkowski distance				
Data points	K=3	K=5	K=7	K=9
[3, 7]	class1	class1	class2	class2
[7, 7]	class1	class1	class1	class1
[4, 3]	class2	class2	class2	class2
[2, 8]	class2	class2	class2	class2
[3, 5]	class2	class2	class2	class2
[1, 2]	class2	class2	class2	class2
[4, 8]	class1	class1	class1	class1
[8, 3]	class1	class1	class1	class1
[8, 4]	class1	class1	class1	class1

For further testing, I can choose K as 11,13, up to training data points.

IV. Conclusion

KNN performs well for given testing data sets. But when the dimension of the dataset is **larger the cost of calculating the distance between the new point and each existing train point is huge which degrades the performance of the algorithm**. Also, when **K value is even the algorithm has a huge probability of having an equal chance to label them in any of the classes which may create miss classification of test sets.**Overall for sample dataset with K as an odd value the algorithm performs quite well.

V. Algorithm Implementation / Code

```
#import necessary libaries
import math
import pandas as pd
import numpy as np
from matplotlib import cm
import matplotlib.pyplot as plt
import matplotlib.pyplot as plt

traindataset=pd.read_csv('train_knn.txt'
,sep=" ",header=None)
train=traindataset.to_numpy();
#print(train)
testdataset=pd.read_csv('test_knn.txt'
,sep=" ",header=None)
#print(testdataset)
test=testdataset.to_numpy();
print(train[0])
print(test)

Trainclass1 =([i[0],i[1]]) for i
in train if i[2] == 1]
Trainclass2 =([i[0],i[1]]) for i
in train if i[2] == 2]
Trainclass1 = np.array(Trainclass1)
Trainclass2 = np.array(Trainclass2)
print(Trainclass1)
print(Trainclass2)
x = np.concatenate((Trainclass1,
Trainclass2), axis=0)
x

fig, axis = plt.subplots()
fig.set_figheight(10)
fig.set_figwidth(15)
#scatting train values
axis.scatter(Trainclass1[:,0],Trainclass1[:,1],
marker='x',color='blue',s=70,label=
'Train Class 1')
axis.scatter(Trainclass2[:,0],Trainclass2[:,1],
marker='x',color='red',s=70,label=
'Train Class 2')
#labeling decoration
legend = axis.legend(loc='upper left',
fontsize='large',labelspacing=1.0)
legend.get_frame().set_facecolor('pink')
axis.set_xlabel('x')
axis.set_ylabel('y')

def euclidian_dist(p1, p2):
    sum = 0
    sum = math.pow(p1[0] - p2[0], 2)
    + math.pow(p1[1] - p2[1], 2)
    return round(math.sqrt(sum),2)
def knn_eu(x,train_set, new_sample, K):
    dists = {}

    for i in range(len(x)):
      d = euclidian_dist(x[i], new_sample)
      dists[i] = d

    k_neighbors = sorted(dists, key=dists.get)
    [:K]

    qty_label1 = 0
    qty_label2 = 0
    for index in k_neighbors:
        if train_set[index][-1] == 1:
            qty_label1 += 1
        else:
            qty_label2 += 1

    if qty_label1 > qty_label2:
        return 1
    else:
        return 2
def manhattan_dist(p1,p2):
    sum = 0
    sum = abs(p1[0]-p2[0])
    +  abs(p1[1]-p2[1])
    return sum
def knn_man(x,train_set, new_sample, K):
    dists = {}
```

```
    for i in range(len(x)):
      d = manhattan_dist(x[i], new_sample)
      dists[i] = d

    k_neighbors = sorted(dists,
    key=dists.get)[:K]

    qty_label1 = 0
    qty_label2 = 0
    for index in k_neighbors:
        if train_set[index][-1] == 1:
            qty_label1 += 1
        else:
            qty_label2 += 1

    if qty_label1 > qty_label2:
        return 1
    else:
        return 2
def minkowski_dist(p1,p2):
    sum = 0
    p=1.5
    pp = 1/p
    sum = math.pow((abs(p1[0]-p2[0])),p)
    +  math.pow((abs(p1[1]-p2[1])),p)
    return round(sum**pp,2)
def knn_min(x,train_set, new_sample, K):
    dists = {}

    for i in range(len(x)):
      d = minkowski_dist(x[i], new_sample)
      dists[i] = d

    k_neighbors = sorted(dists,
    key=dists.get)[:K]

    qty_label1 = 0
    qty_label2 = 0
    for index in k_neighbors:
        if train_set[index][-1] == 1:
            qty_label1 += 1
        else:
            qty_label2 += 1

    if qty_label1 > qty_label2:
        return 1
    else:
        return 2
K = int(input("Enter your value of k: ") )
l = []
for j in range(len(test)):
    label = knn_eu(x,train, test[j], K)
    l.append(label)
l

K = int(input("Enter your value of k: ") )
l = []
for j in range(len(test)):
    label = knn_man(x,train, test[j], K)
    l.append(label)
l

K = int(input("Enter your value of k: ") )
l = []
for j in range(len(test)):
    label = knn_min(x,train, test[j], K)
    l.append(label)
l

test_label = np.array(l)
print(test_label)

y=np.zeros([9,3])
y[:,0]=test[:,0]
y[:,1]=test[:,1]
y[:,2]=test_label

y

test1 =[([i[0],i[1]]) for i in y if
i[2] == 1]
test2 =[([i[0],i[1]]) for i in y if
i[2] == 2]
test1 = np.array(test1)
test2 = np.array(test2)

fig, axis = plt.subplots()
fig.set_figheight(10)
fig.set_figwidth(15)
#scatting train values
axis.scatter(test1[:,0],test1[:,1],marker='x'
,color='red',s=70,label='Test Class 1')
axis.scatter(test2[:,0],test2[:,1],marker='x'
,color='blue',s=70,label='Test Class 2')
#labeling decoration
legend = axis.legend(loc='upper left'
,fontsize='large',labelspacing=1.0)
legend.get_frame().set_facecolor('pink')
axis.set_xlabel('x')
axis.set_ylabel('y')

def knn_eu1(x,train_set, new_sample, K):
    dists = {}
    dx = []
    dx1 = []

    qty_label1 = 0
    qty_label2 = 0

    for i in range(len(x)):
      d = euclidian_dist(x[i], new_sample)
      dists[i] = d
```

```
      dx.append(d)
      #print(dx)
    dx.sort()
    k_neighbors = sorted(dists, key=dists.get)[:K]

    for i in range(K):
        dx1.append(dx[i])

    for index in range(len(k_neighbors)):
        #print(index,':',dx1[index])

        if train_set[k_neighbors[index]][-1] == 1:
            f1.write("%s %s %s %s %s %s \n"
            %("distance",index+1,":"
            ,dx1[index],"class","1"))
            print('distance ',index+1,':'
            ,dx1[index],'class',1)
            qty_label1 += 1
        else:
            f1.write("%s %s %s %s %s %s \n"
            %("distance",index+1,":"
            ,dx1[index],"class","2"))

            print('distance ',index+1,':'
            ,dx1[index],'class',2)
            qty_label2 += 1

    if qty_label1 > qty_label2:

        print('predicted  class 1')
        f1.write("%s\n" %("predicted
class 1"))

    elif qty_label1 < qty_label2:
        f1.write("%s\n" %("predicted
class 2"))
        print('predicted  class 2')

    else:
        f1.write("%s\n" %("can not predict
        into any class"))
        print('can not predict into any class')
def knn_man1(x,train_set, new_sample, K):
    dists = {}
    dx = []
    dx1 = []

    qty_label1 = 0
    qty_label2 = 0

    for i in range(len(x)):
      d = manhattan_dist(x[i], new_sample)
      dists[i] = d
      dx.append(d)
      #print(dx)
    dx.sort()
    k_neighbors = sorted(dists, key=dists.get)[:K]

    for i in range(K):
      dx1.append(dx[i])

    for index in range(len(k_neighbors)):
        #print(index,':',dx1[index])

        if train_set[k_neighbors[index]][-1] == 1:
            print('distance ',index+1,':'
            ,dx1[index],'class',1)
            f2.write("%s %s %s %s %s %s \n"
            %("distance",index+1,
            ":",dx1[index],"class","1"))
            qty_label1 += 1
        else:
            print('distance ',index+1,':'
            ,dx1[index],'class',2)
            f2.write("%s %s %s %s %s %s \n"
            %("distance",index+1,":"
            ,dx1[index],"class","2"))
            qty_label2 += 1

    if qty_label1 > qty_label2:
        print('predicted  class 1')
        f2.write("%s\n" %("predicted
class 1"))

    elif qty_label1 < qty_label2:
        print('predicted  class 2')
        f2.write("%s\n" %("predicted
class 2"))

    else:
        print('can not predict
        into any class')
        f2.write("%s\n" %("can not predict
        into any class"))
def knn_min1(x,train_set, new_sample, K):
    dists = {}
    dx = []
    dx1 = []

    qty_label1 = 0
    qty_label2 = 0

    for i in range(len(x)):
      d = minkowski_dist(x[i], new_sample)
      dists[i] = d
      dx.append(d)
      #print(dx)
    dx.sort()
    k_neighbors = sorted(dists, key=dists.get)[:K]

    for i in range(K):
        dx1.append(dx[i])
```

```
    for index in range(len(k_neighbors)):
        #print(index,':',dx1[index])

        if train_set[k_neighbors[index]]
        [-1] == 1:
            print('distance ',index+1,':'
            ,dx1[index],'class',1)
            f3.write("%s %s %s %s %s %s \n"
            %("distance",index+1,":"
            ,dx1[index],"class","1"))
            qty_label1 += 1
        else:
            print('distance ',index+1,':'
            ,dx1[index],'class',2)
            f3.write("%s %s %s %s %s %s \n"
            %("distance",index+1,":"
            ,dx1[index],"class","2"))
            qty_label2 += 1

    if qty_label1 > qty_label2:
        print('predicted  class 1')
        f3.write("%s\n" %("predicted
class 1"))

    elif qty_label1 < qty_label2:
        print('predicted  class 2')
        f3.write("%s\n" %("predicted
class 2"))

    else:
        print('can not predict into any class')
        f3.write("%s\n" %("can not predict
        into any class"))
f1 = open("predict_eu.txt", "w")
K = int(input("Enter your value of k: "))
for j in range(len(test)):

    f1.write("%s %s\n" %("test point",test[j]))
    print('test point',test[j])
    knn_eu1(x,train, test[j], K)
    f1.write('\n')
    print('\n')
f1.close()
f2 = open("predict_man.txt", "w")
K = int(input("Enter your value of k: "))
for j in range(len(test)):
    f2.write("%s %s\n" %("test point",test[j]))
    print('test point',test[j])
    knn_man1(x,train, test[j], K)
    print('\n')
    f2.write('\n')
f2.close()
f3 = open("predict_min.txt", "w")
K = int(input("Enter your value of k: "))
for j in range(len(test)):
                                        f3.write("%s %s\n" %("test point",test[j]))
                                        print('test point',test[j])
                                        knn_min1(x,train, test[j], K)
                                        print('\n')
                                        f3.write('\n')
                                f3.close()
```