Name : Rupak Saha
Roll : 1703134
Course No : CSE 4204
Course Name : Sessional Based on CSE 4203
Department : Computer Science And Engineering
Institute : Rajshahi University of Engineering And Technology

Name Of The Experiment : Implementation of K nearest neighbor classification with and without distorted pattern.

Theory:
In statistics, the k-nearest neighbors' algorithm (k-NN) is a non-parametric supervised learning method first developed by Evelyn Fix and Joseph Hodges in 1951. It is used for both classification and regression.

Algorithm:
- Define the value of K.
- Prepare a training data set with labeled class and a test data set with unlabeled class and analyze the data set.
- For each test data, calculate Euclidean Distance between the test data and every train data and take the closest K numbers of distances.
- After taking the K closest neighbor for each test data, evaluate the majority of each test data's neighbor's class and assign that class-label to the test data.
- Evaluate accuracy by comparing tested outcome of each test data's class and the true class of each test data.
- Adjust the value of K for more accuracy
- Now prepare a more distorted train dataset and repeat these steps.

Dataset:
A dataset containing two physical features of a person, weight and height was taken(normalized measuring unit). The data set has two classes, rugby player and ballet dancer.
A test dataset has also been taken. The test data set is not labeled.

Ratio of train data set and test dataset is 10/3=0.33 based on number of entries in each dataset

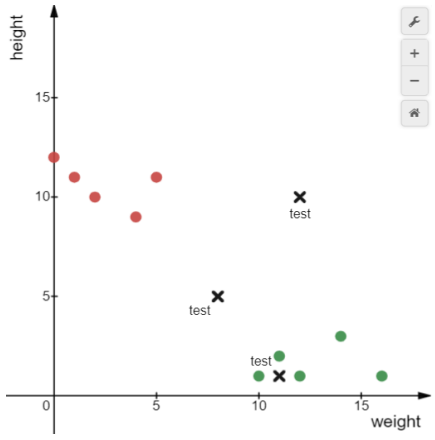

Fig1 : Train Dataset and Test Dataset plotted in graph.
- Rugby players
- Ballet dancer
- Test Subjects

Table1 : Train Data Set

Entries	**Weight**	**Height**	**Class**
Entry 0	10	1	1
Entry1	11	2	1
Entry2	12	1	1
Entry 3	14	3	1
Entry 4	16	1	1
Entry 5	2	10	2
Entry 6	1	11	2
Entry 7	0	12	2
Entry 8	5	11	2
Entry9	4	9	2

Table2 : Test Data Set

Entries	**Weight**	**Height**	**True class**
Entry0	**11**	**1**	**1**
Entry1	**8**	**5**	**2**
Entry2	**12**	**10**	**2**

Source Code:

Includes and initializing the train data set:

```
#include<iostream>
#include<math.h>
double euclid_dist(double x1,double x2,double y1,double y2)
{
    return pow(pow(x1-x2,2)+pow(y1-y2,2),0.5);
}
int main()
{
    //initialising variables
    int k;
    std::cout<<"enter K : ";
    std::cin>>k;
    int num_of_traindata=10;
    int num_of_testdata=3;


    double** Train_data;
    Train_data=new double*[num_of_traindata];
    for(int i=0; i<num_of_traindata; i++)
        Train_data[i]=new double[3];//for now only two dimension or in other words two feature vectors
    //TRAIN DATA TABLE
    std::cout<<"------TRAIN DATA TABLE-------"<<std::endl;
    Train_data[0][0]=10;Train_data[0][1]=1;Train_data[0][2]=1;//weight | height | class(1=A and 2=B)
    Train_data[1][0]=11;Train_data[1][1]=2;Train_data[1][2]=1;
    Train_data[2][0]=12;Train_data[2][1]=1;Train_data[2][2]=1;
    Train_data[3][0]=14;Train_data[3][1]=3;Train_data[3][2]=1;
    Train_data[4][0]=16;Train_data[4][1]=1;Train_data[4][2]=1;
    Train_data[5][0]=2;Train_data[5][1]=10;Train_data[5][2]=2;
    Train_data[6][0]=1;Train_data[6][1]=11;Train_data[6][2]=2;
    Train_data[7][0]=0;Train_data[7][1]=12;Train_data[7][2]=2;
    Train_data[8][0]=5;Train_data[8][1]=11;Train_data[8][2]=2;
    Train_data[9][0]=4;Train_data[9][1]=9;Train_data[9][2]=2;
    //view train_data table in console
    std::cout<<"\t"<<"weight"<<"\theight\tclass"<<std::endl;
    for(int i=0; i<num_of_traindata; i++)
    {

        std::cout<<"entry"<<i<<"\t";
        for(int j=0; j<3; j++)
            std::cout<<Train_data[i][j]<<"\t";
        std::cout<<std::endl;
    }
```

Initializing test data set

```
    Test_data=new double*[num_of_testdata];
    for(int i=0; i<num_of_testdata; i++)
        Test_data[i]=new double[3];
    //TEST DATA TABLE
    std::cout<<"\n\n-------TEST DATA TABLE-------"<<std::endl;
    Test_data[0][0]=11;Test_data[0][1]=1;Test_data[0][2]=1;//weight | height | true class(1 for classA & 2 for class
    Test_data[1][0]=8;Test_data[1][1]=5;Test_data[1][2]=2;
    Test_data[2][0]=12;Test_data[2][1]=10;Test_data[2][2]=2;
    //view test data table in console
    std::cout<<"\t"<<"weight"<<"\theight\ttrue class"<<std::endl;
    for(int i=0; i<num_of_testdata; i++)
    {
        std::cout<<"entry"<<i<<"\t";
        for(int j=0; j<3; j++)
            std::cout<<Test_data[i][j]<<"\t";
        std::cout<<std::endl;
    }
std::cout<<std::endl;
```

K nearest neighbor algo

```
    //K NEAREST NEIGHBOR
    double distances[k][2];//an array to store measured distances from the test subject to its nearest
    //k neighbours and the class of the neighbour
    int closest_dist_nums_of_cA;//a variable to store the num of class1 closest neighbiors in the distances array
    int closest_dist_nums_of_cB;//a variable to store the num of class2 closest neighbiors in the distances array
    for(int i=0; i<num_of_testdata; i++)//iterate through test subjects
    {
        std::cout<<"\n\n-->For Test entry"<<i<<" closest neighbors distances and class :"<<std::endl;
        for(int j=0; j<k; j++)
        {
            distances[j][0]=10000;//initilize the distance value
            distances[j][1]=-1;//class
        }
        closest_dist_nums_of_cA=0;
        closest_dist_nums_of_cB=0;
```

```
81          for(int j=0; j<num_of_traindata; j++)//iterate through the Train subjects for every test subject
82          {
83
84                  double dist=euclid_dist(Train_data[j][0],Test_data[i][0],Train_data[j][1],Test_data[i][1]);
85                  for(int n=0;n<k;n++)
86                  {
87                      if(dist<distances[n][0])
88                      {
89                          distances[n][0]=dist; // k closest distances from the test subject
90                          distances[n][1]=Train_data[j][2]; //and their classes
91                          break;
92                      }
93                  }
94
95          }
96          for(int j=0;j<k;j++)
97          {
98              if(distances[j][1]==1)
99                  closest_dist_nums_of_cA+=1;
100             else
101                 closest_dist_nums_of_cB+=1;
102         }
```

Printing the tested result

```
103
104         //printing result in console
105         for(int n=0;n<k;n++)
106         {
107         std::cout<<"dist"<<n<<"="<<distances[n][0]<<" & cls="<<distances[n][1]<<",\t";
108         }
109
110         std::cout<<"\nnums of c1="<<closest_dist_nums_of_cA<<" nums of c2="<<closest_dist_nums_of_cB<<std::endl;
111         if(closest_dist_nums_of_cA>=closest_dist_nums_of_cB)
112             std::cout<<"So tested class of "<<i<<"'s class = 1"<<std::endl;
113         else
114             std::cout<<"tested class of "<<i<<"'s class = 2"<<std::endl;
115     }
116
117     return 0;
118
119
120 }
```

Console Output

```
enter K : 3
------TRAIN DATA TABLE-------
       weight  height  class
entry0  10      1       1
entry1  11      2       1
entry2  12      1       1
entry3  14      3       1
entry4  16      1       1
entry5  2       10      2
entry6  1       11      2
entry7  0       12      2
entry8  5       11      2
entry9  4       9       2


-------TEST DATA TABLE-------
       weight  height  true class
entry0  11      1       1
entry1  8       5       2
entry2  12      10      2



-->For Test entry0 closest neighbors distances and class :
dist0=1 & cls=1,        dist1=1 & cls=1,        dist2=1 & cls=1,
nums of c1=3 nums of c2=0
So tested class of 0's class = 1


-->For Test entry1 closest neighbors distances and class :
dist0=4.24264 & cls=1,  dist1=5.65685 & cls=1,  dist2=5.65685 & cls=2,
nums of c1=2 nums of c2=1
So tested class of 1's class = 1


-->For Test entry2 closest neighbors distances and class :
dist0=7.07107 & cls=2,  dist1=8.06226 & cls=2,  dist2=9.84886 & cls=1,
nums of c1=1 nums of c2=2
tested class of 2's class = 2

Process returned 0 (0x0)   execution time : 3.110 s
Press any key to continue.
```

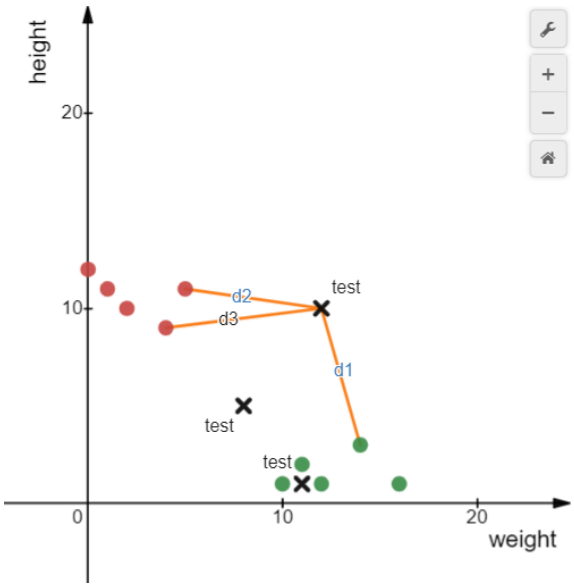

Fig2 : Visualization of the algorithm for one test data. Closest 3 neighbors has been chosen based on euclidean distance As most of the closest neighbors are from class ballet dancer so the test data will be assigned to ballet dancer class

Error Analysis:

For K =3

we see that test_data_0's class has been assigned as 1=rugby player which is correct as the true class of test_data_0 is also 1.

Test_data_1's class has been assigned as 1=rugby player which is incorrect as test_data_1's true class is 2.

Test_data_2's class has been assigned as 2=ballet dancer which is correct as the true class of it is also 2.

So for K =3 error = 0.33% for this test dataset.

Now for K=5 console output of the test result segment is below:

```
-->For Test entry0 closest neighbors distances and class :
dist0=1 & cls=1,        dist1=1 & cls=1,        dist2=1 & cls=1,        dist3=3.60555 & cls=1,  dist4=5 & cls=1,

nums of c1=5 nums of c2=0
So tested class of 0's class = 1


-->For Test entry1 closest neighbors distances and class :
dist0=4.24264 & cls=1,  dist1=5.65685 & cls=1,  dist2=5.65685 & cls=2,  dist3=6.7082 & cls=2,   dist4=9.21954 & cls=2

nums of c1=2 nums of c2=3
tested class of 1's class = 2


-->For Test entry2 closest neighbors distances and class :
dist0=7.07107 & cls=2,  dist1=8.06226 & cls=2,  dist2=9.84886 & cls=1,  dist3=10 & cls=2,       dist4=11.0454 & cls=2

nums of c1=1 nums of c2=4
tested class of 2's class = 2

Process returned 0 (0x0)   execution time : 2.700 s
Press any key to continue.
```

Now we see that for k=5 every test_data_entrie's assigned class matches the true class of that test data. Thus for K=5 the error is 0% for this test data set.

K_N_N for distorted Data set:

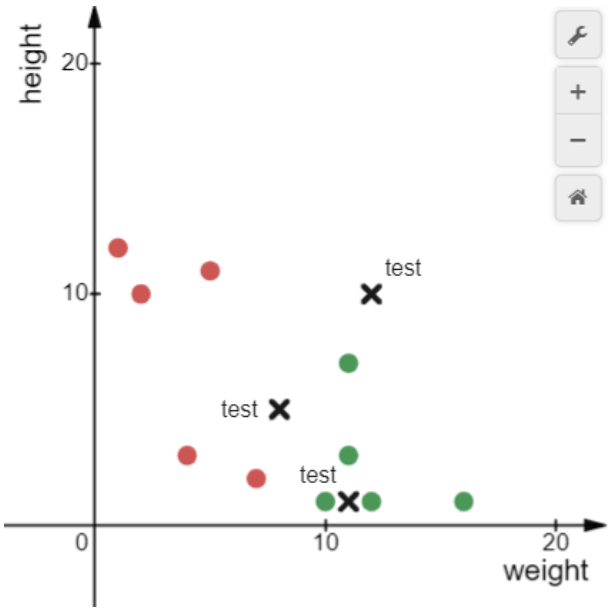

Fig3 : Distorted Train Dataset.

If the above k_n_n algorithm is applied to the above plotted data set it will not produce good results for the previous test data. Reason behind this is the classes are almost scattered around resulting in so many rogue entries. Thus K_N_N won't be able to assign the test data accurate classes

Conclusions:

According to the given tasks

First the characteristics of the datasets were defined. It was visualized that for the Train data set and Test data set that error rate decreased with the value of K increased at about a certain amount.

It was also shown that K_N_N doesn't work well with distorted training data sets.

Now with more dimensions (more features) added the distortion among class-data increases. Thus K_N_N doesn't perform well for higher dimensional data sets.

Also with the huge amount of entries in the training data set degrades the accuracy of K_N_N.