K Nearest Neighbors (KNN) Algorithm Implementation using Euclidean Distance

```
clear all; %Remove items from workspace, freeing up system memory.
clc; %Clear Command Window.
```

Step 1: Reading the dataset

```
rng("default"); %Control random number generator, produces a sequence of random numbers by defa
full_data=table2array(readtable('Data.csv')); %Create table from our sample data file and conve
A=normalize(full_data); %Normalize returns the vectorwise z-score((datapoint(x)-columnmean(XBAR
data=A(:,1:13); %Data includes all the predictor variables.
label=A(:,14); %Data includes the target variables.
k=input("Enter the number of nearest neighbors:"); %Get input from user.
```

Step 2: Preparing the data - Splitting the data into Training Set and Testing Set in a ratio of 0.9:0.1.

```
[m,n]=size(A); %Size of matrix A. Store row variables(instances) in m and columns variables(fea
P = 0.90;
idx=randperm(m); %Returns a row vector containing a random permutation of m without repeating e
datatrain = A(idx(1:round(P*m)),:); %Considering 90% of normalized dataset as our training set.
datatest = A(idx(round(P*m)+1:end),:); %Considering 10% of normalized dataset as our testing se
data_train_label = datatrain(:,14); %Data includes the target variables of training dataset.
data_test_label = datatest(:,14); %Data includes the target variables of testing dataset.
data_train_data = datatrain(:,1:13); %Data includes the predictor variables of training dataset
data_test_data = datatest(:,1:13); %Data includes the predictor variables of testing dataset.
numoftestdata = size(data_test_data,1); %Returns the number of rows of test dataset.
numoftrainingdata = size(data_train_data,1); %Returns the number of rows of training dataset.
```

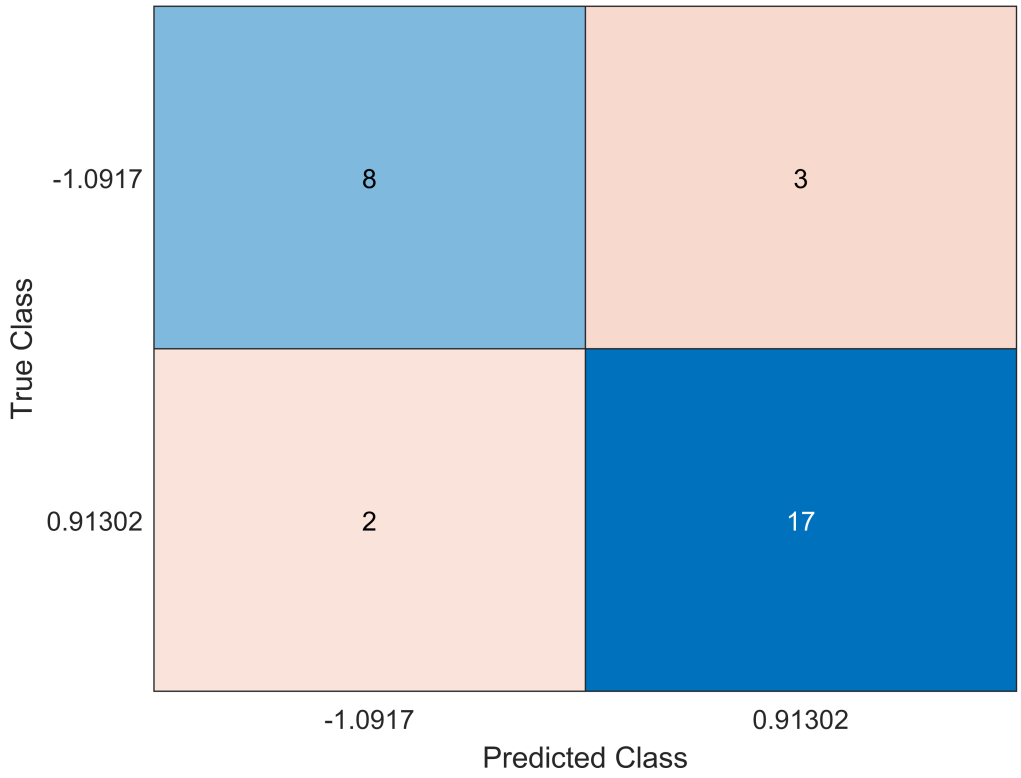
Step 3: Calculating the Euclidean Distance for all the features and instances of test data.

```
for sample=1:numoftestdata %Running a for-loop to generate each row of test data set
euclideandistance = sqrt(sum((repmat(data_test_data(sample,:),numoftrainingdata,1)-data_train_d
[dist, position] = sort(euclideandistance,'ascend'); %Sort the array and its position in ascend
nearestdistances=dist(1:k); %Nearest distance for given value of k.
nearestneighbors=position(1:k); %Nearest neighbor for given value of k.
    if(nearestdistances(1)<nearestdistances(2))
        testlabel(sample)=data_train_label(position(1));
    end

    if (nearestdistances(1)==nearestdistances(2)&&nearestdistances(1)<nearestdistances(3))
            index=randi(2,1,1);
            testlabel(sample)=data_train_label(position(index));
    end

    if (nearestdistances(1)==nearestdistances(2)&&nearestdistances(2)==nearestdistances(3)&&nea
            index=randi(3,1,1);
            testlabel(sample)=data_train_label(position(index));
    end
```

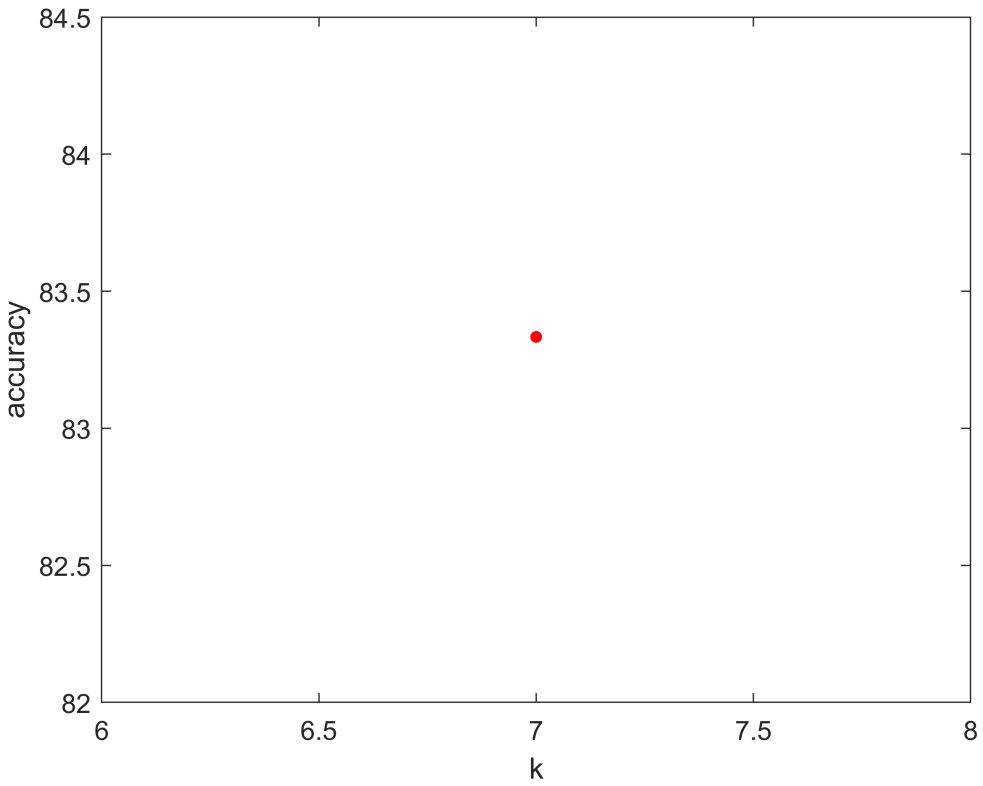
```
    if (nearestdistances(1)==nearestdistances(2)&&nearestdistances(2)==nearestdistances(3)&&nea
            index=randi(4,1,1);
            testlabel(sample)=data_train_label(position(index));
    else
        index=randi(k,1,1);
        testlabel(sample)=data_train_label(position(index));
    end
end
```

Step 4: Checking the accuracy

```
originallabel=data_test_label';
[confmat order]=confusionmat(originallabel,testlabel); %Confusion Matrix to determine the accur
accuracy=(sum(diag(confmat)))/sum(sum(confmat))*100;
figure
confusionchart(originallabel,testlabel)
```




```
figure
gscatter(k,accuracy)
```


84.5
84
83.5
83
82.5
82
accuracy
6
6.5
7
7.5
8
k