SMAI Assignment 1

Abhinav Moudgil
201331039

Introduction:

Classification of data is the process of organizing data into categories for its most efficient and effective use. A well planned data classification system makes it easy to find and retrieve.
For this assignment, kNN or k-Nearest Neighbour classifier is built. It is *non parametric lazy learning algorithm*. It doesn't make any assumptions for the underlying data distribution. By lazy learning, I mean that it does not use the training data to do any generalization. Training phase is pretty fast. During the test phase, whole training data is used. So the cost of testing phase in terms of both time and memory is high.

Datasets:

S.No.	Name	Number of classes	Number of features	Number of instances
1	Iris	3	4	150
2	Wine	3	13	178
3	Banknote Authentication	2	5	1327
4	Ionosphere	2	34	351
5	Magic Gamma Telescope	2	10	19020

Choice of distance function:

1. Iris - Here dataset is linearly separable and no data point has some exceptional distance. There euclidean distance function is chosen.

2. Wine - This dataset has values ranging from 0.32 to 1034. So, feature with large values can dominate in Euclidean distance if other features have very low values. Therefore each dimension is normalized with maximum value in that dimension.

After normalization, accuracy of kNN improved a lot.

3. Banknote Authentication - The features of this dataset nearly have same values. So there is no need for normalization or weighted euclidean distance. Normal Euclidean distance function has done a good job.

4. Ionosphere - Dataset has 34 features but all the features nearly have the same values ranging from 0 to 1. So normal Euclidean distance is considered.

5. Magic Gamma Telescope - Normalization is done with respect and then Euclidean distance is calculated.

kNN Algorithm:

First data is stored and divided into *y* equal parts (*y*-fold). One part is declared as test data and rest is training data. This completes the training phase. During test phase, a test sample is picked and all the training samples are sorted according to normal or (weighted) euclidean distance from test sample. For first *k* data points (in sorted list) polling is done. The class with maximum frequency is allotted to test data sample. Same procedure is repeated for all the test data points.
For a particular dataset, *k* is varied from 1 to 5 and *y* is varied from 2 to 5.
Mean accuracy for test dataset is calculated as follows:

(No. of test data samples - No. of wrong class samples predicted by kNN) / (No. of test data samples) * 100

<u>Tie break:</u>

It may happen when k is even. Two classes may have same frequency during polling. In this case, sum of distances for both the classes is calculated. Class with minimum sum is allotted to test data sample.

Plots:

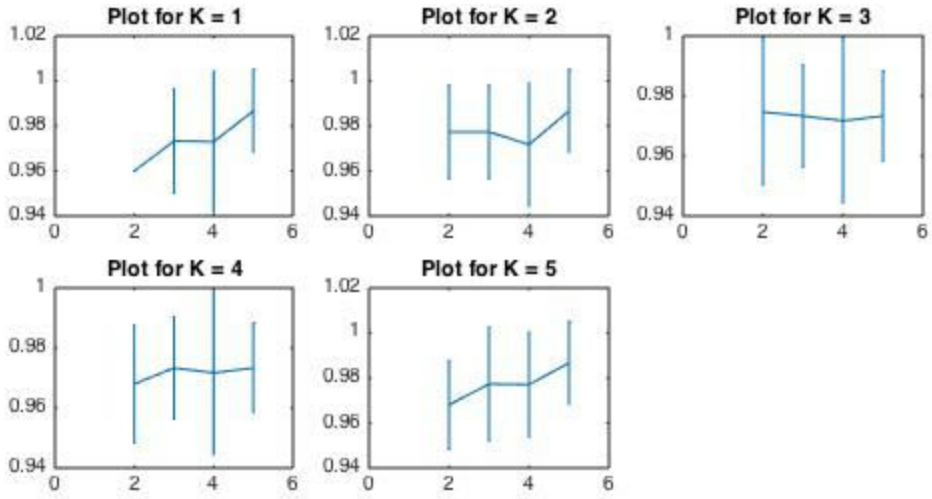
Y label - Mean Accuracy

X label - No. of folds

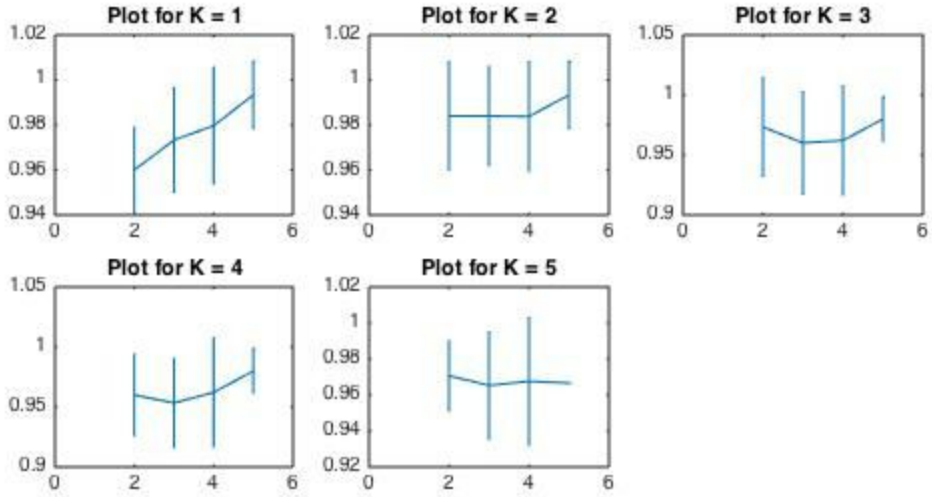
Since data set is randomly divided into training and test data. So two plots for each data set is shown for better assessment.

<u>Iris</u>

Plot 1:

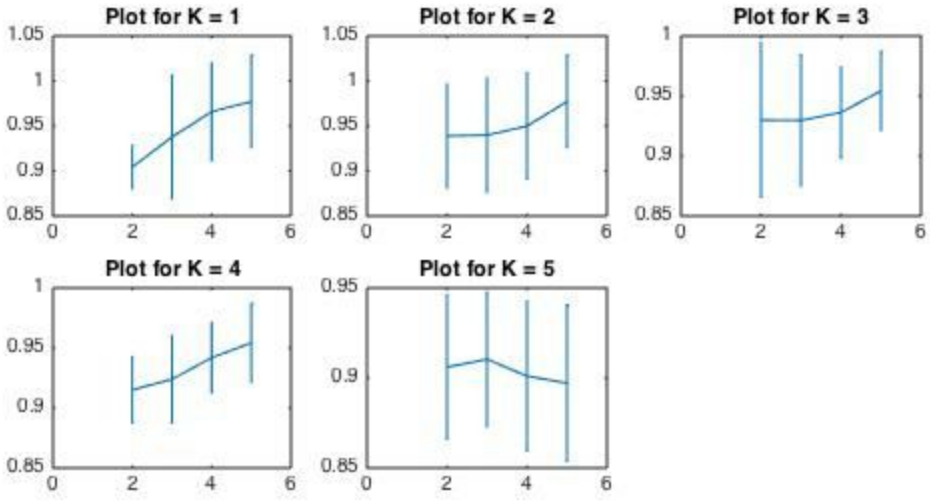


Plot 2:

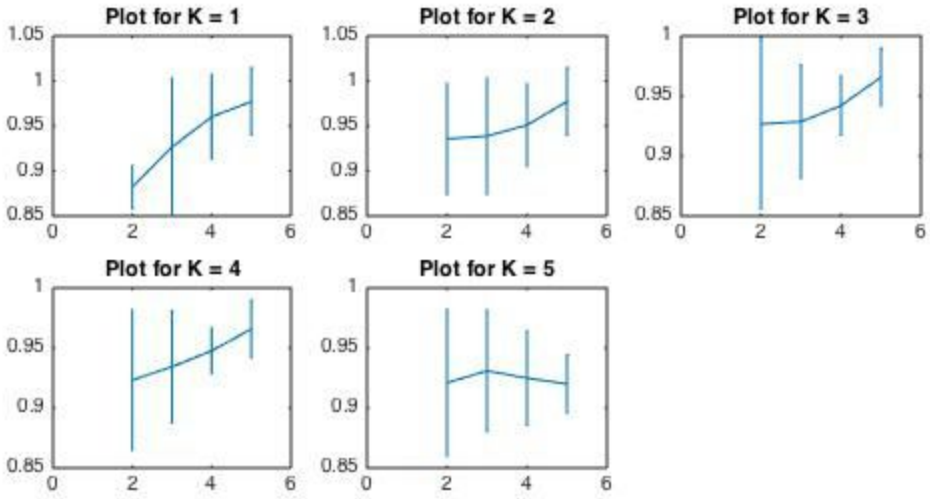


Wine

Plot 1:

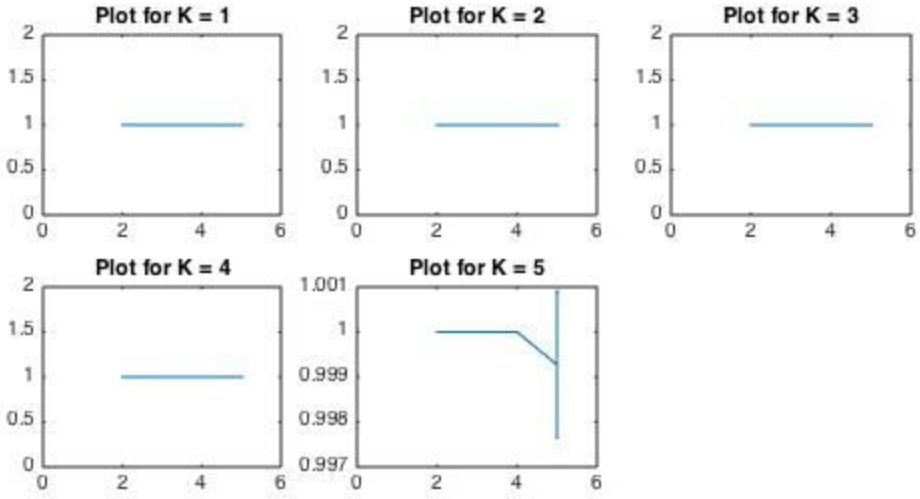


Plot 2:

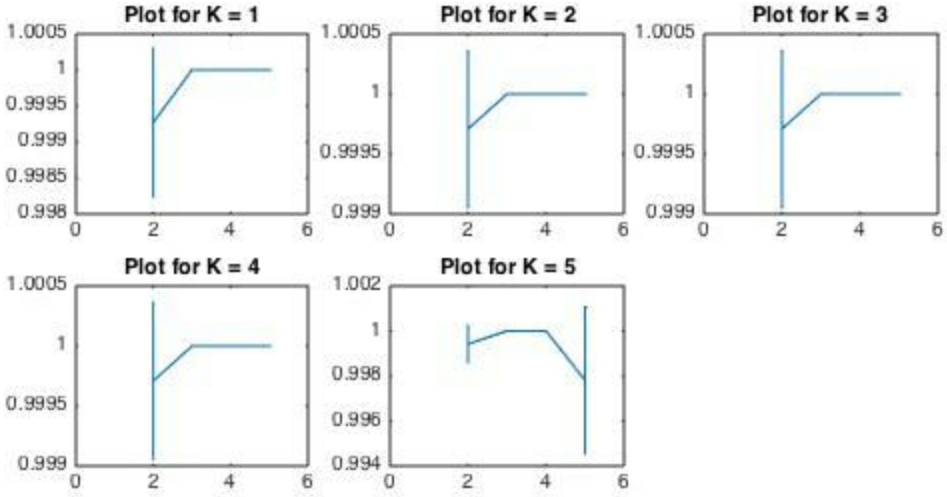


Banknote Authentication

Plot 1:

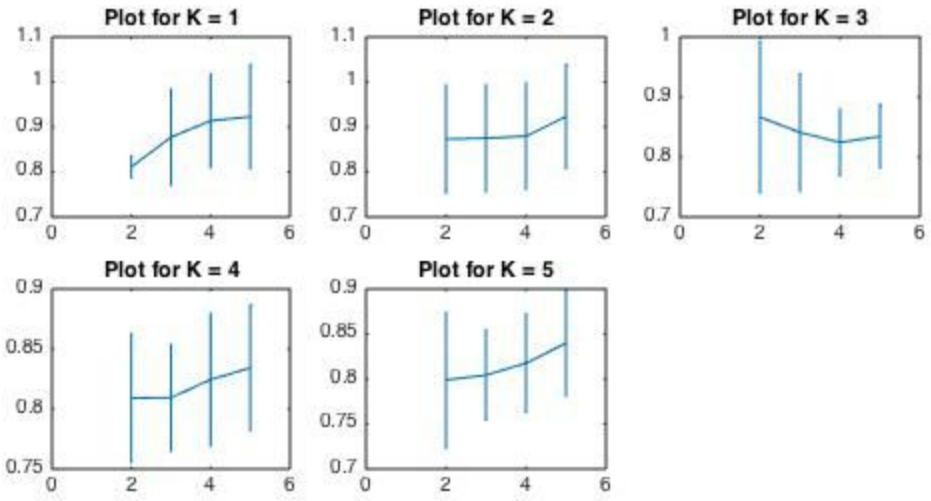


Plot 2:

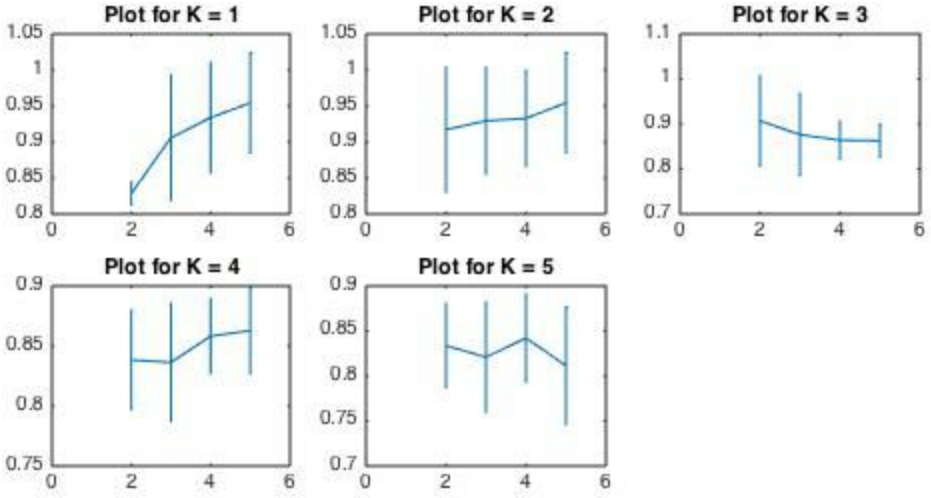


Ionosphere

Plot 1:

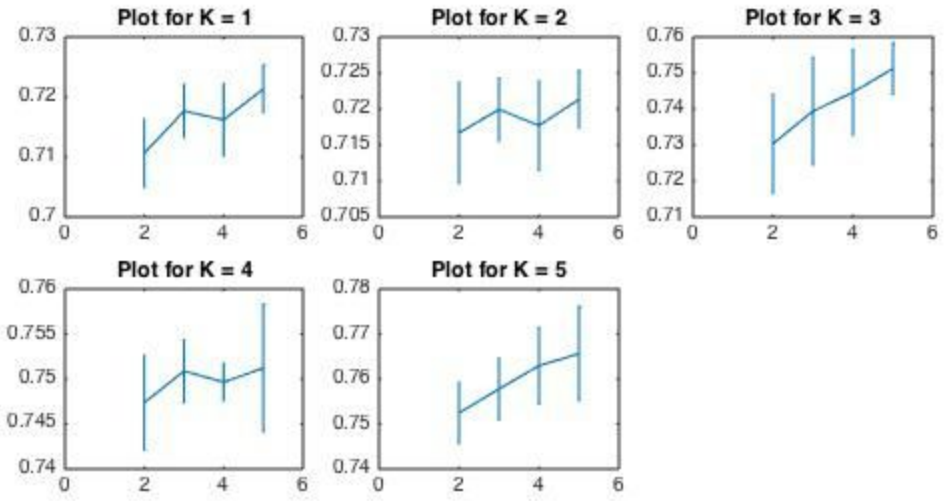


Plot 2:

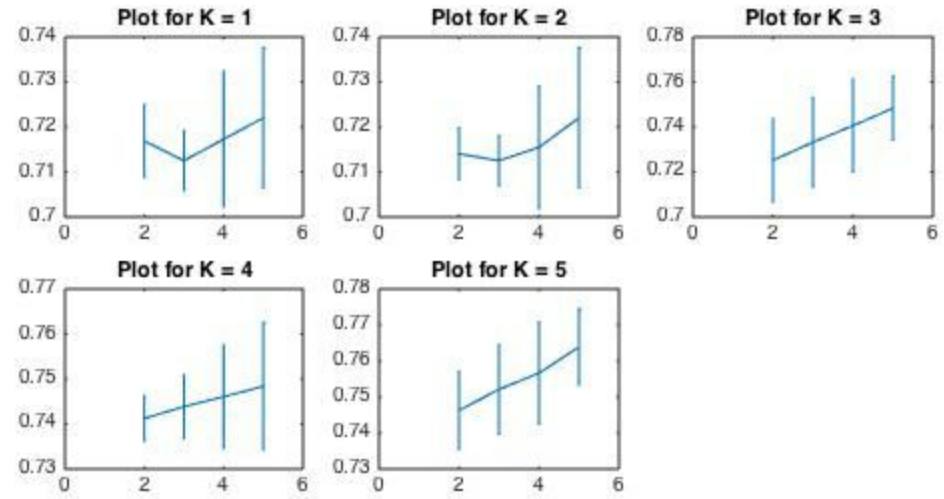


Magic Gamma Telescope

Plot 1:



Plot 2:



Observations:

Dataset	Average Execution Time (in seconds)
Iris	0.504282
Wine	0.576542
Banknote Authentication	4.298854
Ionosphere	1.146183
Magic Gamma Telescope	502.878534

1. Significant improvement in accuracy is observed when normalization with respect to each feature is done.
2. Time of execution is mainly depends upon the number of data samples. E.g. we can observe from above table that there is hardly any time difference between Wine and Ionosphere datasets. Wine has 13 features whereas Ionosphere has 34 features.
3. Generally, accuracy increases with increase in number of folds and value of K.
4. Matlab works pretty fast with matrices. Using loops, execution becomes very slow. Therefore, I tried to do maximum things with matrix manipulation.
   E.g. giving unique number to classes.

References:

1. http://archive.ics.uci.edu/ml/
2. http://in.mathworks.com/help/matlab/ref/errorbar.html
3. https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

Matlab Code

Only one sample code for dataset 'Ionosphere' is included in this report.
Rest of the codes are there in zip file.
For each dataset, I have two files:

1. *Main index file 'main.m'*
2. *kNN classifier function 'knnclassifier.m'*

main.m

```
%Main index file
clear;
close all;

%Calculation of time of execution
tic

filename = 'ionosphere.data.txt';
A = dataset('File', filename, 'Delimiter', ',', 'ReadVarNames', false);
l = size(A, 2);

%Assigning classes a unique number for conversion from dataset to matrix
%Even if classes are more in number with randomly arranged datapoints, this
method would work

u = unique(A(:, l));
u = dataset2cell(u);
u = u(2 : end, :);
p = [1 : size(u, 1)];
C = containers.Map(u, p);
t = A(:, l);
t = dataset2cell(t);
t = t(2 : end);
A.Var35 = C.values(t);
z = A(:, l);
z = dataset2cell(z);
z = z(2 : end);
z = cell2mat(z);
z = mat2dataset(z);
A(:, l) = z;
A = double(A);
```

```
%Generating random permutation for division into test and train data
nrows = size(A, 1);
randrows = randperm(nrows);

%meanaccuracy contains the accuracy data
%rows in meanaccuracy denote fold
%and column denote the corresponding value of K

meanaccuracy = zeros(4, 5);
X = A;
k = [2, 3, 4, 5];
e = zeros(4, 1);
for K = 1 : 5
for fold = 2 : 5
        for chunk = 1 : fold
            chunksize = floor(nrows/fold);
            x = (chunk - 1) * chunksize + 1;
            y = chunk * chunksize;
            testdata = X(randrows(x:y), :);
            if chunk == 1
                traindata = X(randrows(y + 1:end), :);
            elseif chunk == fold
                traindata = X(randrows(1 : x-1), :);
            else
                traindata = X(randrows(1, x-1:y+1, end), :);
            end
            currentacc = knnclassifier(traindata, testdata, K);
            s(chunk) = currentacc;
        end
        meanaccuracy(fold - 1, K) = mean(s);
         out(fold - 1) = mean(s);
         e(fold - 1) = std(s);
    end
    subplot(3,3, K);
    errorbar(k, out, e);
    title(['Plot for K = ', num2str(K)])
end

toc
```

knnclassifier.m

```
%KNN Classifier function

function accur = knnclassifier(traindata, testdata, K)

dist = zeros(size(traindata, 1), 1);

w = size(traindata, 2);
```

```
%Find distance with all training datapoints, sort and poll
for i = 1 : size(testdata)
x = testdata(i,:);
for j = 1 : w - 1
dist(:, 1) =  dist(:, 1) + (traindata(:, j) - x(j)) .^ 2;
end
dist(:, 1) = sqrt(dist(:, 1));

classes = traindata(:, w);
dist(:, 2) = classes;
poll = sortrows(dist, 1);

%For tiebreak in case of even K
if (mod(K, 2) == 1)
expclass(i) = mode(poll(1 : K, 2));
else
    temp = poll(1 : K, 2);
    uniq = unique(temp);
    p = size(uniq);
    bincounts = histc(temp, uniq);
    q = max(bincounts);
    %if number of unique elements = 2 && highest frequency is K/2, then there
is tie
    M = (p == 2) & (q == K/2);
    %Allotted the class which is at closest distance
    expclass(i) = mode(poll(1 : K - M, 2));
end
end

%Error percentage calculation
error = transpose(expclass) - testdata(:, w);
accur = ((size(error, 1) - nnz(error))/size(error, 1));

end
```