The University of British Columbia

CPSC 340

Machine Learning and Data Mining

Homework #2

Student name:

ALI SIAHKOOHI

Student ID:

84264167

Instructor:

Professor Schmidt

Date Submitted:

October 7th, 2016

Question #1

1.1.1

```
function [yhat] = predict(model,Xtest)
% Write me!

X = model.X;
y = model.y;
k = model.k;

[n,d] = size(X);
[t,d] = size(Xtest);

D = X.^2*ones(d, t) + ones(n, d)*(Xtest').^2 - 2*X*Xtest';
D = D.^.5;
yhat = zeros(t, 1);

for i = 1:t

    [tmp, ind] = sort(D(:, i));
    yhat(i) = mode(y(ind(1:k)));
end


end
```

1.1.2

The output is as below:

Error is defined as number of entries that are different between the the given clustering (y) and the predicted clustering ($\hat{y}$) divided by number of data points.

```
(Question 1.1.2) Training error for (k = 1) is: 0
(Question 1.1.2) Testing error for (k = 1) is: 0.0645
(Question 1.1.2) Training error for (k = 3) is: 0.0275
(Question 1.1.2) Testing error for (k = 3) is: 0.066
(Question 1.1.2) Training error for (k = 10) is: 0.0725
(Question 1.1.2) Testing error for (k = 10) is: 0.097
```

1.1.3

The output odd classifier2Dplot is shown in figure (1 – 1)

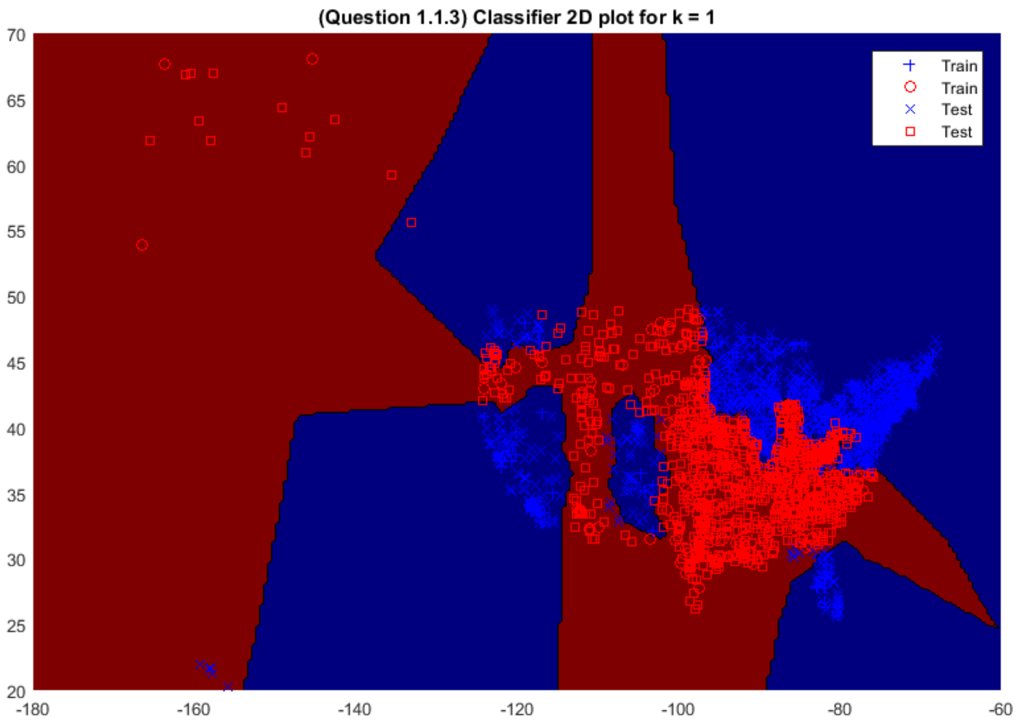

Figure (1 – 1)

1.1.4

k=1 means for n data points, we have n clusters. In other words, each point is its own cluster. In consequence the training error is zero.

1.1.5

We can use cross-validation to use to estimate the test error.

1.2.1

```
function [model] = cnn(X,y,k)

% Implementation of condensed k-nearest neighbour classifer
S(1, 1:2) = X(1, 1:2);
y_s = y(1);
j = 2;
```

```
for i = 2:size(X, 1)

    [model] = knn(S, y_s, k);
    yhat_test = model.predict(model,X(i, :));
    error_test = (y(i) ~= yhat_test);

    if error_test == 1
        S(j, 1:2) = X(i, :);
        y_s(j, 1) = y(i);
        j = j + 1;
    end
end


model.X = S;
model.y = y_s;
model.k = k;
model.c = max(y);
model.predict = @predict;

end

function [yhat] = predict(model,Xtest)
% Write me!

X = model.X;
y = model.y;
k = model.k;

[n,d] = size(X);
[t,d] = size(Xtest);

D = X.^2*ones(d, t) + ones(n, d)*(Xtest').^2 - 2*X*Xtest';
D = D.^.5;
yhat = zeros(t, 1);

for i = 1:t

    [tmp, ind] = sort(D(:, i));
    yhat(i) = mode(y(ind(1:k)));
end


end
```

1.2.2

```
(Question 1.2.2) Number of variables in the subset (k = 1) is: 457
(Question 1.2.2) Training error for (k = 1) is: 0.00753308
(Question 1.2.2) Testing error for (k = 1) is: 0.0175772
```

1.2.3

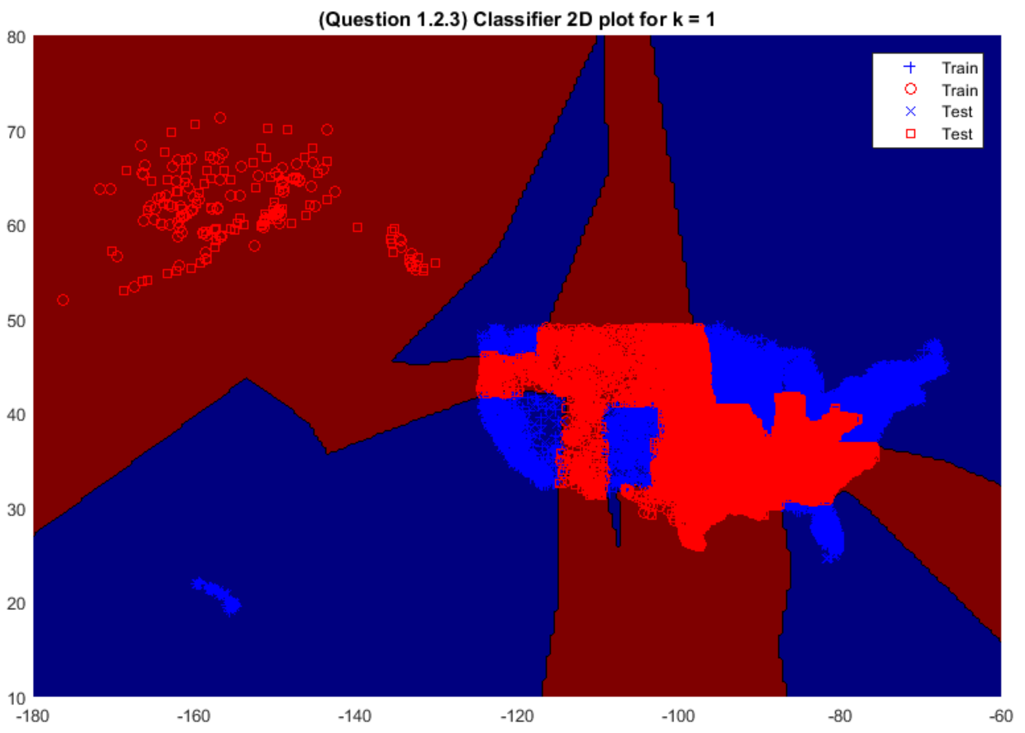

Figure (1 – 2)

1.2.4

Since now instead of 14735 data points for k=1in knn, we have only 457 points in the subset. Therefore there is a possibility that other (14735 – 457) can be predicted wrong. And the reason is we have went through data set just once, so there is no guarantee that the data points which were predicted correctly during the training phase, still get predicted right because the data points added to subset after that particular point could have changed the decision of the learning system towards it.

1.2.5

Since the algorithm has O(nd) computational complexity for predicting one example in knn, in ccn it is O(tkn) .

1.2.5

It’s because the training data matrix X in this data set is not randomly distributed. By looking at vector y it can be seen that there are a lot of data points in order. This not random order of data, affects the training phase. By permuting the rows of X and y correspondingly, I got the following error.

Question #2

2.1.1

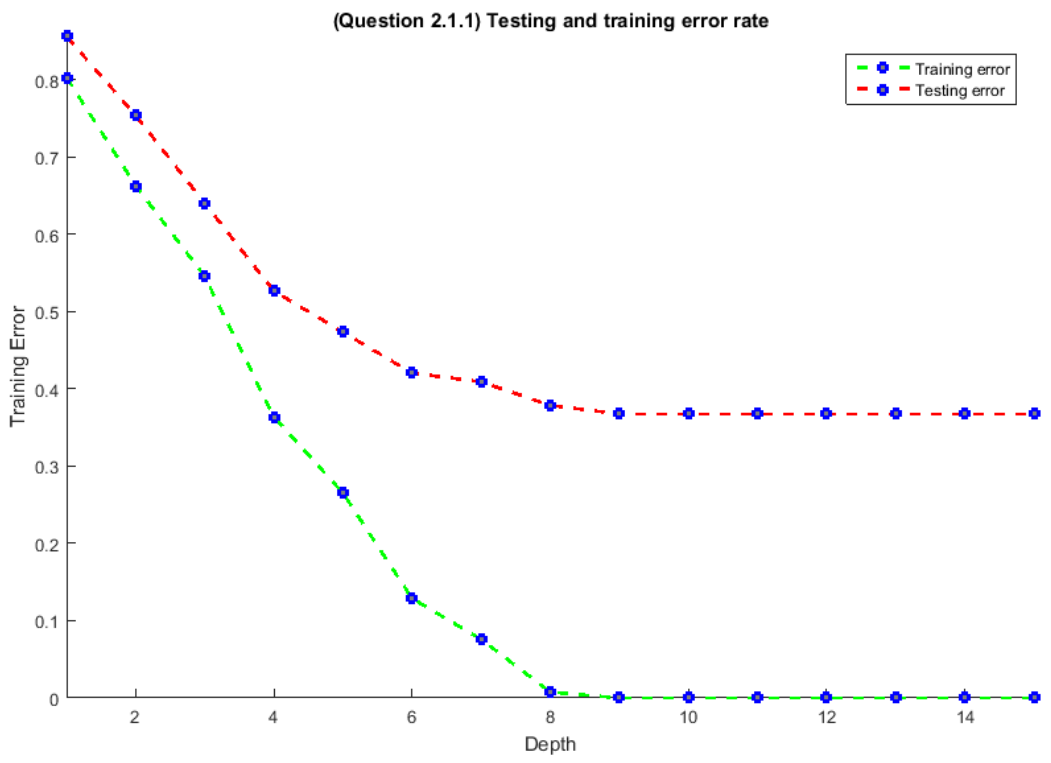

Figure (2 – 1)

2.1.2

Because it reaches a state which there is no splitting scheme exists which can reduce the error and regarding to the following line in decisionTree.m

```
if maxDepth <= 1 || isempty(splitModel.splitVariable)
    % If we have reached the maximum depth or the decision stump does
    % nothing, use the decision stump
```

2.1.3

```
function [model] = randomStump(X,y)
% [model] = randomStump(X,y)
%
% Fits a decision stump that splits on a single variable.
% Compute number of training examples and number of features
[n,d] = size(X);

% Choosing the random features
```

```
rand_generator = round((d - 1)*rand(1, floor(sqrt(d))) + 1);

% Computer number of class lables
k = max(y);

% Address the trivial case where we do not split
count = accumarray(y,ones(size(y)),[k 1]); % Counts the number of occurrences
of each class
[maxCount,maxLabel] = max(count);

% Compute total entropy (needed for information gain)
p = count/sum(count); % Convert counts to probabilities
entropyTotal = -sum(p.*log0(p));

maxGain = 0;
splitVariable = [];
splitThreshold = [];
splitLabel0 = maxLabel;
splitLabel1 = [];

% Loop over features looking for the best split
if any(y ~= y(1))
    % for randomly choosen features
    for j = rand_generator
        thresholds = sort(unique(X(:,j)));

        for t = thresholds'

            % Count number of class labels where the feature is greater than
threshold
            yVals = y(X(:,j) > t);
            count1 = accumarray(yVals,ones(size(yVals)),[k 1]);
            count0 = count-count1;

            % Compute infogain
            p1 = count1/sum(count1);
            p0 = count0/sum(count0);
            H1 = -sum(p1.*log0(p1));
            H0 = -sum(p0.*log0(p0));
            prob1 = sum(X(:,j) > t)/n;
            prob0 = 1-prob1;
            infoGain = entropyTotal - prob1*H1 - prob0*H0;

            % Compare to minimum error so far
            if infoGain > maxGain
                % This is the lowest error, store this value
                maxGain = infoGain;
                splitVariable = j;
                splitThreshold = t;

                % Compute majority class
                [maxCount,splitLabel1] = max(count1);
                [maxCount,splitLabel0] = max(count0);
            end
        end
    end
end
```

```
model.splitVariable = splitVariable;
model.splitThreshold = splitThreshold;
model.label1 = splitLabel1;
model.label0 = splitLabel0;
model.predict = @predict;
end
```

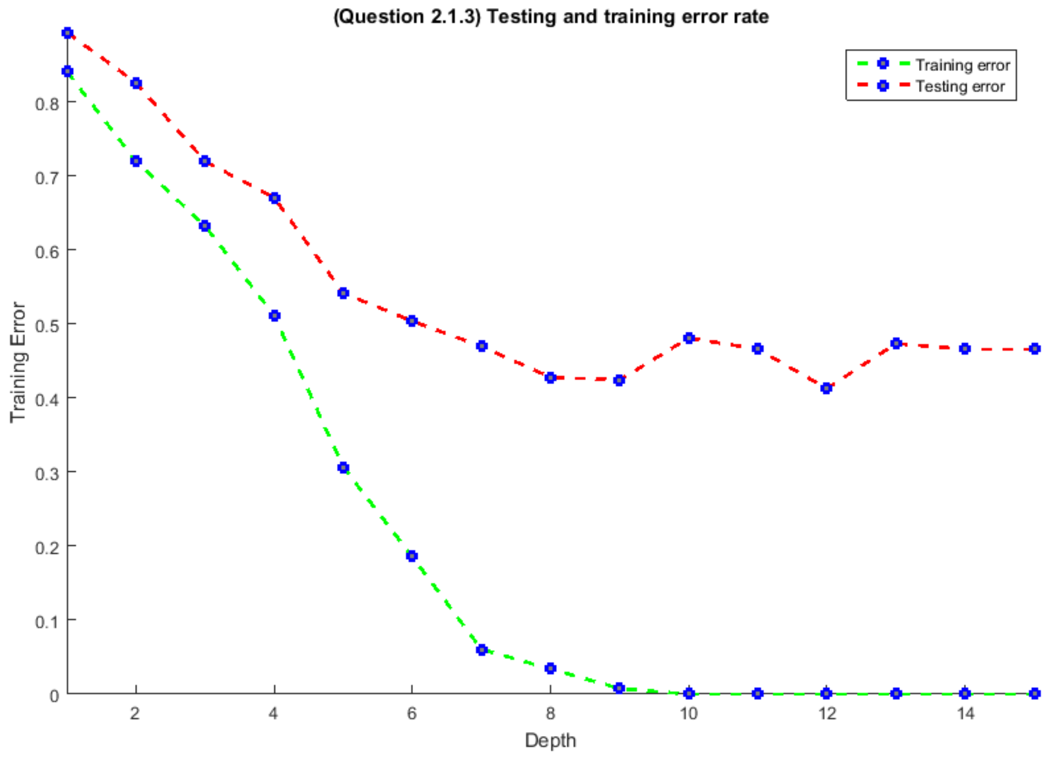

Figure (2 – 2)

2.1.4

The figure is in the next page.

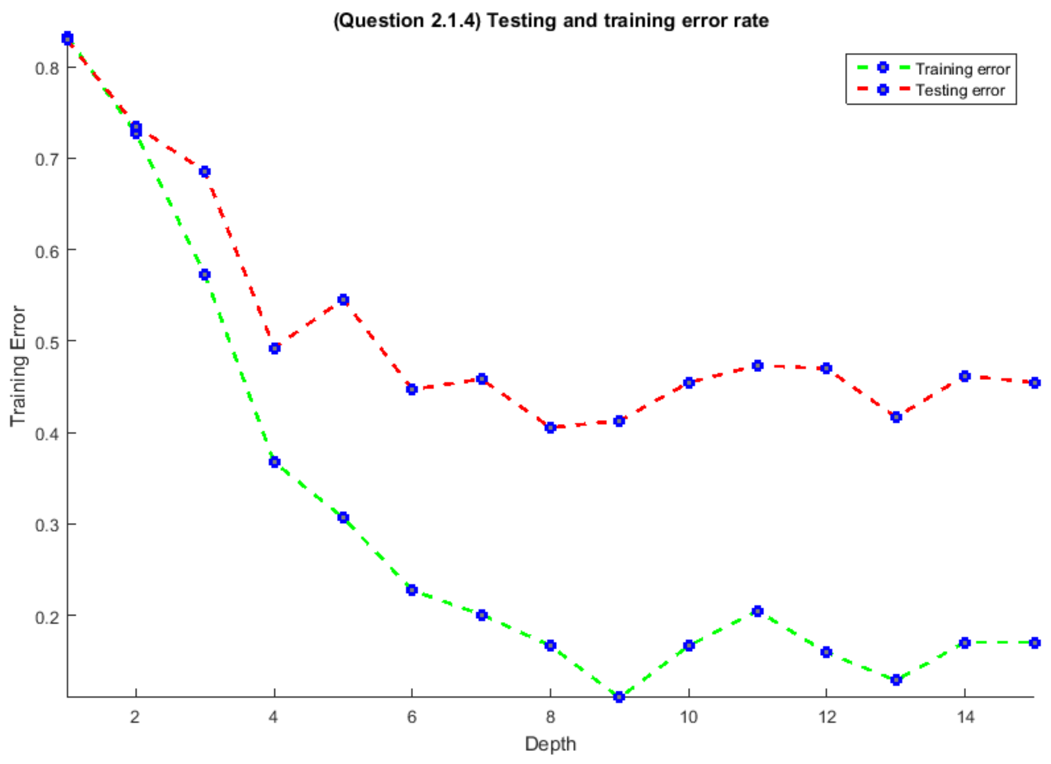

Figure (2 – 3)

2.2.1

```
(Question 2.2.1) Testing error is: 0.367424
```

2.2.2

```
(Question 2.2.2) Testing error is: 0.295455
```

2.2.3

```
(Question 2.2.3) Testing error is: 0.189394
```

2.2.4

```
(Question 2.2.4) Testing error is: 0.17803
```

```matlab
function [model] = decisionForest_randomTree_rand(X,y,depth,nBootstraps)

% Fit model to each boostrap sample of data
for m = 1:nBootstraps
    model.subModel{m} = randomTree_rand(X,y,depth);
end

model.predict = @predict;

end

function [y] = predict(model,X)

% Predict using each model
for m = 1:length(model.subModel)
    y(:,m) = model.subModel{m}.predict(model.subModel{m},X);
end

% Take the most common label
y = mode(y,2);
end
```

```matlab
function [model] = randomTree_rand(X,y,maxDepth)
% [model] = randomTree(X,y)
%
% Fits a decision tree that splits on a sequence of single variables.

[n,d] = size(X);
rand_generator = round((n - 1)*rand(1, n) + 1);
X = X(rand_generator, :);
y = y(rand_generator);
% Learn a random decision stump
splitModel = randomStump(X,y);

if maxDepth <= 1 || isempty(splitModel.splitVariable)
    % If we have reached the maximum depth or the decision stump does
    % nothing, use the decision stump
    model = splitModel;
else
    % Fit a decision tree to each split, decreasing maximum depth by 1
    d = splitModel.splitVariable;
    t = splitModel.splitThreshold;
    model.splitModel = splitModel;

    % Find indices of examples in each split
    splitIndex1 = find(X(:,d) > t);
    splitIndex0 = find(X(:,d) <= t);

    % Fit decision tree to each split
    model.subModel1 = randomTree(X(splitIndex1,:),y(splitIndex1),maxDepth-1);
    model.subModel0 = randomTree(X(splitIndex0,:),y(splitIndex0),maxDepth-1);
    % Assign prediction function
    model.predict = @predict;
end
end

function [y] = predict(model,X)
[t,d] = size(X);
y = zeros(t,1);

splitModel = model.splitModel;
if isempty(splitModel.splitVariable)
    % If no further splitting, return the majority label
    y = splitModel.label1*ones(t,1);
else
    % Recurse on both sub-models
    d = splitModel.splitVariable;
    t = splitModel.splitThreshold;

    splitIndex1 = find(X(:,d) > t);
    splitIndex0 = find(X(:,d) <= t);

    subModel1 = model.subModel1;
    subModel0 = model.subModel0;

    y(splitIndex1) = subModel1.predict(subModel1,X(splitIndex1,:));
    y(splitIndex0) = subModel0.predict(subModel0,X(splitIndex0,:));
end
end
```

```matlab
function [model] = randomStump(X,y)
% [model] = randomStump(X,y)
%
% Fits a decision stump that splits on a single variable.
% Compute number of training examples and number of features
[n,d] = size(X);

% Choosing the random features
rand_generator = round((d - 1)*rand(1, floor(sqrt(d))) + 1);

% Computer number of class lables
k = max(y);

% Address the trivial case where we do not split
count = accumarray(y,ones(size(y)),[k 1]); % Counts the number of occurrences
of each class
[maxCount,maxLabel] = max(count);

% Compute total entropy (needed for information gain)
p = count/sum(count); % Convert counts to probabilities
entropyTotal = -sum(p.*log0(p));

maxGain = 0;
```

```
splitVariable = [];
splitThreshold = [];
splitLabel0 = maxLabel;
splitLabel1 = [];

% Loop over features looking for the best split
if any(y ~= y(1))
    % for randomly choosen features
    for j = rand_generator
        thresholds = sort(unique(X(:,j)));

        for t = thresholds'

            % Count number of class labels where the feature is greater than
threshold
            yVals = y(X(:,j) > t);
            count1 = accumarray(yVals,ones(size(yVals)),[k 1]);
            count0 = count-count1;

            % Compute infogain
            p1 = count1/sum(count1);
            p0 = count0/sum(count0);
            H1 = -sum(p1.*log0(p1));
            H0 = -sum(p0.*log0(p0));
            prob1 = sum(X(:,j) > t)/n;
            prob0 = 1-prob1;
            infoGain = entropyTotal - prob1*H1 - prob0*H0;

            % Compare to minimum error so far
            if infoGain > maxGain
                % This is the lowest error, store this value
                maxGain = infoGain;
                splitVariable = j;
                splitThreshold = t;

                % Compute majority class
                [maxCount,splitLabel1] = max(count1);
                [maxCount,splitLabel0] = max(count0);
            end
        end
    end
end
model.splitVariable = splitVariable;
model.splitThreshold = splitThreshold;
model.label1 = splitLabel1;
model.label0 = splitLabel0;
model.predict = @predict;
end

function [y] = predict(model,X)
[t,d] = size(X);

if isempty(model.splitVariable)
    y = model.label0*ones(t,1);
else
    y = zeros(t,1);
    for n = 1:t
```

```
        if X(n,model.splitVariable) > model.splitThreshold
            y(n,1) = model.label1;
        else
            y(n,1) = model.label0;
        end
    end
end
end
```

2.2.5

It increases the randomness of data which each tree in the forest has been learned through. In Overall the consequence is that at the same time we can keep the training error and test error both low.

3.1.1

```
function [error] = error(model,X)

[n,d] = size(X);
W = model.W;
k = size(W, 1);
X2 = X.^2*ones(d,k);

distances = X2 + ones(n,d)*(W').^2 - 2*X*W';

error = 0;

for i = 1:size(X, 1)
    error = error + (min(distances(i, :), [], 2));
end

end
```

3.1.2

The figure is in the next page.

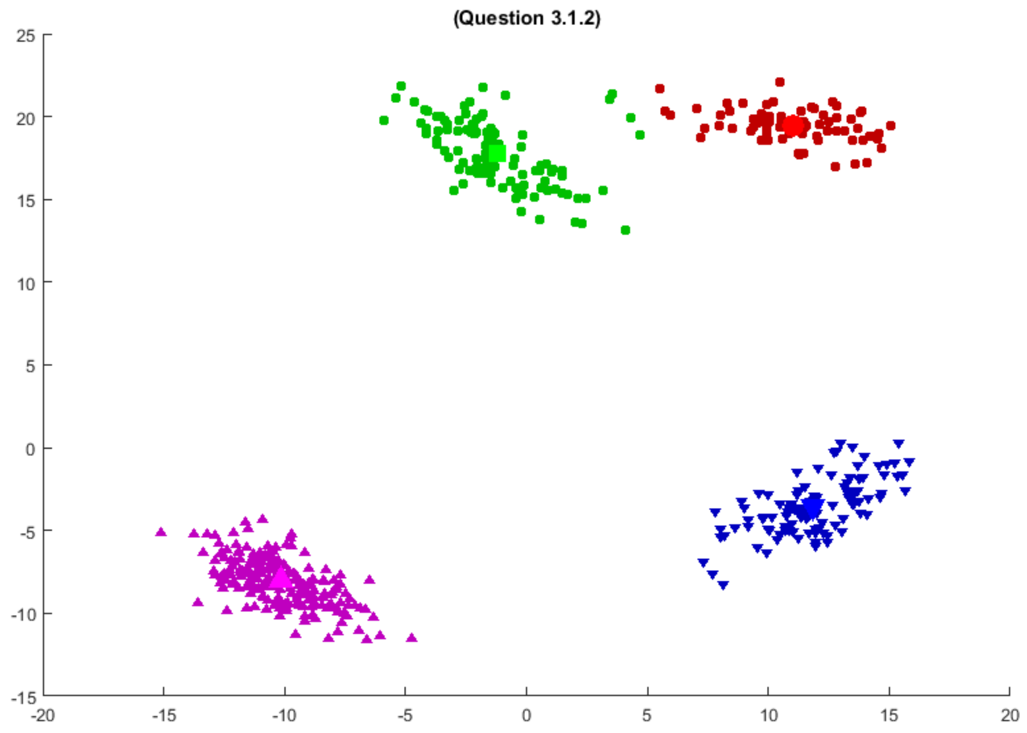

Figure (3 – 1)

3.2.1

It is trivial that we need to minimize the objective function in order to find a good k! but the point is when k is chosen to be one, it seems like we are going to have the most errors. Because adding any additional point makes the objective function smaller. On the other hand having k getting large and to infinity, we will have the objective function zore. In fact if the data point are really clustered in h clusters, even choosing k=h is not going to give us any onformation with this formulation.

3.2.2

In addition to points in 3.21, In order to find k using the objective function in the question we need to try big amount of different k’s and in addition to that, for each k we need to run k-means several time in order to be sure that we are not stuck in a local minimum. In other words, for each k, finding the means which are the best is non-trivial.

3.2.3

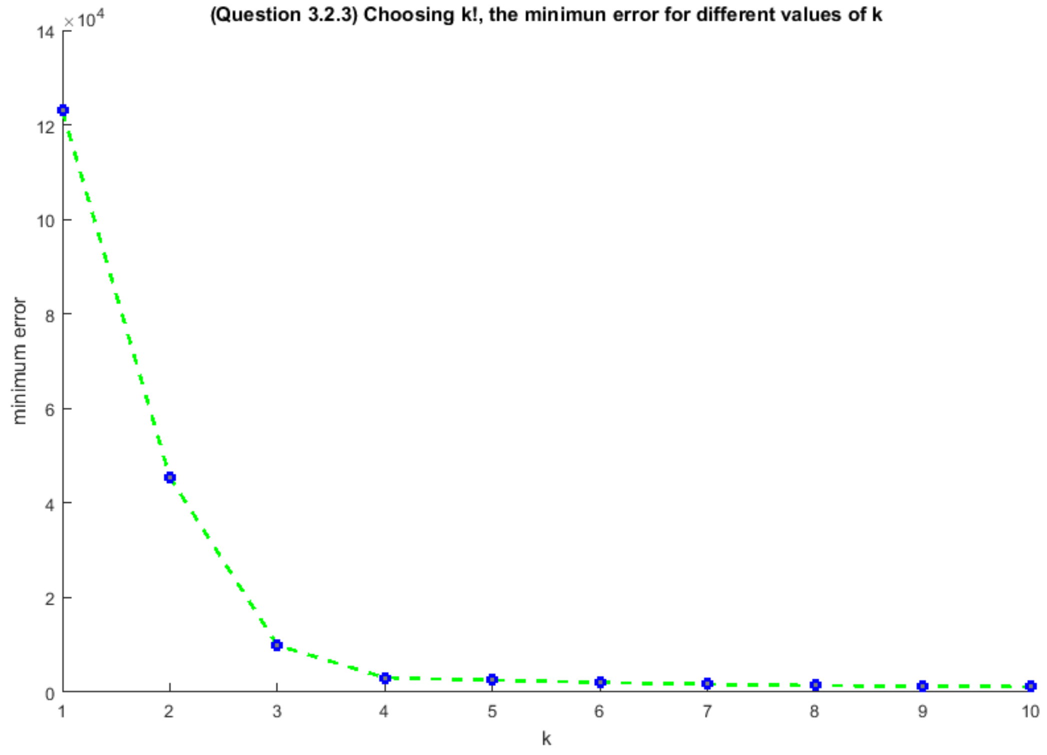

Figure (3 – 2)

3.2.4

Both 3 and 4 are quite similar in the biggest change in slop, but since after 4 the rate of decreasing error gets smaller, and computational cost more expensive, k=4 could be a reasonable choice.

3.3.1

The plot is in the next page.

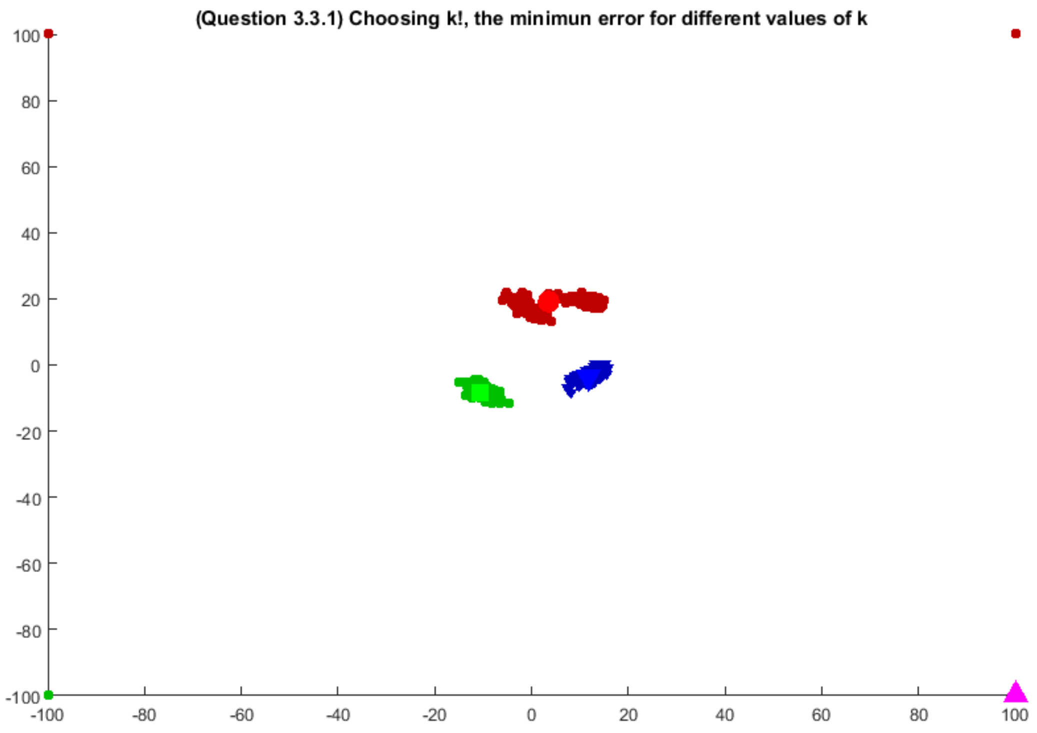

Figure (3 – 3)

3.3.2

According to the plot below, which is produced as like the figure above for this data set, k=8 can be suggested.

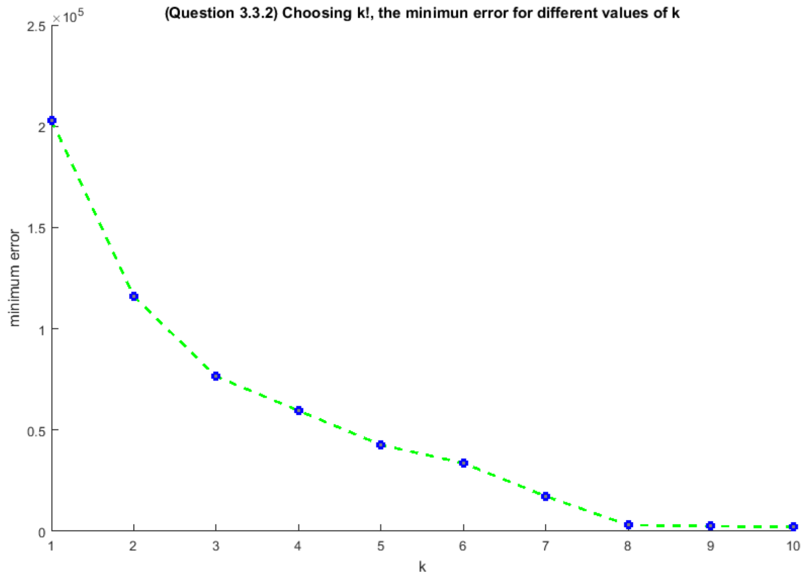

Figure (3 – 4)

3.3.3

```matlab
function [model] = clusterKmedians(X,k,doPlot)
% [model] = clusterKmedians(X,k,doPlot)
%
% K-medians clustering

[n,d] = size(X);
y = ones(n,1);

% Choose random points to initialize medians
W = zeros(k,d);
for k = 1:k
    i = ceil(rand*n);
    W(k,:) = X(i,:);
end

X2 = X.^2*ones(d,k);
while 1
    y_old = y;

    % Draw visualization
    if doPlot && d == 2
        clustering2Dplot(X,y,W)
    end

    % Compute (absolute) Euclidean distance between each data point and
    % each median
    distances = zeros(size(X, 1), size(W, 1));
    for i = 1:size(distances, 1)
        for j = 1:size(distances, 2)
            distances(i, j) = norm(X(i, :) - W(j, :), 1);
        end
    end

    % Assign each data point to closest mmedian
    [~,y] = min(distances,[],2);

    % Draw visualization
    if doPlot && d == 2
        clustering2Dplot(X,y,W)
    end

    % Compute median of each cluster
    for k = 1:k
        W(k,:) = median(X(y==k,:),1);
    end

    changes = sum(y ~= y_old);
    fprintf('Running K-medians, difference = %f\n',changes);

    % Stop if no point changed cluster
    if changes == 0
        break;
    end
end
```

```matlab
model.W = W;
model.y = y;
model.predict = @predict;
model.error = @error;
end

function [y] = predict(model,X)
[t,d] = size(X);
W = model.W;
k = size(W,1);

% Compute Euclidean distance between each data point and each median
distances = zeros(size(X, 1), size(W, 1));
for i = 1:size(distances, 1)
    for j = 1:size(distances, 2)
        distances(i, j) = norm(X(i, :) - W(j, :), 1);
    end
end


% Assign each data point to closest mean
[~,y] = min(distances,[],2);
end

function [error] = error(model,X)

[n,d] = size(X);
W = model.W;
k = size(W, 1);
X2 = X.^2*ones(d,k);

distances = zeros(size(X, 1), size(W, 1));
for i = 1:size(distances, 1)
    for j = 1:size(distances, 2)
        distances(i, j) = norm(X(i, :) - W(j, :), 1);
    end
end

error = 0;

for i = 1:500
    error = error + (min(distances(i, :), [], 2));
end

end
```

3.3.4

k=4 is suggested now!

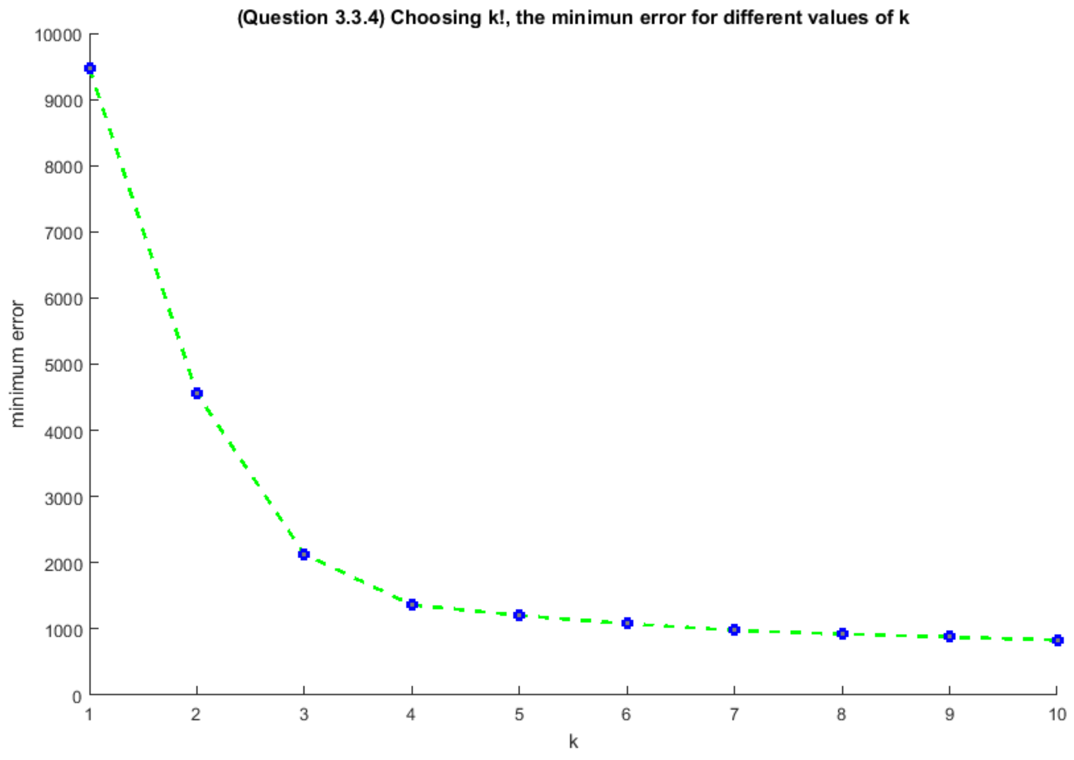

Figure (3 – 5)

Question #4:

4.1.1

```
function [Iquant] = quantizeImage(I,b)

k = 2^b;
% X = zeros(size(I, 1)*size(I, 2), 3);

X = reshape(I, size(I, 1)*size(I, 2), 3);

doPlot = 0;
model = clusterKmeans(X,k,doPlot);
y = model.predict(model,X);

Inew = zeros(length(y), 3);
for i = 1:length(y);
```

```
    Inew(i, :) = model.W(y(i));
end

Iquant = round(reshape(Inew, size(I, 1), size(I, 2), size(I, 3)));

end
```

4.1.2

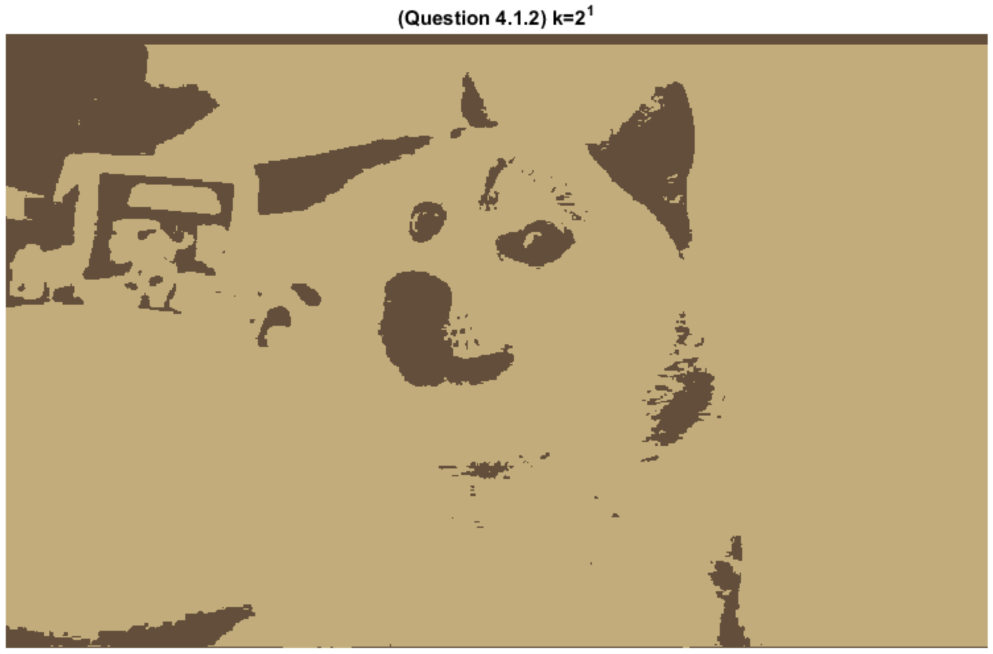

Figure (4 – 1) b=1, k=2

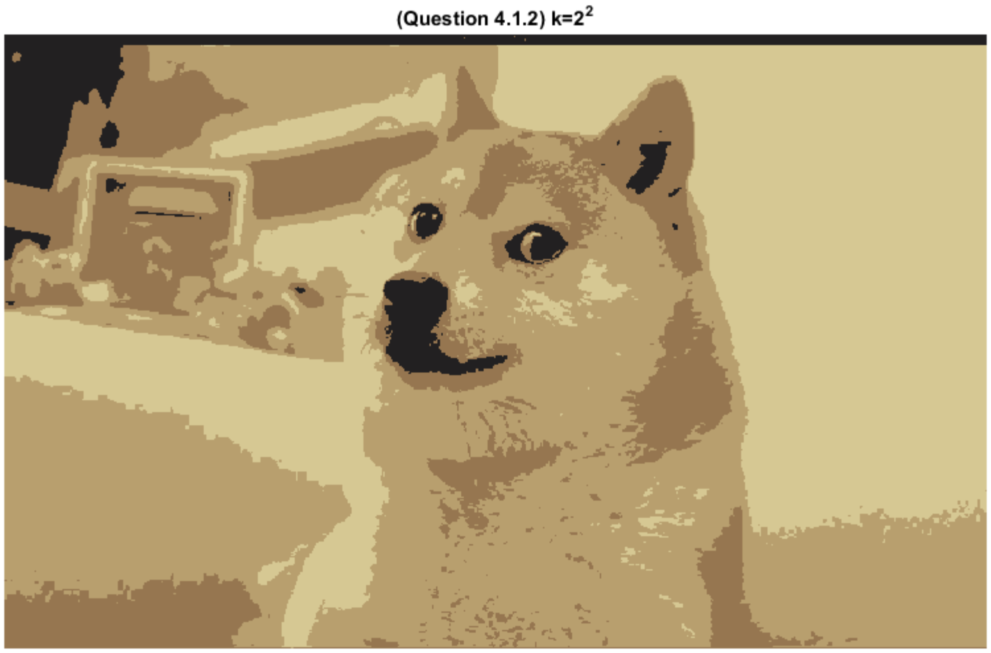

Figure (4 – 2) b=2, k=4

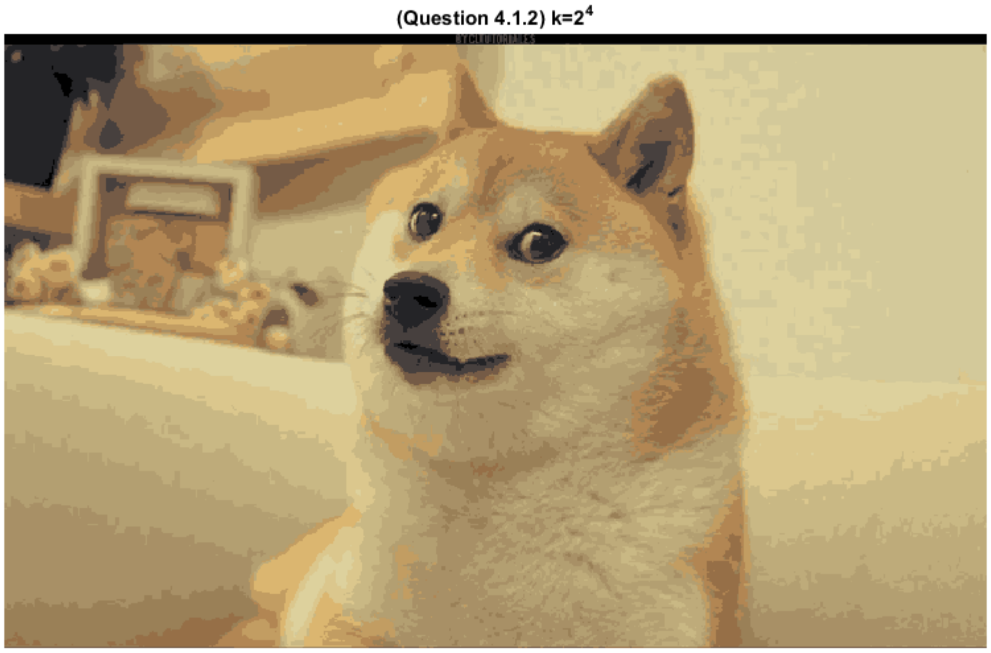

Figure (4 – 3) b=4, k=16

Figure (4 – 4) b=6, k=64

4.2.1
```
radius = 1;
minPts = 6;
```

4.2.2
```
radius = 40;
minPts = 6;
```

4.2.3
```
radius = 200;
minPts = 6;
```

4.2.4
```
radius = 500;
minPts = 6;
```

4.3.1
This indicates that the cluster has no data points assigned to it. The reason could be First, parameter k is not chosen correctly. Second, the algorithm gets stuck in a local minimum which never comes out and this cluster remains empty. The reason that this always does not happen relates to the initial means chosen.

4.3.2

Here is my choice for radius:

```
radius = 13;
```

and the output is:

```
Cluster 1: antelope horse moose ox sheep buffalo zebra deer pig cow
Cluster  2:  dalmatian  german+shepherd  siamese+cat  fox  wolf  chihuahua
rat weasel bobcat collie
Cluster 3: hippopotamus elephant rhinoceros
Cluster 4: spider+monkey gorilla chimpanzee
Cluster 5: hamster rabbit mouse
```

```
%% Animals with attributes data
load animals.mat

%% DBSCAN clustering
radius = 13;
minPts = 3;
doPlot = 0;

model = clusterDBcluster(X,radius,minPts,doPlot);
k = max(model.y) - min(model.y)
for c = 1:k
    fprintf('Cluster %d: ',c);
    fprintf('%s ',animals{model.y==c});
    fprintf('\n');
end


% radius = 500;
% minPts = 6;
```