CAB420 Machine Learning

Queensland University of Technology

Assignment 2 Report

Group 22

Nathan Armishaw – n9157191

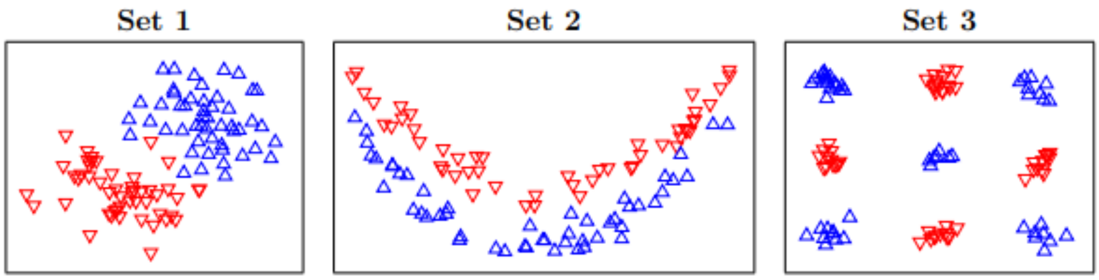
Quang Huy Tran - n10069275

Due: 23/05/20

Table of Contents

Part A: SVMs and Bayes Classifiers [45 Marks]

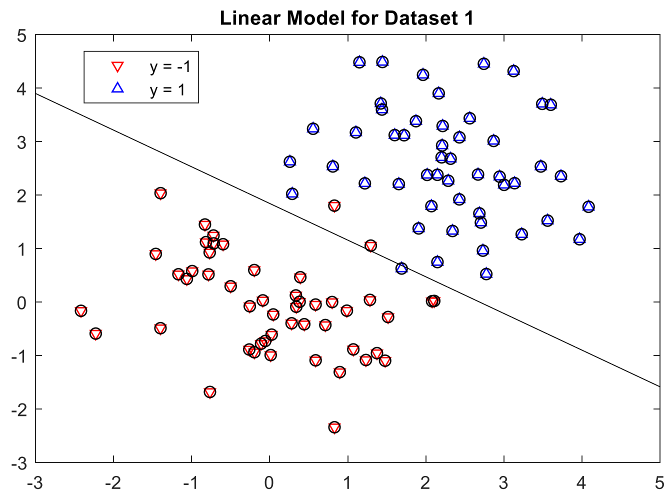
Support Vector Machines [30 Marks]

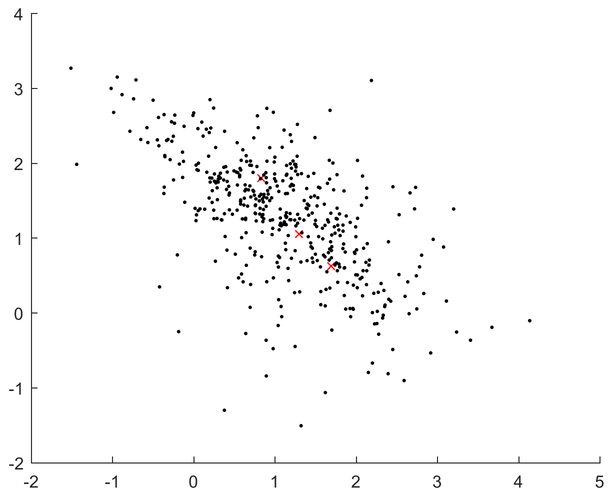


```
%% Section A: Support Vector Machines
% Load the datasets
svm_data = load('data_ps3_2.mat');

% Training Data
set1_train = svm_data.set1_train; % Set 1
set2_train = svm_data.set2_train; % Set 2
set3_train = svm_data.set3_train; % Set 3
set4_train = svm_data.set4_train; % Set 4

% Testing Data
set1_test = svm_data.set1_test; % Set 1
set2_test = svm_data.set2_test; % Set 2
set3_test = svm_data.set3_test; % Set 3
set4_test = svm_data.set4_test; % Set 4

%Set constant values
C = 1000;
polyOrder = 2;
SD1 = 1;
SD2 = 1.5;
```

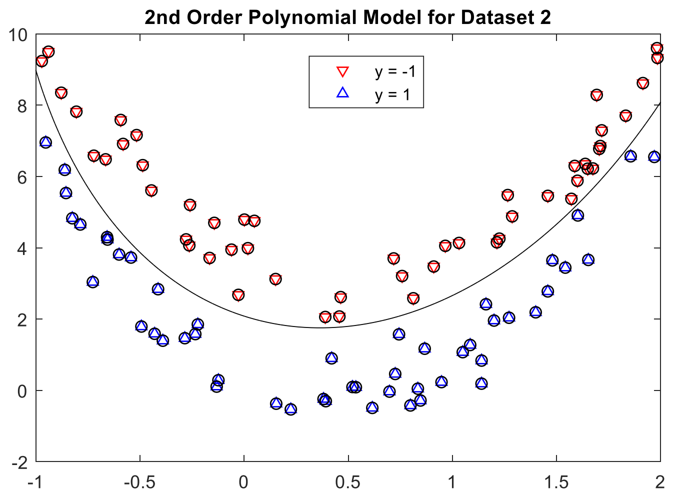
```
%% Question 1:
% For the first three datasets, consider the linear, second
% order polynomial, Gaussian of standard deviation 1 kernels

%% Part 1:
% Train first 3 datasets, plot the decision boundary and print the test
% errors  using svm_test

% Plot the decision boundary and test errors with the linear model
svm_test(@Klinear,[],C,set1_train,set1_test);
svm_test(@Klinear,[],C,set2_train,set2_test);
svm_test(@Klinear,[],C,set3_train,set3_test);

% Plot the decision boundary and test errors with the 2nd order
% polynomial model
svm_test(@Kpoly,polyOrder,C,set1_train,set1_test);
svm_test(@Kpoly,polyOrder,C,set2_train,set2_test);
svm_test(@Kpoly,polyOrder,C,set3_train,set3_test);

% Plot the decision boundary and test errors with the gaussian model of
% standard deviation 1
svm_test(@Kgaussian,SD1,C,set1_train,set1_test);
svm_test(@Kgaussian,SD1,C,set2_train,set2_test);
svm_test(@Kgaussian,SD1,C,set3_train,set3_test);
```

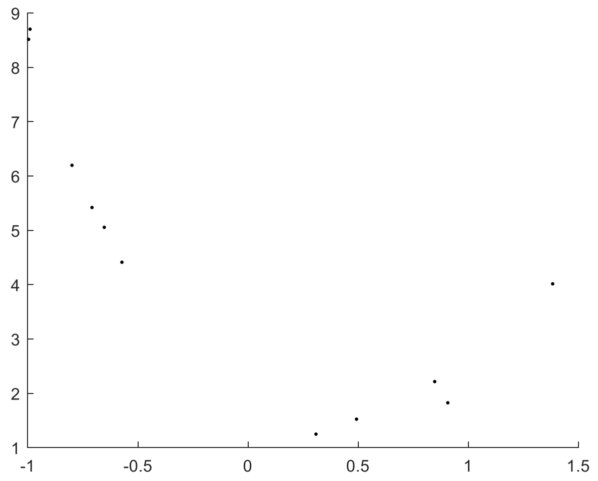
From these results the best model for datasets 1,2 and 3 were chosen. These were Linear for set 1 because the data was linearly clustered. 2nd order polynomial for set 2 because the 2 classes were separable by a curved boundary. 1 SD gaussian for set 3 because the data was separately clustered. The plots of the decision boundary and test errors for each of these three is shown below. Note that the plots below show the decision boundary on the training data, then the testing result on the test data.



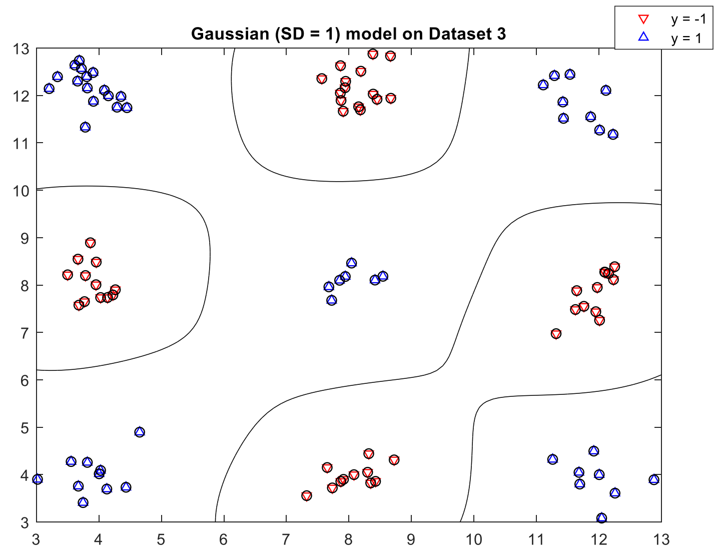


```
WARNING: 3 training examples were misclassified!!!
TEST RESULTS: 0.0446 of test examples were misclassified.
```





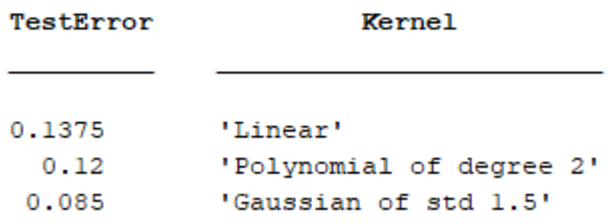
```
TEST RESULTS: 0.011 of test examples were misclassified.
```



```
TEST RESULTS: 0 of test examples were misclassified.
```

```matlab
%% Part 2

% Train 4th dataset with a linear, polynomial of degree 2 and Gaussian of
% standard deviation 1.5 kernels

% Preallocate TestError and set Kernel
TestError = zeros(3,1);
Kernel = {'Linear';'Polynomial of degree 2';'Gaussian of std 1.5'};

% For the following chunks of code the following process was used: First the
% SVM model was trained on the training data, then the errors between the
% prediction and actual results in the test data was calculated. Then the
% fraction of errors was displayed.

% For linear model
svm_linear4 = svm_train(set4_train,@Klinear,[],C);
y_linear4 = sign(svm_discrim_func(set4_test.X,svm_linear4));
errors_linear = find(y_linear4 ~= set4_test.y);
TestError(1) = length(errors_linear)/length(set4_test.y);
fprintf('Linear SVM: %g of 4th test examples  were misclassified.\n',...
    length(errors_linear)/length(set4_test.y));

% For polynomial model
svm_poly4 = svm_train(set4_train,@Kpoly,polyOrder,C);
y_poly4 = sign(svm_discrim_func(set4_test.X,svm_poly4));
errors_poly = find(y_poly4 ~= set4_test.y);
TestError(2) = length(errors_poly)/length(set4_test.y);
fprintf('Polynomial SVM: %g of 4th test examples  were misclassified.\n',...
    length(errors_poly)/length(set4_test.y));

% For gaussian model
svm_gaussian4 = svm_train(set4_train,@Kgaussian,SD2,C);
y_gaussian4 = sign(svm_discrim_func(set4_test.X,svm_gaussian4));
errors_gaussian = find(y_gaussian4 ~= set4_test.y);
TestError(3) =  length(errors_gaussian)/length(set4_test.y);
fprintf('Gaussian SVM: %g of 4th test examples  were misclassified.\n',...
    length(errors_gaussian)/length(set4_test.y));

% The test errors of 4th dataset trained on different kernels is as below:
display(table(TestError,Kernel));
```

This results in the data below. Clearly the gaussian model is the most accurate, followed by the polynomial then the linear.



```
TestError              Kernel
_________    ________________________

0.1375       'Linear'
  0.12       'Polynomial of degree 2'
 0.085       'Gaussian of std 1.5'
```

Bayes Classifiers [15 Marks]

```
%% Section A: Bayes Classifiers
% Code is relevant to parts (A) and (B)
% Set the Training and Test data as well as other constants related to them
Xtr1 = [0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1];
Xtr2 = [0,0,0,0,1,1,1,1,0,0,0,1,1,1,1,1];
Ytr = [0,1,1,1,0,1,1,1,0,0,0,0,0,0,1,1];
Xtest = [0 1; 1 0; 1 1];
YtrLength = length(Ytr);
Ytr0Length = length(Ytr(Ytr==0));
Ytr1Length = length(Ytr(Ytr==1));

% Find the probabilities needed to create Joint/naive Bayes classifier
% Find out the percentage occurrence of each possible class(Ytr) value
% to do this divide number of occurrences by the length of the total array
% considered. Repeat this process to calculate all probabilities needed for
% classification
P_0 = length(Ytr(Ytr==0)) / YtrLength;
P_1 = length(Ytr(Ytr==1)) / YtrLength;

% Find probabilities for Joint Bayes classifier, these are the percentage
% occurrence rates of an (X1,X2) combination for a specific y value
% P(x1,x2|y). The naming logic specifies the x1 value, x2 value then y
% value.
jbc_P_000 = length(Xtr1(Xtr1==0 & Xtr2==0 & Ytr == 0))/Ytr0Length;
jbc_P_010 = length(Xtr1(Xtr1==0 & Xtr2==1 & Ytr == 0))/Ytr0Length;
jbc_P_100 = length(Xtr1(Xtr1==1 & Xtr2==0 & Ytr == 0))/Ytr0Length;
jbc_P_110 = length(Xtr1(Xtr1==1 & Xtr2==1 & Ytr == 0))/Ytr0Length;
jbc_P_001 = length(Xtr1(Xtr1==0 & Xtr2==0 & Ytr == 1))/Ytr1Length;
jbc_P_011 = length(Xtr1(Xtr1==0 & Xtr2==1 & Ytr == 1))/Ytr1Length;
jbc_P_101 = length(Xtr1(Xtr1==1 & Xtr2==0 & Ytr == 1))/Ytr1Length;
jbc_P_111 = length(Xtr1(Xtr1==1 & Xtr2==1 & Ytr == 1))/Ytr1Length;

% Find probabilities for naive Bayes classifier, these are the percentage
% occurrence rates of P(x1|y) and P(x2|y). The naming logic specifies the
% x value then y value.
nbc_x1_P_00 = length(Xtr1(Xtr1==0 & Ytr == 0))/Ytr0Length;
nbc_x1_P_10 = length(Xtr1(Xtr1==1 & Ytr == 0))/Ytr0Length;
nbc_x2_P_00 = length(Xtr1(Xtr2==0 & Ytr == 0))/Ytr0Length;
nbc_x2_P_10 = length(Xtr1(Xtr2==1 & Ytr == 0))/Ytr0Length;
nbc_x1_P_01 = length(Xtr1(Xtr1==0 & Ytr == 1))/Ytr1Length;
nbc_x1_P_11 = length(Xtr1(Xtr1==1 & Ytr == 1))/Ytr1Length;
nbc_x2_P_01 = length(Xtr1(Xtr2==0 & Ytr == 1))/Ytr1Length;
nbc_x2_P_11 = length(Xtr1(Xtr2==1 & Ytr == 1))/Ytr1Length;
```

Parts A and B have been finished in hand written form as seen below. The data in the tables is taken from the MATLAB results.

Bayes Classifiers

a) First construct probability tables using the data. These display how many times each combination occoured. (Their chance)

x_1	x_2	$\hat{P}(x_1, x_2 \mid y=0)$
0	0	1/8
0	1	1/8
1	0	3/8
1	1	3/8

x_1	x_2	$\hat{P}(x_1, x_2 \mid y=1)$
0	0	3/8
0	1	3/8
1	0	0
1	1	2/8

$\hat{P}(y=1) = 8/16 = 0.5$

$\hat{P}(y=0) = 1 - \hat{P}(y=1)$
$= 0.5$

Find class prediction and probability of test set. Use bayes rule.

$\hat{P}(y=c \mid x_1, x_2) = \hat{P}(x_1, x_2 \mid y=c) \times \hat{P}(y=c) / \hat{P}(x_1, x_2)$

Test set

x_1	x_2	y
0	1	1
1	0	1
1	1	0

for the feature values (x_1, x_2) make a class prediction and calculate the probability using the formula above. Note that this is a joint Bayes classifier. To get classification find the larger probability between $y=1$ and $y=2$ for current feature values.

$\hat{P}(y=1 \mid x_1 = 0, x_2 = 1) = 3/8 \times 0.5 / (1/8 \times 0.5 + 3/8 \times 0.5)$

$= 0.75$

classification

so with $\hat{P}(0,1 \mid y=1) \times \hat{P}(y=1) = 3/8 \times 0.5 = 0.1875$, larger
and $\hat{P}(0,1 \mid y=0) \times \hat{P}(y=0) = 1/8 \times 0.5 = 0.0625$, $y=1$ is larger $\therefore$ Classify As $y=1$ At .75

$\hat{P}(y=1 \mid x_1 = 1, x_2 = 0) = 0 \times 0.5 / (3/8 \times 0.5 + 0 \times 0.5) = 0$

classification
with $\hat{P}(1,0 \mid y=1) \times \hat{P}(y=1) = 0 \times 0.5 = 0$
and $\hat{P}(1,0 \mid y=0) \times \hat{P}(y=0) = 3/8 \times 0.5 = 0.1875$ larger
$y=0$ is larger $\therefore$ classify class as $y=0$ with 100% probability.
remember that the chance of the other y value is $1 -$ current probability.
so $1 - 0 = 1$

$\hat{P}(y=0 \mid x_1 = 1, x_2 = 1) = 3/8 \times 0.5 / (3/8 \times 0.5 + 2/8 \times 0.5)$

$= 0.6$

classification

$\hat{P}(1,1 \mid y=0) \times \hat{P}(y=0) = 3/8 \times 0.5 = 0.1875$ larger

$\hat{P}(1,1 \mid y=1) \times \hat{P}(y=1) = 2/8 \times 0.5 = 0.125$

$y=0 ?$, $\therefore$ Classify class as $y=0$ with 60% probability.

b) Naive Bayes classifier

First construct probability tables.

x_1	y	$\hat{P}(x_1 \mid y=c)$
0	0	$2/8$
1	0	$6/8$
0	1	$6/8$
1	1	$2/8$

x_2	y	$\hat{P}(x_2 \mid y=c$
0	0	$4/8$
1	0	$4/8$
0	1	$3/8$
1	1	$5/8$

$\hat{P}(y=1) = 8/16 = 0.5$ $\qquad$ $\hat{P}(y=0) = 1 - 0.5 = 0.5$

$$\hat{P}(y=c \mid x_1, x_2) = \hat{P}(x_1, x_2 \mid y=c) \times \hat{P}(y=c) / \hat{P}(x_1, x_2)$$

sub in test set values to formula to calculate probability and classification prediction.

$$\hat{P}(y=1 \mid x_1=0, x_2=1) = \frac{6/8 \times 5/8 \times 0.5}{(2/8 \times 4/8 \times 0.5 + 6/8 \times 5/8 \times 0.5)}$$

$$\simeq 0.7895$$ probability

now do class prediction

$\hat{P}(x_1=0, x_2=1 \mid y=1)\,\hat{P}(y=1) = 6/8 \times 5/8 \times 0.5 = 0.234$ larger

$\hat{P}(x_1=0, x_2=1 \mid y=0)\,\hat{P}(y=0) = 2/8 \times 4/8 \times 0.5 = 0.0625$

$y=1 >$ $\therefore$ classify as $y=0$ at 78.95% probability.

$$\hat{P}(y=1 \mid x_1=1, x_2=0) = \frac{2/8 \times 3/8 \times 0.5}{(6/8 \times 4/8 \times 0.5 + 2/8 + 3/8 \times 0.5)} = 0.2$$

classification

$\hat{P}(x_1=1, x_2=0 \mid y=1)\,\hat{P}(y=1) = 2/8 \times 3/8 \times 0.5 = 0.0468$

$\hat{P}(y=0)\,\hat{P}(x_1=1, x_2=0 \mid y=1) = 0.5 \times 6/8 \times 4/8 = 0.1875$ larger

$1 - 0.2 = 0.8$

$y=0 >$ $\therefore$ classify $y=0$ with current x_1, x_2 features. with 80% probability.

The fact this probability isnt 100% is a limitation of the Naive Bayes classifier compared to joint.

$$\hat{P}(y=0 \mid x_1=1, x_2=1) = \frac{6/8 \times 4/8 \times 0.5}{(6/8 \times 4/8 \times 0.5 + 2/8 \times 5/8 \times 0.5)}$$

$$= 0.705$$

$\hat{P}(x_1=1, x_2=1 \mid y=0)\,\hat{P}(y=0) = 6/8 \times 4/8 \times 0.5 = 0.1875$ larger

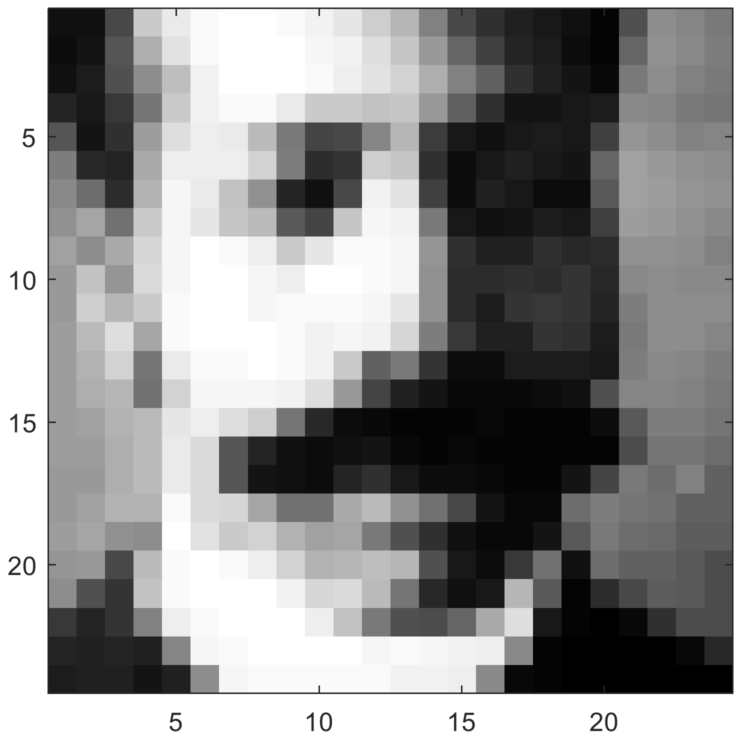
$\hat{P}(x_1=1, x_2=1 \mid y=1)\,\hat{P}(y=1) = 2/8 \times 3/8 \times 0.5 = 0.04687$

$y=0 >$, $\therefore$ classify as $y=0$ with 70.5% probability.

Part B: PCA & Clustering [45 marks]

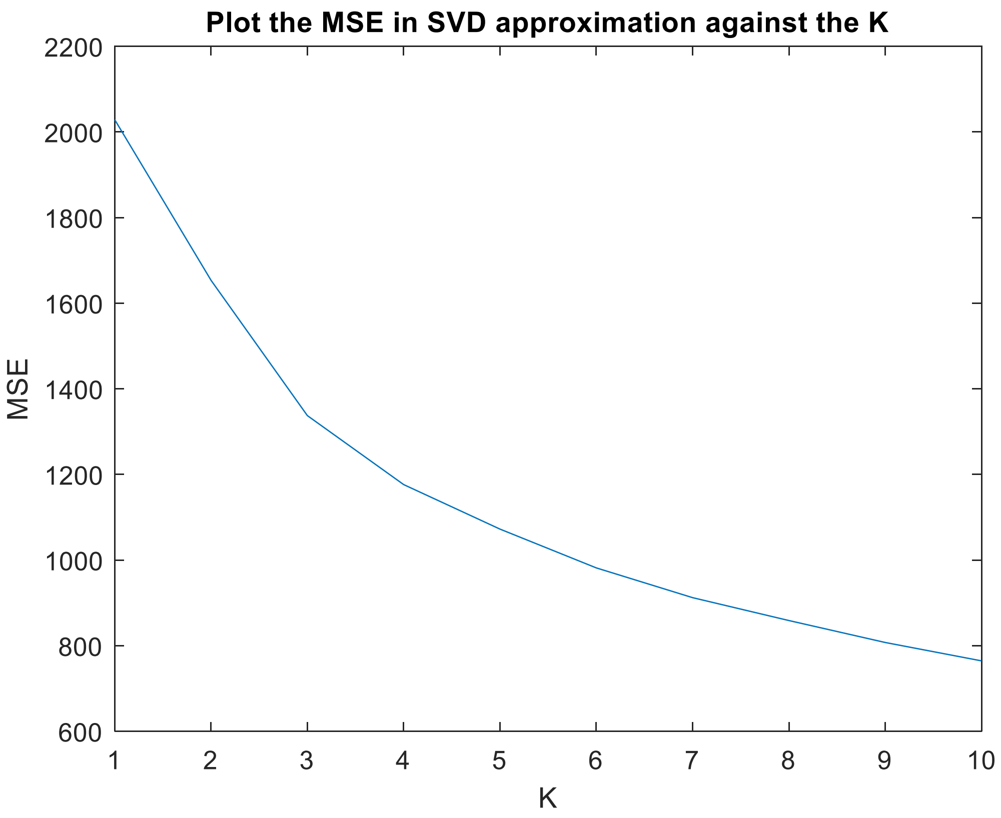
EigenFaces [25 Marks: 5 marks for each section (a) to (e)]

```
%% Section B: PCA
% Eigenfaces - Part A to E

% Load the data and display a few faces
X = load('data/faces.txt');
img = reshape(X(2,:),[24 24]);
imagesc(img); axis square; colormap gray;
```



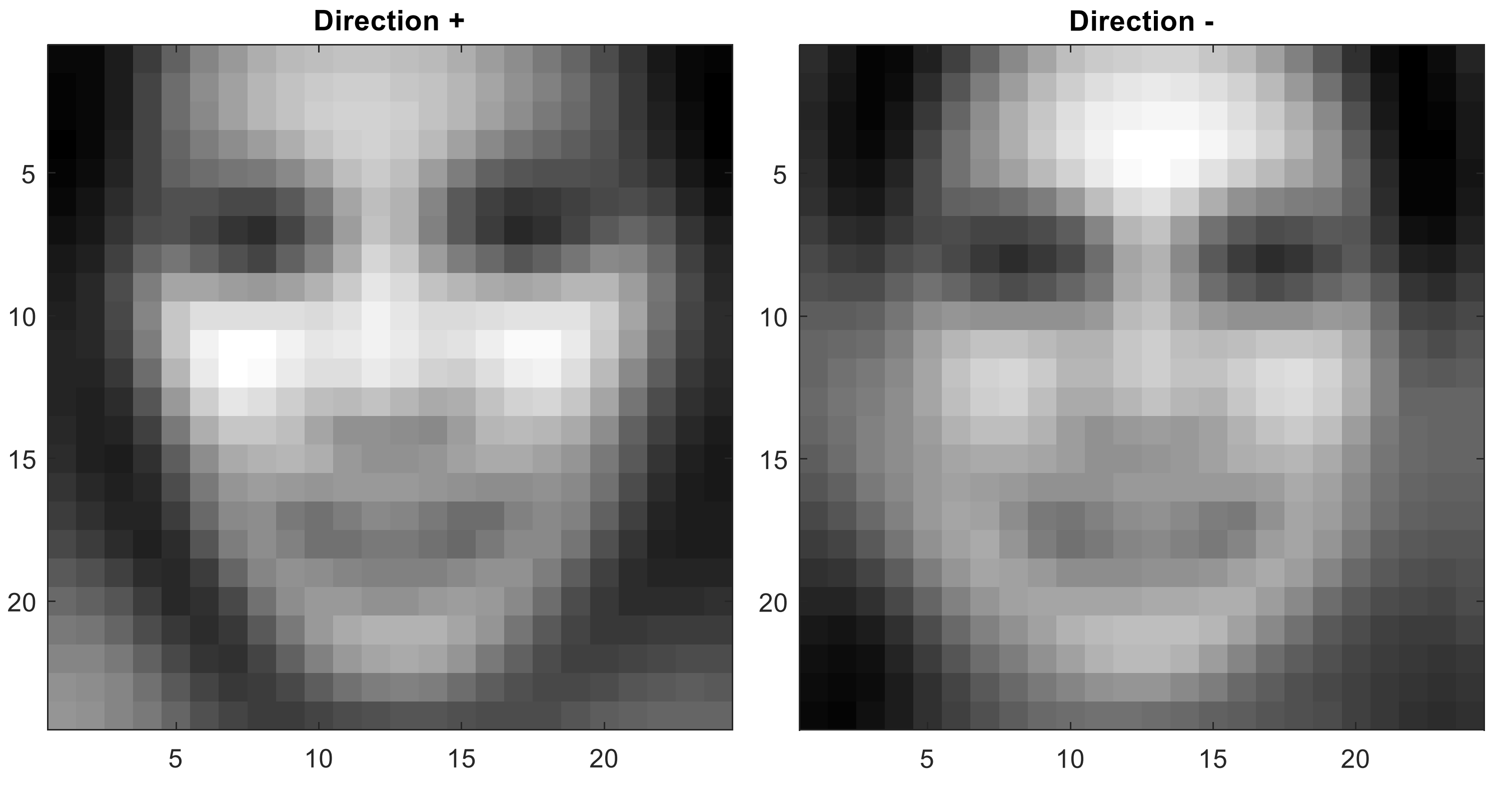
```
%% Part A: Subtract the mean of the face images to make the data zero-mean
mean_X = mean(X);
X0 = X - mean_X;
% Take the SVD of the data
[U, S, V] = svd(X0);
W = U * S;
```

```
%% Part B: Compute the mean square error in SVD's approximation
errors = zeros(1,10); % Store the MSE in SVD's approximation
K = 1:10;
for i = 1:length(K)
    [U_k S_k V_k] = svds(X0,K(i));
    X0_svd = U_k * S_k * V_k';
    mse_svd = mean(mean((X0 - X0_svd).^2));
    errors(i) = mse_svd;
end
figure(1);
plot(K,errors);
title('Plot the MSE in SVD approximation against the K');
xlabel('K'); ylabel('MSE');
```



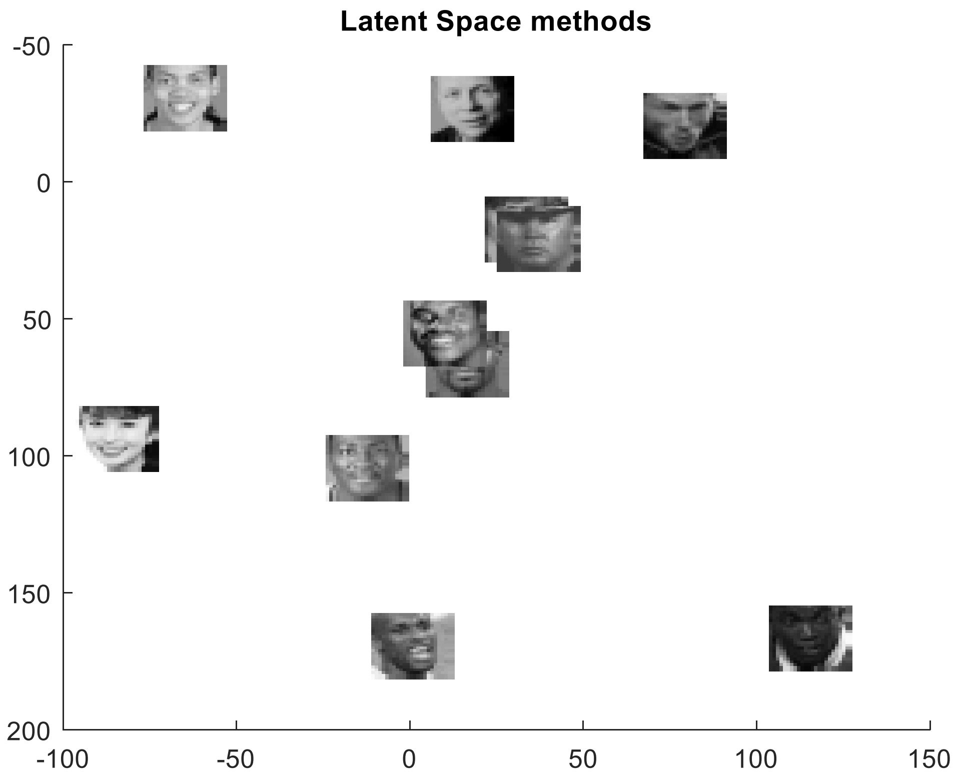
```
%% Part C: Display a first few principal directions of the data

alpha = 2 * median(abs(W(:,10))); % Scale factor
direction1 = reshape(mean_X + alpha * V(:,10)',[24,24]);
direction2 = reshape(mean_X - alpha * V(:,10)',[24,24]);
figure(2);
imagesc(direction1); axis square; colormap gray;
figure(3);
imagesc(direction2); axis square; colormap gray;
```

This results in the following figures.

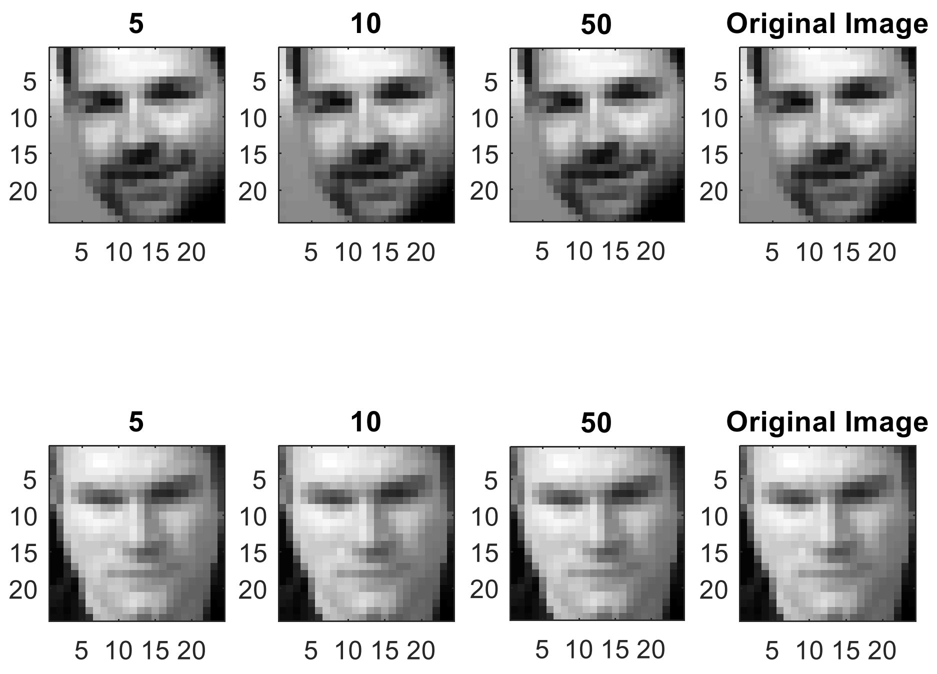


```
%% Part D: Latent Space methods

idx = 15:25; % random indices of data
figure(4); title('Latent Space methods');  hold on; axis ij; colormap(gray);
% find range of coordinates to be plotted
range = max(W(idx,1:2)) - min(W(idx,1:2));

% want 24x24 to be visible
scale = [200 200]./range;

for i=1:length(idx)
    imagesc(W(idx(i),1) * scale(1), W(idx(i),2) * scale(2),
reshape(X(idx(i),:),24,24));
end
```

The resulting figure is below.

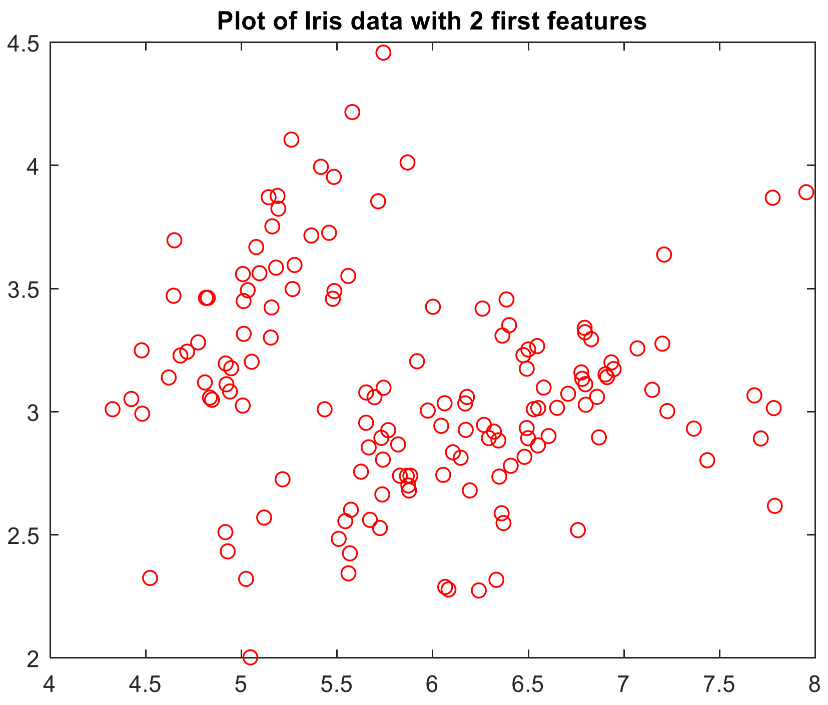


```
%% Part E: Choose two faces and reconstruct using only K principal directions

K_recover = [5,10,50]; % Use K principal directions
% Choose random two faces
indices = randperm(size(X0,1));
index1 = indices(1); index2 = indices(2);
img1 = X(index1,:);
img2 =  X(index2,:);
figure(5);
title('Construct image using K principal directions');

% Iterate over the 3 k values then reconstruct and plot the image using that k
% value, as well as the original image
for i = 1:length(K_recover)
    [U1 S1 V1] = svds(img1,K_recover(i));
    recovered_img1 = U1 * S1 * V1';
    subplot(2,4,i);
    imagesc(reshape(recovered_img1,24,24)); axis square; colormap gray;
    title([num2str(K_recover(i))]);
    [U2 S2 V2] = svds(img2);
    recovered_img2 = U2 * S2 * V2';
    subplot(2,4,i+4);
    imagesc(reshape(recovered_img2,24,24)); axis square; colormap gray;
    title([num2str(K_recover(i))]);
end
subplot(2,4,4); imagesc(reshape(img1,24,24)); axis square; colormap gray;
title('Original Image');
subplot(2,4,8); imagesc(reshape(img2,24,24)); axis square; colormap gray;
title('Original Image');
```

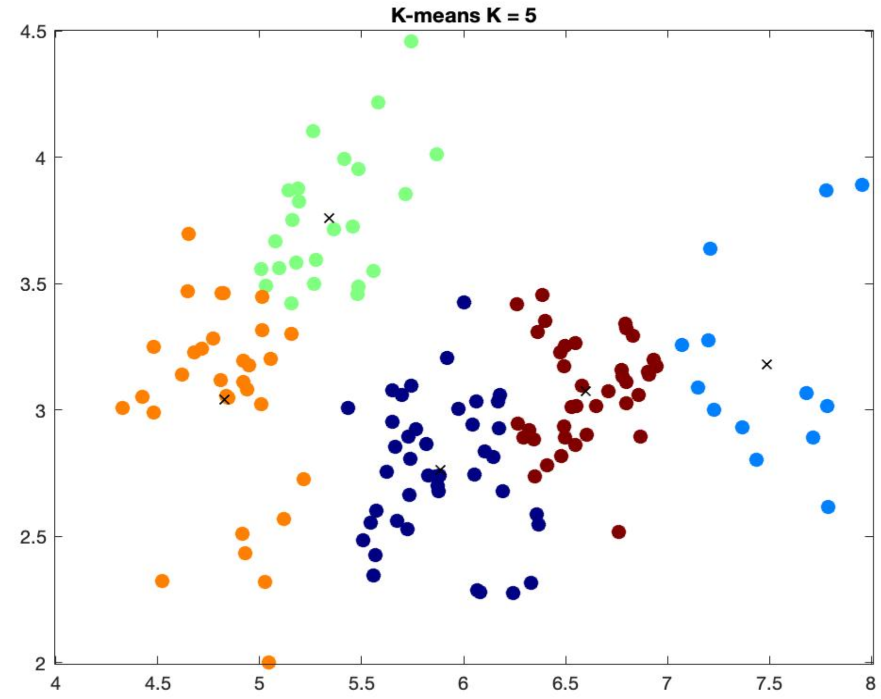


Clustering [20 Marks: 5 marks for each section (a) to (d)]

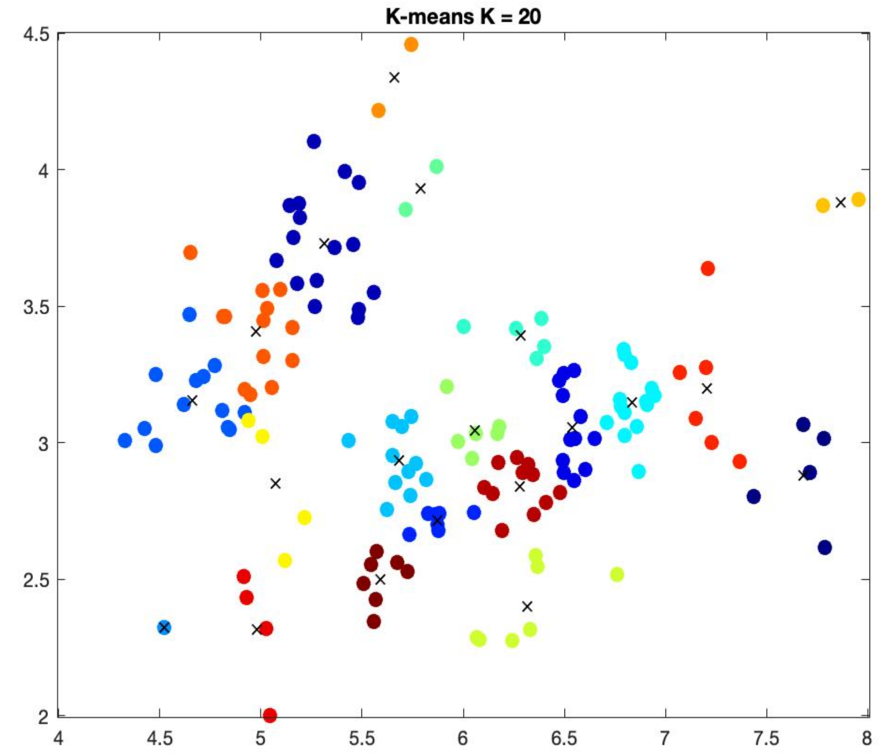
```
%% Part A: Load the usual Iris data with 2 features and plot

iris = load('data/iris.txt'); % Load Iris data
X_iris = iris(:,1:2);         % Use only two first features
figure(6);
plot(X_iris(:,1),X_iris(:,2),'ro');
title('Plot of Iris data with 2 first features');
```



```
%% Part B: K-Means on the data
% Choose the initialization with the best score

% Run k-means with k = 5 with farthest initialization
figure(7);
title('k-means on the data with k = 5');
[z5 c5 sumd5] = kmeans(X_iris,5,'farthest',100);
plotClassify2D([],X_iris,z5);
hold on
plot(c5(:,1),c5(:,2),'kx');
title('K-means K = 5');

% Run k-means with k = 20 with k++ intialization
figure(8);
title('k-means on the data with k = 20');
[z20 c20 sumd20] = kmeans(X_iris,20,'k++',100);
plotClassify2D([],X_iris,z20);
hold on
plot(c20(:,1),c20(:,2),'kx');
title('K-means K = 20');
```



K-means K = 5



K-means K = 20

```
%% Part C: Agglomerative clustering
% Using single linkage on Iiris data
[z5_min_agg join] = agglomCluster(X_iris,5,'min'); % k = 5
[z20_min_agg join] = agglomCluster(X_iris,20,'min'); % k = 20
figure(9);
subplot(1,2,1);
plotClassify2D([],X_iris,z5_min_agg);
title('K = 5');
subplot(1,2,2);
plotClassify2D([],X_iris,z20_min_agg);
title('K = 20');

% Using complete linkage on Iris data
[z5_max_agg join] = agglomCluster(X_iris,5,'max'); % k = 5
[z20_max_agg join] = agglomCluster(X_iris,20,'max'); % k = 20
figure(10);
subplot(1,2,1);
plotClassify2D([],X_iris,z5_max_agg);
title('K = 5');
subplot(1,2,2);
plotClassify2D([],X_iris,z20_max_agg);
title('K = 20');
```

The resulting plots of using single and complete linkage are demonstrated below.

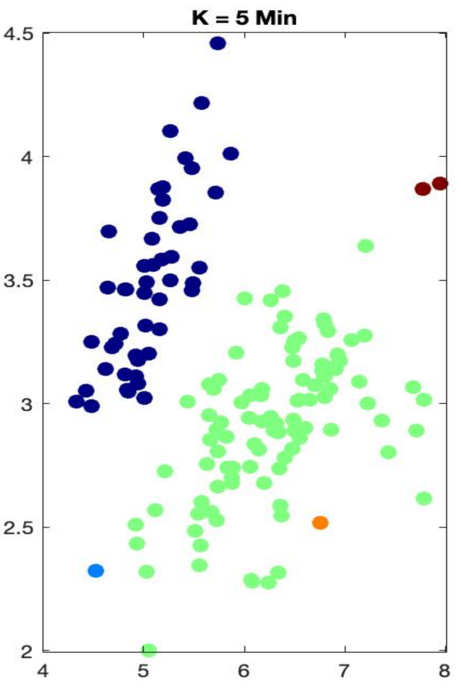
In hierarchical clustering, clusters have a tree like structure or parent child relationship. Here, the two most similar clusters are combine together and continue to combine until all objects are in the same cluster

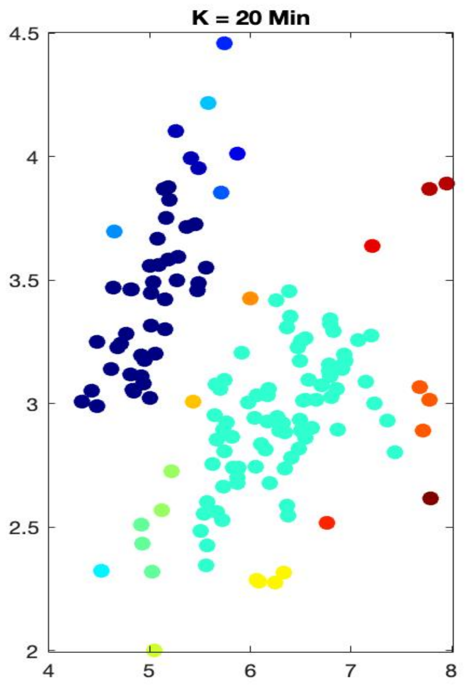
In single linkage clustering, we merge in each step the two clusters with the smallest minimum pairwise distance

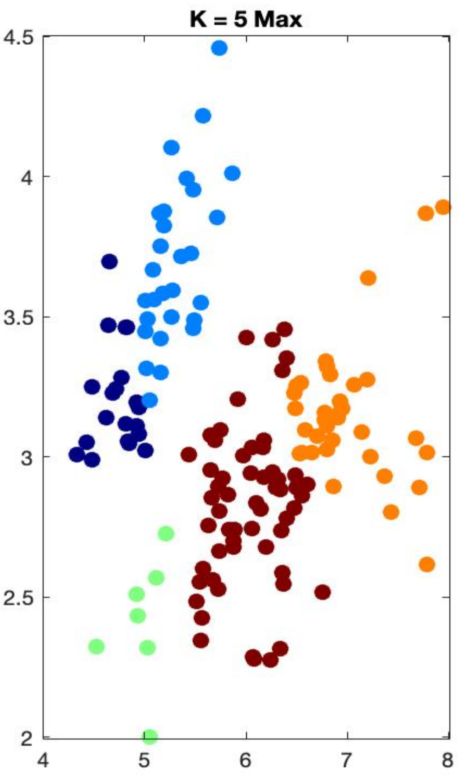
In complete linkage clustering, we merge in each step the two clusters with the smallest maximum pairwise distance

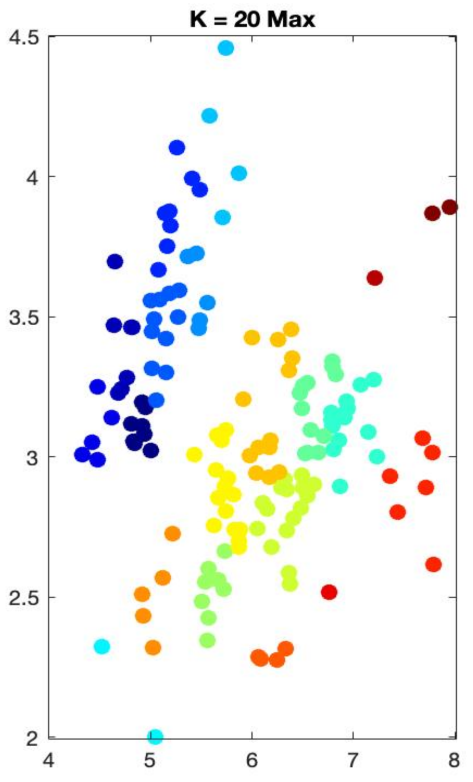
Therefore, it is observed from the graphs that the data points are distributed equally on clusters created using complete linkage compared to single linkage. In single linkage clustering, some clusters are very large.

Unlike hierarchical clustering mentioned above, k-means algorithm assigns each point to the cluster whose center (called centroid) is nearest. The center is the average of all the points in the cluster – that is, its coordinates are the arithmetic mean of each dimension separately over all points in the cluster.

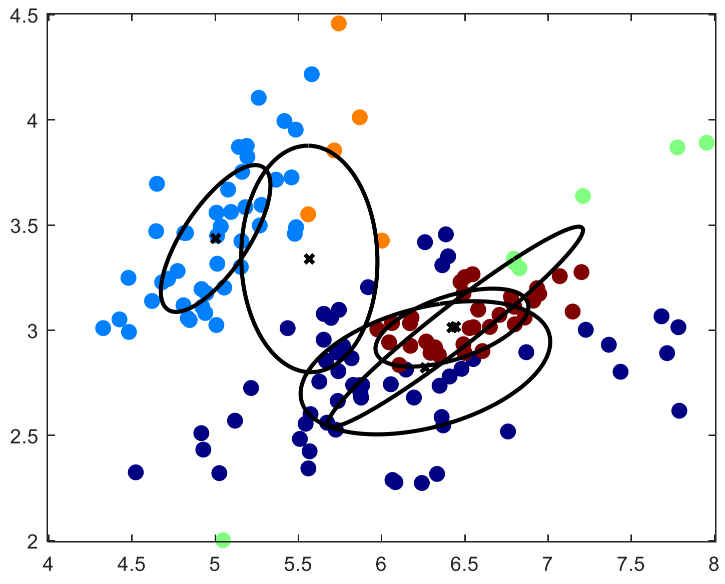

K = 5 Min
4.5
4
3.5
3
2.5
2
4
5
6
7
8

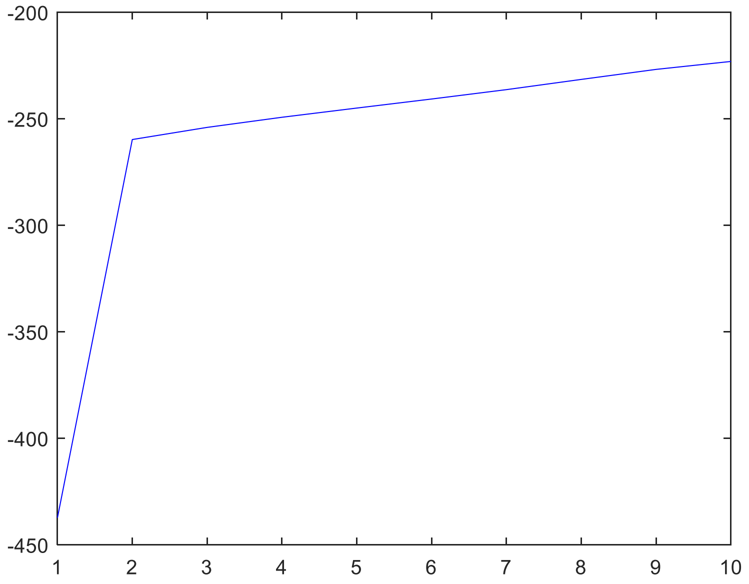

K = 20 Min
4.5
4
3.5
3
2.5
2
4
5
6
7
8

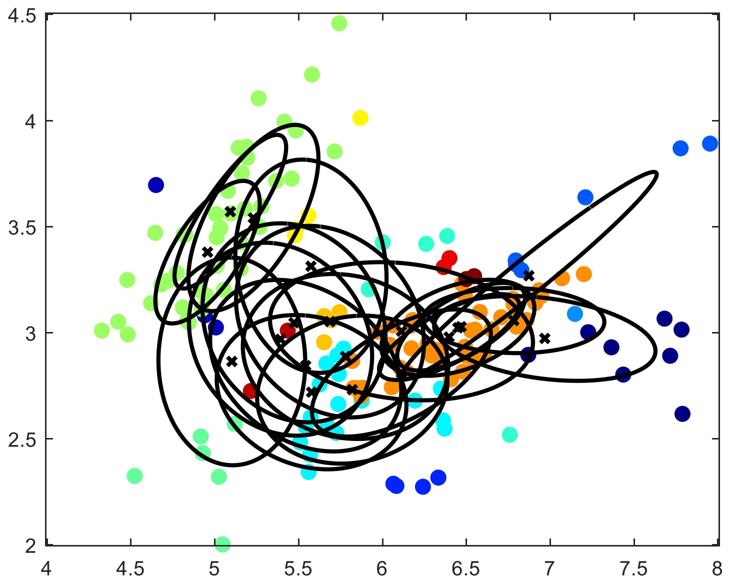

K = 5 Max
4.5
4
3.5
3
2.5
2
4
5
6
7
8

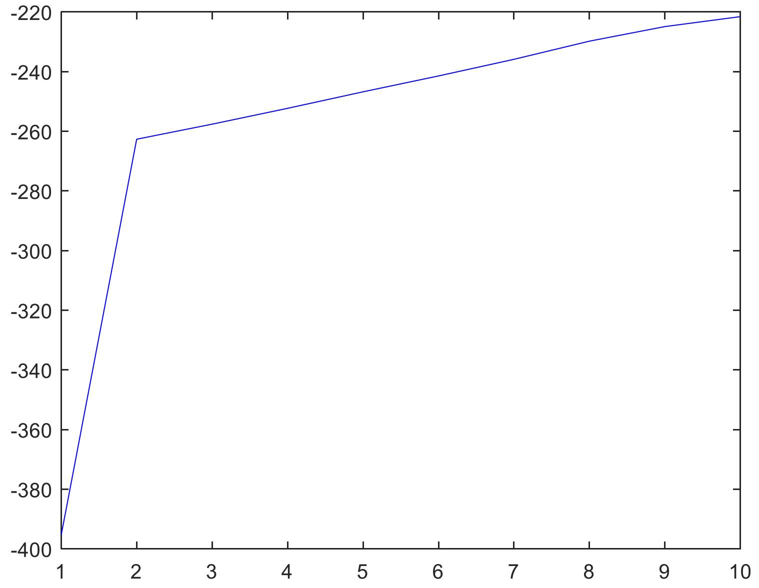

K = 20 Max
4.5
4
3.5
3
2.5
2
4
5
6
7
8

```matlab
%% Part D: Run the EM Gaussian Mixture Model
% Use doPlot = true in emCluster to observe the evolution of mixture
% components' locations and shapes

% EM Gaussian mixture model with k = 5
[z5_em,T,soft,ll] = emCluster(X_iris,5,'farthest',10);


% EM Gaussian mixture model with k = 20
[z20 em,T,soft,ll] = emCluster(X iris,20,'farthest',10);
```









The plots of EM Gaussian mixture model with 5 and 20 components and the loglikehood of the returned model against the number of iterations are illustrated above on the left and right side respectively.

Unlile two clustering methods above, Expectation Maximization works the same way as K-means except the data is assigned to each cluster with the weights being soft probabilities instead of distances. The advantage is that the model becomes generative as we define the probability distribution for each model

In three clustering methods, K-means is the most reasonable one as the region for each cluster colored by assignment has a better decision boundary compared to other two methods.